Mathematical and Algorithmic Puzzles
Dedicated to

LEARNING
MATHEMATICS
PUZZLES
ALGORITHMS
PHILOSOPHY
SPIRITUALITY

and

SHRIDEVI S. BHAT
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>iv</td>
</tr>
<tr>
<td>Preface</td>
<td>vi</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>xv</td>
</tr>
<tr>
<td>Bibliography</td>
<td>xvii</td>
</tr>
<tr>
<td>Philosopher’s Death</td>
<td>1</td>
</tr>
<tr>
<td>Hare and Tortoise</td>
<td>6</td>
</tr>
<tr>
<td>Mosquito to Elephant</td>
<td>15</td>
</tr>
<tr>
<td>Prisoners and a Light Bulb</td>
<td>25</td>
</tr>
<tr>
<td>Mog and Ooga</td>
<td>36</td>
</tr>
<tr>
<td>Larger or Smaller</td>
<td>41</td>
</tr>
<tr>
<td>Lion and Man</td>
<td>50</td>
</tr>
<tr>
<td>Stirring Tea</td>
<td>66</td>
</tr>
<tr>
<td>Hercules and Hydra</td>
<td>74</td>
</tr>
<tr>
<td>Nerdy Needle</td>
<td>84</td>
</tr>
<tr>
<td>Marriage Problem</td>
<td>95</td>
</tr>
<tr>
<td>Princess and Princes</td>
<td>101</td>
</tr>
<tr>
<td>Losing + Losing = Winning</td>
<td>110</td>
</tr>
<tr>
<td>Drunkard’s Dilemma</td>
<td>120</td>
</tr>
<tr>
<td>Uncountability</td>
<td>127</td>
</tr>
<tr>
<td>Prisoners and Hats</td>
<td>138</td>
</tr>
<tr>
<td>Prisoners and Boxes</td>
<td>147</td>
</tr>
<tr>
<td>Electrician and Wires</td>
<td>153</td>
</tr>
<tr>
<td>Monty Hall Problem</td>
<td>159</td>
</tr>
<tr>
<td>Poisoned Wine</td>
<td>168</td>
</tr>
<tr>
<td>Egg Dropping</td>
<td>178</td>
</tr>
<tr>
<td>Fox and Duck</td>
<td>186</td>
</tr>
<tr>
<td>Pirates and Gold Coins</td>
<td>193</td>
</tr>
<tr>
<td>Camels and Bananas</td>
<td>197</td>
</tr>
<tr>
<td>River Crossing</td>
<td>202</td>
</tr>
<tr>
<td>Sand Timers</td>
<td>213</td>
</tr>
<tr>
<td>Water Pouring</td>
<td>225</td>
</tr>
<tr>
<td>Circle of Death</td>
<td>236</td>
</tr>
<tr>
<td>Coin Weighing</td>
<td>257</td>
</tr>
<tr>
<td>Picture Hanging</td>
<td>275</td>
</tr>
<tr>
<td>Bridge Crossing</td>
<td>279</td>
</tr>
<tr>
<td>Horse Racing</td>
<td>288</td>
</tr>
<tr>
<td>Soldiers Firing</td>
<td>295</td>
</tr>
<tr>
<td>Majority Vote</td>
<td>304</td>
</tr>
<tr>
<td>Mouse in a Hole</td>
<td>311</td>
</tr>
</tbody>
</table>
“All life is problem-solving”, said Karl Popper, a famous philosopher. Life is in fact full of problems and we need to solve them as solving problems leads to greater happiness. However, it is literally impossible to come up with a single master method or God-formula to solve all problems. Nevertheless, it is possible to develop a set of principles, techniques, and strategies to solve problems belonging to particular classes. We learn such problem-solving techniques by carefully understanding and analyzing existing problems and their solutions.

This book is about problem-solving. But, why care for problem-solving?

[Why care for problem-solving?] Problems and problem-solvers are everywhere. Scientists solve scientific problems, farmers solve food problems, entrepreneurs solve real-world problems, doctors solve healthcare problems, lawyers/judges solve justice problems, police solve civil disorder problems, detectives solve crime problems, armed forces solve national security problems, software engineers solve programming problems, designers solve design problems, and so on. Most decisions we make involve some kind of understanding, analyzing, thinking, and problem-solving.

As an example, consider the study by Hanus et al. [Hanus et al., 2011]. A long and transparent glass tube was vertically attached to a chimpanzee’s cage. The bottom of the tube was closed and the top was open. Three metal rings held the tube firmly to the cage. A shelled peanut was dropped into the tube. The chimpanzee could not use its fingers to reach the shelled peanut as the tube was long. Also, there were no sticks inside the cage that the chimpanzee could use to remove the shelled peanut. Still, the brainy chimpanzee solved the tough problem. How?

The chimpanzee discovered a surprising solution. There was a water dispenser in the cage from which the chimpanzee drank water regularly. The chimpanzee sucked water from the dispenser and spit into the tube several times until the shelled peanut floated to the top. In this way, the clever chimpy used life-giving water as a tool to solve the unreachable shelled peanut problem.

We humans are where we are and we are what we are because of our learning and problem-solving skills and abilities. From time immemorial, our ancestors have solved thousands of problems, developed hundreds of problem-solving strategies, recorded and stored all this knowledge and wisdom, and passed those lessons to us. Using our problem-solving skills, we have discovered spoken languages, language scripts, agriculture, trade, money, metals, clothing, education, publishing, justice, civil order, philosophy, arts (literature, painting, music, and drama), and sciences (vehicles, electricity, artificial light, telephone, television, DNA as the source code of life, computer, and Internet). Learning the existing problem-solving techniques helps us to solve newer problems and also aids us to discover new problem-solving strategies.

The breathtaking beauty of learning is summarized in a verse from Śaṁskṛtam, the most beautiful literary language of the world.
Can’t be stolen by a thief, can’t be confiscated by government,
Can’t be divided among brothers as ancestral property, isn’t heavy to carry,
Keeps growing as and when spent continuously,
The wealth of learning is the greatest of all wealths.

This book is about mathematical and algorithmic problem-solving. But, why care
for mathematics and algorithms?

[Why care for mathematics?] Isaac Newton developed a mathematical theory to
explain the motions of the planetary systems (i.e., systems including planets, moons,
asteroids, meteoroids, and comets) assuming a gravitational force between two bodies.
Albert Einstein developed a mathematical theory to explain the motions of heavenly
bodies (planetary systems, star systems or galaxies, and interstellar gas clouds or
nebulae) assuming the warping of space and time by the physical bodies; Einstein’s
theory is more accurate than that of Newton. Niels Bohr, Erwin Schrödinger, Werner
Heisenberg, Max Born, and others developed the initial mathematical theory for
describing the properties of physical objects at the scale of atoms and subatomic parti-
cles. Alexander Friedmann and Georges Lemaître developed the initial mathematical
theory to show that the observable universe might have started from a primeval atom
and then might have continuously expanded. Alan Turing developed the simplest, the
most general, and the most-intuitive mathematical model of a computing machine to
perform computations, leading the way to the creation of physical computers. John
Horton Conway designed a cellular automaton that has four simple rules and it can
simulate the evolution of life on an infinite, two-dimensional orthogonal grid in unpre-
dictable ways which shows that simple laws can indeed create the most complicated
systems. In summary, mathematics has been used to understand nature at both large
and small scales, the creation of our universe, the theory of computing machines, the
evolution of life, and many more.

Mathematics has often been considered the queen of sciences and the language
of nature. Galileo Galilei wrote that the grand book of the universe was written in
the language of mathematics. Here is an anecdote [Littlewood, 1986] that beautifully
conveys the infinite power of mathematics. Mathematician Abram S. Besicovitch and
Dean of Chapel Harry Williams once asked mathematician John E. Littlewood what
God was doing before the creation. Littlewood replied “He was doing pure mathemat-
ics and thought it would be a pleasant change to do some applied.”

[Why care for algorithms?] An algorithm is a sequence of step-by-step unambigu-
ous instructions to solve a problem in a finite amount of time. The formal definition
of algorithm is that it is a halting Turing machine (that models deterministic com-
putation terminating after a finite number of timesteps). However, the meaning of
algorithm is used in a more general sense to also include nondeterministic computation such as parallel algorithms, probabilistic algorithms, randomized algorithms, and quantum algorithms. Algorithms are processes that need not take any input and need not give any output. But for simplicity we will assume that algorithms take some input and give some output.

What is the difference between a (mathematical) function, an algorithm, and a program? A function is a mapping from every element of an input set to an element of the output set. It is like a blackbox that maps elements without giving details on how exactly the output is obtained from input. For example, a sorting function maps an array of elements to a sorted array but it does not give information on how it sorts the array. An algorithm can be considered as an implementation of a function, i.e., giving details on the step-by-step sequence of instructions to transform the input data to output data. Hence, algorithms are also called procedures, methods, strategies, and solutions. For example, there are 100+ sorting algorithms that implement the exact same sorting function. Finally, a program is a sequence of instructions that need not terminate. All algorithms are programs but all programs are not algorithms. For example, a code segment that checks if an (input) app crashes or not is a program but not an algorithm because it can be mathematically proved that checking to see if an app crashes is an algorithmically unsolvable problem.

The mathematical formalization of algorithms, the construction of computers, and the development of Internet has been the most beneficial consequence of learning to humankind in the 20th century. Algorithms are used in operating systems, Internet, application software, computer graphics, computer vision, computational sciences, and machine learning. Algorithms are part of software used in hospitals, banks, schools, companies, organizations, vehicles, smartphones, email systems, search engines, navigational maps, and millions of other websites/apps/software/systems. It has become impossible to imagine a world where there are no algorithms, or computation in general.

Science, including mathematics, is the process and product of the discovery of patterns from a sequence of events, numbers, or objects. These patterns, based on different degrees of confidence in the evidences, are known by various names such as strategies, techniques, methods, concepts, ideas, axioms, postulates, formulas, theorems, algorithms, functions, principles, theories, laws, reality, facts, and truths. With deep understanding of these enigmatic patterns, maybe almost all arts and humanities can be reduced to psychology, maybe almost all psychology can be reduced to biology, maybe almost all biology can be reduced to chemistry, maybe almost all chemistry can be reduced to physics, maybe almost all physics can be expressed through mathematics, and maybe almost all mathematics can be implemented/simulated using computation. Hence, it might be possible that almost everything in all universe can be understood through mathematics and computation.

This book is really about mathematical and algorithmic puzzle-solving. But, why should we care for mathematical and algorithmic puzzles?

[Why care for mathematical and algorithmic puzzles?] There are several benefits from learning and solving mathematical and algorithmic puzzles.
The major benefit from solving puzzles are the feelings of thrill and joy. Being puzzled gives us the feeling of thrill. The sense of accomplishment got from solving a puzzle gives us joy. Mystery, suspense, thriller, and horror films are among the most popular movie genres because they produce these same feelings of thrill and joy in audience who like such films. Puzzle addicts get to the highest level of thrill and joy by injecting challenging puzzles and solutions into their minds. They have the heavenly opportunity to fly with ecstasy in the enigmatic words of utter confusion, infinity, recursion, paradoxes, time loops etc.

Other side benefits include the following. [Mathematical and algorithmic thinking.] We develop mathematical and algorithmic thinking from practicing puzzle-solving. This art of logical reasoning helps us not only to solve several technical problems but also to tackle real-life problems in a systematic way. [Entertainment.] Mathematical puzzles keep us entertained during long travels, waiting times, and during boring college classes. [Health.] All kinds of simple not-tough-to-solve puzzles are known to reduce our stress levels thereby improving our mental health. [Interviews.] Solving puzzles helps students in answering some of the toughest interview questions of elite companies.

[What is this book about?] This book does not cover tens of varieties of puzzles such as English language riddles, jigsaw puzzles, crossword puzzles, word puzzles, sudoku puzzles, mechanical puzzles, chess puzzles, paper-and-pencil puzzles, matchstick puzzles, card puzzles, rope puzzles, lateral thinking puzzles, construction puzzles, tiling puzzles, lock puzzles, folding puzzles, and diagram puzzles.

This book presents serious mathematical puzzles that are mostly counterintuitive. The major difference between standard technical problems and puzzles is the entertaining and intriguing factor. Technical problems seem only to entertain experts from a specific technical domain. On the other hand, puzzles arouse curiosity in everyone as they relate to day-to-day real-life scenarios and characters with a slight touch of fantasy. This book consists of fun puzzles.

The presented puzzles are simultaneously entertaining, challenging, intriguing, and haunting. This book introduces its readers to counterintuitive mathematical ideas and revolutionary algorithmic insights from a wide variety of topics such as logic, probability, modular arithmetic, geometry, proof techniques, differential and integral calculus, combinatorics, topology, binary search (or binary representation), divide-and-conquer, dynamic programming, distributed algorithms, theory of computation, theory of games, theory of optimal stopping, theory of group testing, theory of comparative advantage (from economics), and theory of special relativity (from physics).

The presented solutions that are discovered by many mathematicians and computer scientists are highly counterintuitive and show supreme mathematical beauty. These counterintuitive solutions are intriguing to a degree that they shatter our preconceived notions, shake our long-held belief systems, debunk our fundamental intuitions, and finally rob us of sleep and haunt us for a lifetime. Multiple ways of attacking the same puzzle are presented which teach the application of elegant problem-solving strategies.
This book is about mathematical puzzle-solving.
Multiple algorithmic solutions are provided for each puzzle.

[What is the theme of this book? How is this book unique?] There are several books on mathematical ideas and puzzles. How is this book different? The theme, uniqueness, and the significance of the book comes from the two aspects:

1. [Counterintuitive puzzles.] Existing puzzle books do not usually cover counterintuitive puzzles as these puzzles mostly require deep mathematical analysis. In contrast, counterintuition is at the heart of this book. Counterintuitive puzzles/solutions seem counterintuitive because they involve concepts such as infinity, continuum, logical paradoxes, contradictions, self-reference, recursion, nonlinear functions, inaccurate definitions, circular definitions, invalid arguments, and probability. It is possible that humans have not evolved enough to naturally understand and analyze some of these concepts. Counterintuitive puzzles/solutions show that our perceptions are deficient, our experiences are insufficient, our memories are unreliable, our beliefs are biased, our analyses are intuitive, our predictions are erroneous, and hence our minds are fallible. Analyzing counterintuitive puzzles is an excellent way to learn mathematical thinking because the solutions to such puzzles prove us wrong again and again, forcing us to switch our reasoning tool from ordinary intuitive thinking to rigorous mathematical thinking.

2. [Multiple solutions, deep solutions, and generalized solutions.] Existing puzzle books contain at most two/three solutions per puzzle. In contrast, this book presents at most nine solutions per puzzle. As many solutions as possible are provided for each puzzle.

   The three main approaches of teaching puzzles are: (i) **Topic-centric approach:** In this approach, ideas are organized according to topics/domains. (ii) **Technique-centric approach:** In this approach, ideas are organized according to design techniques or puzzle-solving strategies. (iii) **Puzzle-centric approach:** In this approach, ideas are organized according to puzzles and each puzzle is taught through as many techniques as possible. It is important to note that this approach is relatively new in the field of teaching puzzle-solving.

   This book does not follow the topic-centric approach. The reason is that identifying the topic(s) a puzzle belongs to is part of puzzle-solving and categorizing puzzles based on topics is like robbing the readers of the opportunity and excitement of discovering the puzzle domains. This book does not also follow the technique-centric approach. The reason is that classifying puzzles based on puzzle-solving strategies restricts readers to bounded ways of thinking because many puzzles are known to have multiple solutions belonging to seemingly-unrelated branches of mathematics. This book follows the puzzle-centric approach. The reason is that the approach will be tremendously useful in understanding how different techniques attack the same puzzle.

   Existing puzzle books consider puzzles more as a source of recreation and pastime and less as a serious topic for in-depth understanding, analysis, and contemplation. This book aims to view puzzle-solving as a topic for intense study and
There are three major classes of mathematical puzzle-solving books: 

(i) **Read books**: A read book is one that is easy to read. Books that discuss history, philosophy, motivation, process, and impact of mathematics and mathematical puzzles are read books. 

(ii) **Work books**: A work book is one that involves readers in solving puzzles. Many of the standard puzzle books are work books and several of the puzzles in these books are solvable with a decent amount of effort in time and thoughts. 

(iii) **Study books**: A study book is one that cannot be easily worked out or read as it contains deep mathematical solutions that are difficult to discover by most people. There are very few such scholarly books in this category. This book is majorly a study book. Hence, it is fruitful to carefully study the deep solutions given in this book.

This book aims to teach *how to think, not what to think*, in the same spirit of the following nice quote “Education is kindling of a flame, not the filling of a vessel.” The focus in the book is more on the thinking process, reasoning technique, and problem-solving strategies than just obtaining answers and results. Like the saying “Journey is more important than the destination”, solutions and problem-solving techniques are more important than the answers.

Most contemporary books do not provide generalized solutions (e.g.: coin weighing puzzle, water pouring puzzle, bridge crossing puzzle, and so on). The aim of this book is to give generalized solutions and not just to solve some specific instances of puzzles. The beauty of generalization is that those solutions can be programmed to generate output for any instance in a fraction of a second. We can then compare different generalized solutions to check which of the generalized solutions is theoretically and/or practically faster.

**[Who should read this book?]** The target audience of the book includes:

- Computer science [under]graduates/professors interested in mathematical puzzles.
- Mathematics [under]graduates/professors interested in mathematical puzzles.
- Industry professionals interested in mathematical puzzles.

The book is ideal for those who have a passion/obsession for learning mathematical puzzles via algorithmic solutions. Good background in undergraduate-level mathematics and algorithms is required. Recommended textbooks for undergraduate mathematics and algorithms are given in the bibliography section.

**[How are the solutions in the book organized?]** Existing mathematical puzzle books are written using flat essay-styled articles with almost no sections and subsections. In contrast, solutions in this book are organized using a hierarchy of sections and subsections for easy understanding and searching. Generalizations of solutions are given. Extensive citations and references are provided. Philosophies and take-home lessons are discussed wherever possible. Tables, diagrams, and plots are used to enhance the readers’ learning experience. Important observations, intermediate results, and non-optimal answers are highlighted using a light red box and desired answers and generalized results are highlighted using a light green box.

**[How to solve a mathematical puzzle?]** Algorithm SOLVINGAPUZZLE shows an
outline of standard puzzle-solving strategies for solving a mathematical puzzle.

<table>
<thead>
<tr>
<th>SOLVING A PUZZLE (puzzle)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> An interesting mathematical puzzle.</td>
</tr>
<tr>
<td><strong>Output:</strong> Multiple solutions to the puzzle and a couple of new related puzzles.</td>
</tr>
</tbody>
</table>

**[I. Pre-solving phase] .................................**
- **Be cool.** Have mental cool, think positive, relax, be happy, and keep giggling.
- **Believe in yourself.** Believe in yourself that you can shatter the puzzle into pieces.
- **Read puzzle.** Read the puzzle completely multiple times and understand all words.
- **Understand puzzle.** Understand the deep meaning of the puzzle clearly.
- **Write observations.** Catalogue all data that is given, unknown, and to be computed.

**[II. Solving phase] .................................**
- **Find domain.** Find the puzzle area such as probability, calculus, algorithms, etc.
- **Relate puzzle.** Relate the puzzle to a previous known concept/idea/problem.
- **Restate puzzle.** Restate / vary / modify the puzzle to get different perspectives.
- **Ask questions.** Ask lots of questions on data, assumptions, and conditions.
- **Solve small instances.** Solve for smaller instances and generalize the pattern.
- **Solve examples.** Solve examples to find the principles behind the solution.
- **Solve special cases.** Work on the special cases as they might be easier to solve.
- **Write thoughts.** Write and analyze the thought process in detail on a paper.
- **Draw diagrams.** Draw neat diagrams, figures, graphs, tables, functions, charts
- **Create presentations.** Create beautiful color presentations to get more intuitions.
- **Eliminate possibilities.** Eliminate possibilities that cannot lead to the solution.
- **Prove or disprove.** Try to use induction/ contradiction/counterexample/etc.
- **Reverse engineer.** Start from the to-be-proved result and move backwards.
- **Contemplate puzzle.** Think on the puzzle while walking, traveling, sleeping, etc.
- **Code.** Write computer programs, simulate, get plots, and analyze the results.
- **Discuss.** Brainstorm the puzzle with friends, classmates, teachers or anyone.
- **Develop strategies.** Think about many more new strategies not presented here.

**[II. Post-solving phase] .................................**
- **Verify solution.** Rigorously test / verify every single line of the solution.
- **Revise solution.** Revise / review solution to understand the problem deeper.
- **Generalize.** Generalize the puzzle and solution to make it work for all instances.
- **Find other solutions.** Go to Phase II and find a different solution to the puzzle.
- **Learn more solutions.** Learn more solutions from books, Internet, discussions, etc.
- **Create puzzles.** Create new related puzzles by modifying the variables / conditions.

**return** all solutions and new related puzzles

It is extremely important to have a proper mindset to solve a puzzle. A person who strongly believes in herself, has mental cool, thinks positive, is relaxed and happy has a much better chance of cracking a puzzle compared with a person having a complete opposite personality. The reader is strongly recommended to study the revolutionary book by Joseph Murphy [Murphy, 2012], which is one of the greatest books ever written on the infinite power of belief.
It is interesting to see that often there are multiple ways to solve a puzzle. Due to the variegated thinking and experience of people, seemingly-unrelated approaches are attempted to solve a puzzle culminating in multiple solutions. For example, Elisha Scott Loomis’ book \cite{Loomis, 1972} consists of 370 proofs of the Pythagorean theorem.

There is an old proverb: “Give a man a fish and you feed him for a day; teach a man to fish and you feed him for a lifetime.” Adapting the proverb to problem-solving, we can state: “Give a woman a solution and you solve her current problem; teach a woman problem-solving and you solve most of her problems.” The reader is recommended to focus more on the problem-solving techniques a solution uses compared with the solution itself. This book teaches problem-solving implicitly through multiple detailed solutions and explicitly through core ideas, problem-solving techniques, analyses, and take-home lessons.

Each solution will have a core idea. Once the core idea is uncovered, deriving the details can be relatively easy. The most significant step in solving a puzzle is to discover that one core idea using problem-solving techniques which is the crux of the solution.

Our aim should not just be in obtaining a solution to a given problem, rather in understanding the problem very, very deeply. An ordinary puzzle-solver is contented when she/he discovers/learns a correct (but mostly inefficient) solution to the puzzle. On the other hand, a puzzle-lover is interested in the deep understanding of the puzzle itself and discovering a solution does not satiate her/his unfathomable thirst for understanding the puzzle. So, the puzzle-lover keeps trying to learn/discover simpler, better, and more interesting problem-solving methodologies. A puzzle-lover’s favorite question is “Can we do better?” Hence, a puzzle-lover’s true aim is in understanding the puzzle very deeply knowing that solutions are natural byproducts of deep understanding of a given puzzle.

Our aim is to understand a given puzzle very deeply.

We should always keep this thought in mind when we attempt to solve a puzzle.

This a lesson you should heed: try, try, try again.
If at first you don’t succeed, try, try, try again.
If at second you don’t succeed, try, try, try again.
If at third you do succeed, that’s like the first rain.

You should go for knowledge greed, learn and learn in a chain.
More and more ways if you breed, you realize there’s lot to gain.
Give your mind this daily feed, experience your bulging brain.
If you subscribe to this creed, you will avoid the risk of pain.

Sow this beautiful puzzle seed, make your mind a thought fountain.
Bleed and bleed and do your deed, be the greatest thinker insane.
If you learn at this speed, your life will be a roller-coaster train.
Finally at last when you are freed, take another and start again.
Acknowledgements

I sincerely thank the great mathematics teachers of my school and college: Shanthakumari, Vathsala, Shridhar, Prasad, A. V. Shiva Kumar, P. C. Nagaraj, K. S. Anil, and C. Chamaraju for making me fall in love with mathematics — the queen of all sciences. I thank the author Anany Levitin for making me fall in love with algorithms — the princess of all sciences, through his legendary textbook on algorithms.

I am infinitely indebted to Rezaul A. Chowdhury for introducing me to the world of organized research and rigorous algorithmic thinking. I am eternally grateful to Yanhong A. Liu for her neverending support and encouragement. I am very thankful to Michael A. Bender for exponentially increasing my love towards algorithms.

An infinite thanks to Andrey Gorlin for teaching me different ways to attack puzzles. I cannot thank him enough for contributing multiple solutions in this book and for his unparalleled feedback for improvising this book. Many many thanks to Jackson Menezes, Rong Zou, Quan Xie, and Sourabh Daptardar for lengthy and informative discussions on puzzle-solving.

This book would be impossible without the effort of many people. From the bottom of my heart, I thank my students who have worked on the respective concepts/topics as given below.

- Shubham Jindal (MS): exceptional, extraordinary, and unparalleled work on the implementations and pseudocodes of Takakazu-Euler-Tait, Herstein-Kaplansky, Schubert, Busche, Knuth (three algorithms: one based on a recurrence and two others based on inverse permutations), Uchiyama, Halbeisen-Hungerbühler, Gelgi (both recursive and non-recursive), and Booth’s algorithms for the circle of death puzzle; solution of the jealous husbands puzzle
- Keyur Potdar (MS): state diagram solutions (without and with epsilon transitions) for solving the sand timers puzzle; development and implementation of a complicated variant of all-source best-first search algorithm to compute all flips-optimal solutions; generalization, implementation, and pseudocode of Wert’s algorithm for the coin weighing puzzle
- Varshneya Rajagopal (MS): implementations and diagrams of McCarthy-Minsky, Waksman, Balzer, Gerken, Mazoyer, and Clergue-Verel-Formenti’s solutions for the soldiers firing puzzle; recursive formulation, implementation, and pseudocode of Dyson’s algorithm for the coin weighing puzzle
- Chanpreet Kaur (MS): implementation of Morito-Salkin’s algorithm for solving a linear Diophantine equation; and the analysis of the $n$-containers algorithm for the water pouring puzzle
- Qinglu Du (BS): discussion of solutions for the stirring tea puzzle
- Pratik Thorwe (MS), Yashi Sharma (MS), and Priyanka Dighe (MS) for implementing the percentile sort algorithm for the horse racing puzzle
- Mandar Mahajan: understanding proofs based on Cauchy sequences and supremum in the uncountability puzzle
• Racha Harshita (MS): implementation of the branch-and-bound algorithm for the bridge crossing puzzle
• Aishwarya Chintala (MS): implementation of Burde’s algorithm and performance evaluation for the circle of death puzzle
• Xingtong Zhou (BS): geometric interpretation of the princes and princess puzzle
• Sai Bhavana Ambati (MS): implementations of combination Parrondo games to check if they are winning or losing in the losing + losing = winning puzzle
• Nikhil Sira (MS) and Mohit Khandelia (MS): performance evaluation for the prisoners and a light bulb puzzle

I thank Darshan Palasamudram for discussions on the improved multiple dynamic counters strategy from prisoners and a light bulb puzzle and Aaron Gregory for discussions on the improved divide-and-conquer solution from electrician and wires puzzle. Other students who were involved in the discussion of puzzles are: Sahil Shah (MS): helping create Figures 87 and 88 and Sankalp Paradkar (MS): helping create Figure 84.

Thanks to Anany Levitin, Steven Skiena, Leo Bachmair, and Peter Winkler for helpful feedback. I wholeheartedly thank my biological parents Ganapathi Hegde and Varada Ganapathi for all their sacrifices while bringing me up. Finally, I thank everyone and everything that is directly or indirectly responsible for this work.
[Prerequisites: Undergraduate Mathematics.] Recommended textbooks:
- Trigonometry [Larson, 2022] and Calculus [Thomas et al., 2018].
- Linear Algebra [Anton et al., 2019].
- Probability [Pishro-Nik, 2016, Dekking et al., 2005, Tijms, 2012].
- Discrete Mathematics [Epp, 2019].

[Prerequisites: Undergraduate Computer Science.] Recommended textbooks:
- Data Structures [Goodrich et al., 2014, Weiss, 2010].

[Puzzles and Solutions.] The book is written in the puzzles-solutions format. Books that are written in this format and are comparable to the book are:

[Puzzles and Solutions in a Specific Topic.] All puzzles in these books are from a particular topic.

[Solutions for a Specific Puzzle.] Each of these books is on a single puzzle.
- [Wapner, 2005, Rosenhouse, 2009]

[Collection of Beautiful Mathematical Ideas.] These books contain essay-style expositions of several powerful mathematical ideas but do not follow the standard problem-solutions format. Hence, these books are majorly read for ideas, motivation, history, philosophy, and a summary of mathematics and not solely for puzzle-solving, except for some books by Martin Gardner, the greatest of all puzzlers.
- Books by the great Martin Gardner.

[Tutorials on Problem-Solving.] These books teach mathematical problem-solving principles, techniques, and strategies through problems and solutions. The problems that are used to teach problem-solving are not extremely challenging or entertaining.

[Quantitative Aptitude.] Books on mathematical reasoning are:
- [Aggarwal, 2021]
Philosopher’s Death

Problem

An intelligent princess was in love with a philosopher. After learning the relationship between the princess and the philosopher, the king vowed to give the philosopher a death sentence. The next day, the philosopher was brought to the court.

The king commanded the philosopher to speak one sentence, which would decide his fate as follows:
- If the sentence was true, then he would be hanged.
- If the sentence was false, then he would be beheaded.
- If the sentence was not a proposition (a proposition is a declarative sentence for which a truth value can be assigned), then he would be fed to lions.
- If the sentence was inherently self-contradictory, then he would be shot with an arrow.
- If the sentence was a conjecture or a hypothesis (i.e., neither proven nor unproven mathematical statement), then he would be left to die from starvation.
- If the sentence was not stated within ten minutes, then he would be poisoned.
- If the sentence could not be understood easily (e.g.: sentence from an unknown language or sentence spoken in a low audible voice), then he would thrown from a tall mountain.

The philosopher mentally preparing himself to face death for his love, applied his philosophic acumen and came up with a sentence. The sentence was so profound that the king let the philosopher free, united the princess and the philosopher, and finally made the philosopher his successor to the throne.

What was the philosopher’s sentence?

Solution

The problem seems impossible to solve. Surprisingly, there is a beautiful logical solution to the puzzle.

Common solutions (incorrect)

The most common wrong answer given by people to solve the problem is a sentence that does not have a truth value. For example, “The painting is beautiful.” Questions, proposals, suggestions, commands, exclamations, and opinions typically cannot be assigned with truth values. We know that if the philosopher states a sentence for which a truth value cannot be assigned, then he will be fed to lions. Hence, this type of sentence cannot solve the problem. The second most common wrong answer given is a sentence that is self-contradictory. For example, “This statement is false.” We
know that if the philosopher states a logical paradox, then he will be shot with an arrow. Hence, this type of sentence also cannot solve the problem.

Logic solution

The problem clearly belongs to logic, a branch of philosophy, mathematics, and computer science that deals with the principles of truth and falsehood. The problem states that the sentence given by the philosopher must not be true, must not be false, must not be contradictory, however, must have a truth value. It is clear that the answer cannot be so simple.

We address the problem in three parts. First, we give fundamentals of logic necessary to understand the solution. Second, we present the solution. Third and finally, we analyze a couple of issues with the solution.

[Fundamentals.] A proposition is a sentence that is either true or false, but not both. For example, “1 + 1 = 3” is a false proposition. A philosophical paradox is a sentence that seems self-contradictory but might be either true or false. A logical paradox is a self-contradictory sentence that if its assumptions are false then its conclusion is true and if the assumptions are true then its conclusion is false.

We define a few sets of sentences as follows:

- **S** = Not propositions. E.g.: “The painting is beautiful”, “Who is Isaac Newton?”, “Wow!”
- **T** = True propositions. E.g.: “Some husbands cheat on their wives.”
- **F** = False propositions. E.g.: “Politicians never lie.”
- **C** = Conjectures and hypotheses. E.g.: Riemann hypothesis, Goldbach conjecture.
- **L** = Logical paradoxes or self-contradictions. E.g.: “This statement is false.”
- **B** = Sentences that are both true and false.
- **N** = Sentences that are neither true nor false.
- **O** = Sentences whose truth values oscillate depending on context.

The sets of sentences that are least understood are **L**, **B**, **N**, and **O**. They are not rigorously defined and the similarities and differences between them are still unclear.

One extremely intriguing set of sentences that we focus on in our solution is the set **O**. It consists of sentences whose truth value oscillates depending on the consequences of stating the statement based on context/situation. It might seem that the set **O** is the same as set **L**, but this is not the case. We will now show that the answer to our problem must a sentence from this set **O**.

If the philosopher states a sentence from any of sets **S**, **T**, **F**, or **L**, then he will surely die. If he states a sentence from **B** but not **P** i.e., (B − P) or from **N** but not **P** i.e., (N − P), then the philosopher can possibly escape death. However, we don’t really understand the sets **B** and **N**. This leaves us with only one set i.e., set **O**.

If the philosopher states a sentence from the set **O**, then he might escape death, because the truth value of his sentence could oscillate (between, say, true, false, and logical paradox) depending on context (i.e., the way he is going to die). So, let’s construct sentences from this set **O** and analyze the consequences.
[Solution.]

Philosopher's sentence can be any of the following:
- “I will not be hanged.”
- “I will be beheaded.”
- “Either I speak truly and I will be beheaded or I speak falsely and I will be hanged.”

We show the analysis for the sentence “I will not be hanged.” The remaining two sentences can be analyzed similarly.

Consider the philosopher’s sentence: “I will not be hanged.” This sentence is a proposition as it definitely has a truth value of either true or false based on how the philosopher is going to die. It is clearly not a logical paradox as it contains no inherent self-contradiction. Hence, the philosopher will neither be fed to lions nor be shot with an arrow. This means that the only two options which remain are death by hanging or beheading.

If the sentence is true, then “I will not be hanged” is true. This implies that the philosopher would be beheaded. But, he can only be beheaded if his sentence is false. The assumption that the sentence is true led to the conclusion that the sentence is false. Hence, the entire system is a logical paradox.

If the sentence is false, then “I will not be hanged” is false. This implies that the philosopher would be hanged. But, he can only be hanged if his sentence is true. The assumption that the sentence is false led to the conclusion that the sentence is true. Hence, the entire system is a logical paradox.

It is very important to observe that the sentence “I will not be hanged” is not a logical paradox. However,

The system consisting of the following three statements is a logical paradox:
- If the philosopher's sentence is true, then he would be hanged.
- If the philosopher's sentence is false, then he would be beheaded.
- The philosopher's sentence is “I will not be hanged.”

The philosopher's sentence is a good example of a sentence from the set $O$ that oscillates between various truth values depending on context. The truth value of the sentence oscillates between true and false indefinitely depending on the temporary decision of how the philosopher will be killed. Hence, the philosopher can neither be hanged nor beheaded.

The sentence is so profound that the king let the philosopher free, united the princess and the philosopher, and finally made the philosopher his successor to the throne. The beautiful princess and the intelligent philosopher lived happily ever after.

[Analysis.] The solution we discussed is not mathematically rigorous. In fact, the problem is open to a wide variety of philosophical interpretations and analyses based on various assumptions and definitions of philosophical ideas. Here, we analyze three questions related to our solution:

[Question 1. What is the truth value of the philosopher's statement?]
The truth value of the philosopher’s statement depends on the way he is going to die in the near future. In other words, the truth value of the philosopher’s statement depends on the context, i.e., the parameters such as time and methods of death sentence.

A truth value cannot be assigned to the statement at the instantaneous moment when the philosopher states his conditional proposition.

[Question 2. What is the truth value of the king’s promise?]
The philosopher’s statement does not belong to any of the categories listed by the king. The philosopher’s conditional proposition lies outside the framework of the king’s conditional death sentence. So, the king can decide to either punish or pardon the philosopher. If the king decides to punish, he can punish the philosopher through any method whatsoever. For example, the king could even burn or bury the philosopher alive. In the problem, the king let the philosopher free.

A truth value cannot be assigned to the king’s promise because the philosopher’s statement is outside the system of classes and conditions for the death sentence and the king can decide anything when the statement is outside the system.

[Question 3. What should the king do?]
We analyze the question in both logical and moral settings.

Logically speaking, in order to keep his promise, the king has to determine the truth value of the philosopher’s statement and punish him based on the stated conditions. If the statement is outside the conditions of the death penalty, the king is free to kill or let live the philosopher.

Morally speaking, the king has to let the philosopher live. This is what the king did, which implies that the king is morally good.

Problems
1. [Two envelopes and a gold coin.] There are two envelopes with messages written on them. One of the envelopes contains a gold coin and the other envelope is empty. Here are the messages written on the two envelopes:
   - [Envelope 1.] “At least one of the two messages is false.”
   - [Envelope 2.] “The gold coin is in this envelope.”
   The gold coin is factually inside the second envelope. How?
2. [Thomson’s lamp.] Suppose a lamp is switched on for 1/2 minute, switched off for 1/4 minute, switched on for 1/8 minute, switched off for 1/16 minute, etc. Is the lamp on or off at the end of one minute?
3. [Grandi’s series.] Let \( S = 1 - 1 + 1 - 1 + 1 - 1 + \cdots \). Consider the four scenarios:
   - (a) \( S = (1 - 1) + (1 - 1) + (1 - 1) + \cdots = 0 + 0 + 0 + \cdots = 0 \).
   - (b) \( S = 1 + (-1 + 1) + (-1 + 1) + \cdots = 1 + 0 + 0 + 0 + \cdots = 1 \).
   - (c) \( S = 1 - (1 - 1 + 1 - 1 + \cdots) \Rightarrow S = 1 - S \Rightarrow 2S = 1 \Rightarrow S = \frac{1}{2} \).
   - (d) Add the results from (a) and (b). We get \( S + S = 0 + 1 \Rightarrow 2S = 1 \Rightarrow S = \frac{1}{2} \).
What is the value of $S$? Is it 0 or 1 or $\frac{1}{2}$?

4. [Sets $B$ and $N$.] What are the similarities and differences between the two classes $B$ and $N$? How do they compare with class $L$? What are the example sentences that belong to the two sets $B$ and $N$?

References

Hare and Tortoise

Problem

The hare and the tortoise decided to have a race to completely settle a 2500-year old question as to who is the fastest. The race is shown in Figure 1. The referee stands at the origin. The hare and the tortoise start simultaneously at the origin and run in opposite directions. The finish line for the hare and the tortoise is 1 km each from the origin. The participants run at a constant speed of 1 m/s from start to finish. The race ends but there is a problem.

The referee declares that the race is a tie because both the hare and the tortoise reached their finish lines simultaneously. The hare announces that it won the race and the tortoise has lost. The tortoise announces that it won the race and the hare has lost. The claims of the referee, hare, and tortoise are contradictory but we know that none of them are lying. No other race in all history has caused this much confusion and perplexity.

Who has really won the race? Why is there so much confusion?

Solution

The puzzle looks like a silly riddle. But don’t be fooled. This is an advanced mathematical puzzle related to our enigmatic physical world. The solution to this puzzle leads to the understanding of one of the greatest mysteries of the universe.

How can we know what domain the puzzle belongs to? The result of the race is different for different observers and it is clear that none of them are lying. This gives a clue that the problem involves observers or frames of reference and hence belongs to physics.

We make two assumptions regarding the physical reality of our universe:

1. [Laws of physics.] The laws of physics are the same for every observer.
2. [Speed of light.] The speed of light in the vacuum is the same for every inertial observer.

These are carefully selected assumptions for which there is plenty of experimental evidence. From several experiments, it has been confirmed that light travels in the vacuum at the speed of $c = 299,792,458$ m/s. The speed of light in a medium other than vacuum is less than $c$. For example, the speed of light in a few mediums are: air: $\frac{c}{1.0003}$, water: $\frac{c}{1.33}$, crown glass: $\frac{c}{1.52}$, diamond: $\frac{c}{2.42}$, and lead: $\frac{c}{2.6}$. In our puzzle, though there is air, we will use the light speed as $c$ for simplicity and without loss of generality.

[Observations.] Let $O, A$, and $B$ be the origin, the finish line of the hare, and the finish line of the tortoise, respectively. We assume that when the participants reach their respective finish lines, they stop instantaneously and wait until the race ends. There are three frames of reference: the referee, the hare, and the tortoise.

Let

$v_{o,f} = \text{relative velocity of object } o \text{ in the } f \text{ frame of reference}$

$s_{o,f} = \text{relative length of object } o \text{ in the horizontal direction in the } f \text{ frame of reference}$

$t_{e,f} = \text{relative time of event } e \text{ in the } f \text{ frame of reference}$

$v_{hare,referee} = v_{tortoise,referee} = 1 \text{ m/s}$

$s_{OA,referee} = s_{OB,referee} = 1000 \text{ m}$

$t_{hare \text{ travels } OA,referee} = t_{tortoise \text{ travels } OB,referee} = 1000 \text{ s}$

Light delay solution (incorrect)

We all know that the stars we see every night might be thousands of years old because it takes light some non-zero finite time to reach us. In fact, everything that we see in our daily life is its past. Suppose you are standing at a distance of 1 meter from your friend. The light reflected from your friend takes $\frac{1}{c}$ seconds to reach you. So when you see your friend at a distance of 1 m, you see your friend as she or he was $\frac{1}{c}$ seconds ago.

We use the above idea to solve the puzzle. When both the hare and the tortoise reach their finish lines, the light reflected from them reaches the referee simultaneously. Hence, the referee declares that the race is a tie. It takes light $\frac{1000}{c}$ seconds to reach from both $A$ and $B$ to $O$. When the hare reaches its finish line $A$, it sees that the tortoise has not yet reached its finish line $B$ because it takes light some non-zero time to reach $A$. It takes light $\frac{2000}{c}$ seconds to reach from $B$ to $A$. Similarly, when the tortoise reaches its finish line $B$, it sees that the hare has not yet reached its finish line $A$ because it takes light some non-zero time to reach $B$. It takes light $\frac{2000}{c}$ seconds to reach from $A$ to $B$. As per this reasoning, we find that

$t_{hare \text{ travels } OA,referee} = t_{tortoise \text{ travels } OB,referee} = \left(1000 + \frac{1000}{c}\right) \text{ seconds}$

$t_{hare \text{ travels } OA,hare} = t_{tortoise \text{ travels } OB,tortoise} = 1000 \text{ seconds}$

$t_{tortoise \text{ travels } OB,hare} = t_{hare \text{ travels } OA,tortoise} = \left(1000 + \frac{2000}{c}\right) \text{ seconds}$
Hence, the race is a tie according to the referee, the hare won the race according to the hare, and the tortoise won according to the tortoise. Even if the hare and the tortoise run in the same direction and have the same finish line, they would think that they won the race and the other participant lost. This is because it takes light non-zero time to reach from one participant to another. This time delay distorts simultaneity.

The solution seems convincing. However, it is incorrect. Do you see a flaw in this solution? A simple flaw is described here.

If each observer can compute the distance between two points $X$ and $Y$ accurately and then make a note of the time delay i.e., the time taken by light to travel between the two points, then we can adjust the time intervals of all events. Applying this technique to our puzzle, we can eliminate the extra $\frac{1000}{c}$ and $\frac{2000}{c}$ factors as we know for sure that this delay is accounted by light and not by the two participants. With the accounting for the time delay, the time intervals of all events as measured by the observers are the same. That is,

\[
\begin{align*}
\text{t}_{\text{hare travels } OA, \text{referee}} &= \text{t}_{\text{tortoise travels } OB, \text{referee}} = 1000 \text{ seconds} \\
\text{t}_{\text{hare travels } OA, \text{hare}} &= \text{t}_{\text{tortoise travels } OB, \text{hare}} = 1000 \text{ seconds} \\
\text{t}_{\text{hare travels } OA, \text{tortoise}} &= \text{t}_{\text{tortoise travels } OB, \text{tortoise}} = 1000 \text{ seconds}
\end{align*}
\]

There is a more fundamental flaw with this solution. The flaw will be clear when we see the correct solution to this puzzle. Figuring out the fundamental flaw with this solution is left to the reader as an exercise.

**Einstein’s special relativity solution**

During the start of the 20th century, Albert Einstein developed his theory of relativity that shattered our common perceptions on space, time, motion, matter, energy, and gravity. In this section, we use Albert Einstein’s special theory of relativity that deals with space, time, motion, and light for solving the puzzle.

We need to understand some fundamentals of the special relativity theory before proceeding to solve the puzzle. This is not because there is advanced mathematics in the theory but just that the consequences of the theory are highly counter-intuitive.

**[Inertial frames of reference.]** We define a frame of reference as *inertial* if it is at rest or if it is moving at a constant velocity with respect to the frame at rest. No mechanical or optical experiment can be performed inside a closed frame of reference to know whether the frame is at rest or moving with a constant velocity. Every object or frame of reference is at rest in its own frame i.e., $v_{f,f} = 0$, for any frame $f$. In our puzzle, the referee, the hare, and the tortoise can be approximated as inertial frames of reference, though they are not (Why?).

For subsequent discussions, we consider two inertial frames of reference to explain a few fundamental concepts of the special relativity:

1. **[Rest-frame.]** A frame of reference at rest.
2. **[Car-frame.]** A car frame of reference moving with a uniform velocity of $u \text{ m/s}$, w.r.t. the rest-frame.
[Clocks.] A clock is an instrument to measure time with periodic ticks. In the long history of humankind, different clocks have been used to measure time with varying degrees of accuracy. For example, heartbeats, sundials, water clocks, hourglass, candle clocks, oil lamp clocks, mechanical clocks, pendulum clocks, spring-driven clocks, electric clocks, quartz clocks, and atomic clocks. As of the date of this writing, atomic clocks are the most accurate clocks.

There are no perfect physical clocks. Hence, in our future discussions, we use a theoretically and mathematically precise clock to compare time intervals in different inertial frames. However, any physical clock that is least affected by motion and gravity can be used.

[Relative time.] Voltaire asked: “What is that without which nothing can be done, and with which many do nothing?”. One of the answers is time. Time is the gift of Albert Einstein to science.

In this section, we will show that the time interval of an event in a moving frame of reference appears to slow down compared to the time interval of the same event at rest.

We will use a type of theoretical clock called light clock to measure time. The light clock is constructed as follows. There are two mirrors: top and bottom, separated by a distance of $h$ units. A light photon starts from the bottom mirror and hits the top mirror, reflects, and returns back to the bottom mirror. The process continues. We consider this round-trip time interval as a clock tick.

There is a light clock $C$ in the rest-frame. There is a light clock $C'$ in the car-frame. We want to find the relationship between a clock tick $\Delta t$ in $C$ in the rest-frame and a clock tick $\Delta t'$ in $C'$ of the car-frame as measured from the rest frame.

![Figure 2: Left: A clock tick $\Delta t$ as measured in the rest-frame. Right: A clock tick $\Delta t'$ in the car-frame as measured from the rest-frame.](image)

Consider a clock tick $\Delta t$ in $C$ in the rest-frame as shown in the left of Figure 2. We have

$$\Delta t = \frac{\text{distance}}{\text{photon speed}} = \frac{2h}{c}$$

(1)

Suppose a car-frame is moving with a uniform relative velocity of $u$ m/s towards the right. Then, the photon’s travel route (i.e., the zig-zag path) as seen from the
rest-frame is shown in the right part of Figure 2. Consider a clock tick $\Delta t'$ in $C'$ in the car-frame as measured from the rest-frame. We have
\[
\Delta t' = \frac{\text{distance}}{\text{photon speed}} = \frac{2H}{c}
\]
\[
= \frac{2}{c} \sqrt{h^2 + \left(\frac{u\Delta t'}{2}\right)^2} \quad \text{(Substituting $H$ using the Pythagorean theorem)}
\]
\[
= \frac{2}{c} \sqrt{\left(\frac{c\Delta t}{2}\right)^2 + \left(\frac{u\Delta t'}{2}\right)^2} \quad \text{(Substituting $h$ using Equation 1)}
\]

Simplifying, we can measure the relative time (time dilation) using the formula
\[
\Delta t' = \frac{\Delta t}{\sqrt{1 - \left(\frac{u}{c}\right)^2}}. \quad (2)
\]

**[Relative space.]** We show that the length of an object in an inertial frame appears to contract in the direction of motion compared to the length of the same object at rest.

![Diagram](image)

Figure 3: Left: Length $d$ as measured in the rest-frame. Right: Length $d'$ in the car-frame as measured from the rest-frame.

Consider the clock tick $\Delta t$ in $C$ in the rest-frame as shown in Figure 3 left. We have
\[
\Delta t = \frac{\text{distance}}{\text{photon speed}} = \frac{2d}{c}. \quad (3)
\]

The car-frame is moving with a uniform relative velocity of $u$ m/s towards the right. Then, the photon’s travel route as seen from the rest-frame is shown in the right part of Figure 3. Consider a clock tick $\Delta t'$ in $C'$ in the car-frame as measured from the rest-frame. Let $\Delta t'_1$ and $\Delta t'_2$ be the times of travel between the left to the right mirror, and from the right to the left mirror respectively. Then, we can write the following equations involving $\Delta t'_1$ and $\Delta t'_2$.

Distance traveled from the left to the right mirror = $d' + u\Delta t'_1 = c\Delta t'_1$. 

Distance traveled from the right to the left mirror = \( d' - u\Delta t' = c\Delta t' \).

Simplifying the equations above, we find that \( \Delta t'_1 = \frac{d'}{(c-u)} \) and \( \Delta t'_2 = \frac{d'}{(c+u)} \). Thus,

\[
\Delta t' = \Delta t'_1 + \Delta t'_2 = \frac{d'}{c-u} + \frac{d'}{c+u} = \frac{2d'}{c\left(1 - \left(\frac{u}{c}\right)^2\right)}. \tag{4}
\]

We substitute the value of \( \Delta t' \) from Equation 3 and the value of \( \Delta t' \) from Equation 4 into Equation 2 to get

\[
\frac{2d'}{c\left(1 - \left(\frac{u}{c}\right)^2\right)} = \frac{2d}{c\sqrt{1 - \left(\frac{u}{c}\right)^2}}.
\]

Simplifying, we can measure the relative length (length contraction) using the formula

\[
d' = d\sqrt{1 - \left(\frac{u}{c}\right)^2} \tag{5}
\]

[Relative velocity.] Suppose a car-frame is moving with a uniform velocity of \( v_{\text{car,rest}} = u_1 \text{ m/s w.r.t a rest-frame} \). The car fires a bullet that moves in the same direction as the car with a uniform velocity of \( v_{\text{bullet,car}} = u_2 \text{ m/s w.r.t the car-frame} \). Then, the relative velocity of the bullet w.r.t rest-frame is computed as

\[
v_{\text{bullet,rest}} = \frac{u_1 + u_2}{1 + \left(\frac{u_1 u_2}{c^2}\right)} \tag{6}
\]

The derivation of the relative velocity formula mentioned above is left to the reader as an exercise. When objects or frames of reference are moving in any two directions (might not be parallel), the relative velocity formula gets horrendously complicated. The reader need not worry about the general case because for the puzzle we only need to understand the relative velocity of objects moving in parallel lines.

[Solution.] We now solve our puzzle borrowing the counter-intuitive ideas on time (time dilation) and space (length contraction) from the special theory of relativity. In our puzzle, there are 3 inertial frames of reference: the referee (the absolute rest frame), the hare (the moving frame w.r.t the rest frame), and the tortoise (the moving frame w.r.t the rest frame).

[Referee measurements.] Please refer to Figure 1. The observations of the referee are:

\[
\begin{align*}
  v_{\text{hare,referee}} &= v_{\text{tortoise,referee}} = 1 \text{ m/s} \\
  s_{\text{OA,referee}} &= s_{\text{OB,referee}} = 1000 \text{ m} \\
  t_{\text{hare travels OA,referee}} &= t_{\text{tortoise travels OB,referee}} = 1000 \text{ s}
\end{align*}
\]

[Hare measurements.] According to the hare, it is at rest, the referee travels \( AO \) distance and the tortoise travels \( AB \) distance with uniform velocities. We will make use of the formulas of the special relativity here.
Using the relative velocity formula (Equation [6]), we get
\[ v_{\text{referee}, \text{hare}} = 1 \text{ m/s.} \] (Due to symmetry)
\[ v_{\text{tortoise}, \text{hare}} = \frac{v_{\text{referee}, \text{hare}} + v_{\text{tortoise}, \text{referee}}}{1 + \frac{(v_{\text{referee}, \text{hare}} \times v_{\text{tortoise}, \text{referee}})}{c^2}} = \frac{1 + 1}{1 + \left(\frac{1}{c^2}\right)} = \frac{2}{1 + \left(\frac{1}{c^2}\right)} \text{ m/s.} \]

Using the length contraction formula (Equation 5), we get
\[ s_{\text{AO}, \text{hare}} = s_{\text{OA}, \text{referee}} \sqrt{1 - \left(\frac{v_{\text{referee}, \text{hare}}}{c}\right)^2} = 1000 \sqrt{1 - \left(\frac{1}{c}\right)^2} \text{ meters.} \]
\[ s_{\text{AB}, \text{hare}} = s_{\text{AB}, \text{referee}} \sqrt{1 - \left(\frac{v_{\text{referee}, \text{hare}}}{c}\right)^2} = 2000 \sqrt{1 - \left(\frac{1}{c}\right)^2} \text{ meters.} \]

Using distance and speeds, we can compute the race times as
\[ t_{\text{referee travels AO, hare}} = \frac{s_{\text{OA}, \text{hare}}}{v_{\text{referee}, \text{hare}}} = 1000 \sqrt{1 - \left(\frac{1}{c}\right)^2} \text{ seconds} = 999.9999999999943674971973190782365467335308446966807 \ldots \]
\[ t_{\text{tortoise travels AB, hare}} = \frac{s_{\text{AB}, \text{hare}}}{v_{\text{tortoise}, \text{hare}}} = 1000 \sqrt{1 - \left(\frac{1}{c}\right)^2} \times \left(1 + \left(\frac{1}{c}\right)^2\right) \text{ seconds} = 1000.000000000000005563250280268092083496065621982534754444 \ldots \]

We see that
\[ t_{\text{referee travels AO, hare}} < t_{\text{tortoise travels AB, hare}} \]

Hence, according to the hare, it reached its finish line A faster than the tortoise reached its finish line B. Hence, according to the hare, it is the winner and the tortoise is the loser.

[Tortoise measurements.] The tortoise measurements are symmetric to the measurements of the hare. The tortoise will see that
\[ t_{\text{referee travels BO, tortoise}} < t_{\text{hare travels BA, tortoise}} \]

According to the tortoise, it reached its finish line B faster than the hare reached
its finish line $A$. Hence, it believes that it is the winner and the hare is the loser.

Confusion in the race result is due to the relativity of velocity (relative velocity), space (length contraction), and time (time dilation). Relativity is due to the constant speed of light (in vacuum) measured from any inertial frame of reference. Constant speed of light is due to... Sorry, nobody knows!

[Generalization.] Let’s generalize the puzzle. Let, according to the referee, the distance to be covered by the hare and the tortoise be $\ell$ meters. Let the speeds of the hare and the tortoise as measured by the referee be $v$ m/s. Then, the time taken by the hare and the tortoise to reach their respective finish lines as measured by the referee is $(\frac{\ell}{v})$ seconds. The referee declares that the race is a draw. But, the hare and the tortoise think that they won the races, respectively.

Suppose $\alpha = 1 + \left(\frac{x}{2}\right)^2$ and $\gamma = \sqrt{1 - \left(\frac{x}{2}\right)^2}$. When $u \in [0, c]$, then $\alpha \in [1, 2]$ and $\gamma \in [0, 1]$. The measurements of different observers are summarized in Table 1.

<table>
<thead>
<tr>
<th>Observer:Observed</th>
<th>Referee</th>
<th>Hare</th>
<th>Tortoise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referee</td>
<td>$v = 0$</td>
<td>$s = 0$</td>
<td>$u = v$</td>
</tr>
<tr>
<td></td>
<td>$t_{\text{race ends}} = (\frac{\ell}{v})$</td>
<td>$s_{OA} = \ell$</td>
<td>$s_{OB} = \ell$</td>
</tr>
<tr>
<td></td>
<td>$t_{\text{hare travels } OA} = (\frac{\ell}{v})$</td>
<td>$t_{\text{hare travels } OB} = (\frac{\ell}{v})$</td>
<td>$t_{\text{tortoise travels } OB} = (\frac{\ell}{v})$</td>
</tr>
<tr>
<td>Hare</td>
<td>$v = u$</td>
<td>$s = 0$</td>
<td>$v = u$</td>
</tr>
<tr>
<td></td>
<td>$s_{AO} = \ell \gamma$</td>
<td>$t_{\text{race ends}} = (\frac{\ell}{v}) \gamma \alpha$</td>
<td>$s_{AB} = 2\ell \gamma$</td>
</tr>
<tr>
<td></td>
<td>$t_{\text{hare travels } AO} = (\frac{\ell}{v}) \gamma \alpha$</td>
<td>$t_{\text{tortoise travels } AB} = (\frac{\ell}{v}) \gamma \alpha$</td>
<td></td>
</tr>
<tr>
<td>Tortoise</td>
<td>$v = u$</td>
<td>$s = 0$</td>
<td>$v = u$</td>
</tr>
<tr>
<td></td>
<td>$s_{BO} = \ell \gamma$</td>
<td>$t_{\text{race ends}} = (\frac{\ell}{v}) \gamma \alpha$</td>
<td>$s_{BA} = 2\ell \gamma$</td>
</tr>
<tr>
<td></td>
<td>$t_{\text{hare travels } BO} = (\frac{\ell}{v}) \gamma \alpha$</td>
<td></td>
<td>$t_{\text{tortoise travels } BA} = (\frac{\ell}{v}) \gamma \alpha$</td>
</tr>
</tbody>
</table>

Table 1: Observer:observed table. Relative measurements in the race as measured by the referee, the hare, and the tortoise. The leftmost column represents the observers or the frames of reference and the top row represents the observed.

For simplicity, let $\ell = u = xc$, where $x \in [0, 1]$. That is, both $\ell$ and $u$ are constant fraction of $c$. Let $t_{\text{self}}$ and $t_{\text{other}}$ be the times denoted by

$$t_{\text{self}} = t_{\text{referee travels } AO, \text{hare}} = t_{\text{referee travels } BO, \text{tortoise}}.$$  
$$t_{\text{other}} = t_{\text{tortoise travels } AB, \text{hare}} = t_{\text{hare travels } BA, \text{tortoise}}.$$ 

For a given $x$, we have

$$t_{\text{self}} = \sqrt{1 - x^2} \text{ and } t_{\text{other}} = \sqrt{1 - x^2} \cdot (1 + x^2).$$

Then, Figure gives the difference between $t_{\text{self}}$ is less than and $t_{\text{other}}$ when $x$ varies. Note that for every value of $x$ except boundary values, $t_{\text{self}}$ is less than $t_{\text{other}}$. For example, when $x = \frac{\sqrt{3}}{2}$, i.e., for $\ell = u = \left(\frac{\sqrt{3}}{2}\right)c$, we have $t_{\text{self}} = 0.5$ seconds (see point $E$) and $t_{\text{other}} = 0.875$ seconds (see point $F$). This means that both hare and tortoise think that they took 0.5 seconds to travel the required distance themselves but the other participant took 0.875 seconds to cover the same distance. This is the effect of relativity.
Figure 4: Relative time

Now you know who is the culprit of the crime for creating confusion in the race. Yes, you are right, it is light!

References

Mosquito to Elephant

Problem

A magician puts a mosquito in his magic hat and pulls an elephant out. A mathematician watching the magic show jumps out of his seat and shouts that it is just an illusion. He claims he can do a dramatically better trick with absolutely no illusion. He says it is possible to separate a solid mosquito into a finite number of pieces and rearrange them (using translations and rotations) to create an elephant (see Figure 5). The only assumption he needs to make is that the mosquito and elephant are mathematical objects consisting of an infinite number of points.

How on earth is it mathematically possible to separate a mosquito into a finite number of pieces and rearrange them to create an elephant?

Solution

The puzzle is undoubtedly one of the most mind-bending puzzles of all time. The puzzle belongs to the domains of measure theory, topology, and geometry.

In further sections, we give a complicated but elegant mathematical proof to show that it is possible to cut a mosquito into a finite number of pieces that can be rearranged to create an elephant. The major assumption we make of the given objects is that both the mosquito and elephant consist of an infinite number of points.

Hausdorff-Banach-Tarski solution

We prove this brain-drilling, commonsense-defying, counter-intuitive mathematical result in four steps:

1. [Relate rotation sets.] Partition the rotations of surface points on a unit sphere into three sets. Find weird relations between the rotation sets.
2. [Create two spherical surfaces from one spherical surface.] Partition the sphere surface to create two identical sphere surfaces using the concept of rotations. This result is called the Hausdorff theorem.
3. [Create two spheres from one sphere.] Partition the sphere to create two identical spheres. This result is called the Banach-Tarski theorem (duplication version).
4. [Create elephant from mosquito.] Partition a mosquito to create an elephant. This result is called the Banach-Tarski theorem (magnification version).
[Step 1. Relate rotation sets.] In this step, we define rotations, partition them into three sets, and find weird relations between the three rotation sets.

We define rotation of an object along an axis as a transformation (but not distortion) of an object such that each point of the object moves in a circular path about the axis in a plane perpendicular to the axis. Let’s define two basic rotations of a unit sphere:

• $\alpha$ = clockwise rotation of 120° about the $z$-axis.
• $\beta$ = clockwise rotation of 180° about the line $z = x$ in the $xz$-plane.

We can achieve literally any arbitrary rotation of the unit sphere by sequentially combining the two basic rotations a finite number of times. (Proof?) For example, a clockwise rotation of 240° about the $x$-axis can be obtained by two successive $\alpha$ rotations, denoted by $\alpha \alpha$ or $\alpha^2$. We define identity rotation, denoted by $I$, as a rotation that brings an object to its original position. For example, $\alpha^3$ and $\beta^2$ are identity rotations i.e., $\alpha^3 = \beta^2 = I$.

We define rotation sequence as a sequence of rotations applied on an object. For example, $\alpha^3 \beta^2 \alpha^7 \beta^{10} \alpha^4$ is a rotation sequence applied from left-to-right. We define rotation sequence in simplified form as a rotation sequence in its most simplified or reduced form devoid of all identity rotations. For example, $\alpha^9 \beta^3 \alpha^17 \beta^{10} \alpha^4 = (\alpha^3)^3 \beta^2 \beta (\alpha^3)^5 \alpha^2 (\beta^2)^5 \alpha^3 \alpha = \beta$. We define rotation length of a rotation sequence in its simplified form as the number of symbols in the sequence. For example, the rotation length of $\alpha^2 \beta \alpha$ is 3.

Let $R$ be a set of all possible rotations of a unit sphere. Every rotation in $R$ has a simplified form of representation. Also that two different simplified form rotations represent two physically unique rotations. (Proof?) This result is called the uniqueness theorem.

<table>
<thead>
<tr>
<th>GENERATEANDPARTITIONROTATIONS()</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $A \leftarrow I$; $B \leftarrow {\alpha, \beta}$; $C \leftarrow {\alpha^2}$</td>
</tr>
<tr>
<td>2. for rotation length $i \leftarrow 1$ to $\infty$ do</td>
</tr>
<tr>
<td>3. for every rotation sequence $R_i = r_1 r_2 \ldots r_i$ in $A, B, C$ do</td>
</tr>
<tr>
<td>4. Generate rotations $R_{i+1}$ from $R_i$</td>
</tr>
<tr>
<td>5. Assign them to sets $A, B, C$ based on Figure’s partitioning logic.</td>
</tr>
</tbody>
</table>

Figure 6: An algorithm to partition all rotations into rotation sets $A, B,$ and $C$.

<table>
<thead>
<tr>
<th>If $R_i \in A$</th>
<th>If $R_i \in B$</th>
<th>If $R_i \in C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>If $r_i = \alpha$ or $\alpha^2$</td>
<td>$B \leftarrow B \cup R \beta$</td>
<td>$A \leftarrow A \cup R \beta$</td>
</tr>
<tr>
<td>If $r_i = \beta$</td>
<td>$B \leftarrow B \cup R \alpha$</td>
<td>$C \leftarrow C \cup R \alpha$</td>
</tr>
<tr>
<td>$C \leftarrow C \cup R \alpha^2$</td>
<td>$A \leftarrow A \cup R \alpha^2$</td>
<td>$B \leftarrow B \cup R \alpha^2$</td>
</tr>
</tbody>
</table>

Figure 7: Generating new rotations $R_{i+1}$ from a given rotation $R_i = r_1 r_2 \ldots r_i$, and assigning them to rotation sets $A, B,$ or $C$.

We partition the set $R$ into three subsets $A, B,$ and $C$ such that they are mutually exclusive and exhaustive. That is, $A \cap B = A \cap C = B \cap C = \emptyset$ and $R = A \cup B \cup C$, where $\cap$ symbol denotes intersection and $\cup$ symbol denotes union. We generate simplified
form rotations and partition them into sets A, B, and C using the GENERATEAND-
PARTITIONROTATIONS algorithm as shown in Figure 6.

Initially, the identity rotation is assigned to A, α, and β are assigned to B, and α² is assigned to C. These are the rotation sequences having lengths up to 1. We use the algorithm to create and partition rotation sequences of length 2, 3, 4, and so on. At every iteration $i$, the algorithm considers the rotation sequences of length $i$ and uses the logic from Table 7 to generate rotation sequences of length $i+1$ and assign them to A, B, or C. Table 2 shows the rotation sequences generated by the algorithm up to length 4.

<table>
<thead>
<tr>
<th>Length</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{I}</td>
<td>–</td>
<td>{αβ, α²β, βα²}</td>
<td>{βαβ}</td>
<td>{αβαβ, αβα²β, α²βαβ, α²βα²β, βα²βα²}</td>
</tr>
<tr>
<td>B</td>
<td>{α, β}</td>
<td>–</td>
<td>{αβα, α²βα, βα²β}</td>
<td></td>
<td>{βαβα}</td>
</tr>
<tr>
<td>C</td>
<td>–</td>
<td>{α²}</td>
<td>{βα}</td>
<td>{αβα², α²βα²}</td>
<td>{βαβα², βα²βα²}</td>
</tr>
</tbody>
</table>

Table 2: The rotations created by GENERATEANDPARTITIONROTATIONS algorithm up to length 4.

We now arrive at the meaty stuff of this entire step. It is the weird relations between the rotation sets A, B, and C. Any rotation sequence from A followed by the basic rotation α yields a rotation sequence in B. Conversely, every rotation in B is related to some rotation in A. This means that there is a 1-to-1 correspondence between rotations from A with rotations from B. Mathematically, we can write the relation as $Aα = B$. Similarly, we can derive two other relations: $Aα² = C$ and $Aβ = B \cup C$. (Proof?)

The three weird relations between the rotation sets A, B, and C are:

$Aα = B$, $Aα² = C$, and $Aβ = B \cup C$.

Before we really understand the weirdness of these relations, we need to understand the concept of congruence or isometry. Suppose a set of points $Q_1$ is mapped or transformed into another set of points $Q_2$ such that the distance between every pair of points is the same before and after the transformation. Then we say that $Q_1$ is congruent or isometric to $Q_2$, denoted by $Q_1 = Q_2$. Congruence or isometry is a distance-preserving transformation. For example, there is a 1-to-1 correspondence between the sets {1, 2, 3, …} and {2, 4, 6, …} but they are not congruent as the distance is not preserved. Congruence implies 1-to-1 correspondence, however, 1-to-1 correspondence does not imply congruence. As another example, a puppy can undergo a congruent or isometric transformation through translation (or shift), rotation, and mirror reflection but not through stretching and deformation.

See the weirdness of the relations? The equations mean that the sets A, B, C, and $B \cup C$ are congruent! How can A be congruent to B, C, and $B \cup C$, simultaneously? Think!

[Step 2. Create two spherical surfaces from one spherical surface.] In this step, we take the surface of a unit sphere, separate it into a finite number of pieces
and then reassemble them to create two sphere surfaces identical in shape and size with the original unit sphere.

It is important to observe that every rotation sequence has an axis of rotation. The axis of rotation for $\alpha$ is $z$-axis and that for $\beta$ is $z = x$ in the $xz$-plane. Similarly, we can always find exactly one axis of rotation for any given rotation sequence that would allow us to rotate the sphere from an initial position to its final position in exactly one rotation.

The axis of rotation for any given rotation sequence intersects the spherical surface at exactly two points called poles. Let $P$ denote the set of all poles for all the rotation sequences and let $S$ denote the set of all points on the surface of the sphere. Then, $S - P$ denotes the set of all non-poles. The set $P$ is a countably infinite set as $R$ is a countably infinite set and each rotation sequence in $R$ corresponds to two poles in $P$. The set $S - P$ is an uncountably infinite set. Therefore, there are far more points in $S - P$ than in $P$. (Likewise, there are far more numbers in the set of irrational numbers (uncountably infinite set) than the set of rational numbers (countably infinite set)). Each point in $P$ does not move for at least one rotation sequence from $R$. On contrary, each point in $S - P$ moves for every rotation sequence from $R$.

Each point in $S - P$ is connected to a countably infinite number of points in $S - P$ via different rotation sequences. Two distinct points from $S - P$ are said to be in the same orbit if there is a connection between the two points. There is an uncountably infinite number of such orbits. Using axiom of choice, we pick one point from each orbit and denote the choice set i.e., the set of all picked points by $K$.

Axiom of choice states that if $Q$ is a collection of an infinite number of sets, then there exists a choice function $f$ on the collection $Q$ that selects an element from each of the sets present in $Q$. Though the axiom seems believable it is the most debated and controversial axiom after Euclid’s parallel postulate. For example, if $Q$ is the collection of all pairs of shoes, then one choice function could be $f$ be the left shoe. Another example: if $Q$ is a collection of all pairs of identical socks, then $f$ can be an arbitrary choice among a pair of identical socks. Axiom of choice is an important assumption in the solution to our puzzle.

A few observations on the choice set $K$:
- $K$ is an uncountably infinite set.
- $K$ and $P$ do not have common points.
- No two points in $K$ are connected. That is, no point in $K$ can be got by rotating another point in $K$.
- Each point in $S - P$ can be got from some point in $K$ by applying a rotation sequence from $R$. This means that by applying rotation sequences from $R$ on the points in $K$, we get $S - P$.

We partition $S - P$ into three subsets $X$, $Y$, and $Z$ (see Figure 8) such that they are mutually exclusive and exhaustive. That is, $X \cap Y = X \cap Z = Y \cap Z = \emptyset$ and $S - P = X \cup Y \cup Z$. Let $X$ denote the set of all points got by applying the rotation sequences from $A$ to every point in $K$. Similarly, let $Y$ and $Z$ denote the sets of all points got by applying the rotation sequences from $B$ and $C$, respectively, to every point in $K$. Mathematically, we can write the three relations as $X = KA$, $Y = KB$, and
\[ Z = KC. \]

From the definitions of \( X \) and \( Y \) (i.e., \( X = KA \) and \( Y = KB \)) and the relation between the rotation sets \( A \) and \( B \) (i.e. \( A\alpha = B \)), when we apply the basic rotation \( \alpha \) on \( X \), we get \( Y \). This means that \( X \) is congruent to \( Y \), denoted by \( X \cong Y \). Similarly, when the basic rotation \( \alpha^2 \) is applied to \( X \), we get \( Z \) and when the basic rotation \( \beta \) is applied to \( X \), we get \( Y \cup Z \). So, \( X \cong Z \) and \( X \cong Y \cup Z \). This is the Hausdorff paradox, which is a theorem.

The three weird relations between the point sets \( X, Y, \) and \( Z \) are:

\[ X \cong Y, \quad X \cong Z, \quad \text{and} \quad X \cong Y \cup Z. \]

Let’s see the geometrical interpretation of the paradox. The set \( S - P \) denotes almost all of the spherical surface. Because \( X \cong Y \cong Z \), each of \( X, Y, \) and \( Z \) is roughly one-third the entire sphere. Because \( X \cong Y \cong Z \cong Y \cup Z \), each of \( X, Y, \) and \( Z \) is also roughly one-half the entire sphere. This means that each of the sets \( X, Y, \) and \( Z \) is roughly equal to one-third and one-half of the sphere, simultaneously. How is it possible?

We will now use this result to show that it is possible to separate a spherical surface into a finite number of pieces and then reassemble them to create two identical spherical surfaces.

We partition the spherical surface \( S \) into two identical spherical surfaces to \( S \) as follows. We use the symbol + to denote the addition of sets of points. We partition \( S - P \) and reassemble to create two identical copies as shown in Figure 9. We then deal with the poles using the concept of piecewise congruence or equidecomposability. We say that two objects are piecewise congruent or equidecomposable, denoted by \( \equiv \), if one object can be split into a finite number of components and reassembled via rotations and translations to form the other object taking all points of the object into consideration.

\[
S = (S - P) + P = X + Y + Z + P
\]

\[
\cong (Y + Z) + (Y + Z) + (Y + Z) + P \quad (\because X \cong Y \cong Z \cong Y + Z)
\]

\[
\cong X + Y + Z + X + Y + Z + P \quad (\because X \cong Y \cong Z)
\]
\[(X + Y + Z + P) + (X + Y + Z) \equiv S + (S - P) \quad \text{(simplify)}\]
\[\Leftrightarrow S + S \quad (\therefore S - P \text{ is piecewise congruent to } S)\]
\[= 2S\]

We use the technique of *shifting to infinity* to create a complete spherical surface \(S\) from an almost spherical surface \(S - P\). One might wonder if this implies that the whole can equal its proper part. In the world of infinity, the answer is yes!

Before we deal with the problem of creating a spherical surface from a spherical surface having a countably infinite number of holes, let’s understand the problem’s counterparts in 1-D and 2-D.

In 1-D, the sets of points \(Q_1 = \{1, 2, 3, \ldots\}\) and \(Q_2 = \{1, 2, 3\} \cup \{5, 6, 7, \ldots\}\) are piecewise congruent. This is because we can decompose \(Q_2\) into \{1, 2, 3\} and \{5, 6, 7, \ldots\} and shift left by one to get \{1, 2, 3\} \cup \{4, 5, 6, \ldots\} which is in fact \(Q_1\). Observe that \(Q_1\) and \(Q_2\) are piecewise congruent but not congruent (or isometric). Congruence implies piecewise congruence but not vice versa. In summary, we can transform \(Q_2 = \{1, 2, 3, 5, 6, \ldots\}\) which has a hole at 4 to \(Q_1\) which has no holes. In this process of shifting to infinity, the hole at 4 will be absorbed.

![Figure 10: Reassembling a circle with a hole to a complete circle without holes using the technique of shifting to infinity.](image)

In 2-D, a complete circle and a circle with a hole at a point on the circumference are piecewise congruent. Consider the circle on the left in the Figure 10. Starting from the hole, move a distance of 1 unit in the counterclockwise direction and label it as 1. We could have chosen to travel any length that is an irrational multiple of the circumference. Move 1 unit again in the same direction and label the point as 2. Label the remaining points in this way. Now move the point at 1 to location 0. Move point at 2 to point at 1. Move point at 3 to point at 2. Continue the process till infinity. This process is shown in the mid part of the figure. Finally, we end up with a complete circle as shown on the right of the figure. In this process of shifting to infinity, the hole at location 0 will be absorbed.

In 3-D, a complete spherical surface \(S\) and a spherical surface with a countably infinite number of holes \(S - P\) are piecewise congruent. Consider a point hole on the spherical surface. We can imagine a circle on the sphere’s surface passing through this point hole. By using the technique of shifting to infinity, we can add an additional point thereby absorbing the hole. Using the technique repeatedly for all countably infinite number of points, we can transform \(S - P\) to \(S\). In this process of shifting to
infinity, all the points of \( P \) will be absorbed.

We obtain two identical spherical surfaces from one spherical surface using the axiom of choice, Hausdorff paradox, and piecewise congruence.

[Step 3. Create two spheres from one sphere.] In this step, we take a unit sphere, separate it into a finite number of pieces and then reassemble them (through rigid motions of translation and rotation) to create two spheres identical in shape and size with the original unit sphere. This result is called the Banach-Tarski paradox (duplication version).

The method of doubling a spherical surface can be very easily extended to doubling a sphere. Think of a solid 3-D unit sphere \( S \) as a collection of an infinite number of concentric spherical surfaces \( S_r \) having radius \( r \) in the range \([0, 1]\). We consider all spherical surfaces \( S_r \), except the origin \( O = (0, 0, 0) \), and double the spherical surfaces to \( 2S_r \) using the Hausdorff paradox from the previous section. As a result, we would end up with two spheres except for the center point for the second sphere.

We can equidecompose the second incomplete sphere \( S - O \) to a complete sphere \( S \) using the technique of shifting to infinity. We select a circle (or a spherical surface) inside the incomplete sphere that passes through the center and then shift points from infinity to plug in the hole at the center. We then would have our second complete sphere.

Simple steps to double a unit sphere are as follows:

\[
S = \sum_{r=0}^{1} S_r
\]

\[
= O + \sum_{r>0}^{1} S_r
\]

\[
\Leftrightarrow O + \sum_{r>0}^{1} (2S_r) \quad (\because S_r \text{ is piecewise congruent to } 2S_r)
\]

\[
= \left(O + \sum_{r>0}^{1} S_r\right) + \sum_{r>0}^{1} S_r
\]

\[
= S + (S - O) \quad (\text{simplify})
\]

\[
\Leftrightarrow S + S \quad (\because S - O \text{ is piecewise congruent to } S)
\]

\[
= 2S
\]

We obtain two identical spheres from one sphere.

[Step 4. Create an elephant from a mosquito.] In this step, we take a mosquito, separate it into a finite number of pieces and then reassemble them (through rigid motions of translation and rotation) to create an elephant. This result is called the Banach-Tarski paradox (magnification version).
We use the Banach-Schröder-Bernstein theorem to prove our claim. Suppose \( S_1 \) and \( S_2 \) be two bounded 3-D objects with nonempty interiors. The theorem states that if \( S_1 \) is piecewise congruent to a subset of \( S_2 \) and if \( S_2 \) is piecewise congruent to a subset of \( S_1 \), then \( S_1 \) is piecewise congruent to \( S_2 \). Mathematically we write as follows. If \( S_1 \preceq S_2 \) and if \( S_2 \preceq S_1 \), then \( S_1 \Leftrightarrow S_2 \).

First, we show that mosquito is piecewise congruent to a subset of an elephant. Then, we show that the elephant is piecewise congruent to a subset of mosquito. Using the Banach-Schröder-Bernstein theorem we prove that mosquito is piecewise congruent to an elephant.

Let \( S_m \) denote a sphere that contains the mosquito. Let \( S_e \) denote a sphere that is completely inside the elephant. Duplicate \( S_e \) several times, say \( n \) times, such that the \( n \) overlapping \( S_e \) spheres completely contain the sphere \( S_m \). Observe that the \( n \) overlapping \( S_e \) spheres is piecewise congruent to a subset of \( n \) disjoint \( S_e \) spheres. We know from the Banach-Tarski paradox that the \( n \) disjoint \( S_e \) spheres are piecewise congruent to one \( S_e \) sphere.

Mathematically, we have:

\[
\text{Mosquito} \subseteq S_m \subseteq n \text{ overlapping } S_e \text{ spheres} \\
\leq n \text{ disjoint } S_e \text{ spheres} \\
\Leftrightarrow S_e \quad \text{(using Banach-Tarski theorem)} \\
\subseteq \text{Elephant}
\]

This implies that mosquito is piecewise congruent to a subset of an elephant. On similar lines, we can show that the elephant is piecewise congruent to a subset of mosquito. Using the Banach-Schröder-Bernstein theorem, we have the mind-blowing result that a mosquito is piecewise congruent to an elephant.

A mosquito can be partitioned into a finite number of pieces and reassembled (through translations and rotations and allowing the pieces to pass through one another) to form an elephant.

Analysis

The problem has been solved mathematically on paper. But the mind still craves for an intuitive understanding of this weird result. In this section, let’s explore the meaning and consequences of the crazy result.

[Axiom of Choice.] The mosquito-to-elephant result is highly counterintuitive. Is the result true? The proof for the result seems logically flawless except for the assumption of the axiom of choice. The axiom states that given a collection of an infinite number of mutually disjoint nonempty sets, it is possible to create a new set, called the choice set, consisting of a unique element from each set in the collection. The axiom assumes a choice function without proof. The axiom leads to several bizarre counterintuitive results. Hence, the axiom is considered one of the most controversial statements in mathematics.
Why can’t we either prove or disprove the axiom of choice? Kurt Gödel proved that we can have a consistent mathematical system accepting the axiom of choice. In contrast, Paul Cohen proved that we can have a consistent mathematical system rejecting the axiom of choice. As the axiom of choice is independent of the other axioms of the set theory, it is impossible to either prove or disprove the axiom of choice. Therefore, we are free to accept or reject the axiom.

Why can’t we reject the axiom of choice (meaning that the axiom is false)? We can certainly do that. In the mathematical system where the axiom of choice is false, the Hausdorff theorem, the Banach-Tarski theorem, and the mosquito-elephant piecewise congruence do not hold. However, the weird relations between the rotation sets still hold and they are equally paradoxical and defy common sense.

The axiom of choice is like Euclid’s parallel postulate. The parallel postulate says that given a straight line and a point outside the straight line, there can be exactly one line that passes through the point that does not intersect the given straight line. If we accept the postulate, then we have Euclidean geometry (or parabolic geometry), which is consistent. If we reject the postulate, then we get non-Euclidean geometries such as hyperbolic geometry (or Lobachevsky-Bolyai geometry) and elliptic geometry (or Riemannian geometry), which are consistent too. We cannot argue as to which of Euclidean or non-Euclidean geometry is true (or consistent) because both the systems are true in their own way and they both are consistent. Similarly, we can have a consistent mathematical system where the axiom of choice does not hold.

Most of the mathematical community has accepted the axiom of choice because it seems intuitively correct and leads to beautiful mathematical results.

[Intuition.] Are mathematicians joking that we can obtain an elephant-sized object from a mosquito-sized object violating the law of conservation of mass? If that is the case, then we can: (i) print tons of bank money from a single note of the highest denomination and become the richest people on the planet. (ii) clone millions of copies of chickens and fish for food. (iii) clone hundreds of copies of the world’s greatest scientists, engineers, teachers, doctors, entrepreneurs, and most importantly politicians for the rapid development of our society. (iv) create gargantuan and aesthetically beautiful architectural buildings from a grain of sand. (v) create hundreds of solar systems, planets, and moons and distribute the human population among them.

If something does not make sense, if something is against our intuition, and if something is unbelievable, it does not mean that it is not true. There are many counterintuitive truths in nature. For example, Thomas Young’s double-slit experiment which showed that light acts as both particles and waves; Albert Einstein’s special theory of relativity which showed that measured time (also length and mass) is not universal but dependent on the observer; and Werner Heisenberg’s uncertainty principle which showed that the measurement of multiple physical properties of an object simultaneously is inherently probabilistic. On similar lines, the mosquito-to-elephant paradox (or the one sphere to \( n \) spheres paradox) is counterintuitive but true in the mathematical system where the axiom of choice is true.

An important point to note is that the mosquito-to-elephant theorem can be applied to mathematical objects consisting of an uncountable infinity of points. Weird
principles and crazy results are common in the world of infinity. As we are wired to think about finite quantities and work with finite arithmetic, it becomes difficult to comprehend and accept the results from the world of infinity.

It is unclear if the mosquito-to-elephant theorem can be applied to the real physical objects. The real-life objects in the observable universe seem to be made up of a finite number of entities, each of which has a finite nonzero size (and mass). We cannot apply the result to physical objects. Suppose we come across a strange bounded physical object, somewhere in our cosmos, which has either zero or infinite mass and is made up of an uncountable infinity of continuous physical points (equivalent to mathematical points), each of whose size limits to zero. Then, in that imaginary physical world, the theorem might be applicable, wonders might be common, and fantasies might be a reality.

References

Study the uncountability puzzle in this book to understand some of the counterintuitive principles of infinity. The Hausdorff theorem was first proved by Felix Hausdorff. Banach-Tarski theorem was first proved in [Banach and Tarski, 1924]. [Robinson, 1947] showed that doubling a sphere can be achieved with a minimum of five partitioned pieces. The proof of the Hausdorff-Banach-Tarski theorem is excellently presented in [Wapner, 2005]. The book also gives proof for the uniqueness theorem, the weird relations between the rotation sets, and the Banach-Schröder-Bernstein theorem and several interesting and insightful ideas related to the problem. Other articles on the paradox are [Stromberg, 1979] and [French, 1988]. Refer to [Jech, 2008] and [Herrlich, 2006] to know more about the axiom of choice.
Prisoners and a Light Bulb

Problem

100 prisoners who have their own individual rooms in a prison are told that they will be interrogated the next day. The interrogation is held in the interrogation room where a light bulb is present. None of the prisoners can see the light bulb from their own rooms. Once the interrogation starts, the only way of communication between the prisoners will be through toggling the light switch i.e., switching on the bulb if it is off or switching off the bulb if it is on. Initially, the light bulb is switched off and the prisoners know this fact.

A prisoner is called for interrogation every day following a uniform random distribution. During interrogation, a prisoner can do nothing or toggle the light switch or announce that all the 100 prisoners have been interrogated. If his assertion is true, all prisoners are set free. On the other hand, if his assertion is false, all prisoners are executed. This means that the announcement should be made only if the prisoner is 100% confident of the validity of the assertion.

A single day is giving to the 100 prisoners to discuss and come up with a plan to set all prisoners free. Is there a strategy or plan using which the prisoners can be set free? If yes, what is the strategy? If no, prove that such a strategy does not exist.

Solution

The puzzle comes under a broad category of puzzles that involves knowledge, belief, and ignorance. The topics that are closer to the puzzle domain are epistemology, epistemic logic, and game theory. Epistemology is a branch of philosophy that deals with the theory of knowledge. Epistemic logic is a subfield of formal logic that deals with the formal consideration of the strategies for an inquiry into the knowledge and belief. Game theory is a branch of economics that deals with the strategies of cooperation or conflict between intelligent decision-makers in a game or an activity.

The puzzle seems impossible to solve. There is only one light bulb that can be used for communication between all 100 prisoners. The two states of the light bulb: on and off, maps to \( \log_2(2) = 1 \) bit of information. How is it even possible to think of a strategy to communicate between 100 prisoners with only 1 bit of information. Surprisingly, there are several strategies to solve the problem.

We assume that the total number of prisoners is \( n \). For the current problem, \( n = 100 \). A few important observations in the puzzle are: (i) Number of days elapsed is equal to the number of visits to the interrogation room. (ii) The prisoners are brought to the interrogation room in uniformly random order. (iii) The light bulb is initially switched off.

[Fixed-time strategy (probabilistic solution).] This is a simple strategy. A pris-
oner who visits on the $k$th day ($k \geq n$) announces that all the prisoners have visited the interrogation room. It is easy to see that the strategy is a probabilistic solution. Even if we choose $k$ big enough to increase the probability of success to more than 99%, the strategy might still fail. Note that the prisoners are in no way using the status of the light bulb.

The probability of success of the strategy is measured by mapping this problem to the balls and bins problem in probability theory. In the balls and bins problem, we throw $k$ balls to $n$ bins, where $k \geq n$ and the probability of a ball landing in any bin is $\frac{1}{n}$. Now, we compute the probability that all $n$ bins are covered (or non-empty) from $k$ balls. Then,

\[
\text{Probability that } i\text{th ball does not fall in } j\text{th bin} = \frac{(n-1)}{n}^k
\]

\[
\text{Probability that none of the } k \text{ balls falls in } j\text{th bin} = \left(\frac{(n-1)}{n}\right)^k
\]

\[
\text{Probability that at least one of the } k \text{ balls falls in } j\text{th bin} = 1 - \left(\frac{(n-1)}{n}\right)^k
\]

\[
\text{Probability that all the } n \text{ bins are covered in the } k \text{ balls} = \left(1 - \left(\frac{(n-1)}{n}\right)^k\right)^n
\]

The intuition behind the strategy is that we will eventually encounter a block such that every prisoner would have entered the room exactly once. On the first day

Figure 11 gives a plot of the success probability for 100 prisoners when $k$ is varied from 100 to 1000. When $k$ is less than $n$, the success probability is 0.

Figure 11: Success probability of fixed time strategy for 100 prisoners for $k \in [100, 1000]$ days.

[All-in-one-block strategy.] Split the days into blocks each of size $n$. During each $n$-day block, each prisoner follows the ALLINONEBLOCK algorithm.

The intuition behind the strategy is that we will eventually encounter a block such that every prisoner would have entered the room exactly once. On the first day
of every block, the bulb is switched on. A prisoner switches off the bulb if he/she enters the room the second time in a block. So, if the bulb is switched on on the first
day of any block except the first, it means that no prisoner entered the previous block twice.

Let $X$ = number of days in the strategy. We like to compute the expected running time $\mathbb{E}[X]$ of the strategy. Let $B$ = number of $n$-day blocks in the strategy. As $B$ is a Bernoulli trial, $\mathbb{E}[B] = 1$ divided by (probability of a successful block). We have,

$$
\mathbb{E}[X] = n \cdot \mathbb{E}[B] = n \cdot \left( \frac{1}{\text{prob. of all-in-one-block}} \right) = n \cdot \left( \frac{1}{\frac{n^{n+1}}{n!}} \right) = O\left(n^{1/2}e^n\right) \quad \text{(using Stirling’s approximation of } n! \approx \sqrt{2\pi n}(n/e)^n)\
$$

**Expected running time** = $O\left(n^{1/2}e^n\right)$ days.
**Expected running time** ≈ $1.072 \times 10^{44}$ days for $n = 100$.

[**Single counter strategy.**] The core idea of the strategy is as follows. One of the $n$ prisoners is elected as a counter. The counter is responsible for counting the number of prisoners who visit the interrogation room. When the count equals $n$, the counter announces that all prisoners have visited the room.

Initially the counter’s count is initialized to 0. Each non-counter switches on the bulb the first time. The counter switches off the bulb and increments the count by 1. When the count equals 99, the counter announces that all prisoners had been to the room.

Let’s find the expected running time of the strategy. Let $X$ be #days in the strategy, $X_i$ be #days between the 1st day on which count = $i$ and the 1st day on which count = $i+1$, $Y_i$ be #days between the 1st day on which count = $i$ and the 1st day on which count = $i$ and the bulb is on, and $Z_i$ be #days between the 1st day on which count = $i$ and the bulb is on and the 1st day on which count = $i+1$. Then,

$$
\mathbb{E}[X] = \sum_{i=1}^{n-1} \mathbb{E}[X_i] = \sum_{i=1}^{n-1} (\mathbb{E}[Y_i] + \mathbb{E}[Z_i])
$$

$$
= \sum_{i=1}^{n-1} \left( \frac{1}{(n-i)/n} + \frac{1}{1/n} \right) = n(n-1) + n \sum_{i=1}^{n-1} \frac{1}{i}
$$

$$
= n^2 - n + nH_{n-1} \approx n^2 - n + n \ln n
$$

**Expected running time** = $O\left(n^2\right)$ days.
**Expected running time** ≈ 10417.7 days or 28.54 years for $n = 100$.

[**Single dynamic counter strategy.**] Unlike single counter strategy, we can dynamically select the single counter. We can slightly improve the expected running time of the single counter strategy by selecting the counter dynamically. The counts of all the
prisoners will be initialized to 0.

There are two stages in the strategy. In the first stage consisting of $n$ days, a counter is selected dynamically. In the first $n-1$ days of the first stage, when a prisoner visits the interrogation room for the second time on say $k$th day and the bulb is still off, the prisoner becomes the counter. In the second stage, the counter counts $n-(k-1)$ prisoners using the straightforward SINGLE-COUNTER strategy.

**[Generic single counter strategy.]** The single counter strategy works even if the prisoners are called to the interrogation room in an arbitrary order at any time. In this section, we present a generic strategy GENERIC SINGLE COUNTER that is more powerful (but slower) than the single counter strategy that works even when the initial state of the bulb is unknown (either off or on). The count of the counter is initialized to 0.

The only difference between this strategy and the single counter strategy is that in the single counter strategy, the counter announces when his count equals $n-1$, whereas in this strategy, the counter announces when his count equals $2n-2$ to account for the ignorance of the initial state of the bulb. Please note that ignorance is expensive. The proof of why this strategy works is left to the reader as an exercise.

**[Hybrid counters strategy.]** This is a weird strategy that is deterministic but uses probabilistic techniques. The strategy is called hybrid because the first phase of the strategy has multiple counters and the second phase of the strategy has a single counter. The counters are selected dynamically. The counts of all prisoners are initialized to 1.

The core idea of the strategy is that prisoners who have non-zero counts can drop their points with some probability and everyone can collect the points. We choose a probability function that is inversely proportional to the prisoner counts. In other words, prisoners who have more counts will drop their points with very less probability. This implies, the prisoners with more counts will tend to get richer (much more counts) faster and the process converges faster.

As per the strategy, every prisoner can switch on the bulb using a probability function $f : \{0, 1, \ldots, n\} \to [0, 1]$. Let $k = \left\lceil \frac{n}{2} \right\rceil - 1$. Then, we define $f(i)$, where $i \in \{0, \ldots, n\}$, as:

$$f(i) = \begin{cases} 1 & \text{if } i = 0, \\ \frac{(i-k+1)}{k} & \text{if } i \in [1, k], \\ 0 & \text{if } i \in [k+1, n]. \end{cases}$$

In fact, we can choose any function $f(i)$ that satisfies the following constraints: $f(0) = f(1) = 1$, $f\left(2 \ldots \left\lceil \frac{n}{2} \right\rceil \right)$ is decreasing and in the range $(0, 1)$, and $f\left(\left\lceil \frac{n}{2} \right\rceil \ldots n\right) = 0$. The reason for selecting $f(i)$ with these constraints is to make sure that the prisoners with more counts drop their points with less probability. The value of $f\left(\left\lceil \frac{n}{2} \right\rceil \ldots n\right)$ is set to 0 because we want to make sure that the first prisoner whose count reaches $\left\lceil \frac{n}{2} \right\rceil$ not to give away his points but only collect points to reach a total of $n$ points. The reader is recommended to think more deeply on the working on the strategy.

**[Multiple counters strategy.]** In the single counter strategy, there was only one
counter and he was responsible for counting \( n - 1 \) other prisoners. We can generalize this idea to multiple counters. What if there is a tree hierarchy of counters and each
counter at a particular level is responsible for counting some of the counters in the below level? This idea leads to the multiple counter strategy.

In this strategy, we assume that \( n \) is one less than a power of 2. Let each prisoner belong to one of \([\log n] + 1 \) types: \( C_0, C_1, \ldots, C_{[\log n]} \). There are \( 2^i \) number of \( C_{[\log n]−i} \) prisoners, for \( i \in [0, \lfloor \log n \rfloor] \). A \( C_i \) prisoner will be responsible to count two distinct \( C_{i−1} \) prisoners, for \( i \in [1, \lfloor \log n \rfloor] \). This implies \( C_0 \) will not be counting anyone.

We split days into \([\log n] \) stages: \( s_1, s_2, \ldots, s_{[\log n]} \). Each stage \( s_i \) has \(|s_i| \) number of days. In stage \( s_i \), each \( C_i \) prisoner will try to count two \( C_{i−1} \) prisoners each. Initially, the quota of all prisoners is initialized to 0. If a \( C_i \) prisoner succeeds in counting two \( C_{i−1} \) prisoners, then we say that the particular \( C_i \) prisoner’s quota is full. The \( C_i \) prisoners whose quota is full can now be counted by \( C_{i+1} \) prisoners in stage \( s_{i+1} \). In stage \( s_{[\log n]} \), when the \( C_{[\log n]} \) prisoner’s quota is full, he announces that all prisoners had been to the interrogation room. If \( C_{[\log n]} \) prisoner’s quota is not full, the stages \( s_1, s_2, \ldots, s_{[\log n]} \) are repeated. On the last day of stage \( s_i \), if a prisoner apart from \( C_i \) switches off the bulb or if a \( C_i \) prisoner whose quota is full switches off the bulb, then they must remember to switch off the bulb an extra time in future invocations of \( s_i \). The strategy when a prisoner enters the interrogation room during stage \( s_i \) is given in the MULTIPLECOUNTERS algorithm.

Choosing the sizes of the stages \( s_1, \ldots, s_{[\log n]} \) is important. The expected running time of the strategy depends on the sizes of the stages.

[Multiple dynamic counters strategy.] In both single and multiple counter strategies, the counter assignments are fixed. This strategy is similar to the multiple counters strategy, however, the counters are assigned dynamically. The strategy uses the idea of give-and-take. Initially, all prisoners have values equal to 1. Some of the prisoners give out their values and their counts reduces to 0. These extra counts will be taken by the prisoners and their individual count increases to 2. Similarly, some of the prisoners with values 2 give out their values and their counts become 0s. These extra points will be taken by prisoners with values 2 and their individual count increases to 4. This process continues and when a prisoner’s count equals \( n \), it means that the prisoner has counted all prisoners and we stop.

We assume that \( n \) is a power of 2. (Why is the value of \( n \) a power of 2 in this strategy and it is one less than a power of 2 in multiple counters strategy?) In this strategy, the prisoners get different counter roles dynamically. There are \([\log n] \) stages: \( s_1, \ldots, s_{[\log n]} \). Stage \( s_i \) consists of \(|s_i| \) number of days and each day has a value \( v_i = 2^{i−1} \). Initially, the counts of all the prisoners are set to 1. Only prisoners with count values equal to \( v_i \) can participate in stage \( s_i \). The convergence of the strategy can easily be proved with the observation that a prisoner with count \( v \) will either have his count as 0 or count as \( 2v \).

![Stage Diagram]

[Improved multiple dynamic counters strategy.] This strategy is a much improved version of the multiple dynamic counters strategy. Every stage \( s_i \) corresponds
to a power of 2 and hence a bit in the binary representation of $n$. Unlike the multiple dynamic counters strategy, any prisoner can switch off the bulb and increment his/her count and any prisoner whose bit corresponding to the current stage is set to 1 can switch on the bulb and decrement his/her count.

We assume that $n$ is a power of 2. The values for the different days of the stages $s_1$ to $s_{\lfloor \log n \rfloor}$ remains the same as in the previous strategy. However, the sizes of the stages are $s_1 = s_2 = \cdots = s_{\lfloor \log n \rfloor} = \lceil n \log(n) \log(\log(n)) \rceil$. If the announcement does not happen after the last stage, the sequence of stages $s_1, \ldots, s_{\lfloor \log n \rfloor}$, called a cycle, is repeated.

In the best case, the algorithm might end after $O(\log n)$ days. In the worst case,
the algorithm might not end at all. The expected time for the algorithm to end is \( O(n \log^2 n) \) days. Please refer to William Wu [Wu, 2002] and Paul-Olivier Dehaye et al. [Dehaye et al., 2003] for a proof.

\[
\text{Expected running time} = O(n \log^2 n) \text{ days.}
\]
\[
\text{Expected running time} \leq 4400 \text{ days or 12 years for } n = 100.
\]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Asymptotic</th>
<th>Average</th>
<th>Expected</th>
<th>Empirical</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-in-one-block</td>
<td>( O(n^{1/2}e^n) )</td>
<td>( n^{w+1}/n! )</td>
<td>1.75 \times 10^{56}</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Single counter</td>
<td>( O(n^2) )</td>
<td>( n^2 - n + H_{n-1} )</td>
<td>16260.84</td>
<td>17829.4</td>
<td>-</td>
</tr>
<tr>
<td>Single dynamic counter</td>
<td>( O(n^2) )</td>
<td>( n^2 + n + \sum_{k=1}^{n} k(H_{n-k} - k) )</td>
<td>16260.84</td>
<td>13951.1</td>
<td>-</td>
</tr>
<tr>
<td>Generic single counter</td>
<td>( O(n^2) )</td>
<td>( 2(n^2 - n + H_{n-1}) )</td>
<td>32521.68</td>
<td>33194.6</td>
<td>-</td>
</tr>
<tr>
<td>Mult. ctrs</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6304</td>
<td>( n = 2^m - 1 )</td>
</tr>
<tr>
<td>Mult. dyn. ctrs</td>
<td>( O(n \log^2 n) )</td>
<td>-</td>
<td>4608</td>
<td>6348</td>
<td>( n = 2^m )</td>
</tr>
<tr>
<td>Impr. mult. dyn. ctrs</td>
<td>( O(n \log^2 n) )</td>
<td>-</td>
<td>4608</td>
<td>5148</td>
<td>( n = 2^m )</td>
</tr>
</tbody>
</table>

Table 3: Comparative analysis of algorithms for solving the prisoners and a light bulb puzzle.

**Experiments**

Table 3 summarizes both theoretical and experimental performance of algorithms and Figure 13 gives the performance comparison of the algorithms. The multiple dynamic counters and improved multiple dynamic counters algorithms work only when \( n \) is a power of 2. The multiple counters algorithm works only when \( n \) is one less than a power of 2. The asymptotic runtime column gives the order of growth of the number of days. The average runtime column has the exact function for the average number of days. The notation \( H_n \) in the average runtime column represents the \( n \)th Harmonic number defined as the sum of the reciprocals of the first \( n \) natural numbers, i.e., \( H_n = \sum_{i=1}^{n} \frac{1}{i} \). The expected runtime gives a value when we substitute \( n = 2^7 = 128 \) in the average runtime closed-form formula (or the asymptotic runtime when the average runtime formula is absent). The empirical runtime gives the experimental number of days (average over 10 runs) when the implementations of the algorithms are run.

We implement all algorithms except the fixed-time and hybrid counters algorithms (because these two algorithms make use of probability). The algorithms are implemented in Python (Google Colab). The numbers of prisoners are increased in powers of 2 starting from \( n = 2^0 \) to \( n = 2^7 \). The number of days taken by each algorithm is computed and plotted. We plot the number of days by taking the average of 10 runs. Hence, the number of days in the table and plots have fractional parts also.

Refer to Figure 13. We see that all-in-one-block is the slowest algorithm. The multiple counters, multiple dynamic counters, and improved multiple dynamic counters algorithms are among the fastest algorithms. The fastest algorithm is the improved multiple dynamic counters algorithm.
MATHMATICAL AND ALGORITHMIC PUZZLES

Prisoners and a light bulb

![Graph showing performance comparison of algorithms for solving the prisoners and a light bulb puzzle.](Figure 13)

Number of prisoners \( n \)

Number of days

- All in one block
- Generic single counter
- Single counter
- Dynamic single counter
- Multiple counters
- Multiple dynamic counters
- Improved multiple dynamic counters

Figure 13: Performance comparison of algorithms for solving the prisoners and a light bulb puzzle.

[Variant: Turn on all lights in a circle.] There is a circle of light bulbs numbered 0, 1, 2, \ldots, n − 1 with a switch next to each light bulb. Initially all switches are off and hence all light bulbs are turned off. Toggling a switch (on → off or off → on) changes the state of the corresponding light bulb and the two light bulbs adjacent to it. Design an algorithm to turn on all the light bulbs by toggling the minimum number of switches.

[Solution.] The TURNONLIGHTSINACIRCLE algorithm gives the pseudocode of the strategy. The working of the algorithm when \( n = 6, n = 7, \) and \( n = 8 \) are shown in Figure 14.

Number of toggles of the switches = \( \left( \frac{3}{2} \right) \times n \mod 3 = 0 + n \times \left( n \mod 3 \neq 0 \right) \).

Problems

1. [Optimal algorithm.] Design an optimal algorithm for solving the puzzle for known
**TURNONLIGHTSINACIRCLE**(n)

**Input:** Light bulbs 0, 1, 2, ..., n − 1 in a circle, where, n ≥ 3

**Output:** Sequence of toggling of the switches to turn on all bulbs minimizing the number of toggles

1. Create a dynamic array S ← [ ]
2. // Toggle bulbs 1, 4, 7, so on ..............................................
3. for i ← 1 to n − 1 increment 3 do S.Add(i)
4. if n mod 3 ≠ 0 then
5. // Toggle bulbs 0, 3, 6, so on ..............................................
6. for i ← 0 to n − 1 increment 3 do S.Add(i)
7. // Toggle bulbs 2, 5, 8, so on ..............................................
8. for i ← 2 to n − 1 increment 3 do S.Add(i)
9. return S

<table>
<thead>
<tr>
<th>Toggle</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>−</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Toggle</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>−</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 14: The sequence of toggles of the switches and the states of the light bulbs after each toggle for (left) n = 6, i.e., n mod 3 = 0, (middle) n = 7, i.e., n mod 3 = 1, and (right) n = 8, i.e., n mod 3 = 2.

or unknown probability distribution of prisoner calling.

**References**

The puzzle has been discussed in Wu Riddles [Wu, 2016] and Ponder This website of IBM Research [Research, 2002]. The puzzle solutions and the puzzle variants are presented in William Wu’s excellent article [Wu, 2002], [Dehaye et al., 2003], [Winkler, 2003], [van Ditmarsch et al., 2010], and [van Ditmarsch and Kooi, 2015]. The generic single counter strategy is described in [Winkler, 2003] and the hybrid counters strategy is given in [van Ditmarsch et al., 2010]. The multiple counters strategy is a generalization of the two stage protocol given in [Wu, 2002]. The improved multiple dynamic counters strategy is presented in [Dehaye et al., 2003, Wu, 2002].
Mog and Ooga

Problem

In the jungles of Amazon, there lives a tribal couple: husband Mog and wife Ooga. Mog has more muscular strength and stamina than Ooga. Mog can gather 3 mushrooms or catch 6 fish per hour. On the other hand, Ooga can gather 2 mushrooms or catch 1 fish per hour.

It is known that Mog and Ooga each work for 3 hours per meal. To get a balanced diet, Mog requires 6 mushrooms and 6 fish per meal. So Mog spends 2 hours gathering mushrooms and 1 hour fishing. In contrast, Ooga requires 2 mushrooms and 2 fish per meal. So Ooga spends 1 hour gathering mushrooms and 2 hours fishing.

Is there a strategy in which Mog and Ooga can collaborate and gain from each other?

Solution

This puzzle is one of the most important and fundamental puzzles in trade, business, and economics. From the puzzle statement, it is clear that the problem is related to trade. Because, the word collaboration in the context of the puzzle refers to trade.

[Observations.] Let
\[ m_g = \text{Mog's mushroom gathering rate} = 3 \text{ mushrooms/hour}. \]
\[ m_f = \text{Mog's fishing rate} = 6 \text{ fish/hour}. \]
\[ o_g = \text{Ooga's mushroom gathering rate} = 2 \text{ mushrooms/hour}. \]
\[ o_f = \text{Ooga's fishing rate} = 1 \text{ fish/hour}. \]

We see that \( m_g > o_g, m_f > o_f, m_f > m_g, \) and \( o_g > o_f. \) The food matrix showing the food collection of Mog and Ooga are as shown in Table 4.

<table>
<thead>
<tr>
<th>Person</th>
<th>Mushrooms</th>
<th>Fish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mog</td>
<td>6 (2 hours)</td>
<td>6 (1 hour)</td>
</tr>
<tr>
<td>Ooga</td>
<td>2 (1 hour)</td>
<td>2 (2 hours)</td>
</tr>
</tbody>
</table>

Table 4: Food collection matrix of Mog and Ooga.

Common solution (incorrect)

Mog is certainly better than Ooga in both gathering mushrooms \( (m_g > o_g) \) and catching fish \( (m_f > o_f). \) Hence, it is impossible that Mog can gain advantage trading with Ooga.

[Analysis.] The argument above seems clear, compact, and correct. However, the reasoning is flawed. We know that Ooga can definitely gain from trading with Mog.
In the next section, we will show using the power of mathematics that it is possible for even Mog to gain from collaborating (or trading).

We can define absolute advantage of an entity as the ability to produce an item more productively than any other entity. Mog has an absolute advantage over Ooga in both gathering mushrooms \((m_g > o_g)\) and catching fish \((m_f > o_f)\). However, Mog having absolute advantage over Ooga does not mean that he cannot gain by collaborating with Ooga. So, there should be some other concept that is important to solve the puzzle.

**Specialization solution**

Mog works for \(\frac{4}{3}\) hours gathering mushrooms and \(\frac{5}{3}\) hours fishing. In total, he works for 3 hours. Mog collects \(\left(\frac{4}{3}\right) m_g = 4\) mushrooms and \(\left(\frac{5}{3}\right) m_f = 10\) fish. On the other hand, Ooga works for all 3 hours gathering mushrooms. In total, she works for 3 hours. Ooga collects \(3 o_g = 6\) mushrooms and \(0 o_f = 0\) fish. Their food matrix is as shown in Table 5.

Mog and Ooga collaborate to get mutual benefits. Collaboration means trade in the context of the puzzle. Mog gives 3 of his fish to Ooga and Ooga gives back 3 of her mushrooms to Mog. After trade, Mog will have 7 mushrooms and 7 fish and Ooga will have 3 mushrooms and 3 fish. Their food matrix after trade is shown in Table 6. Each value in Table 6 is strictly greater than each value in Table 4. Hence, both Mog and Ooga are benefited through trade.

<table>
<thead>
<tr>
<th>Person</th>
<th>Mushrooms</th>
<th>Fish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mog</td>
<td>(4 \left(\frac{4}{3}\right)) hours</td>
<td>10 (\left(\frac{5}{3}\right)) hours</td>
</tr>
<tr>
<td>Ooga</td>
<td>6 (3 hours)</td>
<td>0 (0 hours)</td>
</tr>
</tbody>
</table>

**Table 5: Before trade.**

<table>
<thead>
<tr>
<th>Person</th>
<th>Mushrooms</th>
<th>Fish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mog</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Ooga</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

**Table 6: After trade.**

Mog must specialize in fishing, spending \(\frac{5}{3}\) hours fishing. Ooga must specialize in gathering mushrooms, spending all her time gathering. Then they trade 3 fish for 3 mushrooms. Both Mog and Ooga can be benefited from specialization and collaboration. Table 6 is better than Table 4.

[Analysis.] In the previous solution, we compared the productivity rates between Mog and Ooga, which led to an incorrect reasoning. In this solution, we compare the productivity rates between different ways to collect food by the same person. So we compare \(m_g\) and \(m_f\) and compare \(o_g\) and \(o_f\). We see that \(m_f > m_g\) and \(o_g > o_f\). We can define comparative advantage or specialization of an entity as the ability to produce an item more productively than any other item by the same entity. The solution uses the theory of comparative advantage or specialization. The difficulty in understanding comparative advantage is summarized in a nice anecdote. Stanislaw Ulam, a famous mathematician and a nuclear physicist who worked on the first nuclear bomb asked Paul Samuelson, a Nobel-winning economist, if there was an idea in economics that
was universally true and non-obvious. Samuelson gave the answer as the theory of comparative advantage.

The comparative advantage theory applied to the puzzle is as follows. Mog must give up \( \left( \frac{m_f}{m_g} = 2 \right) \) fish for every mushroom, and Ooga must give up only \( \left( \frac{o_f}{o_g} = \frac{1}{2} \right) \) fish for every mushroom. Hence, both Mog and Ooga would benefit if Mog specialized in fishing and Ooga specialized in gathering mushrooms.

\[
\left( \frac{m_f}{m_g} = 2 \right) > \left( \frac{o_f}{o_g} = \frac{1}{2} \right)
\]

Here is another argument. Mog can fish \( \left( \frac{m_f}{o_f} = 6 \right) \) times more efficiently than Ooga whereas Mog can gather mushrooms only \( \left( \frac{m_g}{o_g} = \frac{3}{2} \right) \) times more efficiently than Ooga. Hence, both Mog and Ooga would benefit if Mog specialized in fishing and Ooga specialized in gathering mushrooms.

\[
\left( \frac{m_f}{o_f} = 6 \right) > \left( \frac{m_g}{o_g} = \frac{3}{2} \right)
\]

Mog must specialize in fishing and Ooga must specialize in gathering mushrooms. Then they both must collaborate and trade to benefit from each other.

Though the analysis of the specialization solution is sound, it does not provide a concrete step-by-step approach to solve the generalization of the puzzle. In the next sections, we give solutions for solving the generic puzzle.

**Programming solution**

Let \( t_{\text{person.method}} \) denote the time taken by person \( \in \{ \text{Mog, Ooga} \} \) on method \( \in \{ \text{gathering, fishing} \} \). In short, we will denote them by \( t_{m,g}, t_{m,f}, t_{o,g}, t_{o,f} \). We will assume that the total number of mushrooms gathered and fish caught must be at least \( (6 + 2 = 8) \) each. Also, Mog and Ooga each work for exactly 3 hours per day.

The puzzle can then be represented as follows. We need to find all solutions \( [t_{m,g}, t_{m,f}, t_{o,g}, t_{o,f}] \) such that:

\[
3t_{m,g} + 2t_{o,g} \geq 8
\]

\[
6t_{m,f} + 4t_{o,f} \geq 8
\]

\[
t_{m,g} + t_{m,f} = 3
\]

\[
t_{o,g} + t_{o,f} = 3
\]

For all possible values the four time variables can take, we check if the four inequalities are satisfied. The SOLVEINEQUALITIES program can be used to compute all the solutions. The program outputs six possible solutions. This means that Mog and Ooga can collect food in six different ways. The food collected and the time taken to collect the food by Mog and Ooga are shown in Table 7.

Food can be traded in different ways for different trading prices. A trading price refers to the relation between the trading objects. For example, a mushroom can be
traded for a fish. The trading price is set by the market (demand and supply) or the trading parties in case of a barter system. For simplicity, we assume that Mog and Ooga trade one mushroom for one fish.

Consider solution 2 from Table 7. After trading three mushrooms for three fish, the food collection of Mog and Ooga is as shown in solution 2 of Table 8, which is certainly better than the food matrix in Table 4. Note that Mog and Ooga could also have traded four mushrooms for four fish, in which case the food matrix after trade would have been different and is not shown due to space constraints. Table 8 shows the food matrices for all solutions after trade, which are better than that of Table 4.

**Take-home lessons**

There are several life lessons to be learned from the puzzle. Some of them are as follows.

**[Specialization.]** Mog specialized in fishing and Ooga specialized in gathering mushrooms. We must specialize in something. Steve Jobs said, “Your work is going to fill a large part of your life, and the only way to be truly satisfied is to do what you believe is great work. And the only way to do great work is to love what you do. If you haven’t found it yet, keep looking. Don’t settle. As with all matters of the heart, you’ll know
when you find it.” We need to do what we love. When we work for, say, 10,000 hours in the topic we love, we become experts in the field. Hence, it is good for us to love something and specialize in it.  

[Collaboration.] Even though Mog was better than Ooga in both gathering mushrooms and fishing, he can benefit from collaboration. On similar lines, we frequently think that we are superior to others and often underestimate others. But, through humbleness and collaboration with others, we can achieve results that we can never even dream of.  

Charles Darwin once said, “It is the long history of humankind (and animalkind, too) that those who learned to collaborate and improvise most effectively have prevailed.” To this, Henry Ford added, “Coming together is a beginning, staying together is progress, and working together is success.” So, let’s collaborate and bring heavens on earth.  

Collaboration also includes the trade between nations. Countries that are superior in producing goods and services compared with other countries must not underestimate the power of collaboration and trade. They can benefit more from specialization and trade. Free trade between countries leads to economic growth, higher living standards, and prosperity.  

Specialize in whatever you love and/or great at and then collaborate with others to create supernatural and earth-shattering results.  

Problems  

1. [Programming solution.] Suppose there are \(n\) people \(P_1, P_2, \ldots, P_n\) and \(m\) types of food \(F_1, F_2, \ldots, F_m\). The food can be collected in integral quantities only. The food collection rate of person \(P_i\) for food \(F_j\) is \(r_{i,j}\). Initially, the total number of units of food \(F_i\) collected by all people is denoted by \(S_i\). Person \(P_i\) works for \(w_i\) hours per day. Come up with a programming solution for solving this generalized problem identifying the comparative advantage of different people.  

2. [Integer linear programming.] Solve the generalized problem using the integer linear programming technique (from operations research).  

References  

David Ricardo formulated the idea of comparative advantage. The problem and solution are taken from an excellent book [Bauman and Klein, 2010] (and [Bauman and Klein, 2011]), which is certainly one of the best books written on introductory economics. A yet another excellent book on trade, business, entrepreneurship, and economics is [Schiff and Lockman, 1985]. A survey of integer linear programming methods can be found in [Genova and Guliashki, 2011].
Larger or Smaller

Problem

Your friend writes down two arbitrary distinct numbers on two different slips of paper. You randomly choose one of the slips and see the number. Now you need to guess whether the number you saw is larger or smaller than the unseen number. You know that you have a winning probability of 50% through a blind guess. But, can you do better?

Solution

The current puzzle is a beautiful example from the fields of probability theory and randomized algorithms. It shows the weirdness and counter-intuitiveness of probability and the extreme power of randomness.

The theory of probability is often considered the toughest branch of mathematics because it confuses all people all the time. Many events that we so easily believe to be trivially true (resp. false) turn out to be false (resp. true) when we apply the theory of probability. Even when we decide and try to apply the theory of probability to solve problems, it is again so easy to apply the theory incorrectly and get weird results. Let’s hope to solve this beautiful puzzle correctly using probabilistic analysis.

Observations

Let the two numbers written by your friend be $s$ and $\ell$, where $s$ is the smaller number and $\ell$ is the larger number. Let the number seen by you be $x$. The number $x$ can be either smaller (i.e., $x = s$) or larger (i.e., $x = \ell$).

[Random versus arbitrary.] Many readers misinterpret the problem statement assuming that the two numbers are generated randomly, which is incorrect. The two numbers are arbitrarily generated and not randomly generated. We need to understand the difference between random and arbitrary. Choosing a random number means choosing a number based on a specific probability distribution. If the probability distribution is not specified, then random typically means uniform distribution. In contrast, choosing an arbitrary number means choosing a number however we desire with absolutely no rules to follow and no probability associated. In the puzzle, your friend chooses two arbitrary distinct numbers (i.e., literally any two unique numbers) and you choose one of the two slips randomly (i.e., with equal probability of 50%). Your friend can act as an adversary and can even fix the two numbers and use the same two numbers every time the game is played.

[Probability with respect to information.] Probability is not absolute. It varies depending on the amount of known and useful information. Extra useful information
affects the probability of the occurrence of an event. Two different people might compute two different probabilities for the same event depending on the amount of useful information they know. The same person might compute two different probabilities for the same event at two different times depending on the amount of useful information gathered. Sometimes the extra information might not be useful and hence the probability might not change. Moreover, it might be difficult to precisely define and describe the extra information and analyze whether that information is useful. In the case that the information is useful, the process of computing the new probability precisely might still be complicated.

The meaning of probability is unclear in our problem because the sample space for defining probability is not explicitly described. We can define success probability in two different ways: (i) Probability of answering the problem correctly given two numbers, and (i) Probability of answering the problem correctly given a probability distribution for the two numbers. A solution strategy assuming definition (i) implies a solution strategy assuming definition (ii). However, the reverse implication is not necessarily true. (Why?) Hence, given two arbitrary numbers $s$ and $\ell$, we would like to find a strategy for which the probability of answering the problem correctly is greater than 50% when the game is played (with the same two numbers $s$ and $\ell$) an infinite number of times.

**Zero-splitting strategy (not a solution)**

The zero-splitting strategy is as follows. If you see a positive number (i.e., $x > 0$), then guess that it is the larger number (i.e., $x = \ell$). If you see a negative number (i.e., $x < 0$), then guess that it is the smaller number (i.e., $x = s$). If you see a 0, then guess that it is larger or smaller, randomly.

![Figure 15: Two cases in the zero-splitting strategy: (i) When $x > 0$. (ii) When $x < 0$.](image)

**Winning probability.** Let’s analyze the winning probability. We have two cases, as shown in Figure 15

1. [Positive.] When you see a positive number $x > 0$, it seems as if there is more chance for the unseen number to be on the left of $x$ than to be on the right of $x$. This is because the interval length $(-\infty, x)$ seems larger than the interval length $(x, +\infty)$. Hence, we conclude that in this case the probability of $x$ being greater than the unseen number is greater than 50%.

2. [Negative.] Reasoning similar to the previous case, we conclude that in this case where $x < 0$, the winning probability is greater than 50%.

Hence, the zero-splitting strategy guarantees a winning probability greater than 50%. Right? Wrong! The reasoning given above is incorrect. Why?

Even one counterexample is enough to prove the incorrectness of a strategy. Here, we provide two scenarios where the strategy do not guarantee a winning probability
greater than half. Scenario 1: Suppose your friend selects two positive numbers (i.e., \(0 < s < \ell\)). Then, you guess that the number \(x\) is larger (i.e., \(x = \ell\)). In this case, the probability of you being correct is exactly 50%. Scenario 2: Suppose your friend selects two negative numbers (i.e., \(s < \ell < 0\)). Then, you guess that the number \(x\) is smaller (i.e., \(x = s\)). In this case, the probability of you being correct is exactly 50%.

As the strategy does not guarantee a winning probability greater than 50% in several scenarios, the strategy is not a solution to our puzzle.

The zero-splitting strategy doesn’t guarantee a winning probability greater than 50%.

**Cover’s random-splitting strategy**

A highly counter-intuitive strategy called the random-splitting strategy is as follows. The strategy is non-deterministic as it uses randomization. Generate a random number \(r\) from a probability density function \(f\) such that \(f\) is strictly positive everywhere on the real line. If \(x < r\), then announce that the number \(x\) is smaller. If \(x > r\), then announce that the number \(x\) is larger. If \(x = r\), then randomly choose between smaller and larger.

![Figure 16: Three feasible events in the random-splitting strategy.](image)

**[Winning probability.]** Let’s analyze the winning probability. Let \(\mathbb{P}(E)\) denote the winning probability of an event \(E\). The relationship between \(x\), \(s\), and \(\ell\) can fall into any one of the three feasible and mutually-exclusive events \(A\), \(B\), and \(C\), as shown in Figure 16. As the three events are mutually exclusive and exhaustive, \(\mathbb{P}(A) + \mathbb{P}(B) + \mathbb{P}(C) = 1\). We have:

1. **[Event A: \(s < \ell \leq r\).]** In this case, as \(x < r\), you announce that \(x\) is smaller. The probability that \(x\) is smaller is exactly 50% because the number you see can be \(s\) or \(\ell\) with 50% chance. So, the winning probability is \(50\% \times \mathbb{P}(A)\).

2. **[Event B: \(r \leq s < \ell\).]** In this case, because \(x > r\), you announce that \(x\) is larger. The probability that \(x\) is larger is exactly 50% because the number you see can be \(s\) or \(\ell\) with 50% chance. So, the winning probability is \(50\% \times \mathbb{P}(B)\).

3. **[Event C: \(s < r < \ell\).]** In this case, you announce that \(x\) is smaller or larger depending on whether \(x < r\) or \(x > r\), respectively. Irrespective of whatever you announce, you will be correct with 100% probability. So, the winning probability is \(100\% \times \mathbb{P}(C)\).

Moreover, due to the fact that \(r\) is chosen from a probability distribution \(f\) such that \(f(t) > 0\) for all \(t \in (-\infty, +\infty)\), we have \(\mathbb{P}(C) > 0\).

Thus,

Winning probability = Sum of the winning probabilities of events \(A\), \(B\), and \(C\)

\[
\begin{align*}
\text{Winning probability} & = 0.5 \cdot \mathbb{P}(A) + 0.5 \cdot \mathbb{P}(B) + \mathbb{P}(C) \\
& = 0.5 \cdot (\mathbb{P}(A) + \mathbb{P}(B) + \mathbb{P}(C)) + 0.5 \cdot \mathbb{P}(C) \\
& = (\mathbb{P}(A) + \mathbb{P}(B) + \mathbb{P}(C) = 1)
\end{align*}
\]
\[ P(C) = 0.5 + 0.5 \cdot P(C) \quad (P(C) > 0) \]
\[ > 50\% \]

The random-splitting strategy guarantees a winning probability greater than 50%.

The elegant strategy uses a probability distribution function. In subsequent sections, we explore the concepts and examples of probability distribution functions.

[Probability density functions.] The probability of a discrete random variable \( X \) taking a particular value \( a \) can be computed using a probability mass function (PMF) \( f \) as

\[ P(X = a) = f(a) \]

On the other hand, the probability of a continuous random variable \( X \) taking a particular value \( a \) is zero i.e., \( P(X = a) = 0 \). It is more useful to think of the probability of a continuous random variable \( X \) lying in an interval \((a, b)\) as the area under a probability density function (PDF) \( f \) in the interval \((a, b)\). The probability over the interval is then

\[ P(X \in (a, b)) = P(X \in [a, b]) \quad (P(X \in [a, b]) = \int_a^b f(t) \, dt) \]

The probability over the neighborhood of a specific point \( a \) can be written as

\[ P(X \in (a, a + dt)) = \int_a^{a+dt} f(t) \, dt \approx dt \cdot f(a) \]

for a small positive quantity \( dt \).

The probability density \( f(t) \) may or may not be (i) symmetric, (ii) analytically expressible through closed-form formulas, and (iii) differentiable at every point on the real line.

Please carefully observe the difference between probability density \( f(t) \) and probability. Probability density function is used as a tool to compute probabilities over intervals. Intuitively, the probability that \( X \) lies in an interval \((a, b)\) is the area under the probability density function curve \( f \) over the interval \((a, b)\). Consider an example. Suppose the natural lifespan of a person is in the interval \([0, \infty)\) years, where 0 denotes the time at which fertilization (the fusion of a male sperm cell and a female egg cell to produce a fertilized cell) occurs. The probability that a person dies at exactly 70 years (a real number with infinite precision) is 0. However, we can define a probability density function (generalizing and approximating from statistical data using machine learning algorithms) and then find the area under the curve of this function over the interval, say, \([70, 70.1]\) to get the probability that a person dies at approximately 70 years. Similarly, we can find the area under the density function over the interval \([100, 200]\) to find the probability that a person dies between 100 and 200 years.

We define cumulative distribution function (CDF) \( F \) for both discrete and continu-
Probability distribution  | PDF: $f(t)$  | CDF: $F(t)$  | Comments
--- | --- | --- | ---
Gaussian distribution  | $e^{-t^2/2}$  | $\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{t} e^{-x^2} \, dx$  | No closed-form for $F(t)$
Cauchy distribution  | $\frac{1}{\pi (1 + t^2)}$  | $0.5 + \frac{1}{\pi} \tan^{-1} t$  | Asymmetric
Logistic distribution  | $\frac{(e^t + 1)^2}{e^t (e^t + 1)}$  | $\frac{e^t + 1}{e^t - e^{-t}}$  | Nondifferentiable at $t = 0$
Gumbel distribution  | $e^{-t - e^{-t}}$  | $\frac{1}{2} + \left(\frac{\text{sign}(t)}{2}\right) (1 - e^{-|t|})$  |  
Laplace distribution  | $e^{-|t|}/2$  |  |  

Table 9: Continuous probability distributions over the real line with probability densities strictly greater than zero for all numbers.

Table 10: Probability density functions over the real line with probability densities strictly greater than zero for every point.

**ous random variables** as

$$\mathbb{P}(X \leq a) = F(a) = \left( \sum_{-\infty}^{a} f(i) \right) \text{ or } \int_{-\infty}^{a} f(t) \, dt$$

where, $f(t) \geq 0$ for all $t$, $F(-\infty) = 0$, and $F(\infty) = 1$.

When $f(t)$ is continuous, then $f(t)$ can be computed as

$$f(t) = \frac{d}{dt} F(t) = \text{Slope of the tangent of } F(t)$$

For this puzzle, we need a probability density function $f(t)$ such that the probability density at every point on the real line is greater than zero i.e., $f(t) > 0$ for all $t \in (-\infty, +\infty)$. There are an infinite number of families of such probability density functions. Table 9 gives a tiny list of probability density functions satisfying the desired properties and Table 10 presents plots for those density functions.

**RANDOMSPLITTING** and **RANDOMSPLITTINGSIMPLIFIED** are the algorithms for implementing the random-splitting strategy. (Can you prove the correctness of the two algorithms?) Both the algorithms heavily use the concepts of PDF and CDF. The **RANDOMSPLITTING** algorithm uses **RANDOMFROMPDF** to generate a random number from a PDF.
**RANDOMSPLITTING(x)**

**Input:** Number seen by you i.e., $x$.

**Output:** Whether $x$ is larger or smaller than the unseen number.

1. Choose a PDF $f(t)$ such that $f(t) > 0$ for all $t \in (-\infty, +\infty)$
2. $r \leftarrow \text{RANDOMFROMPDF}(f(t))$
3. if $x < r$ then
   4. print $x$ is smaller
4. else if $x > r$ then
   5. print $x$ is larger
5. else if $x = r$ then
   6. print $x$ is RANDOM(larger, smaller)

**RANDOMFROMPDF(f(t))**

1. $u \leftarrow \text{UNIFORMRANDOM}((0,1))$  // from real interval (0,1)
2. find the cumulative distribution function (CDF) $F(t)$ from the given PDF $f(t)$
3. find the inverse CDF i.e., $F^{-1}(t)$
4. return $F^{-1}(u)$

**RANDOMSPLITTINGSIMPLIFIED(x)**

**Input:** Number seen by you i.e., $x$.

**Output:** Whether $x$ is larger or smaller than the unseen number.

1. choose a PDF $f(t)$ such that $f(t) > 0$ for all $t \in (-\infty, +\infty)$
2. find the CDF $F(t)$ from the PDF $f(t)$
3. $u \leftarrow \text{UNIFORMRANDOM}((0,1))$  // from real interval (0,1)
4. if $F(x) < u$ then
   5. print $x$ is smaller
5. else if $F(x) > u$ then
   6. print $x$ is larger
7. else if $F(x) = u$ then
   8. print $x$ is RANDOM(larger, smaller)

[Uniform probability density function.] Consider a variant of the puzzle. Suppose the two arbitrary distinct numbers chosen by your friend are always taken from the real interval $[a, b]$ and you know it. Then, what strategy maximizes your probability of guessing whether the number seen by you is larger or smaller?

We simply use the random-splitting strategy, where a random number is generated from the uniform probability density function over the finite real interval $[a, b]$, which in turn is defined as:

$$
\begin{align*}
  f(t) &= \begin{cases} 
    0 & \text{if } t < a, \\
    \frac{1}{b-a} & \text{if } t \in [a, b], \\
    0 & \text{if } t > b. 
  \end{cases} \\
  F(t) &= \begin{cases} 
    0 & \text{if } t < a, \\
    \frac{(t-a)}{(b-a)} & \text{if } t \in [a, b], \\
    1 & \text{if } t > b. 
  \end{cases}
\end{align*}
$$
Using terminology and analysis from the previous sections, we have:

\[ \mathbb{P}(C) = \text{Winning probability when } s < r < \ell \]
\[ = \frac{(\ell - s)}{(b - a)} > 0 \]

The winning probability using uniform distribution is

\[ \text{Winning probability} = \text{Sum of winning probabilities of events A, B, and C} \]
\[ = 0.5 + 0.5 \cdot \mathbb{P}(C) \]
\[ = 0.5 + 0.5 \cdot \left( \frac{(\ell - s)}{(b - a)} \right) > 50\% \]

When the two arbitrary numbers are selected from the interval \([a, b]\), the random-splitting strategy with uniform probability density function guarantees a winning probability greater than 50%.

Is it possible to extend this strategy to an infinite interval \((-\infty, +\infty)\)? Is it possible to define a uniform probability distribution over the entire real line? It might not be possible to define a uniform distribution over the real line in a conventional way. This is because if we set \(f(t) = 0\) for all values of \(t\), then \(\int_{t=-\infty}^{+\infty} f(t) = 0\), and if we set \(f(t) = \epsilon\) for any fixed \(\epsilon > 0\), then \(\int_{t=-\infty}^{+\infty} f(t) \to \infty\). We are not able to see a way to get \(F(t) = \int_{t=-\infty}^{+\infty} f(t) = 1\) with uniform distribution. However, in the future, mathematicians might find a neat way to define a uniform distribution over the real line using new axioms of probability and/or advanced mathematics possibly making use of an infinite number of functions. Then, we will have a much better solution to our original puzzle.

[Integers.] It is impossible to generate random real numbers with infinite precision. So, let’s generate integers. Suppose your friend chooses two arbitrary numbers from the set of all integers. Then, what will be your best strategy?

Let \(N\), \(\mathbb{W}\), and \(Z\) denote the sets of natural numbers, whole numbers, and integers, respectively. To solve this puzzle variant, we need a probability mass function \(f(t)\) such that \(f(t) > 0\) for all \(t \in \mathbb{Z}\). Table 11 gives a tiny list of specific instances of PMFs over \(\mathbb{N}\) satisfying the desired properties and Table 12 presents plots for those probability functions. It is easy to transform PMFs defined over \(\mathbb{N}\) to PMFs defined over \(\mathbb{Z}\). (Can you think of some transformation methods?). Now we can use the PMF \(f(t)\) to solve the puzzle.

Generate a random integer using a PMF \(f(t)\) defined over \(\mathbb{Z}\) and then add 0.5 to it. Call the resulting non-integer random number \(r\). If \(x < r\), then announce that \(x\) is smaller. If \(x > r\), then announce that \(x\) is larger. As \(x \in \mathbb{Z}\) and \(r \notin \mathbb{Z}\), the relation \(x = r\) cannot hold. Inheriting the winning probability analysis (over the real line) from the previous sections, we conclude that:

The random-splitting strategy over the set of integers guarantees a winning probability greater than 50%.

This strategy also applies to the sets of natural numbers and whole numbers.
**Mathematical and Algorithmic Puzzles**

<table>
<thead>
<tr>
<th>Probability distribution</th>
<th>Domain</th>
<th>PMF: $f(t)$</th>
<th>CDF: $F(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometric distribution</td>
<td>$t \in \mathbb{N}$</td>
<td>$2^t$</td>
<td>$1 - 2^{-t}$</td>
</tr>
<tr>
<td>Poisson distribution</td>
<td>$t \in \mathbb{N}$</td>
<td>$\frac{1}{e(t-1)!}$</td>
<td>$\sum_{i=1}^{t} \frac{1}{e(i-1)!}$</td>
</tr>
<tr>
<td>Telescoping series</td>
<td>$t \in \mathbb{N}$</td>
<td>$\frac{1}{t(t+1)}$</td>
<td>$\frac{t+1}{e^{t+1}}$</td>
</tr>
<tr>
<td>Borel distribution</td>
<td>$t \in \mathbb{N}$</td>
<td>$t^{-1}$</td>
<td>$\sum_{i=1}^{t} \frac{1}{e^{i+1}}$</td>
</tr>
<tr>
<td>Logarithmic distribution</td>
<td>$t \in \mathbb{N}$</td>
<td>$-1$</td>
<td>$\sum_{i=1}^{t} -1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\frac{t}{2t \ln(0.5)}$</td>
</tr>
</tbody>
</table>

Table 11: Discrete probability distributions for the set of natural numbers with probabilities strictly greater than zero for all numbers.

Table 12: Probability mass functions over the set of natural numbers with probabilities strictly greater than zero for every point.

**Problems**

1. **Deterministic algorithm.** Design a deterministic algorithm to solve the original puzzle that guarantees a winning probability strictly greater than 50%? (Note: All pseudo-random generating algorithms are deterministic algorithms.)

2. **Variants.** Solve variants of the original puzzle with the following parameters: (i) The two numbers can be continuous or discrete and can be chosen from bounded or unbounded domains. (ii) The two numbers can be generated arbitrarily or from a probability distribution (uniform distribution or others). (iii) The domain intervals and the arbitrary nature or probabilistic distribution of the two numbers may or may not be known to you. (iv) The number you see can be from a probability distribution (known or unknown to you) or arbitrarily selected by your friend.

3. **Generalization.** Generalize the problem and solutions to $n$ numbers for identifying the largest of those $n$ numbers maximizing the winning probability?
References

The problem and the random-splitting strategy was first presented in [Cover, 1987]. Its integer version appeared in [Winkler, 2002]. A list of discrete and continuous probability distributions can be found in Wikipedia. Many nice probability puzzles are covered in [Mosteller, 1987]. Good textbooks on probability theory include [Pishro-Nik, 2016], [Dekking et al., 2005], and [Tijms, 2012]. To know more about randomized algorithms, please refer to [Karp, 1991] and the book [Mitzenmacher and Upfal, 2005].
Lion and Man

Problem

A lion tamer enters a circular cage to tame a wild lion. The lion is damn hungry and wants to eat the man. The man realizes this and tries to get out of the lion cage. However, the cage door is jammed and the man is not able to open the door. The lion and the man both have the same maximum speed but now are restricted by the circular cage. Assume that the lion and the man are two points present strictly inside a circle. Also, assume that the lion and the man are highly intelligent.

What strategy must the lion follow to catch the man soon? What strategy must the man follow to delay getting caught? For how long (distance and time) can the man run before eventually getting caught by the wild hungry lion?

Solution

The puzzle is one of the most beautiful and challenging mathematical puzzles. It belongs to the class of chases and escapes, also called pursuit and evasion problems. Such problems are often solved using geometry and/or calculus.

If the man was not bounded by the lion cage and instead he was on an infinite grassland, then he could have run at his maximum speed and avoided getting caught by the lion forever. But, life is not always fair. The man is unfortunately bounded by the circular cage. Now, what should he do?

Always run towards the man solution (incorrect)

In this approach, the man runs on the circumference of the circular cage and the lion always runs towards the man. Then, the man can escape from the lion forever.

[Observations.] Please refer to Figure 17. Let

- \( v, kv = \) speed of man and lion, respectively, for some real \( k > 0 \).
- \( O, r = \) center and radius of the circular cage, respectively.
- \( M_0 = (r, 0), L_0 = \) starting positions of man and lion, respectively.
- \( M, L(x, y) = \) current positions of man and lion, respectively.
- \( TM, LN = \) tangent and normal, respectively, to the lion's path at point \( L \).
- \( \theta = \) angle between \( L_0M \) and the \( x \)-axis.
- \( \omega = \) angle between \( TM \) and the \( x \)-axis.
- \( \phi = \) angle between \( L_0M \) and \( MT \).
- \( |ML| = d = \) distance between the man and lion.

The man's strategy is to run at the speed of \( v \) on the circumference of the circular cage. The lion's strategy is to always run towards the man at the speed of \( kv \). That is, the instantaneous velocity of the lion is always directed towards the man.
We want to find the curve of chase (or pursuit) of the lion and analyze whether the lion can catch the man or not.

[Differential equations approach.] A lion’s path or pursuit curve can be described from the relations between the three parameters \( d \), \( \omega \), and \( \theta \). First, we determine the equations of the tangent and the normal at position \( L \) in terms of \( d \), \( \omega \), and \( \theta \). Then, we determine the differential equations of the two lines. Finally, we aim to solve the differential equations to obtain closed-form algebraic functions relating \( d \), \( \omega \), and \( \theta \) that would give us the pursuit curve of the lion.

![Figure 17: The path of the lion when it always runs towards the man.](image1)

![Figure 18: The position vectors of the lion and man and also the distance vector.](image2)

Please refer to Figure 17. The man starts at time \( t = 0 \) from the position \( M_0 \) and the lion starts from the position \( L_0 \). The point \( L_0 \) can be anywhere inside the cage. We have chosen \( L_0 \) as the origin \( O \) for simplicity. When the man covers an angle \( \theta \) (or an arc length of \( r\theta \)) to reach \( M \), the lion would have traveled a distance of \( kr\theta \) to reach \( L \).

[Equations for the tangent and the normal at \( L \).] The point-slope equation of a line in the \( xy \)-plane that passes through a point \((x_0, y_0)\) and has a slope \( m \) is \( y - y_0 = m(x - x_0) \). So, the point-slope equation of the tangent \( TM \) that passes through the point \( M = (r \cos(\theta), r \sin(\theta)) \) and has a slope \( \tan(\omega) = \frac{\sin(\omega)}{\cos(\omega)} \) is

\[
y - r \sin(\theta) = \frac{\sin(\omega)}{\cos(\omega)} (x - r \cos(\theta))
\]

\[
\implies x \sin(\omega) - y \cos(\omega) = r(\sin(\omega) \cos(\theta) - \cos(\omega) \sin(\theta))
\]

\[
\implies x \sin(\omega) - y \cos(\omega) = r \sin(\omega - \theta) \quad \text{[Tangent equation]} \] (7)

Similarly, we need to compute the point-slope equation of the normal \( LN \) that passes through the point \( N \) and has a slope of \( -\frac{1}{\tan(\omega)} = -\frac{\cos(\omega)}{\sin(\omega)} \). First, we need to compute the coordinates of \( N \). We observe that \( |NM| = \frac{d}{\cos(\phi)} = \frac{d}{\cos(\omega - \theta)} \) and \( |L_0N| = r - |NM| \). Using \( |L_0N| \), we can compute the coordinates of \( N \) as \((|L_0N| \cos(\theta), |L_0N| \sin(\theta))\). Using the
coordinates of \( N \) and the slope, we get the point-slope equation of the normal \( LN \) as
\[
y - \left( \frac{r \cos(\omega - \theta) - d}{\cos(\omega - \theta)} \right) \sin(\theta) = -\cos(\omega) \left( x - \left( \frac{r \cos(\omega - \theta) - d}{\cos(\omega - \theta)} \right) \cos(\theta) \right)
\]
\[
\Rightarrow x \cos(\omega) + y \sin(\omega) = (r \cos(\omega - \theta) - d) \left( \frac{\sin(\theta) \sin(\omega) + \cos(\theta) \cos(\omega)}{\cos(\omega - \theta)} \right)
\]
\[
\Rightarrow x \cos(\omega) + y \sin(\omega) = r \cos(\omega - \theta) - d \quad \text{[Normal equation]} \quad (8)
\]

[Differential equations for the tangent and normal at \( L \).] We differentiate the tangent equation (Equation 7) w.r.t. \( \theta \) to get
\[
\frac{dx}{d\theta} \sin(\omega) + x \cos(\omega) \frac{d\omega}{d\theta} - \frac{dy}{d\theta} \cos(\omega) + y \sin(\omega) \frac{d\omega}{d\theta} = r \cos(\omega - \theta) \left( \frac{d\omega}{d\theta} - 1 \right)
\]
\[
\Rightarrow \frac{dx}{d\theta} \sin(\omega) - \frac{dy}{d\theta} \cos(\omega) - d \frac{d\omega}{d\theta} = -r \cos(\omega - \theta) \quad \text{(Use Equation 8)}
\]

Note that \( \frac{dx}{d\theta} = rk \cos(\omega) \). Also, \( \frac{dy}{d\theta} = \left( \frac{d\omega}{dx} \right) \left( \frac{dx}{d\omega} \right) = \tan(\omega) rk \cos(\omega) \). Inserting these values into the equation above and simplifying, we get
\[
d \cdot \frac{d\omega}{d\theta} = r \cos(\omega - \theta) \quad \text{[Tangent differential]} \quad (9)
\]

Similarly, we differentiate the normal equation (Equation 8) w.r.t. \( \theta \) to get
\[
\frac{dx}{d\theta} \cos(\omega) - x \sin(\omega) \frac{d\omega}{d\theta} + \frac{dy}{d\theta} \sin(\omega) + y \cos(\omega) \frac{d\omega}{d\theta} = -r \sin(\omega - \theta) \left( \frac{d\omega}{d\theta} - 1 \right) - \frac{dd}{d\theta}
\]
\[
\Rightarrow \frac{dx}{d\theta} \cos(\omega) + \frac{dy}{d\theta} \sin(\omega) = r \sin(\omega - \theta) - \frac{dd}{d\theta} \quad \text{(Use Equation 7)}
\]

Inserting the values of \( \frac{dx}{d\theta} \) and \( \frac{dy}{d\theta} \) into the equation above and simplifying, we get
\[
\frac{dd}{d\theta} = r (\sin(\omega - \theta) - k) \quad \text{[Normal differential]} \quad (10)
\]

Equations 9 and 10 represent a system of complicated and dull nonlinear differential equations. Unfortunately, the crazy system of differential equations cannot be solved algebraically using elementary functions of \( \theta, \omega, \) and \( d \) parameters. Oops! We have reached a dead end. So, instead, we try the simpler programming approach to compute the chase curve traced by the lion.

[Programming approach.] We use vector algebra and some differential calculus to trace the lion’s path algorithmically and programmatically.

Suppose \( \vec{M}[t], \vec{L}[t], \) and \( \vec{D}[t] \) denote the position vector of man, the position vector of lion, and the distance vector between the lion and the man at time \( t \geq 0 \), as shown in Figure 18. We can write
\[
\vec{M}[t] = M_x[t] \hat{i} + M_y[t] \hat{j}
\]
\[
\vec{L}[t] = L_x[t] \hat{i} + L_y[t] \hat{j}
\]
\[
\vec{D}[t] = \vec{M}[t] - \vec{L}[t]
\]
where, \( M_x, L_x \) are the \( x \)-axis coordinates, \( M_y, L_y \) are the \( y \)-axis coordinates, and \( \hat{i}, \hat{j} \) are the unit vectors in positive \( x \)- and \( y \)-axis directions, respectively. For simplicity, we avoid writing the function parameter \( t \) for vectors e.g.: we write \( \vec{L} \) instead of \( \vec{L}[t] \).
We compute the velocity of lion, i.e. \( \frac{d\vec{L}}{dt} \), as follows:

\[
\frac{d\vec{L}}{dt} = \frac{dL_x}{dt} \hat{i} + \frac{dL_y}{dt} \hat{j}
\]

= lion’s velocity = lion’s speed \cdot lion’s direction

= \( k \cdot \) man’s speed \cdot unit distance vector

= \( k \left| \frac{d\vec{M}}{dt} \right| \cdot \frac{\vec{D}}{|\vec{D}|} \)

= \( \frac{k}{|\vec{M} - \vec{L}|} \left( \frac{dM_x}{dt} \hat{i} + \frac{dM_y}{dt} \hat{j} \right) \cdot (\vec{M} - \vec{L}) \)

= \( k \sqrt{\left( \frac{dM_x}{dt} \right)^2 + \left( \frac{dM_y}{dt} \right)^2} \cdot \frac{(M_x - L_x)\hat{i} + (M_y - L_y)\hat{j}}{\sqrt{(M_x - L_x)^2 + (M_y - L_y)^2}} \)

When the man travels along the circular path of radius \( r \), we have

\[
M_x[t] = r \cos \left( \frac{vt}{r} \text{ radians} \right) \quad M_x[t + dt] = r \cos \left( \frac{v(t + dt)}{r} \text{ radians} \right)
\]

\[
M_y[t] = r \sin \left( \frac{vt}{r} \text{ radians} \right) \quad M_y[t + dt] = r \sin \left( \frac{v(t + dt)}{r} \text{ radians} \right)
\]

Substituting these values in the lion’s velocity equation and simplifying, we get

\[
\begin{align*}
L_x[t + dt] &= k \cdot \text{factor} \cdot (M_x[t] - L_x[t]) \quad (11) \\
L_y[t + dt] &= k \cdot \text{factor} \cdot (M_y[t] - L_y[t]) \quad (12)
\end{align*}
\]

where, \( \text{factor} = \sqrt{\frac{(M_x[t + dt] - M_x[t])^2 + (M_y[t + dt] - M_y[t])^2}{(M_x[t] - L_x[t])^2 + (M_y[t] - L_y[t])^2}} \) \quad (13)

[Limit circle and the conditions for capture.] The \textsc{RunTowardsTheMan} algorithm gives the strategy to trace or plot the curve of the lion’s pursuit. Figure [19] shows the lion’s chase or pursuit curve for \( k = 0.7, k = 1, \) and \( k = 1.2 \). In this problem, we are interested in \( k = 1 \). The chosen values of other parameters are: \( v = r = \text{ratio} = 1, dt = 0.01, \) and \( t_{max} = 15 \). The initial positions of the man and the lion were assumed to be \((r, 0)\) and \((0, 0)\), respectively. The readers can experiment setting different values for the parameters.

The chase (or pursuit) curve of the lion, after enough time, is asymptotic to a circle of radius \( \min(kr, r) \). We call this circle the limit circle. The limit circle is concentric to the cage circle of radius \( r \).

When the lion’s speed is strictly greater than the man’s speed, i.e., \( k > 1 \), then the lion will capture the man in a finite amount of time. When the lion’s speed is not strictly greater than the man’s speed i.e., \( k \leq 1 \), then the lion can never capture the man. Table at the bottom of Figure [19] summarizes the conditions for the man’s capture.
**Input:** Speeds of lion and man, cage radius.

**Output:** Plot the curve of pursuit of lion chasing the man.

1. \( v \) ← man’s speed; \( k \) ← (lion’s speed ÷ man’s speed); \( r \) ← cage radius
2. \( t \) ← 0; \( dt \) ← small time increment; \( t_{max} \) ← maximum time; \( \text{ratio} \) ← \( \frac{v}{r} \)
3. \((M_x[0], M_y[0]) \) ← \((r, 0)\); \textbf{plot} \((M_x[0], M_y[0])\) // man’s position
4. \((L_x[0], L_y[0]) \) ← \((0, 0)\); \textbf{plot} \((L_x[0], L_y[0])\) // lion’s position
5. \textbf{while} \((t + dt) \leq t_{max}\) do
6. \hspace{1em} \((M_x[t], M_y[t]) \) ← \((r \cos(\text{ratio} \times t), r \sin(\text{ratio} \times t))\)
7. \hspace{1em} \((M_x[t + dt], M_y[t + dt]) \) ← \((r \cos(\text{ratio} \times (t + dt)), r \sin(\text{ratio} \times (t + dt)))\)
8. \hspace{1em} \text{factor} \ ← k \times \sqrt{\frac{(M_x[t+dt] - M_x[t])^2 + (M_y[t+dt] - M_y[t])^2}{(M_x[t] - L_x[t])^2 + (M_y[t] - L_y[t])^2}}
9. \hspace{1em} \((L_x[t + dt], L_y[t + dt]) \) ← \((L_x[t] + \text{factor} \times (M_x[t] - L_x[t]), L_y[t] + \text{factor} \times (M_y[t] - L_y[t]))\)
10. \hspace{1em} \textbf{plot} \((M_x[t + dt], M_y[t + dt])\) // man’s position
11. \hspace{1em} \textbf{plot} \((L_x[t + dt], L_y[t + dt])\) // lion’s position
12. \hspace{1em} \( t \) ← \( t + dt \)

---

**Figure 19:** Top: Plots of the lion’s paths (in red) for \( k = 0.7 \), \( k = 1 \), and \( k = 1.2 \), respectively. Bottom: Table summarizing the conditions for the man’s capture.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Capture?</th>
<th>Limit circle radius</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k &lt; 1 )</td>
<td>No</td>
<td>( kr )</td>
</tr>
<tr>
<td>( k = 1 )</td>
<td>No</td>
<td>( r )</td>
</tr>
<tr>
<td>( k &gt; 1 )</td>
<td>Yes</td>
<td>( r )</td>
</tr>
</tbody>
</table>

When the man runs on the boundary of the circular cage and the lionchases him continuously, the man will escape from the lion forever, for all \( k \leq 1 \).

There is a flaw in the solution. Though the steps in the solution are logically correct, the strategy adopted by the lion (i.e., always running towards the man) is not optimal. In the subsequent sections, we show that when the man runs on the boundary of the circular cage, the lion can adopt a better strategy to capture the man.
Stay on the radius solution (incorrect)

In this approach, the lion stays on the radius connecting the man’s position, at all times. Then, the lion will catch the man in a finite amount of time.

[Core idea.] Please refer to Figure 20. The initial position of the man is \( M_0 = (r, 0) \) and that of the lion is the origin i.e., \( L_0 = (0, 0) \). When the man moves to position \( M \), the lion stays on the radius \( L_0M \) at position \( L \). When the man covers the quarter-circle \( M_0MT \) of length \( \frac{\pi r^2}{2} \), the lion covers the semicircle \( L_0LT \) of length \( \pi \left( \frac{r}{2} \right) = \frac{\pi r}{2} \). The lion captures the man at point \( T \).

To prove that the method works, we need to prove that when the lion stays on the radius \( L_0M \) at position \( L \), the length of the arc \( L_0L \) is the same as the length of the arc \( M_0M \). Note that for the small circle, the line \( L_0M_0 \) is a tangent at point \( L_0 \). This means that if the angle \( \angle M_0L_0M = \theta \), then the angle \( \angle L_0RL = 2\theta \). The arc length \( M_0M = r\theta \) and the arc length \( L_0L = \left( \frac{r}{2} \right) \cdot (2\theta) = r\theta \). Because \( M_0M = L_0L \), the method works and the lion captures the man at location \( T \).

[Differential equations approach.] We use the differential equations approach to finding a closed-form algebraic function for the pursuit curve of the lion.

Please refer to Figure 21. Suppose \( \vec{M}[t] \) and \( \vec{L}[t] \) denote the position vectors of man and lion, respectively, at time \( t \geq 0 \). We can write the vectors in polar coordinates (with radial length and subtended angle) as follows:

\[
\vec{M}[t] = \left( r, \frac{vt}{r} \right) \quad \vec{L}[t] = \left( L_d[t], \frac{vt}{r} \right)
\]

where, \( L_d[t] = |L_0L| \) is the radial distance of the lion at time \( t \). Let \( M_r[t], L_r[t] \) be the radial speed components and \( M_\theta[t], L_\theta[t] \) be the angular speed components of man and lion, respectively.

The lion’s pursuit curve can be known from computing the three quantities related...
to the lion: (i) radial distance $L_d[t]$, (ii) radial speed component $L_r[t]$, and (iii) angular speed component $L_\theta[t]$. Let’s compute these three quantities.

Consider Figure [21]. We observe that $M_\theta[t] = v$ and $M_r[t] = 0$. If the positions of the lion, the man, and the cage center are collinear, then we have

$$\frac{L_\theta[t]}{M_\theta[t]} = \frac{L_d[t]}{r} \implies L_\theta[t] = \frac{L_d[t]}{r} v$$

We relate the radial and angular speed components of the lion with the lion’s speed using the Pythagorean theorem to get

$$\sqrt{L_r[t]^2 + L_\theta[t]^2} = kv \quad \text{(relate } L_r[t], L_\theta[t] \text{ and the lion speed)}$$

$$\implies L_r[t] = \sqrt{k^2 v^2 - L_\theta[t]^2} \quad \text{(simplify)}$$

$$\implies L_r[t] = v \sqrt{k^2 - \frac{L_\theta[t]^2}{r^2}} \quad \text{(substitute for } L_\theta[t] \text{ and simplify)}$$

The most important observation in this approach is that we can relate the radial speed component and the distance component of the lion’s vector. Using this relation, we compute the lion’s distance component as follows:

$$\frac{dL_d[t]}{dt} = L_r[t] = v \sqrt{k^2 - \frac{L_d[t]^2}{r^2}} \quad \text{(relate } L_d[t] \text{ and } L_r[t])}$$

$$\implies dt = \frac{dL_d[t]}{kv \sqrt{1 - \left(\frac{L_d[t]}{kr}\right)^2}} \quad \text{(reorder and simplify)}$$

$$\implies \int dt = \int \frac{r \cdot dx}{v \sqrt{1 - x^2}} \quad \left(\text{substitute } x = \frac{L_d[t]}{kr} \text{ and integrate}\right)$$

$$\implies t = \frac{r}{v} \sin^{-1} x + \text{constant} \quad \left(\int \frac{dx}{\sqrt{1 - x^2}} = \sin^{-1} x + \text{constant}\right)$$

$$\implies t = \frac{r}{v} \sin^{-1} x \quad \text{(constant = 0 because } L_d[0] = 0)$$

$$\implies L_d[t] = kr \sin \left(\frac{vt}{r}\right) \quad \text{(substitute } x = \frac{L_d[t]}{kr} \text{ and simplify)}$$

Substituting $L_d[t]$ in the equations of $L_r[t]$ and $L_\theta[t]$, we find that

$$L_r[t] = kv \cos \left(\frac{vt}{r}\right) \quad L_\theta[t] = kv \sin \left(\frac{vt}{r}\right)$$

At time $t$, the lion is at a distance of $L_d[t] = kr \sin \left(\frac{vt}{r}\right)$ from the origin, subtends an angle of $\left(\frac{vt}{r}\right)$ radians w.r.t. the $x$-axis, has a radial speed component of $L_r[t] = kv \cos \left(\frac{vt}{r}\right)$, and an angular speed component of $L_\theta[t] = kv \sin \left(\frac{vt}{r}\right)$.

Using the position and speed components of the lion vector, we can compute (How?):

- **[Euclidean coordinates.]** The Euclidean coordinates of the lion’s position is $(L_x[t], L_y[t]) = \left(kr \sin \left(\frac{vt}{r}\right) \cos \left(\frac{vt}{r}\right), kr \sin^2 \left(\frac{vt}{r}\right)\right)$.
- **[Time of capture.]** The time required for the lion to capture the man is $\left(\frac{v}{r}\right) \sin^{-1} \left(\frac{1}{kr}\right)$. 

56
[Location of capture.] The position at which the lion captures the man in polar coordinates is \( (r, \sin^{-1}\left(\frac{1}{k}\right)) \).

[Curve of pursuit.] The closed-form curve of pursuit of the lion satisfies the equation of the semi-circle 
\[
L_x[t]^2 + \left(L_y[t] - \frac{kr}{2}\right)^2 = \left(\frac{kr}{2}\right)^2 \quad \text{for} \quad L_x[t] \geq 0.
\]

**STAYONTHERADIUS()**

**Input:** Speeds of lion and man, cage radius.

**Output:** Plot the curve of pursuit of lion chasing the man.

1. \( v \leftarrow \) man’s speed; \( k \leftarrow \) (lion’s speed \( \div \) man’s speed); \( r \leftarrow \) cage radius
2. \( t \leftarrow 0; \ dt \leftarrow \) small time increment; \( t_{\text{max}} \leftarrow (r/v) \sin^{-1}(1/k) \)
3. **while** \( t \leq t_{\text{max}} \) **do**
4. \( \theta[t] \leftarrow \frac{v}{r}; \ L_d[t] \leftarrow kr \sin(\theta[t]) \)
5. \( (M_x[t], M_y[t]) \leftarrow (r \cos(\theta[t]), r \sin(\theta[t])) \) // man’s position
6. \( (L_x[t], L_y[t]) \leftarrow (L_d[t] \cos(\theta[t]), L_d[t] \sin(\theta[t])) \) // lion’s position
7. **plot** \( (M_x[t], M_y[t]), (L_x[t], L_y[t]) \)
8. \( t \leftarrow t + dt \)

Figure 22: Top: An algorithm to trace the lion’s path if it always stays on the radius. Middle: Plots of the lion’s paths (in red) for \( k = 1 \), \( k = 1.5 \), and \( k = 4 \), respectively. In this problem, we are interested in \( k = 1 \). The values of other parameters we have chosen to plot the curves are: \( v = r = \text{ratio} = 1, \ \text{adt} = 0.007, \ \text{and} \ t_{\text{max}} = \left(\frac{kr}{2}\right)\sin^{-1}\left(\frac{1}{k}\right) \). The initial positions of the man and the lion were assumed to be \((r, 0)\) and \((0, 0)\), respectively. The readers can experiment for different values of the parameters.

[Conditions for capture.] Figure 22 top shows the STAY-ON-THE-RADIUS algorithm to trace or plot the curve of the lion’s pursuit. Figure 22 middle shows the lion’s chase or pursuit curve for \( k = 1 \), \( k = 1.5 \), and \( k = 4 \), respectively. In this problem, we are interested in \( k = 1 \). The values of other parameters we have chosen to plot the curves are: \( v = r = \text{ratio} = 1, \ \text{adt} = 0.007, \ \text{and} \ t_{\text{max}} = \left(\frac{kr}{2}\right)\sin^{-1}\left(\frac{1}{k}\right) \). The initial positions of the man and the lion were assumed to be \((r, 0)\) and \((0, 0)\), respectively. The readers can experiment for different values of the parameters.
When the lion’s speed is greater than or equal to the man’s speed, i.e., \( k \geq 1 \), then the lion will capture the man in a finite amount of time. Even if the man changes directions an arbitrary number of times at arbitrary locations, the lion can also change directions to ultimately capture the man. When the lion’s speed is strictly lesser than the man’s speed i.e., \( k < 1 \), then the lion can never capture the man. Table at the bottom of Figure 22 summarizes the conditions for the man’s capture.

When the man runs on the boundary of the circular cage and the lion always stays on the radius, the lion will capture the man, for all \( k \geq 1 \).

There is a flaw in the solution. Though the steps in the solution are logically correct, the strategy adopted by the man (i.e., always running along the circumference of the circular cage) is non-optimal. In the subsequent sections, we show that the man can adopt a better strategy by running on a polygonal path and surprisingly escape from the lion forever being inside the cage.

**Besicovitch’s polygonal path of infinite length solution**

The two previous solutions are incorrect because they assume non-optimal strategies for the lion and the man, respectively. Nearly 25 years after the problem was posed, a supremely beautiful and insanely counterintuitive correct solution was given by the great Russian-born mathematician Abram S. Besicovitch.

In this mind-blowingly awesome solution, the man follows a polygonal path of an infinite length to escape from the lion forever.

**[Observations.]** We consider the journey of the lion and the man in an infinite number of small intervals of time \( t_1, t_2, t_3, \) so on up to infinity. Let’s define several parameters that might help us to solve the problem.

- \( v \) = maximum speed of both lion and man.
- \( O \) = center of the circular cage.
- \( r \) = radius of the circular cage.
- \( M_0, L_0 \) = starting positions of man and lion, respectively.
- \( d_0 = |M_0L_0| \) = initial distance between man and lion.
- \( s_0 = |M_0C| \) = initial distance between man and center of the cage.
- \( t_i \) = time interval during step \( i \).
- \( M_i, L_i \) = positions of the man and lion, respectively, after time interval \( t_i \).
- \( x_i = |M_{i-1}M_i| = |L_{i-1}L_i| \) = distance traveled by the man or lion during time interval \( t_i \).
- \( d_i = |M_iL_i| \) = distance between the man and lion after time interval \( t_i \).
- \( s_i = |M_iC| \) = distance of the man from the center of the cage after time interval \( t_i \).

**[Core idea: Divergent and convergent infinite series.]** The mind-blowing idea for the solution comes from two infinite series. One series is divergent (the sum limits to infinity) and another series is convergent (the sum limits to a finite quantity). The divergent series is for computing the total time required to implement the strategies of man and lion. The convergent series is for computing the distance of the man from the center of the circular cage. We have:
The two series make sure that the man is always inside the circular cage and it takes an infinite time to implement the strategies. With some more computations, we make sure that there is always a nonzero distance between the man and the lion. In this way, the man can escape from the lion forever.

**[Strategies.]** The strategy of the lion is always to run towards the man at a speed $v$. The strategy of the man is always to run at a speed of $v$ in a direction perpendicular to the line joining his position and the lion’s position.

![Figure 23: The paths of the man and the lion during time interval $t_1$.](image)

**[Step 1.]** Refer to Figure 23 to see the paths of the man and the lion during time interval $t_1$. In the first time interval $t_1$, the man moves a distance of $x_1$ perpendicular to the line $M_0L_0$ at a speed of $v$, always choosing the direction that keeps him closer to the center. We choose $x_1$ as follows.

$$x_1 = \frac{r - s_0}{2}$$

Then the time interval $t_1$ can be computed by taking the ratio of the distance covered by the man and his maximum speed.

$$t_1 = \frac{x_1}{v} = \frac{r - s_0}{2v}$$

After the first time interval $t_1$, the man would have moved to a new position $M_1$ and the lion to its new position $L_1$. As the instantaneous velocity of the lion is $v$ and is always directed towards the man, the lion would reach the position $L_1$ when the man reaches the position $M_1$. In triangle $\triangle CM_0M_1$, we know that $\angle CM_0M_1 \leq 90^\circ$ (The angle $\angle CM_0M_1 = 90^\circ$ when $C$ is present on the line segment $M_0L_0$). Using Pythagorean theorem in $\triangle CM_0M_1$, we can upper bound the value of $s_1$ (the distance of the man from
the center of the cage) as
\[ s_i^2 \leq s_0^2 + x_i^2 \]

In triangle \( \triangle L_0M_0M_i \), we know that \( \angle L_0M_0M_i = 90^\circ \). Using Pythagorean theorem in \( \triangle L_0M_0M_i \) and considering the distance traveled by the lion, we can lower bound the value of \( d_i \) (distance between the lion and the man) as
\[ d_i = |M_iL_i| \geq |M_iL_0| - |L_1L_0| = \sqrt{d_0^2 + x_i^2} - x_i \]

[Step i.] In the \( i \)th \((i \geq 2)\) time interval \( t_i \), the man moves a distance of \( x_i \) perpendicular to the line \( M_iL_i \) at a speed of \( v \), always choosing the direction that keeps him closer to the center. The value of \( x_i \) is chosen as follows.
\[ x_i = \frac{r - s_0}{i + 1} \]

We use a reasoning similar as given in Step 1 to compute \( t_i, s_i, \) and \( d_i \) values or bounds.
\[ t_i = \frac{r - s_0}{iv}, \quad s_i^2 \leq s_{i-1}^2 + x_i^2, \quad d_i \geq \sqrt{d_{i-1}^2 + x_i^2 - x_i} \]

<table>
<thead>
<tr>
<th>Step (i)</th>
<th>Time interval ( (t_i) )</th>
<th>Distance traveled by the man ( (x_i) )</th>
<th>Distance b/w man and center ( (s_i) )</th>
<th>Distance b/w man and lion ( (d_i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( t_1 = \frac{r - s_0}{2v} )</td>
<td>( x_1 = \frac{r - s_0}{2} )</td>
<td>( s_1^2 \leq s_0^2 + x_1^2 )</td>
<td>( d_1 \geq \sqrt{d_0^2 + x_1^2 - x_1} )</td>
</tr>
<tr>
<td>2</td>
<td>( t_2 = \frac{r - s_0}{3v} )</td>
<td>( x_2 = \frac{r - s_0}{3} )</td>
<td>( s_2^2 \leq s_1^2 + x_2^2 )</td>
<td>( d_2 \geq \sqrt{d_1^2 + x_2^2 - x_2} )</td>
</tr>
<tr>
<td>3</td>
<td>( t_3 = \frac{r - s_0}{4v} )</td>
<td>( x_3 = \frac{r - s_0}{4} )</td>
<td>( s_3^2 \leq s_2^2 + x_3^2 )</td>
<td>( d_3 \geq \sqrt{d_2^2 + x_3^2 - x_3} )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>i</td>
<td>( t_i = \frac{r - s_0}{(iv)i!} )</td>
<td>( x_i = \frac{r - s_0}{i!} )</td>
<td>( s_i^2 \leq s_{i-1}^2 + x_i^2 )</td>
<td>( d_i \geq \sqrt{d_{i-1}^2 + x_i^2 - x_i} )</td>
</tr>
</tbody>
</table>

Table 13: Table summarizing \( t_i, x_i, s_i, d_i \) values for each step \( i \geq 1 \).

[Analysis.] The man can never get caught by the lion because the following three necessary and sufficient conditions are satisfied.

1. [Infinite time.] The sum of all time intervals is infinite. That is, \( t_1 + t_2 + t_3 + \cdots = \infty \).
\[ t_1 + t_2 + t_3 + \cdots = \frac{r - s_0}{2v} + \frac{r - s_0}{3v} + \frac{r - s_0}{4v} + \cdots = \frac{r - s_0}{v} \left( \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots \right) = \infty \quad \left( \sum_{i=1}^{\infty} \frac{1}{i} = \infty \right) \quad \text{[divergent series]} \]

2. [Nonzero distance.] At any finite time, the distance between the man and the lion is nonzero. That is, \( d_i > 0 \) for all \( i \).
**Input:** Speed of lion or man, cage radius, initial positions of lion and man.

**Output:** Plot the curve of pursuit of lion chasing the man using Besicovitch's polygonal path algorithm.

1. \( v \leftarrow \text{speed}; r \leftarrow \text{cage radius}; dt \leftarrow \text{small time increment}; \text{intervals} \leftarrow \#\text{intervals} \)
2. \((L_x, L_y) \leftarrow \text{lion's position}; (M_x, M_y) \leftarrow \text{man's position} \)
3. \(s \leftarrow \text{distance of point} (M_x, M_y) \text{ from the origin} \)
4. **for** \( i \leftarrow 1 \text{ to } \text{intervals} \text{ do} \)
5. \(x \leftarrow \left( \frac{r - x}{r^2} \right); \text{steps} \leftarrow \left( \frac{x}{v dt} \right); \text{flag} \leftarrow 1 \)
6. \(P_s \leftarrow \left( \frac{M_x - L_x}{L_y - M_y} \right), \text{factor} \leftarrow \frac{x}{\sqrt{1 + P_s^2}} \quad \text{// } P_s = \text{perpendicular slope} \)
7. \((A_x, A_y) \leftarrow (M_x + \text{factor}, M_y + P_s \cdot \text{factor}) \quad \text{// perpendicular endpoint } A \)
8. \((B_x, B_y) \leftarrow (M_x - \text{factor}, M_y - P_s \cdot \text{factor}) \quad \text{// perpendicular endpoint } B \)
9. **if** origin is closer to \( A \) than \( B \) **then**
10. \((M_x^{\text{next}}, M_y^{\text{next}}) \leftarrow (A_x, A_y) \quad \text{// next milestone is } A \)
11. **else**
12. \((M_x^{\text{next}}, M_y^{\text{next}}) \leftarrow (B_x, B_y), \text{flag} \leftarrow -1 \quad \text{// next milestone is } B \)
13. \(M_s \leftarrow \left( \frac{M_y^{\text{new}} - M_y}{M_x^{\text{new}} - M_x} \right) \quad \text{// man's line segment slope} \)
14. **plot** \((L_x, L_y), (M_x, M_y) \quad \text{// lion and man's positions} \)
15. **for** \( j \leftarrow 1 \text{ to steps} \text{ do} \)
16. \(L_s \leftarrow \left( \frac{(M_x - L_x)}{(M_y - L_y)} \right) \quad \text{// lion's curve slope} \)
17. \((M_x^{\text{new}}, M_y^{\text{new}}) \leftarrow \left( M_x + \text{flag} \cdot \frac{v dt}{\sqrt{1 + P_s^2}}, M_y + M_y \cdot (M_x^{\text{new}} - M_x) \right) \quad \text{// } M \text{'s new point} \)
18. \((E_x, E_y) \leftarrow \left( L_x + \frac{v dt}{\sqrt{1 + P_s^2}}, L_y + L_y \cdot (A_x - L_x) \right) \quad \text{// } L \text{'s feasible new point} \)
19. \((F_x, F_y) \leftarrow \left( L_x - \frac{v dt}{\sqrt{1 + P_s^2}}, L_y + L_y \cdot (B_x - L_x) \right) \quad \text{// } L \text{'s feasible new point} \)
20. **if** \( M^{\text{new}} \text{ is closer to } E \text{ than } F \text{ then} \)
21. \((L_x, L_y) \leftarrow (E_x, E_y) \quad \text{// } L \text{'s new point} \)
22. **else**
23. \((L_x, L_y) \leftarrow (F_x, F_y) \quad \text{// } L \text{'s new point} \)
24. **plot** \((L_x, L_y), (M_x, M_y) \quad \text{// lion and man's positions} \)
25. \((M_x, M_y) \leftarrow (M_x^{\text{next}}, M_y^{\text{next}}) \quad \text{// milestone of interval } i \text{ reached} \)

If \( d_0 > 0 \), then \( d_1 > 0 \) from the formula of \( d_1 \). If \( d_1 > 0 \), then \( d_2 > 0 \) from the formula of \( d_2 \). This process continues up to infinity. Thus, we have \( d_i > 0 \) for all \( i \).

3. **[Inside the cage.]** At any finite time, the man is inside the circular cage. That is, \( s_i \leq r \) for all \( i \).
   \[ s_i^2 \leq s_{i-1}^2 + x_i^2 \]
Figure 24: Plots of the man’s and lion’s paths for the following initial positions: (left) $L_0 = (0, 0)$ and $M_0 = (0.4, 0.4)$ and (right) $L_0 = (0.7, 0.7)$ and $M_0 = (-0.2, -0.2)$.

$$\leq (s_{i-2}^2 + x_{i-1}^2) + x_i^2$$
$$\leq s_0^2 + x_1^2 + x_2^2 + \cdots + x_i^2 \quad \text{(expand)}$$
$$= s_0^2 + (r - s_0)^2 + \left(\frac{r - s_0}{3}\right)^2 + \cdots + \left(\frac{r - s_0}{i}\right)^2$$
$$= s_0^2 + (r - s_0)^2 \left(\frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{i^2}\right)$$
$$< s_0^2 + (r - s_0)^2 \left(1 - \frac{1}{i + 1}\right) \quad \left(\therefore \sum_{k=2}^{i^2} \frac{1}{k^2} < 1 - \frac{1}{i + 1} \quad \text{[convergent series]}\right)$$
$$< s_0^2 + (r - s_0)^2$$
$$< (s_0 + (r - s_0))^2 = r^2$$

This implies $s_i \leq r$

We proved the three conditions.

The first condition says that it takes an infinite amount of time to follow the strategies. The second condition says that at any finite time there is a nonzero gap between the lion and the man which means the lion has not yet caught the man. The first and the second condition together mean that it takes an infinite amount of time for the lion to catch the man. Or in simple words, the lion can never catch the man.

The third condition says that the man in following his strategy never steps out of the cage and is always bounded by the circular lion cage. Also, for any finite amount of time, the man is inside the circular cage. The man being inside the circular cage implies that the lion is also inside the circular cage. The three conditions together solve the puzzle.
The distance traveled by the man or lion can be computed as

\[ \text{Distance traveled} = x_1 + x_2 + x_3 + \cdots = v \times (t_1 + t_2 + t_3 + \cdots) = \infty \]

\text{POLYGONALPATH gives the algorithm to trace or plot the chase and escape curves lion and man, respectively. Figure 24 shows the chase and escape curves given the initial positions of lion and man. The left plot is when the initial positions are } L_0 = (0, 0) \text{ and } M_0 = (0.4, 0.4) \text{ and the right plot is when the initial positions are } L_0 = (0.7, 0.7) \text{ and } M_0 = (-0.2, -0.2). \text{ The chosen values of the other parameters are: } v = r = 1, ddt = 0.01, \text{ intervals } = 50. \text{ The readers can experiment for different values of the parameters. The algorithm and the plots shown in the figure are for teaching purpose only. It is possible to optimize and fine-tune the algorithm to improve accuracy either by modifying the algorithmic details or by simply making use of sophisticated special-purpose programming language libraries.}

The algorithm execution is divided into \textit{intervals} number of time intervals or stages or milestones. Let the positions of the man and the lion be } M_{i-1} \text{ and } L_{i-1} \text{ after the end of stage } i-1. \text{ In stage } i, \text{ the man travels along a line segment with slope } P_s \text{ (perpendicular to the man’s direction at } M_{i-1}) \text{ for a time period of } t_i = \frac{(r - s_0)}{(i+1)v} \text{ covering a distance of } x_i = \frac{(r - s_0)}{(i+1)} \text{ so that } |AM_{i-1}| = |M_{i-1}B| = x_i. \text{ The points } A \text{ and } B \text{ are the two endpoints of the perpendicular line segment with } M_{i-1} \text{ at its center. The man moves to his next milestone } M_i, \text{ which is either } A \text{ or } B, \text{ depending on whichever location is closer to the origin. In stage } i \text{ of the simulation algorithm, the lion always moves towards the man. In fact, the lion can move in any arbitrary strategy.}

Each stage is subdivided into \textit{steps} number of tiny time steps. In step } j, \text{ the man moves on a straight line for a distance of } vd\text{ to his new position. The lion is directed towards the man. The slope of the lion’s curve is given by } L_s. \text{ The points } E \text{ and } F \text{ are the two endpoints of the line segment having slope } L_s \text{ with } L_{j-1} \text{ at its center and having } |EL_{j-1}| = |L_{j-1}F| = vd. \text{ The lion moves to its new position } L_j, \text{ which is either } E \text{ or } F, \text{ depending on whichever location is closer to the man’s new position. The process continues for all time steps in all stages. In this way, the paths of the man and the lion can be traced.}

\text{The man can escape from the lion forever and survive. After any finite time } t_1 + t_2 + \cdots + t_i, \text{ the distance between the man and the lion is } d_i > 0. \text{ The total distance traveled by the man (or lion) is } \infty.

We assume that the man and the lion are pure mathematical points having 0 sizes. If they have any nonzero size, the solution does not work. We assume that the points can travel in any direction at any time without following the laws of motion. We also assume that the point man and the point lion never die and live for an infinite amount of time. All these assumptions imply that the solution works only in the mathematical world and not in the real physical world. So, if possible, try not to enter a circular lion cage with a wild hungry lion to test the validity and practicality of the solution.
Take-home lessons

What can we learn from the puzzle? Two take-home lessons are presented here. We expect the reader to discover many more lessons.

[Dead end.] The differential equations approach for the always-run-towards-the-man solution led to a dead end. In life too, there are times when we end up at dead ends due to various reasons. But, we should never get disappointed and we should never quit. Often, a given problem can be solved in multiple ways. So, our mantra should be: “Try and try again till you succeed.”

Consider these mathematical and algorithmic examples: (i) Many complicated differential equations that cannot be solved through algebraic analysis can be solved approximately (with a given degree of accuracy) through numerical methods and algorithms. (ii) Many hard computer science problems for which there are no known fast (i.e., polynomial-time) algorithms can be solved efficiently using approximation algorithms (that give approximate answers) or randomized algorithms (that sometimes give incorrect answers). (iii) Many problems that require us to find the hidden function (or relationship) from streaming data, which cannot be solved to get 100% correct hidden functions (or relationships) can be solved through machine learning algorithms to get good approximations to the hidden functions.

So, never quit. Try and try again until you succeed.

[Thinking about present only.] The lion could catch the man in the stay-on-the-radius solution and not in the always-run-towards-the-man solution. In the always-run-towards-the-man solution, the lion always thought about the present or the current situation only by charging towards the man. In contrast, in the stay-on-the-radius solution, the lion thought carefully analyzing where the man would be in the future and took action based on that information. In simple words, the lion planned for the future. Remembering Benjamin Franklin’s words “If you fail to plan, you are planning to fail!” aptly applies to this situation.

For example, (i) Greedy algorithms take action considering only the next immediate best step. Hence, they often lead to non-optimal solutions because locally optimal choices don’t necessarily lead to globally optimal solutions. In contrast, dynamic programming algorithms take action considering all possible steps from the present to the distant future. Hence, they yield optimal solutions. (ii) Those who aim to maximize their material happiness spending a large amount of hard-earned money in the present often struggle financially and emotionally in later years, especially during retirement. On the other hand, those who work hard in their early years planning for the future usually will be happy in their later years reaping the benefits of their hard work.

So, always think and plan for the future instead of thinking about the present only.

Problems

1. [Different speeds.] Analyze the polygonal path of infinite length solution when \( k < 1 \) and \( k > 1 \), where \( k \) is the speed ratio i.e., lion speed ÷ man speed.
2. [Two lions.] Can the man survive if there are two lions in the circular cage?
3. [Birds and fly.] What is the necessary and sufficient number of birds to catch a fly in an \( n \)-dimensional spherical cage? What is the number in an unbounded space?

4. [Cop and robber game.] Consider a finite connected undirected graph. Initially, a cop and a robber are located on two different vertices. Then, the cop and the robber move simultaneously (variant: alternatively) to their neighboring vertices. The cop catches the robber if they end up at the same vertex. Given a graph, is it possible for the robber to escape from getting caught, forever? Given a graph, what is the minimum number of cops required to catch the robber?

**References**

The problem was posed by Richard Rado in the mid-1920s. Always-run-towards-the-man solution is presented in [Nahin, 2012], a nice book on chases and escapes puzzles, and [Davis, 1962]. Stay-on-the-radius solution is presented in [Nahin, 2012]. Abram S. Besicovitch discovered the polygonal path of infinite length solution. The solution is presented in [Littlewood, 1986], [Rado, 1973], [Petkovic, 2009], [Nahin, 2012], and [Schleicher and Lackmann, 2011]. Variants of the problem appeared in [Croft, 1964] and [Rado and Rado, 1974]. For a nice survey of several results related to the problem, refer to [Chung et al., 2011]. [Bonato and Nowakowski, 2011] is a detailed survey on the cop(s) and robber(s) puzzle.
Stirring Tea

Problem

Given a cup of tea, stirring and mixing transforms the tea in three dimensions. Show that there is at least one point in the cup of tea that ends up exactly where it began before mixing. We assume that the tea points are continuous; the tea is stirred in a continuous fashion without spilling or lifting the spoon; and there is no loss of tea after stirring.

Solution

This is a beautiful problem from topology! How should we proceed? This problem is all about proving one of the most celebrated results in topology called Brouwer’s fixed-point theorem. The terms continuous, closed, bounded, and convex will be defined in the subsequent section.

**Theorem 1** ([Brouwer’s fixed-point theorem](#)). For any continuous function mapping a closed bounded convex set to itself, there is a point in the set that is mapped to the exact same point. More formally, for any continuous function \( f : C \rightarrow C \), where \( C \in \mathbb{R}^d \) is a closed bounded convex set, then there is a point \( c \in C \) such that \( f(c) = c \).

[Definitions.] Given a function \( f : X \rightarrow X \), a **fixed point** \( c \in X \) is a point where \( f(c) = c \). When a function has a fixed point \( c \), the point \( (c, c) \) is on its graph. For example, \( y = x \) has an infinite number of fixed points (every point is a fixed point), \( y = 5 \) has exactly one fixed point, and \( y = -1/x \) has no fixed points.

An **n-D sphere** \( S^n \) of radius \( r \) is the generalization of circle to \( n \) dimensions having radius \( r \). It is formally defined as \( S^n = \{ (x_1^2 + x_2^2 + \cdots + x_{n+1}^2 = r) \mid (x_1, x_2, \ldots, x_{n+1}) \in \mathbb{R}^{n+1} \} \). An **n-D ball** \( B^n \) of radius \( r \) is the interior of the sphere \( S^n \) of radius \( r \). A **closed ball** includes the boundary i.e., \( B^n = \{ (x_1^2 + x_2^2 + \cdots + x_n^2 \leq r) \} \), whereas an **open ball** excludes the boundary, i.e., \( B^n = \{ (x_1^2 + x_2^2 + \cdots + x_n^2 < r) \} \). Unless otherwise mentioned, \( S^n \) and \( B^n \) will mean a unit sphere and a unit ball, respectively.

A **closed set** is a mathematical object in a Euclidean space consisting of a continuous interior region and a boundary. In contrast, an **open set** does not include the boundary points. The difference between the closed and the open regions forms the boundary of the object. It is possible for a continuous region in space to be neither closed or open. For example, \([-3, 7], [2, \infty)\), discs, balls, closed half-spaces, \( \{(x, y) \in \mathbb{R}^2 \mid |y| \geq \frac{1}{2} \} \), etc are closed sets; \((-3, 7), (2, \infty)\), open discs, open balls, open half-spaces, \( \{(x, y) \in \mathbb{R}^2 \mid |x + y| < 1 \} \) etc are open sets; the empty set \( \emptyset \) and \( \mathbb{R} \) are both open and closed sets; and \((-3, 7) \) and \( \{(x, y) \in \mathbb{R}^2 \mid x \geq 0, y > 0 \} \) are neither closed nor open.
A continuous region (i.e., closed/open set) in $d$-dimensional Euclidean space is convex if for any two points $p_1$ and $p_2$ in the region, all points on the line segment connecting $p_1$ and $p_2$ are also in the region. A continuous region in $d$-D Euclidean space is bounded if it is contained in some $d$-D ball of finite radius. Table 14 gives functions that satisfy or do not satisfy the three properties of boundedness, convexity, and closure.

A point $x$ is a point of closure of set $X$ in a Euclidean space if every open ball centered at point $x$ contains a point of $X$ (where this point might be $x$ itself). A point $x$ is a point of closure of set $X$ if and only if the point $x$ is an element of $X$ or if $x$ is a limit point of $X$. The closure of a set $C$ is the set of all points of closure of the set $C$. The closure of a set $C$ is the set of all points of $C$ together with its limit points.

A topological space $X$ is disconnected if it is the union of two disjoint non-empty open sets; else it is connected.

A function $f : X \to Y$ between two topological spaces is a homeomorphism or topological isomorphism if it satisfies all the three properties: (i) $f$ is a one-to-one correspondence, (ii) $f$ is continuous, and (iii) the inverse function $f^{-1}$ is continuous. Two topological spaces that have a homeomorphism between them are homeomorphic. For example, open interval $(a, b)$ and $\mathbb{R}$ are homeomorphic for any $a < b$, $n$-D solid hypercube and $n$-D ball are homeomorphic, coffee mug and donut are homeomorphic, $n$-D ball and solid $n$-D simplex are homeomorphic, closed interval $[a, b]$ and $\mathbb{R}$ are not homeomorphic for any $a < b$, Euclidean spaces $\mathbb{R}^m$ and $\mathbb{R}^n$ are not homeomorphic for $m \neq n$, real line and circle are not homeomorphic, and finally $[a, b]$ and $(a, b)$ are not homeomorphic for any $a < b$.

| Function | $|x| + |y| \leq 1$ | $|x| + |y| < 1$ | $|x| + |y| \in [1, 2]$ | $|x| + |y| \in (1, 2)$ | $y \geq x^2$ | $y > x^2$ | $y \leq x^2$ | $y < x^2$ |
|----------|------------------|----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| Bounded  | ✓                | ✓              | ✓               | ✓               | ✓               | ✓               | ✓               | ✓               |
| Convex   | ✓                | ✓              | ✓               | ✓               | ✓               | ✓               | ✓               | ✓               |
| Closed   | ✓                | ✓              | ✓               | ✓               | ✓               | ✓               | ✓               | ✓               |

Table 14: Regions in 2-D Euclidean space that satisfy or do not satisfy bounded, convex, and closed properties.

**Solution using intermediate value theorem (1-D)**

Let $f : [a, b] \to [a, b]$ be a continuous function. We see that the interval $[a, b]$ is closed, bounded, and convex. We will prove that there exists a point $c \in [a, b]$ such that $f(c) = c$, ultimately proving the Brouwer’s fixed-point theorem.

We first define a helper function $g(x) = x - f(x)$. We then use this function and the intermediate value theorem to prove the Brouwer’s theorem.

Using the definition of $g(x)$ we get $g(a) = a - f(a) \leq 0$ and $g(b) = b - f(b) \geq 0$. So, $g(a) \leq 0 \leq g(b)$. The intermediate value theorem from calculus states that if $f$ is a continuous function defined on the interval $[\ell, h]$, then it takes on any given value from $f(\ell)$ to $f(h)$ at some point within the interval. Applying this theorem to the continuous function $g(x)$ means that there is a point $c \in [a, b]$ such that $g(c) = 0$. Simplifying, we get $g(c) = c - f(c) = 0$, $c = f(c)$, which proves Brouwer’s fixed-point theorem.

It is possible to extend this 1-D proof to higher dimensions.
Solution using no-retraction theorem (2-D)

We first prove the no-retraction theorem in 2-D. We then use this theorem to prove the Brouwer’s theorem.

[Step 1. No-retraction theorem.] A function \( r \) is called a \textit{retraction} if it is a continuous function from a set to its proper subset such that \( r \) restricted to the proper subset is the identity. Let’s formalize this concept. Suppose \( Y \subseteq X \subseteq \mathbb{R}^n \), i.e., \( Y \) is a proper subset of a set \( X \) in a Euclidean space. Then the function \( r : X \to Y \) is called a retraction if \( r \) is a continuous function such that \( r(y) = y \) for all \( y \in Y \). \textit{No-retraction theorem} states that there is no retraction from the closed unit disk to its boundary the unit circle. That is, there is no retraction from \( n \)-D ball \( B^n \) to \((n - 1)\)-D circle \( S^{n-1} \), both of unit radius.

We now prove this no-retraction theorem for 2-D using proof by contradiction. Suppose \( r : B^2 \to S^1 \) is a retraction from the unit disk to its boundary. Consider two points \( s_1 \) and \( s_2 \) on the unit circle. Note that \( (S^1 - \{s_1, s_2\}) \) is a disconnected set as it consists of two open arcs. Let \( I_1 \) and \( I_2 \) be two subsets of \( B^2 \) such that the function \( r \) maps \( I_1 \) and \( I_2 \) to \( s_1 \) and \( s_2 \), respectively. That is, \( r(I_1) = s_1 \) and \( r(I_2) = s_2 \). Each of the two subsets \( I_1 \) and \( I_2 \) is closed and might be disconnected. Consider the remaining set \( I_3 = B^2 - (I_1 \cup I_2) \). The important observation here is that \( I_3 \) must be a \textit{connected subset} because \( I_1 \) and \( I_2 \) might be disconnected subsets and if \( I_3 \) was also disconnected, then by definition, \( B^2 = I_1 \cup I_2 \cup I_3 \) would be a disconnected set (but \( B^2 \) is connected).

Consider a point \( s_3 \) inside a clockwise arc between \( s_1 \) and \( s_2 \), not including \( s_1 \) and \( s_2 \). Similarly, consider a point \( s_4 \) inside a clockwise arc between \( s_2 \) and \( s_1 \), not including \( s_1 \) and \( s_2 \). We see that \( s_3 \) and \( s_4 \) are two points inside the connected set \( I_3 \). This implies that there is a continuous path from point \( s_3 \) to point \( s_4 \) inside the set \( I_3 \). As the retraction function \( r \) is continuous, it should map all points in the continuous path from point \( s_3 \) to point \( s_4 \) (inside the connected set \( I_3 \) in the ball \( B^2 \)) to a continuous arc from point \( s_3 \) to point \( s_4 \) (inside the connected set \( I_3 \) on the circle \( S^1 \)). However, the arc from point \( s_3 \) to point \( s_4 \) is discontinuous because of discontinuity at a point (either \( s_1 \) or \( s_2 \)) that is mapped from points not belonging to \( I_3 \) (e.g.: \( I_1 \) or \( I_2 \)). The arc from \( s_3 \) to \( s_4 \) can be discontinuous only if the set \( I_3 \) itself is disconnected. This implies the set \( I_3 \) is disconnected.

The set \( I_3 \) is simultaneously connected and disconnected. This is a contradiction! So, there does not exist a retraction function from the closed unit disk to its boundary.

[Step 2. Brouwer’s fixed-point theorem.] In this section, we prove Brouwer’s fixed-point theorem using proof by contradiction by making use of the no-retraction theorem.

Let’s first assume that the Brouwer’s theorem is false. That is, for any continuous function \( f \) mapping from disk \( B^2 \) to itself, there is no point \( c \in B^2 \) such that \( f(c) = c \). This means that for all \( x \in B^2 \), \( f(x) \neq x \). Suppose we now define a new continuous function \( g(x) \) as the intersection point of the circle \( S^1 \) with the extended ray that starts from \( f(x) \) and passes through \( x \) to hit the circle, as shown in Figure 25. Observe that if point \( x \) lies on the circle \( S^1 \), then \( g(x) = x \). As the continuous function \( g : B^2 \to S^1 \) and \( g(x) = x \) for all \( x \in S^1 \), the function \( g \) is a retraction. However, in Step 1, we proved
that there is no retraction from $B^2$ to its boundary $S^1$. This is a contradiction! So, for any continuous function $f$ mapping from disk $B^2$ to itself, there is a point $c \in B^2$ such that $f(c) = c$.

It is possible to extend this 2-D proof to higher dimensions.

**Solution using Sperner’s lemma (n-D)**

We first prove Sperner’s lemma. We then use this theorem to prove the Brouwer’s theorem.

**[Step 1. Sperner’s lemma.]** We first define Sperner’s lemma in 2-D. Consider a triangle $ABC$ as shown in Figure 26 (top). Divide this triangle into a set of triangles, we call cells, as shown in the figure. This process is called triangulation. We then properly color the vertices of the cells in the triangulation using the following rules.

From now on, colors red, blue, and green represent color numbers 1, 2, and 3, respectively. [Rule (i).] The vertices $A$, $B$, and $C$ are colored (or labeled) with red (or 1), blue (or 2), and green (or 3), respectively. [Rule (ii).] The interior vertices of every edge use only the two colors of the edge’s endpoints. That is, the vertices along the edge $AB$ are colored red or blue, the vertices along $BC$ are colored blue or green, and the vertices along $AC$ are colored red or green. [Rule (iii).] The interior vertices of the triangulation of $ABC$ (i.e., the vertices of the triangulation that do not lie on any edge) can have any of the three colors red, blue, or green. Figure 26 summarizes the set of vertex coloring rules for a triangulation, which we call Sperner coloring. Let’s define a rainbow cell as a cell with vertices having all different colors. Sperner’s lemma in 2-D states that there are an odd number of rainbow cells in the triangulation of $ABC$.

We now prove Sperner’s lemma in 2-D. Suppose we have properly colored (using Sperner coloring) any triangulation of a cell. Let $t_{123}$ be the number of cells with vertex colors $\{1, 2, 3\}$ (or rainbow cells) in that triangulation and let $t_{12}$ be the number of cells with vertex colors $\{1, 1, 2\}$ or $\{1, 2, 2\}$. Let $e_{12}^{\text{ext}}$ be the number of boundary/exterior edges with vertex colors $\{1, 2\}$ and let $e_{12}^{\text{int}}$ be the number of interior edges with vertex colors $\{1, 2\}$. Let’s try to relate $t_{123}$, $t_{12}$, $e_{12}^{\text{ext}}$ and $e_{12}^{\text{int}}$ by counting the number of edges $\{1, 2\}$ in all cells in the triangulation. We have

\[
\#\text{Edges } \{1, 2\} \text{ in all cells} = \#\text{Cells with vertex colors } \{1, 2, 3\} \\
+ 2 \times \#\text{Cells with vertex colors } 1 \text{ and } 2 \\
= t_{123} + 2 \times t_{12}
\]
This is because each cell with vertex colors 1 and 2 only (i.e., \{1,1,2\} and \{1,2,2\}) have exactly two edges with vertex colors 1 and 2. Similarly, we have

\[\text{#Edges } \{1,2\} \text{ in all cells} = \text{#Boundary edges } \{1,2\} + 2 \times \text{#Interior edges } \{1,2\}\]

= \(e^{\text{ext}}_{12} + 2 \times e^{\text{int}}_{12}\)

This is because of the observation that each interior edge is shared by two adjacent cells. From the two equations above, we have

\[t_{123} + 2 \times t_{12} = e^{\text{ext}}_{12} + 2 \times e^{\text{int}}_{12}\]

\[\implies t_{123} = e^{\text{ext}}_{12} + 2 \times (e^{\text{int}}_{12} - t_{12})\]

Let’s focus on \(e^{\text{ext}}_{12}\). The boundary edges colored \{1,2\} can only be present between vertices \(A\) and \(B\) as per rule (ii). If the number of such \{1,2\} boundary edges between \(A\) and \(B\) is even, then it is not possible to have different colors for vertices \(A\) and \(B\). This means that \(e^{\text{ext}}_{12}\) is odd which automatically implies that \(t_{123}\) is odd. In other words, there is at least one rainbow cell in the triangulation of \(ABC\). This proves Sperner’s lemma in 2-D.

We first define a few terms before stating and proving Sperner’s lemma in \(n\)-D. An \(n\)-D \textit{simplex} is the generalization of the concept of triangle to arbitrary dimensions. For example, a 0-simplex is a point, a 1-simplex is line segment, a 2-simplex is a triangle, a 3-simplex is a tetrahedron, etc. A \textit{simplicial subdivision} of an \(n\)-D simplex is a partition of the simplex into smaller simplices called cells such that any two cells are either disjoint or they share a full face (an \((n-1)\)-D simplex). The cells need not be regular and the simplicial subdivision can be of any granularity. The \textit{Sperner coloring} of a simplicial subdivision of an \(n\)-D simplex \(T\) is a set of vertex coloring rules: (i) the \((n+1)\) vertices of \(T\) receive different colors from \(\{1,2,\ldots,n+1\}\), (ii) vertices on any face of \(T\) use only the colors on the face’s \(n\) vertices, and (iii) the interior vertices of the simplicial subdivision of \(T\) can have any of the colors from \(\{1,2,\ldots,n+1\}\). A \textit{rainbow cell} is a cell with vertices having all different colors. \textit{Sperner’s lemma} in \(n\)-D states that there are an odd number of rainbow cells in the simplicial subdivision of \(T\).

We prove the lemma using mathematical induction. Suppose \(P(n)\) denote the statement of Sperner’s lemma for \(n\)-D. [\textit{Basis step.}] It is straightforward to prove that \(P(1)\) is true. In the previous paragraphs, we showed that \(P(2)\) (Sperner’s lemma in 2-D) is also true. [\textit{Induction step.}] We assume that \(P(n-1)\) is true for \(n \geq 2\). We will now prove that \(P(n)\) is true. Let \(t_{12\ldots(n+1)}\) be the number of cells with vertex colors \(\{1,2,\ldots,(n+1)\}\) (or rainbow cells) in that simplicial subdivision. Let \(t_{12\ldots n}\) be the number of cells with vertex colors \(\{1,2,\ldots,n\}\) such that exactly one of these colors is used twice. Let \(f^{\text{ext}}_{12\ldots n}\) be the number of boundary/interior faces with vertex colors \(\{1,2,\ldots,n\}\) and let \(f^{\text{int}}_{12\ldots n}\) be the number of interior faces with vertex colors \(\{1,2,\ldots,n\}\). Let’s try to relate \(t_{12\ldots(n+1)}\), \(t_{12\ldots n}\), \(f^{\text{ext}}_{12\ldots n}\) and \(f^{\text{int}}_{12\ldots n}\) by counting the number of faces using colors \(\{1,2,\ldots,n\}\) in all cells in the simplicial subdivision. We have

\[\text{#Faces } \{1,2,\ldots,n\} \text{ in all cells} = \text{#Cells with vertex colors } \{1,2,\ldots,(n+1)\}\]

\[+ 2 \times \text{#Cells with vertex colors } \{1,2,\ldots,n\}\]

\[= t_{12\ldots(n+1)} + 2 \times t_{12\ldots n}\]

\[\text{#Faces } \{1,2,\ldots,n\} \text{ in all cells} = \text{#Boundary faces } \{1,2,\ldots,n\} + 2 \times \text{#Interior faces } \{1,2,\ldots,n\}\]
Combining the two equations above, we have

\[ t_{12\ldots(n+1)} + 2 \times t_{12\ldots n} = f^\text{ext}_{12\ldots n} + 2 \times f^\text{int}_{12\ldots n} \]

\[ \implies t_{12\ldots(n+1)} = f^\text{ext}_{12\ldots n} + 2 \times (f^\text{int}_{12\ldots n} - t_{12\ldots n}) \]

Let’s focus on \( f^\text{ext}_{12\ldots n} \). The boundary faces colored \( \{1, 2, \ldots, n\} \) can only be present on the big face between \( n \) vertices having colors \( 1, 2, \ldots, n \) of the \( n \)-D simplex as per rule (ii). Due to inductive hypothesis, \( P(n-1) \) is true and hence \( f^\text{ext}_{12\ldots n} \) is odd. This automatically implies that \( t_{12\ldots(n+1)} \) is odd. In other words, there are an odd number of rainbow cells in the simplicial subdivision of \( T \). This proves Sperner’s lemma in \( n \)-D.

**Figure 26:** Top left: Sample vertex coloring of a triangulation of \( ABC \) with red-1, blue-2, and green-3. Top right: Barycentric coordinate system for the triangulation of \( ABC \). Bottom: Sperner coloring of a triangulation.

**[Step 2. Brouwer’s fixed-point theorem.]** In this section, we prove Brouwer’s fixed-point theorem using proof by contradiction by making use of Sperner’s lemma.

We now prove Brouwer’s fixed-point theorem in 2-D for a convex triangle. Brouwer’s fixed-point theorem applies to a convex triangle also because a convex triangle is homeomorphic to a ball \( B^2 \). That is, we will prove that for any continuous function \( f \) mapping from a convex triangle \( T \) to itself, there is a point \( c \in T \) such that \( f(c) = c \).

Let’s consider a sequence of triangulations \( \mathcal{T}_1 (= T), \mathcal{T}_2, \mathcal{T}_3, \ldots \) such that each triangulation \( \mathcal{T}_{i+1} \) is generated by triangulating triangles of \( \mathcal{T}_i \). Now consider the triangulation at the \( i \)th iteration \( \mathcal{T}_i \). Let \( x = (x[1], x[2], x[3]) \) be a vertex of \( \mathcal{T}_i \). Let \( f : \mathcal{T}_i \rightarrow \mathcal{T}_i \) be an arbitrary function that maps a vertex in \( \mathcal{T}_i \) to a vertex in \( \mathcal{T}_i \) i.e., \( f(x[1], x[2], x[3]) = (f(x)[1], f(x)[2], f(x)[3]) \). Let \( g : \mathcal{T}_i \rightarrow \{1, 2, 3\} \) be a coloring function that maps vertices of \( \mathcal{T}_i \) to colors \( \{1, 2, 3\} \) defined using the function \( f \) as:

\[
g(x) = \begin{cases} 
1, & \text{if } f(x)[1] < x[1], \\
2, & \text{if } f(x)[1] \geq x[1] \text{ and } f(x)[2] < x[2], \\
3, & \text{if } f(x)[1] \geq x[1], f(x)[2] \geq x[2], \text{ and } f(x)[3] < x[3], 
\end{cases}
\] (14)
Is the function $g$ well-defined (i.e., is the function defined for every vertex of the triangulation)? We show that the function $g$ is well-defined if the function $f$ has no fixed point. We use the barycentric coordinate system (see top right part of Figure 26) to uniquely represent each vertex $x$ of triangulation $T_i$ as $(x[1], x[2], x[3])$, where the coordinates $x[1], x[2], x[3]$ are in the range $[0, 1]$ and $x[1] + x[2] + x[3] = 1$. The function $g$ is not defined for a vertex $(x[1], x[2], x[3])$ if $f(x[1]) \geq x[1]$, $f(x[2]) \geq x[2]$, and $f(x[3]) \geq x[3]$. However, it is impossible to have the scenario where $f(x[1]) > x[1]$, $f(x)[2] > x[2]$, and $f(x)[3] > x[3]$ because if all three coordinates increase simultaneously then $f(x)[1] + f(x)[2] + f(x)[3] > 1$ which is not allowed. Hence, the only possible case when $g$ is not defined is when $f(x)[1] = x[1]$, $f(x)[2] = x[2]$, and $f(x)[3] = x[3]$, i.e., $f(x) = x$. In other words, $g$ is not defined if and only if $f$ has a fixed point. That is, $g$ is well-defined if there $f$ has no fixed point.

We now show that the function $g$ represents a Sperner coloring of triangulation $T_i$. From the definition of $g$, we see that $g$ satisfies all the three rules of Sperner coloring as follows.

\[
\begin{align*}
    i. & \quad g(1, 0, 0) = 1, g(0, 1, 0) = 2, g(0, 0, 1) = 3 \quad \text{[red, blue, green]} \\
    ii. & \quad g(x[1], x[2], 0) = 1 \mid 2, g(x[1], 0, x[3]) = 1 \mid 3, g(0, x[2], x[3]) = 2 \mid 3 \quad \text{colors of edge’s endpoints} \\
    iii. & \quad g(x[1], x[2], x[3]) = 1 \mid 2 \mid 3 \quad \text{any of [red, blue, green]}
\end{align*}
\]

Hence, the function $g$ represents a Sperner coloring.

Let’s use proof by contradiction to show that $f$ has a fixed point. Initially we assume that $f$ does not have a fixed point. This implies that $g$ is well-defined. We showed that $g$ is a Sperner coloring of the triangulation $T_i$. Using Sperner’s lemma in 2-D, there are an odd number of rainbow cells in triangulation $T_i$. Let

\[
\mathbf{\triangle}_i = [x'_1, x'_2, x'_3] = \text{a rainbow cell in } T_i, \text{ where } x'_j \text{ is a vertex in } T_i \text{ with color } k.
\]

Consider the sequences of vertices in rainbow cells $\langle \mathbf{\triangle}_1, \mathbf{\triangle}_2, \mathbf{\triangle}_3, \mathbf{\triangle}_4, \mathbf{\triangle}_5, \ldots \rangle$:

\[
\begin{align*}
    \text{Sequence of vertices with color 1: } & \ (x'_1, x'_2, x'_3, x'_4, x'_5, \ldots) \\
    \text{Sequence of vertices with color 2: } & \ (x'_2, x'_2, x'_3, x'_4, x'_5, \ldots) \\
    \text{Sequence of vertices with color 3: } & \ (x'_3, x'_3, x'_3, x'_4, x'_5, \ldots)
\end{align*}
\]

We show that all three sequences above converge to the same limiting point. Bolzano-Weierstrass theorem states that every bounded infinite set of real numbers has a limit point. This theorem applied on sequence 1 implies that sequence 1 converges to a limit point, say, $x^\ast$. Similarly, sequences 2 and 3 also converge to the same limit point $x^\ast$ because the size of a rainbow cell $\mathbf{\triangle}_i$ tends to zero as $i$ tends to infinity. In summary,

\[
\lim_{i \to \infty} x'_1 = \lim_{i \to \infty} x'_2 = \lim_{i \to \infty} x'_3. \quad (16)
\]

We now prove that the limiting point $x^\ast$ is itself the fixed point of $f$. As we assumed initially that the function $f$ does not have any fixed point, we have $f(x^\ast) \neq x^\ast$. As $f(x^\ast) \neq x^\ast$, we have $f(x^\ast)[j] > x^\ast[j]$ for some coordinate $j \in \{1, 2, 3\}$. From Equations 14 and 15, $f(x^\ast)[j] < x'_j[j]$ for all $i \in \{1, 2, 3, 4, 5, \ldots\}$ and from Equation 16, $\lim_{i \to \infty} x'_j = x^\ast$, and these two observations imply that $f(x^\ast)[j] \leq x^\ast[j]$ by continuity. We derived that $f(x^\ast)[j]$ is simultaneously greater than $x^\ast[j]$ and less than or equal to $x^\ast[j]$, which is a
contradiction. This implies that our initial assumption that \( f \) does not have a fixed point is incorrect and \( f \) does indeed have a fixed point. Hence, for any continuous function \( f \) mapping a convex triangle \( T \) to itself, there is a point \( c \in T \) such that \( f(c) = c \).

This proof can be generalized to \( n \) dimensions.

**Problems**

- **[Mapping a map.]** Given that a map (scaled and/or rotated) of any rectangular region on Earth is placed completely inside that exact rectangular region, prove that there is at least one point on the map that is located on top of that point on the Earth’s surface.

- **[Twin places.]** Prove that at any time there is always a pair of diametrically opposite (or antipodal) places/points on the Earth’s surface with the exact same temperature and barometric pressure. We assume that all the physical parameters vary continuously and Earth is a perfect sphere.

**References**

The Brouwer’s fixed point theorem was proved by Luitzen Egbertus Jan Brouwer (1911). There are many proofs of the Brouwer’s theorem using topics such as topology, combinatorics, analysis, or game theory. Brouwer’s theorem in \( n \)-D can be proved by the generalization of the intermediate value theorem called Bolzano-Poincaré-Miranda theorem. The proof of the Brouwer’s theorem in 2-D using no-retraction theorem can be found in [Munkres, 2000]. Sperner’s lemma was first proved by Emanuel Sperner (1928). We follow the presentation of the proof of the Brouwer’s theorem using Sperner’s lemma from [Fox, 2021] (see also the thesis [Maliwal, 2016]). Please refer to [Park, 1999] for a detailed survey of many results related to Brouwer’s theorem. The twin places puzzle can be generalized to higher dimensions and solved using the Borsuk-Ulam theorem (see [Munkres, 2000]). Refer to [Gamelin and Greene, 1999] for an introduction to topology.

Brouwer’s theorem does not give any information about the location of fixed points. There are algorithms (e.g.: [Scarf, 1967]) to approximately compute those fixed points, which are useful in the calculation of economic equilibria.
Hercules and Hydra

Problem

In Greek mythology, the mighty demigod Hercules defeated and killed a monster water snake called *Hydra*. According to legends, Hydra had the magical ability to regrow heads when one of its heads was cut off. After several millenia, Hydras have evolved and have become more intelligent with mathematical abilities. Below are three types of Hydras that Hercules has to defeat and kill, to save the world. Are there winning strategies for Hercules to defeat and kill these Hydras?

The Hydra is so powerful that Hercules can strike the Hydra exactly once per day. Let \( n \in \mathbb{N} \) be the initial number of heads of the Hydra.

**[Goodstein Hydra.]** Let \( G(n, i) \) denote the number of heads of Hydra after \( i \)th day. The computation of \( G(n, i) \) is explained through an example.

Suppose the Hydra has \( n = 266 \) number of heads. First, we express 266 in base-2 representation. Then, we write each of the exponents in base-2 representation and repeat the process to get the *hereditary base-2 representation*. That is,

\[
G(266, 0) = 2^8 + 2^3 + 2 = 2^2 + 2^{2+1} + 2 = 2^{2^{2+1}} + 2^{2+1} + 2
\]

After the first day of battle, we increase the base (i.e., from 2 to 3) and subtract the resulting number by 1. The process of increasing the base and subtracting by 1 is continued after each day until the process stops. The number of heads of the Hydra for the first few days are as shown in Table 15.

<table>
<thead>
<tr>
<th>#Days ( i )</th>
<th>Base</th>
<th>Computation ( G(n, i) )</th>
<th>#Heads ( G(n, i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>( G(266, 0) = 2^{2^8} + 2^{2+1} + 2 )</td>
<td>266</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>( G(266, 1) = 3^{3^8} + 3^{3+1} + 2 )</td>
<td>( \approx 4.4 \times 10^{38} )</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>( G(266, 2) = 4^{4^8} + 4^{4+1} + 1 )</td>
<td>( \approx 3.2 \times 10^{616} )</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>( G(266, 3) = 5^{5^{3^8}} + 5^{5+1} )</td>
<td>( \approx 2.5 \times 10^{10^{9921}} )</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>( G(266, 4) = 6^{6^{3^8}} + 6^{6+1} - 1 = 6^{6^{6^4}} + 6^8 \times 5 + 6^5 \times 5 + \cdots + 6 \times 5 + 5 )</td>
<td>( \approx 3.5 \times 10^{10^{27832}} )</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>( G(266, 5) = 7^{7^{2^8}} + 7^2 \times 5 + 7^5 \times 5 + \cdots + 7 \times 5 + 4 )</td>
<td>( \approx 1.1 \times 10^{10^{4571822}} )</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>( G(266, 6) = 8^{8^{6^8}} + 8^8 \times 5 + 8^5 \times 5 + \cdots + 8 \times 5 + 3 )</td>
<td>( \approx 1.7 \times 10^{10^{21210686}} )</td>
</tr>
</tbody>
</table>

Table 15: #Heads of the Goodstein Hydra for the first few days when \( n = 266 \).

Similarly, for a Hydra with arbitrary \( n \) number of heads, we compute \( G(n, i) \) as follows. Initially, represent \( n \) using the hereditary base-2 system and use the process of increasing the base by 1 and subtracting the result by 1 to compute the rest of the values.

**[Hypersphere Hydra.]** This type of Hydra has its soul in the form of a hypersphere and hence shows properties of a hypersphere. Let \( H(n, i) \) denote the number of heads of Hydra after \( i \)th day. Then, \( H(n, i) \) can be computed as:

\[
H(n, i) = \text{round}(V(n, i))
\]
where, $V(n, i)$ is the volume of a hypersphere of radius $n$ in $i$ dimensions and round function rounds off a given real number to its nearest integer.

For $n = 266$, #heads of Hydra on the first few days are shown in Table 16.

<table>
<thead>
<tr>
<th>#Days $i$</th>
<th>$V(n, i)$</th>
<th>#Heads $H(266, i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$-266$</td>
<td>266</td>
</tr>
<tr>
<td>1</td>
<td>$2r$</td>
<td>532</td>
</tr>
<tr>
<td>2</td>
<td>$\pi r^2$</td>
<td>222286</td>
</tr>
<tr>
<td>3</td>
<td>$\left(\frac{1}{2}\right)\pi r^3$</td>
<td>78837623</td>
</tr>
<tr>
<td>4</td>
<td>$\left(\frac{1}{2}\right)\pi^2 r^4$</td>
<td>$\approx 2.5 \times 10^{10}$</td>
</tr>
<tr>
<td>5</td>
<td>$\left(\frac{2}{3}\right)\pi^3 r^5$</td>
<td>$\approx 7 \times 10^{12}$</td>
</tr>
<tr>
<td>6</td>
<td>$\left(\frac{1}{2}\right)\pi^3 r^6$</td>
<td>$\approx 1.8 \times 10^{15}$</td>
</tr>
<tr>
<td>7</td>
<td>$\left(\frac{16}{1003}\right)\pi^3 r^7$</td>
<td>$\approx 4.4 \times 10^{17}$</td>
</tr>
</tbody>
</table>

Table 16: #Heads of the Hypersphere Hydra for the first few days when $n = 266$.

[Collatz Hydra.] The number of heads of this type of Hydra changes as follows. Initially, Hydra has $n > 1$ number of heads on day 0. If Hydra has even number of heads on day $i$ then Hercules knows a way to cut half the number of those heads by day $i + 1$. If Hydra has an odd (and greater than one) number of heads on day $i$, then the number of heads on day $i + 1$ will be one plus three times the number of heads its previous day. Finally, if Hydra has a single head on any day, then Hercules can easily cut off Hydra’s head and kill Hydra the following day.

As an example, when Hydra has $n = 266$ heads, Hercules can fight with Hydra and kill it in 30 days. The number of heads of Hydra on 30 different days are:

$266 \rightarrow 133 \rightarrow 400 \rightarrow 200 \rightarrow 100 \rightarrow 50 \rightarrow 25 \rightarrow 76 \rightarrow 38 \rightarrow 19 \rightarrow 58 \rightarrow 29 \rightarrow 88 \rightarrow 44 \rightarrow 22 \rightarrow 11 \rightarrow 34 \rightarrow 17 \rightarrow 52 \rightarrow 26 \rightarrow 13 \rightarrow 40 \rightarrow 20 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 0$

Can Hercules always defeat and kill these three types of Hydras?

Solution

This problem looks ridiculously difficult. It is easy to see that Goodstein and Collatz Hydra subproblems belong to number theory and the hypersphere Hydra subproblem belongs to geometry. Let’s see if Hercules can win in each of the three subproblems.

Goodstein Hydra

We attack this fantastic problem in two steps.
1. [Define transfinite numbers.] We define Cantor’s transfinite numbers.
2. [Prove that Hercules can kill Hydra.] We introduce and prove Goodstein’s theorem using transfinite numbers.

[Step 1. Define transfinite numbers.] The great mathematician Georg Cantor introduced transfinite numbers when he was developing the first mathematical theory
on sets consisting of an infinite number of elements. Transfinite numbers are counting numbers that transcend the finite numbers. That is, transfinite numbers are greater than all whole numbers (or non-negative integers).

A well-ordered set is an increasing sequence of elements having a minimum element. For example, the set of integers is not well-ordered because there is no minimum element. In contrast, the set of whole numbers is well-ordered.

The two types of transfinite numbers are ordinal numbers (or ordinals) and cardinal numbers. Ordinal numbers are defined for a well-ordered set. Ordinal elements depend on both the set elements and their order. On the other hand, cardinal numbers are defined for classes of sets that have the same cardinality i.e. the number of elements in the set. Cardinal numbers depend only on the set elements and not on their order.

The well-ordered set of counting numbers as thought by Cantor is:

\[
0, 1, 2, \ldots, \omega, \omega + 1, \ldots, \omega \times 2, \omega \times 2 + 1, \ldots, \omega^2, \ldots, \omega^\omega, \ldots, \omega^{\omega + 1}, \ldots, \omega^{\omega \times 2} , \ldots,
\]


Don’t panic! We will understand this ocean of Greek letters. The intuition behind ordinal numbers is that no matter how many elements we count, there is always one more element to be counted.

Let \( \{a_0, a_1, a_2, \ldots\} \) denote the first number that is greater than all elements in the well-ordered set. Then, the first number that is greater than all finite numbers is denoted by \( \omega \), i.e.,

\[
\omega = \{0, 1, 2, \ldots\}.
\]

The number \( \omega \) is followed by numbers \( \omega + 1, \omega + 2 \), and so on until we reach \( \omega \times 2 \).

\[
\omega \times 2 = \omega + \omega = \{0, 1, 2, \ldots, \omega, \omega + 1, \omega + 2, \ldots\}.
\]

A very surprising property of ordinal numbers is that the addition and multiplication operations on these numbers are not commutative and hence may lead to different results. For example, \( 1 + \omega \neq \omega + 1 \) and \( 2 \times \omega \neq \omega \times 2 \), as shown below:

[Addition:] 
\[
1 + \omega = \begin{array}{c}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\end{array}
= \begin{array}{c}
0 \\
1 \\
2 \\
3 \\
\omega \\
\omega \\
\omega \\
\omega \\
\omega \\
\omega \\
\end{array}
= \omega
\]

\[
\omega + 1 = \begin{array}{c}
0 \\
1 \\
2 \\
3 \\
\omega \\
\omega \\
\omega \\
\omega \\
\omega \\
\omega \\
\end{array}
= \begin{array}{c}
0 \\
1 \\
2 \\
3 \\
\omega \\
\omega \\
\omega \\
\omega \\
\omega \\
\omega \\
\end{array}
= \omega + 1
\]

[Multiplication:] 
\[
2 \times \omega = \begin{array}{c}
0 \\
1 \\
2 \\
3 \\
\omega \\
\omega \\
\omega \\
\omega \\
\omega \\
\omega \\
\end{array}
= \begin{array}{c}
0 \\
1 \\
2 \\
3 \\
\omega \\
\omega \\
\omega \\
\omega \\
\omega \\
\omega \\
\end{array}
= \omega
\]

\[
\omega \times 2 = \begin{array}{c}
0 \\
1 \\
2 \\
3 \\
\omega \\
\omega \\
\omega \\
\omega \\
\omega \\
\omega \\
\end{array}
= \begin{array}{c}
0 \\
1 \\
2 \\
3 \\
\omega \\
\omega \\
\omega \\
\omega \\
\omega \\
\omega \\
\end{array}
= \omega + \omega
\]

When we add two numbers \( a \) and \( b \), the sum \( a + b \) is obtained by appending or con-
catenating \( b \) to the end of \( a \), followed by recounting. When we multiply two numbers \( a \) and \( b \), the product \( a \times b \) is obtained by placing \( b \) copies of \( a \) side-by-side, followed by recounting. Note that the resulting values depend on the order of elements.

We continue the process of counting from \( \omega \times 2 \) until we reach \( \omega \times \omega = \omega^2 \). Moving on, we reach milestones \( \omega^2 \times \omega = \omega^3, \omega^4, \) and so on until we encounter \( \omega^\omega \), where,

\[
\omega^\omega = 1 + \omega + \omega^2 + \omega^3 + \cdots
\]

Much further in this journey of counting we reach \( \epsilon_0 \), where,

\[
\epsilon_0 = 1 + \omega + \omega^\omega + \cdots
\]

The number \( \epsilon_0 \) consists of \( \omega \) omegas and it is the first ordinal number that cannot be obtained from its previous ordinal numbers using a finite number of additions, multiplications, and exponentiations. It is also the first ordinal number that satisfies the equation \( \omega^\omega = \epsilon \). Further counting leads to \( \epsilon_1 \), where,

\[
\epsilon_1 = (\epsilon_0 + 1) + \omega^\omega \omega + \omega^\omega \omega + \cdots
\]

Eventually, we reach a big milestone \( \epsilon_{\omega \epsilon} \), which is the first ordinal number that satisfies the equation \( \epsilon_\alpha = \alpha \).

Note that this journey to the ultimate infinity never stops. We might exhaust all symbolic letters or get bored or even run out of time and die one day but the journey to the ultimate infinity never stops.

[Step 2. Prove that Hercules can kill Hydra.] To show that Hercules can kill Hydra, we need to prove that the number of heads of the Hydra eventually reaches 0 after some finite number of days. That is, we need to prove that \( G(n, i) = 0 \) for given \( n \in \mathbb{N} \) and some finite \( i \in \mathbb{N} \).

Each term \( G(n, i) \) is written in hereditary base-\((i+2)\) representation. This representation is also called the Cantor normal form. The sequence \( \langle G(n, i) \rangle \), i.e., \( \langle G(n, 0), G(n, 1), G(n, 2), \ldots \rangle \) is called the Goodstein sequence, named after Reuben Louis Goodstein. To show that Hercules can kill Hydra, we simply need to prove the Goodstein’s theorem.

**Theorem 2 (Goodstein’s theorem).** For any given \( n \), the Goodstein sequence eventually terminates at zero. That is, \( G(n, i) = 0 \) for some finite \( i \in \mathbb{N} \).

The core idea to prove Goodstein’s theorem is that we define a sequence \( \langle U(n, i) \rangle \) of ordinal numbers, which is an upper bound to the Goodstein sequence \( \langle G(n, i) \rangle \) and is decreasing. We use Cantor’s ordinal numbers to define \( U(n, i) \).

We prove the theorem in two parts.

[Part 1. Define an upper bound \( U(n, i) \geq G(n, i) \).] The term \( G(n, i) \) is represented in hereditary base-\((i+2)\) system. For example, \( G(266, 5) \) is represented in base-7. We define and construct \( U(n, i) \) as

\[
U(n, i) = \text{Value obtained by replacing base-}\((i+2)\) to base-\(\omega\) in \( G(n, i) \).
\]

Why do we need to replace the bases to \( \omega \)? For the simple reason that \( \omega \) is defined
to be greater than all finite numbers. This makes sure that the resulting number \( U(n, i) \geq G(n, i) \) for every finite number \( i \), until they both reach zero. Table 17 compares \( G(266, i) \) and \( U(266, i) \) for the first few values of \( i \). Note that in the representation of \( U(n, i) \), the coefficients are always written after the omega power terms (e.g.: \( \omega^5 \times 5 \) instead of \( 5 \times \omega^5 \)).

<table>
<thead>
<tr>
<th>( i )</th>
<th>( G(266, i) )</th>
<th>( U(266, i) )</th>
<th>( G(266, i) &lt; U(266, i) )?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( 2^{2^{i+1}} + 2^{i+1} + 2 )</td>
<td>( \omega^{i+1} + \omega^i + \omega )</td>
<td>( G(266, 0) &lt; U(266, 0) )</td>
</tr>
<tr>
<td>1</td>
<td>( 3^{3^{i+1}} + 3^{i+1} + 2 )</td>
<td>( \omega^{i+1} + \omega^i + 2 )</td>
<td>( G(266, 1) &lt; U(266, 1) )</td>
</tr>
<tr>
<td>2</td>
<td>( 4^{4^{i+1}} + 4^{i+1} + 1 )</td>
<td>( \omega^{i+1} + \omega^i + 1 )</td>
<td>( G(266, 2) &lt; U(266, 2) )</td>
</tr>
<tr>
<td>3</td>
<td>( 5^{5^{i+1}} + 5^{i+1} )</td>
<td>( \omega^{i+1} + \omega^i )</td>
<td>( G(266, 3) &lt; U(266, 3) )</td>
</tr>
<tr>
<td>4</td>
<td>( 6^{6^{i+1}} + 6^i \times 5 + 6^i \times 5 + \cdots + 6 \times 5 + 5 )</td>
<td>( \omega^{i+1} + \omega^i \times 5 + \omega^i \times 5 + \cdots + \omega \times 5 + 5 )</td>
<td>( G(266, 4) &lt; U(266, 4) )</td>
</tr>
<tr>
<td>5</td>
<td>( 7^{7^{i+1}} + 7^i \times 5 + 7^i \times 5 + \cdots + 7 \times 5 + 4 )</td>
<td>( \omega^{i+1} + \omega^i \times 5 + \omega^i \times 5 + \cdots + \omega \times 5 + 4 )</td>
<td>( G(266, 5) &lt; U(266, 5) )</td>
</tr>
<tr>
<td>6</td>
<td>( 8^{8^{i+1}} + 8^i \times 5 + 8^i \times 5 + \cdots + 8 \times 5 + 3 )</td>
<td>( \omega^{i+1} + \omega^i \times 5 + \omega^i \times 5 + \cdots + \omega \times 5 + 3 )</td>
<td>( G(266, 6) &lt; U(266, 6) )</td>
</tr>
</tbody>
</table>

**Table 17: Comparison of \( G(n, i) \) and \( U(n, i) \) for \( n = 266 \).**

**[Part 2. Show that \( \langle U(n,i) \rangle \) is a decreasing sequence.]** First, we define

\[
R(n, i) = \text{Value obtained by replacing base-}i\text{ to base-(}i+1\text{) in } n \\
R(n, i \rightarrow \omega) = \text{Value obtained by replacing base-}i\text{ to base-}\omega\text{ in } n
\]

Then, we write functions \( G \) and \( U \) in terms of function \( R \) as:

\[
G(n, i) = R(G(n, i - 1), i + 1) - 1 \\
U(n, i) = R(G(n, i), (i + 2) \rightarrow \omega)
\]

Now we can show that \( U(n, i) \) is a decreasing sequence by proving that \( U(n, i) > U(n, i + 1) \) for every \( i \) until \( U(n, i) = 0 \). We make use of the functions \( G \) and \( U \) written in terms of the function \( R \). We have:

\[
U(n, i) = R(G(n, i), (i + 2) \rightarrow \omega) \\
= R(R(G(n, i), (i + 2)), (i + 3) \rightarrow \omega) \\
> R(R(G(n, i), (i + 2)) - 1, (i + 3) \rightarrow \omega) \\
= R(G(n, i + 1), (i + 3) \rightarrow \omega) \\
= U(n, i + 1)
\]

This means that \( \langle U(n, i) \rangle \) is a decreasing sequence until \( U(n, i) \) reaches zero at some finite \( i = k \). After this point, \( U(n, i) = 0 \) for every \( i > k \). As an example,

\[
U(266, 0) > U(266, 1) > U(266, 2) > U(266, 3) > U(266, 4) > U(266, 5) > U(266, 6) > \cdots
\]

The value \( U(n, i) \) eventually reaches zero. As \( U(n, i) \geq G(n, i) \), the value \( G(n, i) \), too, eventually reaches zero.

This means that Hercules will eventually kill Hydra. The number of days Hercules takes to kill Hydra for small values of \( n \) are shown in Table 18.

Goodstein’s theorem is true. Hence, Hercules can kill Hydra after a finite #days.
<table>
<thead>
<tr>
<th>n</th>
<th>Sequence of Hydra heads ⟨G(n,i)⟩</th>
<th>#Days</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 → 0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2 → 2 → 1 → 0</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3 → 3 → 3 → 2 → 1 → 0</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4 → 26 → 41 → 60 → 83 → 109 → 139 → ··· → 0</td>
<td>3 · (2^10053211 − 1)</td>
</tr>
<tr>
<td>n</td>
<td>n → ··· → 0</td>
<td>Finite</td>
</tr>
</tbody>
</table>

Table 18: #Days required to kill the Goodstein Hydra for the first few values of the initial #heads n.

**Hypersphere Hydra**

We solve the problem in three steps. The first two steps solves the problem. The third step gives deeper insights on the problem.

1. [Compute the volume of a hypersphere.]
2. [Show that the volume of a hypersphere is zero in infinite dimensions.]
3. [Analyze.]

[Step 1. Compute the volume of a hypersphere.] We define a hypersphere of radius $r$ having its center at point $O$ in $n$-dimensional Euclidean space as a set of points that are at most a distance of $r$ from the point $O$. Let $V(r, i)$ denote the volume of an $i$-dimensional hypersphere of radius $r$. We compute $V(r, i)$ using geometry and calculus to obtain:

$$V(r, i) = \left(\frac{\pi^{\frac{i}{2}}}{\Gamma\left(\frac{i}{2} + 1\right)} \cdot r^i\right), \text{ where, } \Gamma \text{ is the Gamma function.}$$

The Gamma function is a generalization of the factorial function. It is defined for real numbers except non-positive integers as follows:

$$\Gamma(x) = \begin{cases} 
(x-1) \cdot \Gamma(x-1) & \text{if } x > 1, \\
\int_{0}^{\infty} t^{x-1} e^{-t} \, dt & \text{if } x \in (0,1], \\
\frac{\Gamma(x+1)}{x} & \text{if } x < 0 \text{ and } x \notin \mathbb{Z}.
\end{cases}$$

Substituting for Gamma function, we get:

$$V(r, i) = \begin{cases} 
\left(\frac{\pi^{\frac{i}{2}}}{(i/2)!} \cdot r^i\right) & \text{if } i \text{ is even,} \\
2 \cdot \left(\frac{i-1}{2}\right)! \cdot \left(4\pi\right)^{\frac{i-1}{2}} \cdot \left(\frac{i}{2}\right)! \cdot r^i & \text{if } i \text{ is odd.}
\end{cases}$$

[Step 2. Show that the volume of a hypersphere is zero in infinite dimensions.] Consider the original formula for $V(r, i)$. We assume that $r$ is fixed. The numerator of $V(r, i)$ function increases as $\pi^{i/2} \cdot r^i$. The denominator of $V(r, i)$ increases as $\Gamma\left(\frac{i}{2} + 1\right)$, which is of the order of $\left(\frac{i}{2}\right)!$. The rate of growth of $\left(\frac{i}{2}\right)!$ is greater than the rate of growth of $\pi^{i/2} \cdot r^i$. Hence, after some finite point, the function $V(r, i)$ keeps decreasing and limits to zero when the number of dimensions $i$ reaches infinity. An
interesting point to note is that \( V(r, i) \) goes below any arbitrarily small \( \epsilon > 0 \) in finite number of dimensions.

We could also prove the result using Stirling’s approximation in high dimensions. We obtain

\[
V(r, i) \approx \frac{1}{i^n} \left( \frac{r \sqrt{2\pi e}}{i} \right)^i
\]

\[
\implies V(r, \infty) \to 0.
\]

When \( i \) is large, \( \left( \frac{r \sqrt{2\pi e}}{i} \right) \) is less than 1, and the result follows.

The initial number of heads of the Hydra is \( n \). The Hydra will be killed when the number of heads of the Hydra reaches 0 i.e., \( H(n, i) = 0 \). This happens when \( V(n, i) \in [0, 0.5] \). From the reasoning above, \( V(n, i) \) reaches \( 0.5 \) after a finite number of dimensions. Hence,

[Hercules can kill Hydra after a finite #days.]

[Step 3. Analyze.] We have absolutely no understanding of the solution, even though we solved the problem fairly easily. What is the intuition behind the result? What is the geometric meaning of the solution? Let’s explore.

We analyze the result in three parts.

[Part 1. Analyze the hypercube volume.] Hypercubes are simpler objects than hyperspheres. We know that a hypercube is a line segment in 1-D, is a square in 2-D, and is a cube in 3-D. We also know that

Hypercube volume in \( i \) dimensions = \( s^i \), where, \( s \) is the side length.

We get a weird result when we use the formula for \( \infty \) dimensions (see Table 19).

\[
\text{Hypercube volume in } \infty \text{ dimensions} = \begin{cases} 
0 & \text{if side length } s < 1, \\
1 & \text{if side length } s = 1, \\
\infty & \text{if side length } s > 1.
\end{cases}
\]

<table>
<thead>
<tr>
<th>( s )</th>
<th>( i = 1 )</th>
<th>( i = 2 )</th>
<th>( i = 3 )</th>
<th>( i = 4 )</th>
<th>( i = 5 )</th>
<th>( i = 6 )</th>
<th>( i \to \infty )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0.5 )</td>
<td>5.0 \times 10^{-1}</td>
<td>2.5 \times 10^{-1}</td>
<td>1.3 \times 10^{-1}</td>
<td>6.3 \times 10^{-2}</td>
<td>3.1 \times 10^{-2}</td>
<td>1.6 \times 10^{-2}</td>
<td>0</td>
</tr>
<tr>
<td>( 1 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( 2 )</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>\infty</td>
</tr>
<tr>
<td>( 0.5 )</td>
<td>1.0 \times 10^0</td>
<td>7.9 \times 10^{-1}</td>
<td>5.2 \times 10^{-1}</td>
<td>3.1 \times 10^{-1}</td>
<td>1.6 \times 10^{-1}</td>
<td>8.1 \times 10^{-2}</td>
<td>0</td>
</tr>
<tr>
<td>( 1 )</td>
<td>2.0 \times 10^0</td>
<td>3.1 \times 10^0</td>
<td>4.2 \times 10^0</td>
<td>4.9 \times 10^0</td>
<td>5.3 \times 10^0</td>
<td>5.2 \times 10^0</td>
<td>0</td>
</tr>
<tr>
<td>( 2 )</td>
<td>4.0 \times 10^0</td>
<td>1.3 \times 10^1</td>
<td>3.4 \times 10^1</td>
<td>7.9 \times 10^1</td>
<td>1.7 \times 10^2</td>
<td>3.3 \times 10^2</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 19: Volumes (approximate) of hypercubes and hyperspheres with various side (\( s \)) and diameter lengths (\( r \)), respectively, in \( i \) dimensions.

The reason for the confusion and perplexity is as follows. Suppose the hypercube’s side length is \( s < 1 \). Then, the volume of a cube is smaller than the volume of a square because \( s^3 < s^2 \) for \( s < 1 \). However, according to our perception, an \( s \times s \) square is just an infinitesimally thin slice of an \( s \times s \times s \) cube. Because a cube can be considered
as a stack of height $s$ consisting of an infinite number of $s \times s$ squares. Hence, the space occupied by the cube must be larger than the space occupied by the square, irrespective of whether $s < 1$, $s = 1$, or $s > 1$. Extending this reasoning based on perception to higher dimensions, the hypercube volume should have been infinite in infinite dimensions for all three cases of $s < 1$, $s = 1$, and $s > 1$.

There is a big difference between how we perceive/imagine space/volume in the physical world and the mathematical definition of volume.

[Part 2. Understand how volume is measured.] We cannot let our perception define volume. In that case, a cube is a collection of an infinite number of squares and hence, the cube's volume will be an infinite times the square's area. So, our perception leads to nothing but infinity and confusion.

Volume is mathematically defined as follows. A unit length is 1 unit, a unit area is 1 unit $\times$ 1 unit, and a unit cube is 1 unit $\times$ 1 unit $\times$ 1 unit. The process continues to infinity. Observe carefully that we cannot compare a 2-D area and a 3-D volume. As an example, we cannot say that 5 square units is greater than 3 cubic units because it would be like saying 5 apples is greater than 3 houses. We know that $5 > 3$ as per mathematics, but a house occupies more space than an apple as per our perception. Applying this analogy to our subproblem, when $s < 1$, we cannot say that $s^3$ cubic units is smaller than $s^2$ square units because they cannot be compared.

The perceived spatial content (as thought in terms of the amount of solid matter) of an $i$-D hypercube is larger than that of an $(i-1)$-D hypercube, assuming that the side length $s$ is a fixed constant. However, the mathematical spatial content (i.e., volume) of an $i$-D hypercube cannot be compared with that of an $(i-1)$-D hypercube due to the difference in the units. Furthermore, when $s < 1$, the mathematical volume of a hypercube in infinite dimensions of size $(s \times s \times \cdots)$ vanishes to zero, only in comparison with a hypercube of size $(1 \times 1 \times \cdots)$.

[Part 3. Analyze the hypersphere volume.] Let’s now focus on hyperspheres. Let

\[ V_{\text{cube}}(s, i) = \text{Mathematical volume of an } i\text{-D hypercube with side length } s, \]
\[ V_{\text{sphere}}(r, i) = \text{Mathematical volume of an } i\text{-D hypersphere with radius } r. \]

The spatial content of an $i$-D hypersphere, according to our perception, is definitely larger than that of an $(i-1)$-D hypersphere, irrespective of whether the fixed radius $r < 1$, $r = 1$, or $r > 1$. However, $V_{\text{sphere}}(r < 1, \infty) = 0$ due to the difference in our perception and the mathematical definition of volume, as explained in the previous sections. But, why is $V_{\text{sphere}}(r > 1, \infty) = 0$, whereas $V_{\text{cube}}(s > 1, \infty) = \infty$? This implies that the mathematical volume of a hypersphere in infinite dimensions with radius, say, the width of our entire observable universe is zero. We can understand the difference between the geometrical interpretations of $V_{\text{sphere}}(r > 1, \infty) = 0$ and $V_{\text{cube}}(s > 1, \infty) = \infty$ as follows.

Suppose a hypersphere of radius $r$ is inscribed in a hypercube of side length $2r$. Then the resulting 2-D and 3-D objects are shown in Figure [27]. Let’s define density $d_i$
in $i$ dimensions as the ratio of volumes of the inscribed hypersphere and the hypercube in $i$ dimensions, i.e.,

$$d_i = \frac{V_{\text{sphere}}(r, i)}{V_{\text{cube}}(2r, i)} = \frac{\pi^\frac{i}{2}}{2^i \cdot \Gamma\left(\frac{i}{2} + 1\right)}.$$ 

![Figure 27: Hyperspheres inscribed inside hypercubes in 2-D and 3-D.](image)

The densities $d_i$ of the inscribed hyperspheres in various dimensions are shown in Figure 28. We see that $d_2 \approx 78.5\%$ for a square and $d_3 \approx 52.4\%$ for a cube and it decreases for higher dimensions.

<table>
<thead>
<tr>
<th>#Dimensions $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Density $d_i$</td>
<td>100</td>
<td>78.5</td>
<td>52.4</td>
<td>30.8</td>
<td>16.4</td>
<td>8.1</td>
<td>3.7</td>
<td>1.6</td>
<td>0.6</td>
<td>0.2</td>
</tr>
</tbody>
</table>

![Figure 28: Approx. density $d_i$ of inscribed hypersphere in an $i$-dimension hypercube.](image)

The density function $d_i$ decreases due to the hypercube corners. An $i$-D hypercube has $2^i$ corners. As dimensionality increases, these corners account for most of the volume of the hypercube and leave very little room for the hypersphere at the center of the hypercube. Hence, even though a hypersphere in infinite dimensions with radius the width of our observable universe is physically large enough to completely encompass our universe, its mathematical volume measured in (unit $\times$ unit $\times$ $\cdots$) reaches zero due to the geometric interpretation of decreasing density. This analysis leads to the following highly-confusing conclusion:

A property of an object can be measured to be either 0 or $\infty$ or any quantity between them, depending on the frame of reference and the choice of a measuring method.

**Collatz Hydra**

This problem is one of the famous open problems in *number theory*. It is also called the *Collatz conjecture* and the 3$n$+1 problem. As of the date of writing, nobody knows how to solve the problem.

Let $C_i$ denote the number of heads of Hydra after $i$th day. Initially, Hydra has $C_0 = n$ (> 1) heads. Then, $C_i$ can be computed as:

$$C_i = \begin{cases} \frac{C_{i-1}}{2} & \text{if } C_{i-1} \text{ is even,} \\ 3C_{i-1} + 1 & \text{if } C_{i-1} \text{ is odd and greater than 1.} \end{cases}$$

When $C_i$ reaches 1, then Hercules can easily kill Hydra the following day, i.e., $C_{i+1} = 0$. 

82
Given any initial number of Hydra heads \( n \), three cases exist for the number of Hydra heads on different days: 

(i) [Convergence.] In this case, \( C_i \) eventually reaches 0. So, Hercules will be able to kill Hydra. 

(ii) [Divergence.] In this case, \( C_i \) grows to infinity. So, Hercules will not be able to kill Hydra. 

(iii) [Loop.] In this case, \( C_i \) gets into a cycle of numbers and loops forever. So, Hercules will not be able to kill Hydra.

The sequence of Hydra heads \( \langle C_i \rangle \) on different days for a few initial values of \( n \) is given in Figure 29. A plot for the number of steps/iterations to reach 1 from any \( n \in [1, 30] \) is given in Figure 30. As of 2017, for \( n \in [1, 87 \times 2^{60}] \), convergence has been shown using computer calculations. For this problem, unfortunately, nobody knows if the sequence converges, diverges, or loops.

<table>
<thead>
<tr>
<th>( n )</th>
<th>Sequence of Hydra heads ( \langle C_i \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \langle 1, 0 \rangle )</td>
</tr>
<tr>
<td>2</td>
<td>( \langle 2, 1, 0 \rangle )</td>
</tr>
<tr>
<td>3</td>
<td>( \langle 3, 10, 5, 16, 8, 4, 2, 1, 0 \rangle )</td>
</tr>
<tr>
<td>4</td>
<td>( \langle 4, 2, 1, 0 \rangle )</td>
</tr>
<tr>
<td>5</td>
<td>( \langle 5, 16, 8, 4, 2, 1, 0 \rangle )</td>
</tr>
<tr>
<td>6</td>
<td>( \langle 6, 3, 10, 5, 16, 8, 4, 2, 1, 0 \rangle )</td>
</tr>
<tr>
<td>7</td>
<td>( \langle 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 0 \rangle )</td>
</tr>
<tr>
<td>8</td>
<td>( \langle 8, 4, 2, 1, 0 \rangle )</td>
</tr>
</tbody>
</table>

Figure 29: #Days required to kill the Collatz Hydra for the first few values of the initial #heads \( n \).

Figure 30: Plot of #steps required to reach 1 for \( n \in [1, 50] \).

It is not known if Hercules will be able to kill Collatz Hydra.

References
Georg Cantor introduced transfinite or ordinal numbers in 1883. The highly counter-intuitive Goodstein theorem is proved by and named after Reuben Louis Goodstein [Goodstein, 1944]. [Kirby and Paris, 1982], [Cichon, 1983], and [Buchholz and Wainer, 1987] proved that the Goodstein's theorem cannot be proved in Peano arithmetic. Transfinite numbers (i.e., ordinal and cardinal numbers) and several other types of numbers are beautifully presented in [Conway and Guy, 2012] and in [Rucker, 2013]. Goodstein's theorem, its proof, and its unprovability in Peano arithmetic is nicely presented in [Sladek, 2007]. A standard book on mathematical logic is [Hodel, 2013]. Gödel numbering is nicely presented in the book.

Hypersphere properties are covered in [Zaki and Meira Jr, 2014]. For a survey on the \( 3n + 1 \) problem, refer to [Lagarias, 2003] [Lagarias, 2006] [Lagarias, 2010].
Nerdy Needle

Problem

Can you find a plane figure of least area, in which a needle of length 1 can be freely turned around or rotated, possibly with translations, and always remaining inside the figure? Assume that the thickness of the needle is 0.

Solution

This problem, famously called Kakeya needle problem, is from the field of geometry. Though it seems like a dull geometry problem, it has a skull-drilling solution that will blow your mind. The puzzle is one of the best examples that elucidates the power of mathematical thinking and problem-solving.

Some solutions (non-optimal)

The following are a few decent non-optimal solutions to solve the problem.

[Circle.] Consider a circle of diameter 1 as shown in Figure 31. Place the needle on the line segment $AB$ so that the midpoint of the needle is exactly on the center $O$ of the circle. Rotate the needle by freely and see that the needle stays inside the circle. The area of the circle with radius 0.5 is $\pi \times \left(\frac{1}{2}\right)^2 = \frac{\pi}{4}$.

Area of a circle with diameter 1 is approximately 0.79.

[Reuleaux triangle.] Reuleaux triangles are named after Franz Reuleaux, a 19th century German engineer who used these shapes in his machine designs. Consider a Reuleaux triangle of width 1 as shown in Figure 31. The Reuleaux triangle of width 1 is constructed as follows: (i) $ABC$ is an equilateral triangle of side length 1, and (ii) Arc $AB$ is an arc of the circle centered at $C$ with radius 1 and having end points at $A$ and $B$. Arcs $BC$ and $AC$ are constructed similarly.

Place the needle on the line segment $AB$. Rotate the needle from line segment $BA$ to $BC$, then from line segment $BC$ to $AC$, and finally from line segment $AC$ to $AB$ and observe that the needle always stays inside the Reuleaux triangle. The area of the Reuleaux triangle with width 1 is $\frac{(\pi - \sqrt{3})}{2} \approx 0.70$.

Area of a Reuleaux triangle with width 1 is approximately 0.70.

The Reuleaux triangle idea can be generalized to other regular polygons with an odd number of sides having width 1. However, their areas will be in between the area...
Figure 31: Some good non-optimal solutions to the Kakeya needle problem.

of a Reuleaux triangle and a circle (i.e., Reuleaux polygon with an infinite number of sides).

[Pentagram.] Pentagram is a star-shaped polygon with five vertices. In popular culture, pentagram is often used as a symbolism of occult, magic, ghosts, demons,
and spirits. Consider a pentagram of height 1 as shown in Figure 31. A pentagram can be constructed as follows. Draw a pentagon i.e., a regular polygon with five vertices \( \{A, B, C, D, E\} \) and five sides \( \{AB, BC, CD, DE, EA\} \). Then draw diagonals (or edges) \( \{AC, AD, BD, BE, CE\} \) between every two vertices which do not have an edge. Finally, remove the five sides of the pentagon \( \{AB, BC, CD, DE, EA\} \). The figure we end up with is a pentagram.

Place the needle on the line \( AD \) such that one end of the needle is at \( A \). Rotate the needle from line \( AD \) to \( AC \) keeping the needle’s end point fixed at \( A \). Slide the needle on the line \( AC \) such that the other end of the needle touches \( C \). Rotate the needle from line \( CA \) to \( CE \) keeping the needle’s end point fixed at \( C \). Continue the process until the needle returns to its original position.

The movement of the needle can be represented as follows. Let the notation \( PR \) mean that the needle is on the line \( PR \) with a needle’s end point at \( P \). Let the notation \( \rightarrow \) mean a rotation and/or translation. Then, the needle’s travel itinerary in the pentagram is:

\[
AD \rightarrow AC \rightarrow CA \rightarrow CE \rightarrow EC \rightarrow EB \rightarrow BE \rightarrow BD \rightarrow DB \rightarrow DA \rightarrow AD.
\]

In this process, the needle always remains inside the pentagram. (How do you show that the needle rotates a complete 360° inside the pentagram?)

Let’s compute the area of the pentagram. Let the lengths \( |AF|, |AC|, |FC| \), and the angle \( \angle DAC = \theta \) denote the pentagram’s height, side, edge, and the internal angle, respectively. Then, we directly use the following formulas of the pentagram:

\[
\text{Edge} = \left( \frac{3 - \sqrt{5}}{2} \right) \cdot \text{Side}, \quad \theta = 36^\circ
\]

\[
\text{Height} = \text{Side} \cdot \cos \left( \frac{\theta}{2} \right) - \text{Edge} \cdot \cos \left( \frac{\pi}{2} - \theta \right)
\]

\[
\text{Area} = \left( \frac{1}{4} \right) \sqrt{650 - 290 \sqrt{5}} \cdot \text{Side}^2
\]

Substituting height as 1 (the needle length) and simplifying the equations above, we get area \( \approx 0.59 \).

Area of a pentagram with height 1 is approximately 0.59.

The generalization of a pentagram is a polygram. It would be interesting to analyze the Kakeya needle problem for different types of polygrams. However, analyzing polygrams is complicated.

[Equilateral triangle.] Consider an equilateral triangle \( ABC \) of height 1 (or side length \( 1/\cos 30^\circ \)) as shown in Figure 31. Place the needle on the line \( AB \) such that one end of the needle is at \( A \). Rotate the needle from line \( AB \) to \( AC \) keeping the needle’s end point fixed at \( A \). Slide the needle on the line \( AC \) such that the other end of the needle touches \( C \). Rotate the needle from line \( CA \) to \( CB \) keeping the needle’s end point fixed at \( C \). Slide the needle on the line \( CB \) such that the opposite end of the needle touches \( B \). Rotate the needle from line \( BC \) to \( BA \) keeping the needle’s end point fixed at \( B \). Slide the needle on the line \( BA \) such that the opposite end of the needle touches \( A \)
and thus we have arrived at the initial position of the needle. Observe that the needle remains inside triangle throughout the process. The area of the equilateral triangle with height 1 is $\frac{1}{\sqrt{3}} \approx 0.58$.

Area of an equilateral triangle with height 1 is approximately 0.58.

[Semicircular tube:] Consider a semicircular tube as shown in Figure 31. Let $R$ and $r$ be the radii of the outer and the inner circles such that the length of a tangent inside the figure is of length 1. Place the needle on the line segment $AB$. Rotate and translate the needle inside the tube such that it travels to the other end of the tube and reaches the line segment $DC$. Slide the needle along the line $CD$ such that the needle’s end point reaches $F$. Rotate the needle from $EF$ to $GF$ by an angle $\alpha$. Slide the needle along the line $FG$ such that the needle’s end point reaches $A$. Continue the process for a second time to cover $360^\circ$. The area of the final figure is computed as:

$$
\text{Area} = \text{Area of the semicircular tube} + \text{Area of the sector } EFG \\
= \left( \frac{\pi R^2}{2} - \frac{\pi r^2}{2} \right) + \alpha = \pi/8 + \alpha \\
\therefore R^2 = r^2 + \left( \frac{1}{2} \right)^2
$$

The area of the entire figure depends on $R$ (or $r$) because the area (and angle) of the sector $EFG$ is indirectly proportional to $R$. When we increase $R$, which in turn increases $r$, the area (and angle) $\alpha$ decreases. Hence, the area $\alpha$ can be made arbitrarily small by making $R$ arbitrarily large.

Area of a semicircular tube with tangent length 1 is approximately $0.39 + \alpha$ for arbitrarily small $\alpha > 0$.

[Three-cornered hypocycloid:] A hypocycloid is a plane curve traced by a fixed point on a circle that rolls around the interior of another circle. A three-cornered hypocycloid, also called a deltoid, inscribed in a circle of diameter $\frac{3}{2}$ is shown in Figure 31. A hypocycloid can be constructed as follows. Let $R$ and $r$ be the radii of the fixed and the rolling circles. Let $k = \frac{R}{r}$. If $k$ is a natural number, then the hypocycloid will have $k$ corners. The parametric equations of the hypocycloid are:

$$(x(\theta), y(\theta)) = (r(k-1) \cos(\theta) + r \cos((k-1)\theta), r(k-1) \sin(\theta) - r \sin((k-1)\theta)).$$

where $\theta \in [0, \infty)$ is the angle between the X-axis and the center of the rolling circle. For our current problem, $R = \frac{3}{4}$, $r = 1/4$, and hence $k = 3$ and we get a three-cornered hypocycloid.

The three-cornered hypocycloid inscribed in a circle of diameter $\frac{3}{2}$ has a special property that every tangent whose end points touch the curve is of length 1. Place the needle inside the three-cornered hypocycloid on a tangent at any point $M$ such that the two end points of the needle touch the curve. By simply rotating the needle as a tangent along the curve, the needle can be made to rotate a complete $360^\circ$ being totally inside the figure.

The area of a three-cornered hypocycloid is $2\pi r^2$. For our current problem, as $r = \frac{1}{4}$, we get the area as $\frac{\pi}{8} \approx 0.39$. Note that this area is exactly half of the area for the circle.
of diameter 1 (i.e., our first solution).

Area of a three-cornered hypocycloid inscribed in a circle of diameter $\frac{3}{2}$ is approximately 0.39.

Let’s generalize. A **hypotrochoid** is a curve traced by a fixed point attached to a circle or radius $r$ that rolls around the interior of a circle of radius $R$, where the fixed point is a distance $d$ from the center of the rolling circle. Let $k = \frac{R}{r}$. Then the parametric equations of the hypotrochoid are:

$$(x(\theta), y(\theta)) = (r(k - 1) \cos(\theta) + d \cos((k - 1)\theta), r(k - 1) \sin(\theta) - d \sin((k - 1)\theta)).$$

Similarly, an **epitrochoid** is a curve traced by a fixed point attached to a circle or radius $r$ that rolls outside a circle of radius $R$, where the fixed point is a distance $d$ from the center of the rolling circle. The parametric equations of the epitrochoid are:

$$(x(\theta), y(\theta)) = (r(k + 1) \cos(\theta) - d \cos((k + 1)\theta), r(k + 1) \sin(\theta) - d \sin((k + 1)\theta)).$$

A **spirograph** is a geometric drawing toy that can be used to create a variety of hypotrochoids and epitrochoids on paper. It uses plastic rings having gear teeth on both the interior and exterior of their circumferences.

By simply changing the parameters $R$, $r$, and $d$, we can generate aesthetically beautiful and enigmatic shapes and figures. Readers are strongly recommended to play around tweaking the three parameters using either a graph plotting software or a spirograph.

**Besicovitch-Perron-Schoenberg trees (non-optimal)**

We present this extremely beautiful and highly counterintuitive solution in three steps:

1. **[Construct a Besicovitch shape using bisection-and-expansion method.]** Construct a plane figure with arbitrarily small area that contains a line segment of length 1 in every direction.
2. **[Construct a Kakeya shape using Pal’s joins on the Besicovitch shape.]** Construct a plane figure with arbitrarily small area such that a line segment of length 1 can be rotated by $360^\circ$ continuously.
3. **[Compute and minimize the area of the Kakeya shape.]** Minimize the area of the Besicovitch shape and the area needed for Pal’s joins.

**[Step 1. Construct a Besicovitch shape using bisection-and-expansion method.]**

In this step, we define and construct a Besicovitch shape.

We define a **Besicovitch shape** as a plane figure that contains a line segment of length 1 in every direction. In further sections, we show how to construct a Besicovitch shape with arbitrarily small area. We first show how to construct a plane figure that contains a unit line segment in $90^\circ$ interval with arbitrarily small area. We then use this result four times, with necessary rotations covering $360^\circ$, to construct the desired Besicovitch shape.

Let a natural number $p \geq 2$ be our parameter. We will set the value of $p$ later. Let $\Delta_p$ denote a plane figure that contains a unit line segment in $90^\circ$ interval. The
Figure 32: The first five iterative steps to construct a Besicovitch shape with: (top) elementary triangles, (middle) ears at each step, and (bottom) only the boundary lines. The height is represented by the level number.

The iterative construction of $\Delta_p$ is as follows.

Please refer to Figure 32 (top). Let level-$i$ denote a height of $\frac{i}{p}$ from the horizontal line, where $i \in [0, p]$. Construct a right-angled isosceles triangle $\Delta_i$ with hypotenuse on the horizontal line and top vertex at level-1. Construct a right-angled isosceles triangle $\Delta_2$ with hypotenuse on the horizontal line and top vertex at level-2. We construct $\Delta_i$ from $\Delta_{i-1}$, where $i \in [3, p]$, using the bisection-and-expansion method.

Figure 33 shows how to use the bisection-and-expansion method to construct $\Delta_3$ from $\Delta_2$. Triangle $OAB$ denotes $\Delta_2$. Bisect the top angle $\angle AOB$ at level-2 using the median $OC$ to obtain two triangles $OAC$ and $OBC$. Expand the triangles $OAC$ and $OBC$ up till level-3 to obtain similar triangles $DAF$ and $EBG$, respectively. Observe that the new line segments $DF$ and $EG$ are parallel to the median $OC$. The resultant shape that has its top vertices at level-3 is $\Delta_3$. Similarly, we can construct $\Delta_i$ from $\Delta_{i-1}$ by applying the bisection-and-expansion method to all triangles that have top vertices at level-$(i-1)$. Consider each triangle that has a top vertex at level-$(i-1)$, bisect the
angle so that the median creates two new triangles, and finally expand the triangles to obtain similar triangles at level-i.

The triangles of \( \Delta_i \) at level \( i \) are called \textit{elementary triangles}. Figure 32 (top) shows the iterative steps for the construction of \( \Delta_1, \Delta_2, \Delta_3, \Delta_4, \) and \( \Delta_5 \) using the bisection-and-expansion method, along with the elementary triangles. The medians of the elementary triangles are shown using dotted lines.

Figure 34 shows that reordering the eight elementary triangles of \( \Delta_5 \) we obtain a right-angled isosceles triangle with its top vertex at level-5. Similarly, reordering the \( 2^{i-2} \) elementary triangles of \( \Delta_i \) we obtain a right-angled isosceles triangle with its top vertex at level-i.

<table>
<thead>
<tr>
<th>Mathematical property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of elementary triangles in ( \Delta_i )</td>
<td>( 2^{i-2} )</td>
</tr>
<tr>
<td>Top angle of an elementary triangle in ( \Delta_i )</td>
<td>( \left(\frac{90}{2^i}\right) )</td>
</tr>
<tr>
<td>Height of an elementary triangle in ( \Delta_i )</td>
<td>( \left(\frac{i}{2^i}\right) )</td>
</tr>
<tr>
<td>Base length of an elementary triangle in ( \Delta_i )</td>
<td>( \left(\frac{i}{2^i}\right) )</td>
</tr>
<tr>
<td>Area of an elementary triangle in ( \Delta_i )</td>
<td>( \frac{1}{2} \cdot \text{base} \cdot \text{height} = \left(\frac{i}{2^i}\right)^2 )</td>
</tr>
</tbody>
</table>

Table 20: A few important mathematical properties of \( \Delta_i \) for \( i \in [3, p] \).
continuous rotation of unit length needle in $\Delta_p$ is as follows. Enumerate the elementary triangles of $\Delta_p$ from left to right. Place the needle initially on a side of the leftmost elementary triangle with one of its end points at level $p$. Rotate the needle through the top angle till it reaches the other side of the elementary triangle. Similarly, the needle can sweep the top angles of all elementary triangles of $\Delta_p$ covering a total of $90^\circ$. Four $\Delta_p$’s make a Besicovitch shape in which the needle can rotate and cover $360^\circ$.

It is easy to see that the needle’s rotation is continuous in each elementary triangle. But, how does the needle move from one elementary triangle to the next elementary triangle continuously? We answer this important question in Step 2 using a very nice idea called Pal’s joins.

[Step 2. Construct a Kakeya shape using Pal’s joins on the Besicovitch shape.] In this step, we define and construct a Kakeya shape. We show how we can continuously move from an elementary triangle to the next elementary triangle using an idea called Pal’s joins.

We define a Kakeya shape as a plane figure such that a line segment of length 1 can be rotated by $360^\circ$ continuously. The relationship between Kakeya and Besicovitch shapes is

$$
\text{Kakeya shape} = \text{Besicovitch shape} + \text{Pal’s joins}
$$

Suppose that $\Delta_p$ is drawn above the horizontal line. We make use of the following two observations to construct a Kakeya shape:

1. [Adjacent elementary triangles have a pair of parallel sides.] The left side of the elementary triangle $T_i$ is parallel to the right side of the elementary triangle $T_{i+1}$, as can be seen in Figure 32 (top).

2. [Pal’s join between two parallel lines.] Given two parallel lines, a needle of length 1 can be moved (translated + rotated) from one line to the other sweeping an arbitrarily small area. This join is called Pal’s join.

Figure 35 shows a Pal’s join between two parallel lines. Suppose the unit length needle on the line segment $AB$ needs to be moved to $EF$ such that $AB$ is parallel to $EF$. The idea of Pal’s join to move from $AB$ to $EF$ consists of four steps: (i) Rotate the needle from $AB$ by an angle $\alpha$ to lie on the line $AD$, (ii) Slide the needle along the line $AD$ until one end of the needle touches $D$, (iii) Rotate the needle by an angle $\alpha$ so that the needle moves to $CD$, and finally (iv) Slide the needle from the line segment $CD$ to the line segment $EF$.

Figure 35: Pal’s join to move a needle continuously between two parallel lines.

Thus, we move from an elementary triangle to the next elementary triangle con-
tinuously using Pal’s joins. The movement of the needle can be represented as follows. Let $T_i,left$ and $T_i,right$ denote the left and right sides of an elementary triangle. Let the notations $\sim$ and $\Join$ mean rotation and Pal’s join, respectively. Then, the needle’s travel itinerary in $\Delta_p$ is:

$$T_i,right \sim T_i,left \Join T_{i+1},right \quad \text{for } i = 1, 2, 3, \ldots$$

[Step 3. Compute and minimize the area of the Kakeya shape.] Let $|\Delta_p|$ represent the area of $\Delta_p$. We compute the required areas as follows:

Area of the Besicovitch shape

$$= 4 \cdot |\Delta_p|$$

$$= 4 \cdot (|\Delta_{p-1}| + \text{Sum of the areas of all ears at level-}p)$$

$$\leq 4 \cdot (|\Delta_{p-1}| + 2|\Delta|)$$

$$\leq 4 \cdot (|\Delta_2| + 2(p - 2)|\Delta|) \quad \text{(Expand the recurrence)}$$

$$= 4 \cdot \left( \left( \frac{1}{2} \cdot \frac{4}{p} \cdot \frac{2}{p} \right) + 2(p - 2) \left( \frac{1}{2} \cdot \frac{2}{p} \cdot \frac{1}{p} \right) \right)$$

$$= 4 \cdot \frac{2}{p} = \frac{8}{p}$$

$$< \frac{\epsilon}{2} \quad \left( \text{Set } p > \frac{16}{\epsilon} \right)$$

Area for all Pal’s joins = Number of Pal’s joins $\times$ Area per Pal’s join

$$= (4n - 1) \cdot 2\alpha \quad (n = 2^{p-2})$$

$$< \frac{\epsilon}{2} \quad \left( \text{Set } p > \frac{16}{\epsilon} \text{ and } \alpha < \frac{\epsilon}{2(4n - 1)} \right)$$

Area of the Kakeya shape = Area of the Besicovitch shape + Area for all Pal’s joins

$$< \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon$$

Area of (Besicovitch-Perron-Schoenberg trees + Pal’s joins) is less than $\epsilon$ for arbitrarily small $\epsilon > 0$.

As an example, if the area of the Kakeya shape is upper bounded by $\epsilon = 0.01$, then the number of overlapping elementary triangles in these Besicovitch-Perron-Schoenberg trees is at least $2^{1600}(\approx 10^{481})$.

Note that it is possible to reduce the area of a Besicovitch shape to zero using other complicated methods of construction (e.g.: Besicovitch’s original construction, Jean-Pierre Kahane’s use of Cantor-like sets, etc). However, as the area required for Pal’s joins is positive, the area of a Kakeya shape that uses Pal’s joins is positive.

**Cunningham trees (non-optimal)**

Frederic Cunningham Jr.’s trees solves the Kakeya problem with arbitrarily small area and improves upon the Besicovitch’s mind-blowing solution in two aspects:

1. **[Diameter.]** The diameter of the Besicovitch-Perron-Schoenberg trees is unbounded and increases drastically with the parameter $p$. However, Cunningham trees is
2. [Connectedness.] The Besicovitch-Perron-Schoenberg trees are multiply connected. In contrast, Cunningham trees is simply connected.

In this section, we only give an intuitive sketch for the construction of Cunningham trees as the solution is quite involved with a lot of intricate mathematical details. Figure 36 shows a starting point to construct Cunningham trees. Draw a circle of radius 1. Draw a regular polygon $P$ with a large odd number of vertices $v$ inside the circle and concentric to the circle. The vertices of $P$ are numbered $C_0 = C_v, C_1, C_2, \ldots, C_{v+1} = C_1$ such that adjacent vertices are almost opposite to each other and connected through diagonals. The two diagonals $C_{i-1}C_i$ and $C_iC_{i+1}$ are extended through $C_i$ to intersect the circle at points $A_i$ and $B_i$, respectively, for $i \in [1, v]$. The isosceles triangle $C_iA_iB_i$ is denoted as triangle $T_i$. The perpendicular distance between parallel lines $C_{i-1}C_{i+1}$ and $A_iB_i$ must be at least 1. The Kakeya shape is the union of the polygon and the $v$ isosceles triangles.

Let the notations $\approx_Q$ and $\rightarrow$ denote rotation around point $Q$ and sliding (or translation), respectively. Then, the needle’s travel itinerary is:

$$C_{i-1}C_iA_i \approx_{C_i} C_{i+1}C_iB_i \rightarrow C_iC_{i+1}A_{i+1} \quad \text{for } i = 1, 2, 3, \ldots, v$$

The area of Cunningham trees is equal to the sum of the areas of the polygon $P$ and the triangles $T_1, T_2, \ldots, T_v$. We construct $P$ such that its area is $\epsilon/2$, for arbitrarily small $\epsilon > 0$. Initially, the sum of the areas of the triangles $T_1, T_2, \ldots, T_v$ is large. The large triangles are replaced with several thin triangles whose total area decreases in every iteration. The process is continued for several iterations using insights from the Besicovitch's solution to reduce the total area required by the thin triangles and joins to less than $\epsilon/2$. Hence, the total area of Cunningham trees is $\epsilon$.

Area of Cunningham trees is less than $\epsilon$ for arbitrarily small $\epsilon > 0$. 
<table>
<thead>
<tr>
<th>Kakeya shape</th>
<th>Size</th>
<th>Area</th>
<th>Convex?</th>
<th>Simply connected?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle</td>
<td>Diameter 1</td>
<td>≈ 0.79</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reuleaux triangle</td>
<td>Width 1</td>
<td>≈ 0.70</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Pentagram</td>
<td>Height 1</td>
<td>≈ 0.59</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Equilateral triangle</td>
<td>Height 1</td>
<td>≈ 0.58</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Semicircular tube</td>
<td>Tangent length 1</td>
<td>≈ 0.39 + α</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>3-cornered hypocycloid</td>
<td>Tangent length 1</td>
<td>≈ 0.39</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>BSP trees</td>
<td>Triangle height 1</td>
<td>&lt; ϵ</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Cunningham trees</td>
<td>Radius 1</td>
<td>&lt; ϵ</td>
<td>×</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 21: Comparison of several Kakeya shapes. Parameters ϵ and α are arbitrarily small positive numbers. BSP stands for Besicovitch-Perron-Schoenberg.

Table 21 compares several Kakeya shapes. Finally, we have the following result.

There is no planar figure of least area, in which a needle of length 1 can be freely turned around or rotated, possibly with translations, and always remaining inside the figure.

**Problems**

1. [Real needle.] Solve the problem assuming that the needle has an arbitrarily small but nonzero thickness δ > 0.

**References**

The problem was originally posed and the three-cornered hypocycloid solution was given by the Japanese mathematician Soichi Kakeya in 1917. The semicircular tube solution is presented in [Ogilvy, 1990]. The highly counterintuitive solution was discovered by the great mathematician Abram Samoilovitch Besicovitch [Besicovitch, 1928], [Falconer, 1986]. Pal’s join was proposed by Julius Pal. Besicovitch’s extremely complicated solution was simplified by Oskar Perron and Isaac Jacob Schoenberg [Besicovitch, 1963]. Burkard Polster explains Besicovitch’s solution in a nice video [Polster, 2015]. A presentation on Jean-Pierre Kahane’s use of Cantor-like sets is given in [Barrowclough, 2008]. Cunningham trees was discovered in [Cunningham Jr, 1971].

Details on polyhedra can be found in [Cromwell, 1999]. Good books on fractals are by [Mandelbrot, 1982], [Schroeder, 2009], and [Briggs, 1992].
Marriage Problem

Problem

A crazy algorithmist is searching for a life partner. The algorithmist wants to inter-
view 100 persons and select the best (i.e., most compatible) of them based on physical,
emotional, intellectual, and spiritual matching. We assume that the persons are given
ranks from 1 to 100 with no ties. The persons appear for the interview in random or-
der. The algorithmist can only deduce the relative ranks of the persons when they
appear for the interview and not their absolute ranks. If a person is selected the algo-
rithmist has to marry the person. If a person is rejected, the person will never come
back again.

What is the strategy to maximize the probability of selecting the best person
among 100 persons?

Solution

The problem is also known as the secretary problem (selecting the best secretary) and
the best choice problem (selecting the ball with largest number written on it). The
puzzle is from the domains of optimal stopping theory and decision theory, where we
need to take a decision and stop at an optimal stopping point.

In life, we search and select the best. We search for the best schools and colleges,
we search for the best jobs, we search for the best friends, we search for the best life-
partners, we search for the best products and deals, we search for the best stocks and
shares, and in general, we search for the best solutions to our problems. However,
often, either we give up too soon or we give up too late. This puzzle elucidates a
beautiful mathematical idea that we can implement in our lives to aid in our journey
of searching for the best.

Basic strategies

[Random select strategy (non-optimal).] We simply select a random person among
$n$ persons. The probability of a randomly selected person being the best person among
$n$ persons is $\frac{1}{n}$.

| Success probability = $\frac{1}{n}$. |

[Reject-half strategy (non-optimal).] Reject the first $\left\lfloor \frac{n}{2} \right\rfloor$ persons but remember
the best person among them. While interviewing the remaining $\left\lceil \frac{n}{2} \right\rceil$ persons, if we
Mathematical and Algorithmic Puzzles

encounter a person who is better than the best person among the first \( \left\lfloor \frac{n}{2} \right\rfloor \) persons, then select her. If we do not find any such person, we select the last person.

Let \( B_1 \) and \( B_2 \) be the 1st and the 2nd best persons among the \( n \) persons. The first \( \left\lfloor \frac{n}{2} \right\rfloor \) persons and the last \( \left\lceil \frac{n}{2} \right\rceil \) persons are denoted as 1st and 2nd halves, respectively. The best persons \( B_1 \) and \( B_2 \) can be anywhere in the 1st or 2nd halves as shown in Table 22.

<table>
<thead>
<tr>
<th>Case</th>
<th>1st half</th>
<th>2nd half</th>
<th>Chance</th>
<th>Correctness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( B_1 )</td>
<td>( B_2 )</td>
<td>( \frac{1}{4} )</td>
<td>( \times )</td>
</tr>
<tr>
<td>2</td>
<td>( B_2 )</td>
<td>( B_1 )</td>
<td>( \frac{1}{4} )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>3</td>
<td>( B_1, B_2 )</td>
<td>-</td>
<td>( \frac{1}{4} )</td>
<td>( \times )</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>( B_1, B_2 )</td>
<td>( \frac{1}{4} )</td>
<td>( \checkmark, \checkmark )</td>
</tr>
</tbody>
</table>

Table 22: Cases for two halves solution.

The strategy always answers correctly for case 2 i.e., when \( B_2 \) is present in the 1st half and \( B_1 \) is in the 2nd half. The strategy always answers incorrectly for cases 1 and 3. For case 4, the strategy answers correctly for some instances and incorrectly for some other instances. We recursively apply the same idea for case 4 i.e., one of the four subcases in case 4 leads to winning. The process continues. Hence,

\[
\text{Success probability} \approx \left( \frac{1}{4^1} \right) + \left( \frac{1}{4^2} \right) + \cdots + \left( \frac{1}{4^\log_4 n} \right) \approx \frac{1}{3}
\]

Though this strategy is better than the random select strategy it is not the best. It is simply unbelievable how through a better strategy the winning probability increased from \( \frac{1}{n} \) (dependent on the number of persons) to \( \frac{1}{3} \) (independent of the number of persons).

Reject-\( k \) strategy

Here is a simple and elegant mathematical idea. Reject the first \( k \in [0, n-1] \) persons and remember the best person among them. Let \( A \) be the best person among the first \( k \) persons. Now, while interviewing the remaining \( (n-k) \) persons if we encounter a person \( B \) better than \( A \) or if \( B \) is the last person, then we say that the \( B \) is the best person.

When \( k = 0 \), we have reject-0 strategy and in this strategy the winning probability is \( \frac{1}{n} \) (the same as that of the random select strategy). When \( k = n-1 \), we have reject-(\( n-1 \)) strategy and in this strategy the winning probability is \( \frac{1}{n} \) (the same as that of the random select strategy). When \( k = \left\lfloor \frac{n}{2} \right\rfloor \), we have our reject-half strategy and in this strategy we have seen that the winning probability is around 33\%. Different values of \( k \) leads to different strategies and different winning probabilities. Our aim is to find that particular value of \( k \) as a function of \( n \) for which the winning probability is maximized.
Let \( P(n, k) \) denote the success probability of reject-\( k \) strategy for \( n \) persons. We have seen \( P(n, 0) = \frac{1}{n} \), \( P(n, n - 1) = \frac{1}{n} \), and \( P \left( n, \left\lfloor \frac{n}{2} \right\rfloor \right) \approx \frac{1}{3} \). By intuition we can reason out that if \( k \) is very small, then it is likely that we select a person before we encounter the best person. On the other hand, if \( k \) is very large, then it is likely that we reject the best person as per the strategy. Hence, \( k \) must be selected with utmost caution. We want to graph the function \( P(n, k) \) and select that \( k \), say \( k_{\text{opt}} \), which maximizes \( P(n, k) \). Then, the optimal strategy will be reject-\( k_{\text{opt}} \) strategy. We give several approaches to identify \( k_{\text{opt}} \).

[Programming approach.] In this approach, for a given \( n \), we list all possible \( n! \) permutations of 1 to \( n \) and then for a particular value of \( k \) we see how many of those \( n! \) possible cases lead to the correct selection of the best person.

Assume that the relative numbers of the persons are in the range \([1, n]\). Larger the person’s number, the better the person. The person with number \( n \) is the best person. We call an arrangement of numbers of the \( n \) persons that leads to the correct selection of the best person as the winning order. Table 23 shows the winning orders for 4 persons. Consider the case when \( n = 4 \). The value of \( k \) can be in the range \([0, 3]\). The total number of permutations possible is \( 4! = 24 \). When \( k = 0 \), the first person selected is considered the best person. To select the best person correctly, the number 4 must occur at the first position. It is easy to see that there are six winning orders: \( \{4123, 4132, 4213, 4231, 4312, 4321\} \). When \( k = 1 \), we reject the first person and select the next person who has a higher number than the that of the first person. In this case, there are 11 winning orders. Similarly, when \( k = 2 \) and \( k = 3 \), there are 10 and 6 winning orders, respectively, as shown in the table.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( k = 0 )</th>
<th>( k = 1 )</th>
<th>( k = 2 )</th>
<th>( k = 3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>{21}</td>
<td>{12}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>{312, 321}</td>
<td>{132, 213, 231}</td>
<td>{123, 213}</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>{4123, 4132, 4213, 4231, 4312, 4321}</td>
<td>{1243, 1324, 1342, 1234, 1324, 2134}</td>
<td>{2143, 2314, 2341}</td>
<td>{2314, 3124, 3214}</td>
</tr>
</tbody>
</table>

Table 23: Winning orders for \( n \in [1, 4] \) and \( k \in [0, n - 1] \).

We can write a computer program to generate all \( n! \) permutations, find the number of winning orders for different values of \( k \), and then plot the success probability \( P(n, k) \). Table 24 gives the success probability for different values of \( n \) and \( k \). The table also shows the optimal \( k \) i.e., \( k_{\text{opt}} \) for different values of \( n \). The problem with this approach is that it becomes extremely difficult to find the values of \( k_{\text{opt}} \) and \( P(n, k_{\text{opt}}) \) for larger values of \( n \).

[Lindley’s probabilistic approach.] In this approach, we find a closed-form formula to compute \( P(n, k) \) as a function of \( n \) and \( k \). We then differentiate \( P(n, k) \) w.r.t. \( k \) and set the differential to zero to compute the value of \( k_{\text{opt}} \).
In the reject-\(k\) strategy, we reject the first \(k\) persons. We select a person \(i\), where \(i \in [k + 1, n]\) as the best person if the \(i\)th person is the first person we meet who is better than the best of the first \(k\) persons. This means that the best person among the first \((i - 1)\) persons is in the first \(k\) persons.

The success probability \(P(n, k)\) can now be computed as \(P(n, k) = P(\text{selected person is the best})\)

\[
= \sum_{i=1}^{n} P(\text{person } i \text{ is selected and person } i \text{ is the best})
\]

\[
= \sum_{i=k+1}^{n} P(\text{person } i \text{ is selected and person } i \text{ is the best})
\]

\(\because\) persons 1, \ldots, \(k\) are never selected

\[
= \sum_{i=k+1}^{n} P(\text{person } i \text{ is the best}) \cdot P(\text{person } i \text{ is selected given person } i \text{ is the best})
\]

\(\because\) Bayes' theorem

\[
= \sum_{i=k+1}^{n} P(\text{person } i \text{ is the best}) \times
\]

\[
P(\text{best person among first } (i - 1) \text{ persons is in the first } k \text{ persons}
\text{ given that person } i \text{ is the best})
\]

\[
= \sum_{i=k+1}^{n} \frac{1}{n} \times \frac{k}{i - 1}
\]

\[
= \frac{k}{n} \sum_{i=k+1}^{n} \frac{1}{i - 1} = \frac{k}{n} \sum_{i=k}^{n-1} \frac{1}{i}
\]

\[
= \frac{k}{n} \left( \frac{1}{k} + \frac{1}{k+1} + \cdots + \frac{1}{n-1} \right) \quad \text{(where, } k \in [1, n-1])
\]

Table 24: Success probability for different values of \(n\) and \(k\).

<table>
<thead>
<tr>
<th>(n)</th>
<th>(k)</th>
<th>(k_{\text{opt}}/n)</th>
<th>(P(n, k_{\text{opt}}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.00</td>
<td>0.50</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.33</td>
<td>0.50</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0.25</td>
<td>0.49</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0.40</td>
<td>0.43</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0.33</td>
<td>0.43</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0.29</td>
<td>0.41</td>
</tr>
</tbody>
</table>

MATHEMATICAL AND ALGORITHMIC PUZZLES
For the reject-\(k\) strategy, the success probability is given by:

\[
\text{Success probability } = \begin{cases} 
\frac{1}{n} & \text{if } k = 0, \\
\left(\frac{k}{n}\right) \sum_{i=k}^{n-1} \left(\frac{1}{i}\right) & \text{if } k \in [1, n-1].
\end{cases}
\]

As a final step in solving the problem we want to find \(k_{\text{opt}}\). We solve this subproblem as follows. The success probability \(P(n, k)\) can be rewritten using a good approximation for large values of \(n\):

\[
P(n, k) \approx \left(\frac{k}{n}\right) \left(\ln(n) - \ln(k) - \sum_{i=1}^{n-1} \frac{1}{i}\right) \approx \ln(n)
\]

As we are interested in maximizing \(P(n, k)\), we need to differentiate it w.r.t. \(k\), equate it to zero, and then find \(k_{\text{opt}}\) value. We must also confirm that the second derivative w.r.t. \(k\) is negative, which is left to the reader. We get:

\[
\frac{d}{dk}(P(n, k)) = \frac{1}{n} \left(\ln(n) - \ln(k) - 1\right) = 0
\]

\[
\implies k_{\text{opt}} = \text{round} \left(\frac{n}{e}\right) \quad (e = 2.718281828...) \]

Substituting \(k_{\text{opt}} = \text{round} \left(\frac{n}{e}\right)\) in the equation of \(P(n, k)\), we get:

\[
P(n, k_{\text{opt}}) \approx \left(\frac{k_{\text{opt}}}{n}\right) \left(\ln(n) - \ln(k_{\text{opt}})\right)
\]

\[
\approx \frac{1}{e} \approx 36.79\%
\]

For the reject-\(k\) \((k \in [0, n-1])\) strategy, for larger values of \(n\), we have \(k_{\text{opt}} = \text{round} \left(\frac{n}{e}\right)\), where \(e = 2.718281828\ldots\)

Success probability \(= P(n, k_{\text{opt}}) \approx \frac{1}{e} \approx 36.79\%\)

![Success probability graph](image)

**Figure 37:** Success probability of the reject-\(k\) strategy for \(n = 100\) and \(k \in [0, n-1]\).
We apply this result to our original problem for \( n = 100 \) persons. The success probability plot for \( n = 100 \) is shown in Figure [37]. For 100 persons, \( k_{\text{opt}} = \text{round}\left(\frac{100}{e}\right) = 37 \). This means that we need to reject the first 37 persons and choose the next person who is better than the best of the first 37 persons. Then our success probability will be approximately 36.79%.

Problems

1. If the reject-\( k_{\text{opt}} \) strategy is the optimal strategy (maximizes success probability) among all possible strategies to solve the problem, prove it mathematically. Else, come up with an optimal strategy to solve the problem.
2. What is the strategy to choose the best person with maximum success probability (i) if we have an unknown but finite number of persons, and (ii) if we have an uncountable number of persons?
3. What is the strategy maximizing the success probability to select (i) \( i \)th best person out of \( n \) persons, and (ii) \( m \) best persons out of \( n \) persons?

References

The problem appeared first in print in Martin Gardner’s Scientific American article [Gardner, 1995] in 1960. The probabilistic solution was given in [Lindley, 1961] and then extended in [Chow et al., 1964]. The odds theorem and the odds algorithm were given in [Bruss, 2000]. Several variations of the problem are discussed in [Freeman, 1983]. A nice history of the problem is given in [Ferguson, 1989]. [Gusein-Zade, 1966] discusses the problem of picking one of the \( k \) best among \( n \). [Vanderbei, 1980] discusses the problem of picking all of the \( k \) best among \( n \). The problem has appeared in Data Genetics blog [Genetics, 2016] and several other websites.
Princess and Princes

Problem

Long long ago there lived a beautiful and intelligent princess who was obsessed with mathematics. When the time came she decided to marry one of the two most intelligent and good-natured princes: A and B. The two princes were called to the court and were asked to stand in front of the princess.

A natural number was written on the crown of each prince. No prince could see the number on his own crown but he could see the number written on the other crown. The princess wrote two distinct natural numbers on a board and announced that one of the board numbers is the sum of the numbers written on their crowns. The princess would ask prince A: “do you know the number written on your crown?” If A’s answer was “no”, the princess would ask prince B the same question. If B’s answer was “no”, the princess would ask prince A the same question. This process would continue in the cyclic order A, B, A, B, . . . until a prince answers “yes”. The prince who would correctly guess the number written on his crown would win the opportunity to marry the princess.

The assumptions in the problem are: the two princes were equally intelligent and truthful, no other form of communication was allowed, and each prince was able to hear the answer of the other prince. What happened eventually? Did the princess get married? If no, how? If yes, to whom?

The counterintuitive result is that the princess got married eventually to one of the two princes! How is this possible?

Solution

A beautiful puzzle! It seems incredible that it is possible to solve the puzzle. The puzzle belongs to the topic of distributed knowledge, which deals with the study of the evolution of knowledge/information across multiple people/players/computers.

Observe that if the question of the princess is changed to “do you know the sum of numbers on all crowns?”, the solution to the puzzle remains the same. This is because knowing the number on one’s crown is equivalent to knowing the sum of numbers on all crowns. To keep our analysis simpler, we assume that the objective is to know the sum of numbers on all crowns.

[Conway-Paterson’s method of decision tree.] Let

\((A, B) = \) the first and the second prince, respectively
\((a, b) = \) natural numbers written on the crown of prince A and B, respectively
\((S_1, S_2) = \) smallest and the largest natural numbers written on the board, respectively
\(d = (S_2 - S_1) = \) the absolute difference between the two numbers written on the board
\(A_i\) = conclusion of prince \(A\) after the \(i\)th consecutive “no” of prince \(B\)

\(B_i\) = conclusion of prince \(B\) after the \(i\)th consecutive “no” of prince \(A\)

We now prove that the princess eventually got married to one of the princes \(A\) and \(B\) using proof by contradiction. To this end, we assume the negation of what needs to be proved i.e., we assume that the princess never got married due to infinite “no”s. We then show a contradiction.

[A’s first “no”.] If \(A\) says “no”, then \(B\) concludes that \(b \in (0, S_1)\). How?

If \(b \in [S_1, S_2)\), then \(A\) can deduce his number using the below analysis

\[
\begin{align*}
b &\geq S_1 & (b \text{ is in the range } [S_1, S_2)) \\
\implies a + b &> S_1 & (\text{adding positive number } a \text{ on the left side}) \\
\implies a + b &= S_2 & (\text{sum greater than } S_1 \text{ must equal } S_2) \\
\implies a &= S_2 - b & (\text{move } b \text{ to the right side})
\end{align*}
\]

and \(A\) would have answered “yes”. As \(A\) says “no”, \(B\) concludes that \(b \in (0, S_1)\).

\(B_1 : b \in (0, S_1)\).

[B’s first “no”.] If \(B\) says “no”, then \(A\) concludes that \(a \in (d, S_1)\). How?

If \(a \in [S_1, S_2)\), then \(B\) can deduce his number using the below analysis

\[
\begin{align*}
a &\geq S_1 & (a \text{ is in the range } [S_1, S_2)) \\
\implies a + b &> S_1 & (\text{adding positive number } b \text{ on the left side}) \\
\implies a + b &= S_2 & (\text{sum greater than } S_1 \text{ must equal } S_2) \\
\implies b &= S_2 - a & (\text{move } a \text{ to the right side})
\end{align*}
\]

and \(B\) would have answered “yes”. As \(B\) says “no”, \(A\) infers that \(a \in (0, S_1)\).

\(A_1 : a \in (0, S_1)\).

\(A\) will not stop reasoning here. He will continue to reason as follows. \(B\) would have already concluded that \(b \in (0, S_1)\). If \(a\) were less than or equal to \(d\), then \(B\) would also have inferred that

\[
\begin{align*}
a &\leq d \\
\implies a + b &\leq d + b & (\text{adding positive number } b \text{ on both sides}) \\
\implies a + b &< (S_2 - S_1) + S_1 & (d = (S_2 - S_1) \text{ and } b < S_1 \text{ from conclusion } B_1) \\
\implies a + b &< S_2 & (\text{simplify}) \\
\implies a + b &= S_1 & (\text{sum smaller than } S_2 \text{ must equal } S_1) \\
\implies b &= S_1 - a & (\text{move } a \text{ to the right side})
\end{align*}
\]

and \(B\) would have answered “yes”. As \(B\) says “no”, \(A\) concludes that \(a \in (d, S_1)\).

\(A_1 : a \in (d, S_1)\).

[A’s second “no”.] If \(A\) says “no”, then \(B\) concludes that \(b \in (0, S_1 - d)\). How?

If \(b \in [S_1 - d, S_1)\), then \(A\) would have inferred his number using the below analysis

\[
\begin{align*}
b &\geq S_1 - d & (b \text{ is in the range } [S_1 - d, S_1)) \\
\implies a + b &\geq a + (S_1 - d) & (\text{adding positive number } a \text{ to both sides}) \\
\implies a + b &> d + (S_1 - d) & (a > d \text{ from conclusion } A_1)
\end{align*}
\]

102
\[ \Rightarrow a + b > S_1 \]  \hspace{1cm} (simplify)
\[ \Rightarrow a + b = S_2 \]  \hspace{1cm} (sum greater than \( S_1 \) must equal \( S_2 \))
\[ \Rightarrow a = S_2 - b \]  \hspace{1cm} (move \( b \) to the right side)

and \( A \) would have answered “yes”. As \( A \) says “no”, \( B \) concludes that \( b \in (0, S_1 - d) \).
\[ B_2 : b \in (0, S_1 - d). \]

[B’s second “no”.] If \( B \) says “no”, then \( A \) concludes that \( a \in (2d, S_1) \). How?
\( A \) knows that \( a \in (0, S_1) \) from conclusion \( A_1 \). \( A \) will continue to reason as follows.
\( B \) would have already concluded that \( b \in (0, S_1 - d) \). If \( a \) were less than or equal to \( 2d \), then \( B \) would also have inferred that
\[ a \leq 2d \]
\[ \Rightarrow a + b \leq d + d + b \]  \hspace{1cm} (adding positive number \( b \) to both sides)
\[ \Rightarrow a + b < d + (S_2 - S_1) + (S_1 - d) \]  \hspace{1cm} (\( d = (S_2 - S_1) \) and \( b < S_1 - d \) from \( B_2 \))
\[ \Rightarrow a + b < S_2 \]  \hspace{1cm} (simplify)
\[ \Rightarrow a + b = S_1 \]  \hspace{1cm} (sum smaller than \( S_2 \) must equal \( S_1 \))
\[ \Rightarrow b = S_1 - a \]  \hspace{1cm} (move \( a \) to the right side)

and \( B \) would have answered “yes”. As \( B \) says “no”, \( A \) concludes that \( a \in (2d, S_1) \).
\[ A_2 : a \in (2d, S_1). \]

[A’s kth “no”.] If \( A \) says “no” for the \( k \)th time, then \( B \) concludes that \( b \in (0, S_1 - (k-1)d) \).
\[ B_k : b \in (0, S_1 - (k-1)d). \]

[B’s kth “no”.] If \( B \) says “no” for the \( k \)th time, then \( A \) concludes that \( a \in (kd, S_1) \).
\[ A_k : a \in (kd, S_1). \]

\( A \) will know that \( a \in (kd, S_1) \). However, as \( d \) is positive, \( kd \) will be greater than or equal to \( S_1 \) for some natural number \( k \). It is impossible to simultaneously satisfy the contradictory conditions \( a \in (kd, S_1) \) and \( kd > S_1 \). This implies that the assumption of princess not getting married due to infinite “no”s is incorrect. So, the princess eventually got married to one of the princes. \textsc{TWOPRINCES} gives the algorithm for solving the problem. The number of questions asked in any instance of two princes puzzle can be computed in constant time as shown in the \textsc{TWOPRINCES-QUESTIONS} algorithm. Figure 38 gives the decision trees of princes \( A \) and \( B \).

\textbf{Conway-Paterson’s method of elimination.} In this section, we give a different way to solve the puzzle. Let’s assume that the two numbers written on the board are \( S_1 = 8 \) and \( S_2 = 11 \) (observe that \( S_1 < S_2 \)). This implies that the numbers written on the crowns of the two princes must be an ordered pair from the following set:
\[ ([1, 7], [1, 10], [2, 6], [2, 9], [3, 5], [3, 8], [4, 4], [4, 7], [5, 3], [5, 6], [6, 2], [6, 5], [7, 1], [7, 4], [8, 3], [9, 2], [10, 1]). \]

An ordered pair \([p, q]\) considers the case when prince \( A \)’s number is \( p \) and prince \( B \)’s number is \( q \). We want to find the set of ordered pairs (i.e., numbers on the princes) in which one of the princes answers correctly after \( i \)th question. We know that the
**TwoPrinces**$(a, b, S_1, S_2)$

**Input:** Numbers $a$ and $b$ on princes $A$ and $B$’s crowns, respectively. Numbers $S_1$ and $S_2$ on board such that $0 < S_1 < S_2$ and one of $S_1$ or $S_2$ is equal to $a + b$. $A$ starts first.

**Output:** Sequence of answers and conclusions of the princes $A$ and $B$.

1. $d \leftarrow (S_2 - S_1)$
2. **while** true do
   3. $k \leftarrow 1$
   4. // computation of prince $A$ ...........................................
      5. **if** $b \geq S_1 - (k - 1)d$ **then**
         6. $A$ computes $a = S_2 - b$
         7. $A$ says “yes”; **return**
      **else**
         8. $A$ says “no”
         9. $B$ concludes $B_k : b \in (0, S_1 - (k - 1)d)$
   10. // computation of prince $B$ ...........................................
    11. **if** $k = 1$ **and** $a \geq S_1$ **then**
        12. $B$ computes $b = S_2 - a$
        13. $B$ says “yes”; **return**
    14. **if** $a \leq kd$ **then**
        15. $B$ computes $b = S_1 - a$
        16. $B$ says “yes”; **return**
    17. **else**
        18. $B$ says “no”
        19. $A$ concludes $A_k : a \in (kd, S_1)$
   20. $k \leftarrow k + 1$

**TwoPrinces-Questions**$(a, b, S_1, S_2)$

**Input:** Crown numbers $a, b$ and board numbers $0 < S_1 < S_2$ such that one of $S_1$ or $S_2$ is equal to $a + b$.

**Output:** Number of questions needed to identify one’s own crown number

1. $d \leftarrow (S_2 - S_1)$
2. **if** $b \geq S_1$ **then** questions $\leftarrow 1$
3. **else if** $a \leq d$ **or** $a \geq S_1$ **then** questions $\leftarrow 2$
4. **else**
   5. $k = \left\lceil \frac{a}{d} \right\rceil$; questions $\leftarrow 2k$
   6. **if** $b \geq S_1 - (k - 1)d$ **then** questions $\leftarrow$ questions $- 1$
7. **return** questions

first question will go to prince $A$, the second question will go to prince $B$, the third question will go to prince $A$, and so on. Figure [39] first column shows all possible pairs
If $A \in S_2$, then $a = S_2 - b$. If $b \in [S_1, S_2)$, then $a = S_2 - b$.

Figure 38: Decision trees of princes $A$ and $B$.

of numbers on the crowns whose sum is either 8 or 11. We want to prove that all pairs that represent the numbers of the princes that sum to 8 or 11 will be correctly identified after a finite number of questions.

Figure 39: Identifying numbers on the crowns of the two princes when the board numbers are 8 and 11.

After the first question, if prince $A$ sees a number greater than or equal to 8, he will easily know that his number is the difference between 11 and the number he sees. In other words, after the first question $Q_1$, the following pairs will be identified: $[1, 10], [2, 9],$ and $[3, 8]$. Consider $[3, 8]$ as an example. In this case, prince $A$ sees 8 on prince $B$’s crown and infers that his number must be 3 because $S_2 - 8 = 11 - 8 = 3$. His number cannot be $S_1 - 8 = 8 - 8 = 0$ because his number has to be a natural number. In general, if the princes have numbers $[1, 10], [2, 9],$ or $[3, 8]$, the prince $A$ can find out his number.
After question 1 as highlighted in the second column of the figure.

After the second question, prince $B$ can answer “yes” if the princes numbers are any of the following pairs: $[1, 7]$, $[2, 6]$, $[3, 5]$, $[8, 3]$, $[9, 2]$, and $[10, 1]$. Consider $[8, 3]$ as an example. In this case, prince $B$ sees 8 on prince $A$’s crown and infers that his number must be 3 because $S_2 - 8 = 11 - 8 = 3$. Prince $B$ can infer his number in $[8, 3]$ after question 2 for the same reason that prince $A$ can infer his number in $[3, 8]$ after question 1. Consider $[2, 6]$ as another example. In this case, prince $B$ sees 2 on prince $A$’s crown and infers that his number must be 6. How? Prince $B$ knows that his number must be $S_1 - 2 = 8 - 2 = 6$ or $S_2 - 2 = 11 - 2 = 9$. $B$ then reasons that if his number was 9, then prince $A$ would have guessed $A$’s number as 2 after question 1 as described in the previous paragraph. So $B$’s number has to be 6 and he answers correctly after question 2 in this case. Similar reasoning can be applied to other pairs.

We continue this process until all one of the princes in each pair identifies his number after a finite number of questions. Figure 40 shows a tree of how different pairs are identified in different rounds.

![Decision Tree](image)

Figure 40: Identifying numbers on the crowns of the two princes when the board numbers are 8 and 11 using a decision tree.

**Generalization**

The puzzle and the elimination method can be generalized to $n$ princes and $k$ board numbers. Let $A_1, A_2, A_3, \ldots, A_n$ be the princes and $S_1, S_2, S_3, \ldots, S_k$. An $n$-tuple represents the possible tuple of numbers on the crowns of the princes $A_1, A_2, A_3, \ldots, A_n$. Let $T$ be the set of all $n$-tuples of the form $[a_1, a_2, a_3, \ldots, a_n]$
such that the tuple sum \((a_1+a_2+a_3+\cdots+a_n)\) is one of the board numbers \(\{S_1,S_2,S_3,\ldots,S_k\}\).

Initially, the set \(T\) has a finite number of \(n\)-tuples. In every round starting from the first round, apply the elimination rule as given below.

**Elimination rule.** Identify those numbers such that each of the numbers appear in exactly one of the \(n\)-tuples and then eliminate or strike-off such \(n\)-tuples that have those numbers.

The method described above is the generalization of the process used in Figure 39. Does this elimination process always end in a finite number of rounds? In other words, can this process reach a state where it is impossible to eliminate tuples with the elimination rule described above? We will prove in subsequent paragraphs that the \(n\)-princes \(k\)-boardnumbers game will always terminate when \(k \leq n\).

We first define a few terms and conditions before giving a formal proof. **Termination** in our problem means that it is always possible to identify and eliminate at least one \(n\)-tuple every round. This implies that all possible \(n\)-tuples will be identified in a finite number of rounds. Instead of proving that this game must terminate, we prove that it is impossible for the game not to terminate. We define a state as a set of \(n\)-tuples. We call a state ambiguous iff

\[
\forall \text{ tuple } t \text{ in the state, } \forall \text{ index } i, \exists \text{ tuple } t' \text{ in the state such that } t \text{ and } t' \text{ differ only in the } i\text{th coordinate.}
\]

That is, a state is ambiguous iff for any tuple \(t\) in the state and any index \(i\) in the set \(\{1,2,\ldots,n\}\) there is a tuple \(t'\) in the state that differs from \(t\) only in the \(i\)th coordinate.

What is the significance of an ambiguous state? If the situation (or state) reaches an ambiguous state, then for any \(n\)-tuple \([a_1,a_2,\ldots,a_n]\) in the state there are at least two possible values for his number. This automatically implies that if the game reaches an ambiguous state, then the elimination process never terminates.

We will now prove the highly counterintuitive result

**Theorem.** The \(n\)-princes \(k\)-boardnumbers game will always terminate when \(k \leq n\).

We will prove the result above by proving that for any \(n\)-princes game that goes on forever without terminating, there must be at least \(n+1\) distinct board numbers, i.e.,

If \(C_n\) is an ambiguous state (i.e., process never ends) with \(n\)-tuples, then \(C_n\) must have at least \(n+1\) distinct sums.

We will prove this statement using mathematical induction. Let \(P(k)\) denote the given statement. **Basis.** \(P(1)\) is true because the ambiguous 1-tuple set \([\{a_1\},\{a'_1\}]\) has two distinct sums, as \(a_1 \neq a'_1\). **Induction.** Suppose \(P(k)\) is true for some \(k \geq 1\). That is, \(C_k\) or the ambiguous set of \(k\)-tuples of the form \([a_1,a_2,\ldots,a_k]\) has at least \(k+1\) distinct
sums. We want to prove that $P(k+1)$ is true. That is, $C_{k+1}$ or the ambiguous set of $(k+1)$-tuples of the form $[a_1, a_2, \ldots, a_k, a_{k+1}]$ has at least $k+2$ distinct sums.

Observe that we cannot construct $C_{k+1}$ without making use of $C_k$. So let’s construct $C_{k+1}$ using $C_k$. Consider the smallest set of $(k+1)$-tuples $[a_1, a_2, \ldots, a_k, a_{k+1}]$ that can be constructed using $C_k$. In this set $a_{k+1}$ is the smallest value possible for the $k$th coordinate; so the number of distinct sums possible will be at least $k+1$ using the inductive hypothesis. However, this set cannot be $C_{k+1}$ (i.e., ambiguous state) because if we fix the first $k$ coordinates $[a_1, a_2, \ldots, a_k]$, then the $(k+1)$th coordinate is not changing. To make the set ambiguous or $C_{k+1}$ we must have at least one more option for the $k$th coordinate, say $a'_{k+1}$ such that $a'_{k+1} > a_k$. Let $[\ell_1, \ell_2, \ldots, \ell_k]$ be the $k$-tuple with the largest sum in a particular $C_k$. Then the $(k+1)$-tuple $[\ell_1, \ell_2, \ldots, \ell_k, a'_{k+1}]$ represents a sum which is clearly larger than any of the sums considered before; hence represents at least the $(k+2)$th sum. So, $P(k+1)$ is true.

Related puzzle: Red eyed monks

There is an island of monks where everyone has either black eyes or red eyes. Monks who have red eyes are cursed, and are supposed to commit suicide at midnight. There is no form of communication between the monks as they have a vow of silence. Thus, no one knows their own eye color. They can only see the eye colors of other monks. One day, a tourist visits the island, and not knowing that he’s not supposed to talk about eyes, he states the observation: “At least one of you has red eyes.”

How will the monks get to know who have red eyes?

[Solution.] The solution to the problem can be analyzed in two different cases: (i) The tourist is lying, and (ii) The tourist is telling the truth.

[Case 1. Tourist lies.] There is no way to find out whether the tourist is lying or not. If the tourist lies, it means that there are no monks with red eyes. The monks believe whatever the tourist says. Hence, the monks think that at least one of them has red eyes. Each monk starts believing that he himself has red eyes as he sees no other monk having red eyes. Therefore, they all commit suicide on the first night.

If the tourist lies, then all the monks commit suicide on the first night.

[Case 2. Tourist tells the truth.] The tourist is telling the truth, which means in fact there is at least one monk who has red eyes.

If only one monk has red eyes, he sees that no other monks have red eyes and believes that he is the only monk with red eyes. Then, he commits suicide on the first night. If there are two monks who have red eyes, each of them see that the other monk has red eyes. Hence, each of the two monks think that the other monk is going to commit suicide on the first night. But, no one dies after the first night. As no one died on the first night, it is a confirmation that both the monks have red eyes and hence they commit suicide on the second night. A similar reasoning follows for three, four, five, and more number of monks having red eyes. If there are 100 monks with red eyes, then they all commit suicide on the 100th night.
If the tourist tells the truth and there are $n$ monks with red eyes, then they all commit suicide on the $n$th night.

References

The generalized puzzle (formulated in a different setting) and its solution are first presented in [Conway et al., 2020] in 1977. The decision tree method follows the description from [Savchev and Andreescu, 2003] and the elimination method follows the description from [Gale, 1998]. [Lasry et al., 1989] show that even when the number of board numbers is strictly greater than the number of princes, the solution can converge in some cases. For a formalized theory on reasoning about knowledge, see [Fagin et al., 2003].
Losing + Losing = Winning

Problem

If you play game A, you are guaranteed to lose.
If you play game B, you are guaranteed to lose.
If you play games A and B back-and-forth, you are guaranteed to win. How?

Solution

What? Two losing games combines to a winning game. How is this possible? Can two wrongs really make a right? In fact, two losing games can combine to a winning game and two winning games can combine to a losing game. This counterintuitive phenomenon is called Parrondo’s paradox and it belongs to the domain of game theory.

Fundamentals

[Terminologies.] A game consists of a set of rules using which players make independent decisions at every round/timestep for an infinite number of timesteps. A winning game (respectively losing game) is one in which the expected money/points/score a player makes after playing the game for an infinite number of timesteps is positive (respectively negative) infinity. In other words, a winning (respectively losing) game is one in which in the long term we expect to finish up with infinitely more (respectively less) money we started with; which is possible when the average amount made per timestep is positive (respectively negative). Let \( m_t \) denote the money the player has at timestep \( t \). Let \( m_0 \) be 0. Then a game if called winning if \( \lim_{t \to \infty} m_t = \infty \), losing if \( \lim_{t \to \infty} m_t = -\infty \), and neutral if \( \lim_{t \to \infty} m_t = 0 \).

We will consider both probabilistic and deterministic games. Deterministic games can be considered as a subset of probabilistic games such that the probability of occurrence is 1. We will construct probabilistic games (or games of chance) using tosses of an unfair coin. A coin is unfair/biased (respectively fair/unbiased) if the success probability (i.e., the probability of heads turning up) is not equal (respectively equal) to the failure probability (i.e., the probability of tails turning up). In all our deterministic and probabilistic games we assume that the player starts with 0 money.

[Simple neutral games.] A simple example of a deterministic neutral game is as follows. In every timestep the player neither wins nor loses a dollar. This is damn boring game but it is a neutral game. Another example is as follows. In every odd timestep, the player gains \( k > 0 \) dollars. In every even timestep, the player loses \( k \) dollars. This game can be considered as a neutral (or a partially winning game). A simple example of a probabilistic neutral game would be to toss a fair coin repeatedly.
We use Markov chain to model the state transitions as shown in Figure 41. Markov chain is a mathematical system that consists of states; and the probability of transition to any particular state depends on the current state and the time elapsed. This process is also called a 1-D random walk on an integer grid, also known as drunkard’s walk. The money the player makes at timestep \( t \) in this game, denoted by \( m_t \), can be computed as follows:

\[
m_t = \begin{cases} 
0 & \text{if } t = 0, \\
m_{t-1} + 1 & \text{if } t \text{th coin toss is heads}, \\
m_{t-1} - 1 & \text{if } t \text{th coin toss is tails}.
\end{cases}
\]

At time 0, the player has no money. At time 1, he tosses a coin. If it turns up heads his money will be +1, otherwise it will be −1. At time 2, he tosses the coin again. If it turns up heads his money will be incremented by 1 dollar, otherwise it will be decremented by 1 dollar. This process repeats indefinitely. At infinite time, the expected value of player’s money will be 0 dollars. This can be seen from Table 25, which shows that the probability that \( m_\infty \) takes a particular value, forms the normal or Gaussian function with mean 0 and variance 1. That is

\[
\mathbb{P}(m_\infty = k) = \int_{x=-0.5}^{x=0.5} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \, dx.
\]

Setting \( k = 0 \), the probability that the player will have nothing at the end of the game is 38.29%. Hence, the game is a neutral game.

<table>
<thead>
<tr>
<th>Timestep</th>
<th>( \cdots )</th>
<th>−5</th>
<th>−4</th>
<th>−3</th>
<th>−2</th>
<th>−1</th>
<th>0</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>+4</th>
<th>+5</th>
<th>( \cdots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t = 0 )</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t = 1 )</td>
<td>0.5</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t = 2 )</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t = 3 )</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t = 4 )</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t = 5 )</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t = \infty )</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 25: Probabilities that \( m_t \) represents a particular value in a simple neutral game. For example, the probability that \( m_5 = -3 \) is \( \frac{5}{32} \).

**[Simple winning and losing games.]** A simple deterministic game is as follows. In every step, a player makes \( k \) dollars. Clearly, if \( k > 0 \), this game is a winning game and if \( k < 0 \), this game is a losing game. Another simple game is as follows. In every odd timestep, the player gains \( m > 0 \) dollars. In every even timestep, the player loses \( n > 0 \) dollars. If \( m > n \), this game is a winning game and the player gains \( (m - n)/2 \) dollars.
dollars per round. If $m < n$, then this game is a losing game as the player loses $\frac{(n-m)}{2}$ dollars per round.

A simple probabilistic game is as follows. Suppose we toss an unfair coin which has a success probability $p$. Let $\mathbb{P}$(event) denote the success probability of an event.

\[
\begin{align*}
\mathbb{P}(&\text{win}) = p \quad \text{(money will be incremented by a dollar)} \\
\mathbb{P}(&\text{lose}) = 1 - p \quad \text{(money will be decremented by a dollar)}
\end{align*}
\]

The money the player will have at timestep $t$ will be

\[
m_t = \text{total number of coin tosses} \times [\text{prob. of winning} \times \text{amount of gain} + \text{prob. of losing} \times \text{amount of loss}]
\]

\[
= t \times [\mathbb{P}(&\text{win}) \times \$1 + \mathbb{P}(&\text{lose}) \times (-\$1)]
\]

\[
= t \times (p \times \$1 + (1 - p) \times (-\$1)) = t(2p - 1)
\]

If $p = \frac{1}{2}$, we have a neutral game.

If $p \in \left(\frac{1}{2}, 1\right]$, say, $p = \frac{1}{2} + \epsilon$, for a feasible positive real $\epsilon$, we have a winning game. The player gains on an average $2p - 1$ dollars per round in this winning game.

If $p \in \left[0, \frac{1}{2}\right)$, say, $p = \frac{1}{2} - \epsilon$, for a feasible positive real $\epsilon$, we have a losing game. The player loses on an average $2p - 1$ dollars per round in this losing game.

![Figure 42: Markov chain for a simple winning/losing game when $p \neq \frac{1}{2}$](image)

[Combining simple games.] In this section, we combine two simple probabilistic games $A$ and $B$ to construct a combined game $[AB]$. In the combined game $[AB]$, the player plays a round of $A$ and then a round of $B$, and repeats this pattern. Suppose $A$ and $B$ are two simple games that require coins 1 and 2, respectively, and their success probabilities are $p$ and $q$, respectively. We then construct a combined game $AB$ as follows. We first toss coin 1 and then coin 2 and repeat this process. Figure 43 shows the binary decision tree for the first two coin tosses of the combined game $[AB]$.

![Figure 43: Binary decision tree of combining two games](image)

We compute the probabilities of the first two coin tosses of game $[AB]$ as:

\[
\begin{align*}
\mathbb{P}(\text{win}, \text{win}) &= pq \quad \text{(money will be incremented by 2 dollars)} \\
\mathbb{P}(\text{win}, \text{lose}) &= p(1 - q) \quad \text{(money will be unchanged)}
\end{align*}
\]
\[ P(\text{lose, win}) = (1 - p)q \] (money will be unchanged)
\[ P(\text{lose, lose}) = (1 - p)(1 - q) \] (money will be decremented by 2 dollars)

The money the player will have at timestep \( 2t \) playing game \([AB]\) will be
\[
m_{2t} = \text{total number of coin tosses} \times [P(\text{win, win}) \times 2 + P(\text{lose, lose}) \times (-2)]
= 2t \times (pq \times 2 + (1 - p)(1 - q) \times (-2))
= 2t(2(p + q - 1))
\]

\( m_{2t} = 0 \): If \( p + q = 1 \), the combined game \([AB]\) is a neutral game.
\( m_{2t} > 0 \): If \( p + q > 1 \), the combined game \([AB]\) is a winning game.
\( m_{2t} < 0 \): If \( p + q < 1 \), the combined game \([AB]\) is a losing game.

<table>
<thead>
<tr>
<th>Game ([AB])</th>
<th>Game A</th>
<th>Game B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neutral ( p + q = 1 )</td>
<td>Neutral ( p = 0.5 )</td>
<td>Neutral ( q = 0.5 )</td>
</tr>
<tr>
<td>Winning ( p + q &gt; 1 )</td>
<td>Winning ( p \in (0.5, 1) )</td>
<td>Losing ( q \in [0, 0.5) )</td>
</tr>
<tr>
<td>Losing ( p + q &lt; 1 )</td>
<td>Losing ( p \in [0, 0.5) )</td>
<td>Winning ( q \in (0.5, 1) )</td>
</tr>
<tr>
<td>Winning ( p \in (0.5, 1) )</td>
<td>Winning ( p \in (0.5, 1) )</td>
<td>Winning ( q \in (0.5, 1) )</td>
</tr>
<tr>
<td>Winning ( p \in (0.5, 1) )</td>
<td>Losing ( p \in [0, 0.5) )</td>
<td>Winning ( q \in (0.5, 1) )</td>
</tr>
<tr>
<td>Losing ( p \in [0, 0.5) )</td>
<td>Losing ( p \in [0, 0.5) )</td>
<td>Losing ( q \in [0, 0.5) )</td>
</tr>
<tr>
<td>Losing ( p \in [0, 0.5) )</td>
<td>Winning ( p \in (0.5, 1) )</td>
<td>Losing ( q \in [0, 0.5) )</td>
</tr>
<tr>
<td>Losing ( p \in [0, 0.5) )</td>
<td>Losing ( p \in [0, 0.5) )</td>
<td>Winning ( q \in (0.5, 1) )</td>
</tr>
</tbody>
</table>

Figure 44: Possibilities for games \( A \) and \( B \) given the status of the combined game \([AB]\).

Figure 44 shows the states of games \( A \) and \( B \) given the state of the game \([AB]\).

Observe that in Figure 44 we see multiple combinations of states of the two games \( A \) and \( B \) for a given state of game \([AB]\). However, we don’t see the combinations (losing + losing) = winning and (winning + winning) = losing. We cannot imagine that there can exist such unifications that can totally reverse the effects of the specific parts. In fact, such unifications are possible and we will explore them in the next section.

**Parrondo’s example**

In this section, let’s see how two losing games can combine to a winning game. Consider the two games losing \( A \) and \( B \).

**[Losing game \( A \)]** Game \( A \) represents repeated tossing of coin 1 such that the success probability for each coin toss is \( p_1 \), where, \( p_1 = \frac{1}{2} - \epsilon \) for some positive \( \epsilon \). Clearly, game \( A \) is a losing game.

**[Losing game \( B \)]** Game \( B \) is more complicated than game \( A \) as it uses two coins. At each toss, the player uses one of the two coins. The player’s choice of the coin at every timestep depends on the money the player has at that moment. If the player’s money is a multiple of 3, then the player tosses coin 2 with success probability \( p_2 \), where, \( p_2 = \left( \frac{1}{10} \right) - \epsilon \) for some positive \( \epsilon \). Otherwise, the player tosses coin 3 with success probability \( p_3 \), where, \( p_3 = \left( \frac{3}{4} \right) - \epsilon \) for some positive \( \epsilon \).
Let $m_t$ be the player’s money after timestep $t$. Then the decision tree of the player to compute $m_{t+1}$ after timestep $t + 1$ is shown in Figure 45.

![Figure 45: Binary decision tree of game B.](image)

Using simulation, it can be shown that game B is a losing game.

Combining losing games A and B can be winning.] Using simulation, it can be shown that a variety of games that alternate between A and B deterministically or probabilistically can be winning. A probabilistic combination of games A and B, i.e., randomly switching between games A and B is a winning game. Similarly, there are many deterministic winning combinations of games A and B such as [AABB], [ABB], and [ABBAB]. For example, in the game [AABB], a player plays two rounds of game A and then two rounds of game B and then repeats this pattern indefinitely.

It is also possible to get the (winning + winning = losing) result by using different values for $p_1$, $p_2$, and $p_3$.

More examples

Simpler examples to illustrate losing + losing = winning result exist. Such examples can be deterministic or probabilistic, money-dependent or money-independent, timestep-dependent or timestep-independent, and history-dependent or history-independent (here history means two or more previous states). Table 26 gives a small list of examples that show counterintuitive behavior in games.

Parrondo’s paradox is not really a logical paradox; it is just a paradox in the sense that it is counterintuitive even though it is true. Let’s understand this behavior through a very simple example, say Example 1 from Table 26. Figure 46 clearly shows that playing game A during the odd timestep and playing game B during the even timestep increases the player’s fortune.

Parrondo’s example explained

In this section, we will analyze the Parrondo’s example that we saw earlier. Clearly, game A is losing. We will see how to prove mathematically and through simulation that game B is losing and the combined game $C$ by randomly switching between games $A$ and $B$ with probability 0.5 is winning.

Mathematical analysis.] A simple approach to describe the transitions between states of game B is via the 3-state Markov chain. State diagram of game B is given
### Example 1

<table>
<thead>
<tr>
<th>Game</th>
<th>Conditions</th>
<th>Status of the game</th>
</tr>
</thead>
</table>
| Game A| If timestep is even, lose $3  
If timestep is odd, win $1                                      | Losing avg. $1 per round            |
| Game B| If timestep is even, win $1  
If timestep is odd, lose $3                                    | Losing avg. $1 per round            |
| Game [AB]| One round of A and one round of B                                    | Winning avg. $1 per round           |

### Example 2

<table>
<thead>
<tr>
<th>Game</th>
<th>Conditions</th>
<th>Status of the game</th>
</tr>
</thead>
</table>
| Game A| If money is even, win $1  
If money is odd, lose $3                                          | Losing avg. $1 per round            |
| Game B| If money is even, lose $3  
If money is odd, win $1                                                | Losing avg. $1 per round            |
| Game [AB]| One round of A and one round of B                                    | Winning avg. $1 per round           |

### Example 3

<table>
<thead>
<tr>
<th>Game</th>
<th>Conditions</th>
<th>Status of the game</th>
</tr>
</thead>
</table>
| Game A| If money is even, toss a biased coin  
Win $1 with probability $9\over10$  
Lose $1 with probability $1\over10$ | Losing avg. $1\over3$ per round     |
| Game B| If money is even, toss a biased coin  
Win nothing with probability $9\over10$  
Lose $1 with probability $1\over10$ | Losing avg. $1\over3$ per round     |
| Game [AB]| One round of A and one round of B                                    | Winning avg. $1\over3$ per round     |

### Example 4

<table>
<thead>
<tr>
<th>Game</th>
<th>Conditions</th>
<th>Status of the game</th>
</tr>
</thead>
</table>
| Game A| If money is even, win $3  
If money is odd, lose $5                                                  | Losing avg. $1 per round            |
| Game B| Lose $3 every round                                                        | Losing avg. $1 per round            |
| Game [AB]| One round of A and one round of B                                    | Winning avg. $1 per round           |

Table 26: Games to illustrate losing + losing = winning result.

In Figure 47, these states are represented as 0, 1, and 2 to represent the remainders when the player’s money at any moment is divided by 3. That is, the state $i \in \{0, 1, 2\}$ represents the set of all numbers $m_i$ such that $m_i \equiv i \pmod{3}$. So, state 0 represents $\{0, \pm 3, \pm 6, \ldots\}$, state 1 represents $\{\pm 1, \pm 4, \pm 7, \ldots\}$, and state 2 represents $\{\pm 2, \pm 5, \pm 8, \ldots\}$. The transition matrix $M$ of the 3-state Markov chain is given in Figure 49. The transition matrix of the 3-state Markov chain is a $3 \times 3$ matrix such that the $(i, j)$ entry denotes the probability of transitioning from state $j$ to state $i$ in the next coin toss. We know from the game setting that $(i, i)$ entries must be 0 because a state changes with every coin toss. Observe that the sum of values in each row and column is 0.

Similarly, the state diagram and transition matrix of game $C$ are shown in Figures 48 and 50, respectively. The transition matrix of game $C$ is constructed by taking half the values of game $B$’s transition matrix and half the values of game $A$’s transition matrix. Note that game $A$’s transition matrix can be constructed from game $B$’s transition matrix by replacing $p_2$ and $p_3$ with $p_1$.

Let’s consider a general matrix $M$ that generalizes the transition matrices of games $B$ and $C$ as shown in Figure 51. Suppose $x^{(t)}$ be the probability vector that denotes the
Figure 46: Gains and cumulative gains of games A, B, and [AB] of Example 1 from Table 26. The blue and red colors represent the positive and negative gains, respectively.

Figure 47: Game B’s state diagram.

<table>
<thead>
<tr>
<th>Trans. prob.</th>
<th>From state</th>
<th>To state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 1−p₃ p₃</td>
<td>1 2</td>
</tr>
<tr>
<td>1</td>
<td>p₂ 0 1−p₃</td>
<td>2 1−p₂ p₃</td>
</tr>
</tbody>
</table>

Figure 48: Game C’s state diagram.

<table>
<thead>
<tr>
<th>Trans. prob.</th>
<th>From state</th>
<th>To state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1−(p₁+p₂)²</td>
<td>1−(p₁+p₃)²</td>
</tr>
<tr>
<td>1</td>
<td>(p₁+p₂)²</td>
<td>(p₁+p₃)²</td>
</tr>
</tbody>
</table>

Figure 49: Game B’s trans. matrix.

<table>
<thead>
<tr>
<th>Trans. prob.</th>
<th>From state</th>
<th>To state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1−q₂ q₃</td>
<td>1 2</td>
</tr>
<tr>
<td>1</td>
<td>q₁ 0 1−q₃</td>
<td>2 1−q₁ q₂</td>
</tr>
</tbody>
</table>

Figure 50: Game C’s transition matrix.

<table>
<thead>
<tr>
<th>Trans. prob.</th>
<th>From state</th>
<th>To state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 1−q₂ q₃</td>
<td>1 2</td>
</tr>
<tr>
<td>1</td>
<td>q₁ 0 1−q₃</td>
<td>2 1−q₁ q₂</td>
</tr>
</tbody>
</table>

Figure 51: General matrix that generalizes the transition matrices of B and C games.
probabilities of being in each of the states 0, 1, and 2 after the $r$th timestep.

Let $x^{(r)} = \begin{bmatrix} x_0^{(r)} \\ x_1^{(r)} \\ x_2^{(r)} \end{bmatrix}$. Then, $x^{(r)} = M \cdot x^{(r-1)}$ implies $x^{(r)} = M^r \cdot x^{(0)}$ where $x^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$.

The first coordinate of vector $x^{(0)}$ is 1 because the player starts with money 0 i.e., in state 0. We now compute the limiting probability vector $x^{(\infty)}$ at timestep $\infty$, which gives us the probabilities of being in states 0, 1, and 2 when the game $B$ is played for an infinite time. From the mathematical analysis of Markov chains, we have

$$x^{(\infty)} = M \cdot x^{(\infty)} \implies (M - I) \cdot x^{(\infty)} = 0$$

where $I$ is the identity matrix. For simplicity we assume $x = x^{(\infty)}$ to get

$$x = M \cdot x \implies (M - I) \cdot x = 0$$

As $M$ and $I$ are known, $M - I$ is also known. $(M - I) \cdot x = 0$ represents a system of linear equations (3 equations, 3 unknowns). There are several systematic methods to solve such systems of linear equations including Gaussian elimination, lower-upper (LU) decomposition, Cramer’s rule, and singular value decomposition (SVD).

There is yet another direction to solve for $x$ using the eigenvector approach. The equation $x = M \cdot x$ implies that the vector $x$ must be the eigenvector of the transition matrix $M$ corresponding to eigenvalue 1. Solving Equation 17 using the eigenvector approach we get the normalized eigenvector $x$:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} (q_2 q_3 - q_2 + 1)/s \\ (q_1 q_3 - q_3 + 1)/s \\ (q_1 q_2 - q_1 + 1)/s \end{bmatrix}, s = (q_1 q_2 q_3 + q_1 q_2 + q_2 q_3 + q_1 q_3 - q_1 - q_2 - q_3 + 3).$$

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Game A</th>
<th>Game B</th>
<th>Game C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$p_1$</td>
<td>$p_2$</td>
<td>$(p_1 + p_2)/2$</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$0.495$</td>
<td>$0.095$</td>
<td>$0.295$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$p_1$</td>
<td>$p_3$</td>
<td>$(p_1 + p_3)/2$</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$0.495$</td>
<td>$0.745$</td>
<td>$0.62$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$p_1$</td>
<td>$p_3$</td>
<td>$(p_1 + p_3)/2$</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$0.495$</td>
<td>$0.745$</td>
<td>$0.62$</td>
</tr>
<tr>
<td>$x_0$</td>
<td>0.333</td>
<td>0.383612</td>
<td>0.345</td>
</tr>
<tr>
<td>$x_1$</td>
<td>0.333</td>
<td>0.154281</td>
<td>0.399</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0.333</td>
<td>0.462107</td>
<td>0.255</td>
</tr>
</tbody>
</table>

Table 27: Eigenvectors of games for Parrondo’s example with $\epsilon = 0.005$, $p_1 = \frac{1}{2} - \epsilon$, $p_2 = \frac{1}{10} - \epsilon$, and $p_3 = \frac{3}{4} - \epsilon$.

We will analyze games $A$, $B$, and $C$ using values from Table 27. 
[Game A.] Let $P$ and $E$ denote the success probability and expected return of an event. We now compute the expected return for game $A$ and show that game $A$ is a losing game.

$$E(\text{game } A) = P(\text{win}) \times \text{amount gained} + P(\text{lose}) \times \text{amount lost}$$

$$= p_1 \times ($1) + (1 - p_1) \times ($(-1)) = 0.495 \times ($1) + 0.505 \times ($(-1)) = $(-0.01)$ per round

[Game B.] We compute the expected return for game $B$ and show that game $B$ is a
losing game.

\[ E(\text{state } 0) = P(\text{win in state } 0) \times \text{amount gained} + P(\text{lose in state } 0) \times \text{amount lost} \]
\[ = q_1 \times (\$1) + (1 - q_1) \times (\$(-1)) = 0.095 \times (\$1) + 0.905 \times (\$(-1)) \]
\[ = \$(-0.81) \text{ per round} \]
\[ E(\text{state } 1 \text{ or } 2) = P(\text{win in state } 0 \text{ or } 1) \times \text{gain} + P(\text{lose in state } 0 \text{ or } 1) \times \text{loss} \]
\[ = q_2 \times (\$1) + (1 - q_2) \times (\$(-1)) = 0.745 \times (\$1) + 0.255 \times (\$(-1)) \]
\[ = \$(0.49) \text{ per round} \]
\[ E(\text{game } B) = P(\text{state } 0) \times E(\text{state } 0) + P(\text{state } 1 \text{ or } 2) \times E(\text{state } 1 \text{ or } 2) \]
\[ = x_0 \times \$(-0.81) + (x_1 + x_2) \times \$(0.49) \quad (\text{see Table 27 for } x \text{ values}) \]
\[ = 0.383612 \times \$(-0.81) + 0.616388 \times \$(0.49) \]
\[ = \$(-0.0086956) \text{ per round} \]

[Game C.] We compute the expected return for game C and show that game C is a winning game. Note that the game C we are considering here is a uniformly random combination of games A and B.

\[ E(\text{state } 0) = P(\text{win in state } 0) \times \text{amount gained} + P(\text{lose in state } 0) \times \text{amount lost} \]
\[ = q_1 \times (\$1) + (1 - q_1) \times (\$(-1)) = 0.295 \times (\$1) + 0.705 \times (\$(-1)) \]
\[ = \$(-0.41) \text{ per round} \]
\[ E(\text{state } 1 \text{ or } 2) = P(\text{win in state } 0 \text{ or } 1) \times \text{gain} + P(\text{lose in state } 0 \text{ or } 1) \times \text{loss} \]
\[ = q_2 \times (\$1) + (1 - q_2) \times (\$(-1)) = 0.62 \times (\$1) + 0.38 \times (\$(-1)) \]
\[ = \$(0.24) \text{ per round} \]
\[ E(\text{game } C) = P(\text{state } 0) \times E(\text{state } 0) + P(\text{state } 1 \text{ or } 2) \times E(\text{state } 1 \text{ or } 2) \]
\[ = x_0 \times \$(-0.41) + (x_1 + x_2) \times \$(0.24) \quad (\text{see Table 27 for } x \text{ values}) \]
\[ = 0.345 \times \$(-0.41) + 0.655 \times \$(0.24) \]
\[ = \$(0.01575) \text{ per round} \]

[Programming-based analysis.] As we see in the previous section, mathematical analysis is complicated and requires knowledge about probability and statistics. Also, when the combined game is different, the mathematical analysis might also be very different. Probably the simplest approach to analyze whether games are losing or winning are by simulation or programming. Figure 52 shows the expected gains of games \([A], [B]\), a deterministic \([AABB]\) combination, and a probabilistic uniformly random \([(A, B)|0.5]\) combination. The expected gains are computed over 100 thousand runs. The plot clearly shows that games \([A]\) and \([B]\) are losing whereas the combined games \([AABB]\) and \([(A, B)|0.5]\) are winning, visually illustrating Parrondo's paradox.

Among all possible deterministic combination of A and B games with the size of the patterns at most five, the following are the losing games: \([A], [B], [AA], [AB], [BA], [BB], [AAA], [BBB], [ABA], [BAA]\). The remaining deterministic combination games of pattern size at most five are all winning.
Figure 52: The expected gain from games \(A, B, [AABB]\) and the uniformly random combination of \(A\) and \(B\) i.e., \([A, B]|0.5\). The red and blue colors represent the losing and winning games, respectively. The expected gain is computed over 100,000 runs or trials.

**Take-home lessons**

Many of us keep losing money even after buying winning stocks. Why? It is due to the (winning + winning = losing) phenomenon. A simple reason for this phenomenon is because the stocks are winning over long term and we frequently buy-sell over short terms with losses. To elaborate, we typically buy stocks at high prices due to fear of missing out (FOMO) psychology and sell stocks at low prices due to fear, uncertainty, and doubt (FUD) psychology. This type of trading based on extreme emotions leads to losses and the losses accumulate to wipe out the entire capital invested.

What is the solution? Simple. Let’s use the (losing + losing = winning) phenomenon. We will buy stocks at low prices and sell them at high prices. This requires giving the decision-making powers and control over to our logical aspect of our mind and not the emotional aspect of our mind. In that case, we might make money even with the losing stocks.

**References**

Parrondo’s paradox is named after its discoverer Juan Parrondo in 1997 and published in [Harmer and Abbott, 1999]. Mathematical analysis of Parrondo’s paradox is taken from Thomas Pietraho’s presentation [Pietraho, 2014] and Alberto Grunbaum’s talk [Grunbaum, 2017]. Examples 2 and 3 in Table 26 are from [Ekhad and Zeilberger, 2000] and [Berresford and Rockett, 2003], respectively. There are several other phenomena that have counterintuitive behavior like Parrondo’s example: losing stock + losing stock = winning stock portfolio [Abbott, 2010], losing lockdown strategy + losing unlock strategy = winning outcome for the community during pandemics and epidemics [Cheong et al., 2020], and chaos + chaos = order [Almeida et al., 2005].
Drunkard’s Dilemma

Problem

Long ago there were two villages separated by a thick forest. People in the first village grew chicken and people in the second village grew rice. There was scarcity of rice in the first village and scarcity of chicken in the second village. During a trade, a unit of rice was equal to the amount of rice whose weight equals the weight of the chicken being traded. As rice was scarce and in high demand in the first village, people traded 2 chickens for a unit of rice. As chicken was scarce and in high demand in the second village, people traded 2 units of rice for a chicken.

A drunkard lived an amazing life in these two villages. He stole a chicken from the first village and sold it to an alcohol shop owner in the second village. He got a bottle of beer and a change of a unit of rice. He came to the first village and sold the unit of rice to an alcohol shop owner in the first village. He got a bottle of beer and a change of a chicken. The drunkard repeated this process indefinitely until he died one day peacefully by drinking.

The drunkard was in a dilemma throughout his life understanding the paradoxical nature of the trade. (a) He thought that any trade must have profits and losses. If one person was profited, another person incurred a loss. Hence, one party gains and another party loses. (b) He also thought that a trade is based on a mutual benefit theory. That is, both the parties involved in the trade are benefited. Hence, both parties gain and no party loses.

Could you please help solve the dilemma of the drunkard and analyze which perspective is correct or more correct? Did the drunkard gain or lose? Did the alcohol shop owners gain or lose? Who is benefited and who incurred losses?

Solution

The confusing problem arises in the domains of economics, finance, and business.

Fundamentals

Before we can understand the perplexing situation, we need to know the fundamental concepts of trade, money, demand, supply, profits, losses, and equilibrium points.

[Trade.] A trade is a mutual exchange of goods and services. Trade has its origins in the physical interaction of organisms in evolution. Consider a few different types of interaction between two species:

- [Mutualism.] A relationship where both the organisms are benefited. E.g.: The digestive bacteria present in the human intestine help in the digestion of food that cannot be digested by humans.
• [Parasitism.] A relationship where one organism is benefited and the other organism is harmed. E.g.: Parasites such as fleas and ticks live on dogs and cattle living off the blood of the host animal.

• [Commensalism.] A relationship where one organism is benefited and the other organism is neither benefited nor harmed. E.g.: The cattle egret, a bird, follows the cattle and eats insects that are stirred up when the cattle moves through the grass.

• [Synnecrosis.] A relationship where both the organisms are harmed. E.g.: When some bees sting humans, it inflicts pain to the humans and the bees lose their stings and die.

Mutualism dates back to at least 100 million years. It is impossible to imagine a world where different organisms existed without each other. A few more examples of this type are: (i) Flowers provide food (i.e., nectar) to bees. In return, bees provide a service of pollination (pollination is the process of transfer of pollen to female reproductive organs of a plant for fertilization), and (ii) Animals use oxygen and produce carbon dioxide. Plants use carbon dioxide and produce oxygen.

A mutual beneficial partnership helps a civilization to thrive and makes trade and business possible. The most common example of mutualism in trade is barter system. In a barter system, people exchange their goods and services for other goods and services. For example, one person might exchange a unit of wheat with another person’s 1.5 units of rice. The exchange rate is set by mutual agreement of the people involved in the barter. Long ago, as there was no organization that standardized the value of the goods and services, the exchange rate used to vary widely from person to person, place to place, and time to time. Hence, the barter system was extremely confusing and complicated.

**Figure 53:** Difference between barter system and the monetary system.

![Diagram](image)

[Money.] Money is a sexy idea invented to exponentially simplify trade as shown in Figure 53. Money is a tool used as a medium of exchange of goods and services between trading entities. Several advantages of the monetary system over barter system are given in Table 54. Money is a medium of payment accepted by governments, banks, and millions of stores. Any form of money gets its value when people accept it and people must accept a certain form of money when governments make it mandatory. There are variegated opinions about money. There is a common saying “Love of money is the root of all evil.” In contrast, George Bernard Shaw says “Lack
of money is the root of all evil”, which seems to be more true in modern times.

Money in any form (physical or virtual) that is practically used for exchange is called currency. The different forms of currencies that have been used from ancient to modern times are: cattle (and other livestock), Cowry shells (Cowry shells is a type of sea shells made from mollusks that live in the shallow waters of Pacific and Indian oceans), metallic coins, paper notes, cheques, credit cards, debit cards, and bitcoins. Please refer to Table 55 for details on these currencies.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Barter system</th>
<th>Monetary system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trade</td>
<td>Trade is difficult. Suppose a person owns cows and wants to trade for wheat with another person. But, the person with wheat might not want cattle. He/she might want something that the person with cows doesn’t have. The entire process of trading cows for wheat gets extremely complicated. Trade is easy. Suppose a person owns cows and wants to trade for wheat with another person. He/she can sell it to anyone who want cows and in return can get money. Then the person can buy wheat easily with his/her money from any person who sells wheat.</td>
<td></td>
</tr>
<tr>
<td>Carry</td>
<td>It is difficult to carry livestock, metals, grains, and other things. The goods are typically heavy, consume space, and difficult to maintain. Money is easy to carry and maintain. The modern forms of money fit in small pockets and can even buy expensive goods.</td>
<td></td>
</tr>
<tr>
<td>Division</td>
<td>Division is difficult. Suppose a person owns a cow and wants $\frac{1}{7}$th the cow's worth of wheat, $\frac{1}{12}$th the cow's worth of rice, and $\frac{1}{30}$th the cow's worth of chilli. As the cow cannot be cut into pieces for the sake of trade, the process of mutual exchange becomes horrendously complicated. Division is easy. Suppose the person sells the owned cow for $x$ units of money. Then the person can buy wheat, rice, and chilli for $\frac{x}{7}$, $\frac{x}{12}$, and $\frac{x}{30}$ units of money, respectively. The remaining amount of $\left(x - \left(\frac{x}{7} + \frac{x}{12} + \frac{x}{30}\right)\right)$ can be saved for future use.</td>
<td></td>
</tr>
<tr>
<td>Expiry</td>
<td>Goods do not last long. Livestock die, food spoils, grains rot, metals corrode, and things get damaged. Money can last for generations. Money can be saved at homes or banks or can be invested (with risks) in real estate, gold, stocks, and mutual funds.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 54: Differences between barter system and the monetary system.

<table>
<thead>
<tr>
<th>Currency</th>
<th>Origin time</th>
<th>Origin place</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cattle</td>
<td>9000 BC</td>
<td>–</td>
<td>Other livestock used: sheep, goats, and oxen.</td>
</tr>
<tr>
<td>Cowry shells</td>
<td>1200 BC</td>
<td>China</td>
<td>Used majorly in coastal regions.</td>
</tr>
<tr>
<td>Metallic coins</td>
<td>600 BC</td>
<td>Turkey</td>
<td>Made from gold, silver, copper, and bronze.</td>
</tr>
<tr>
<td>Paper notes</td>
<td>806 AD</td>
<td>China</td>
<td>Easy to counterfeit.</td>
</tr>
<tr>
<td>Cheque</td>
<td>1717 AD</td>
<td>UK</td>
<td>Introduced by The Bank of England.</td>
</tr>
<tr>
<td>Credit card</td>
<td>1966 AD</td>
<td>UK</td>
<td>Introduced by Barclaycard.</td>
</tr>
<tr>
<td>Debit card</td>
<td>1987 AD</td>
<td>UK</td>
<td>Introduced by Barclays.</td>
</tr>
<tr>
<td>Bitcoins</td>
<td>2009 AD</td>
<td>–</td>
<td>First decentralized virtual currency.</td>
</tr>
</tbody>
</table>

Figure 55: Different types of currencies.

[Demand, supply, profits, and losses.] The most important assumption in economics is that each person’s goal when dealing with resources is to maximize his/her utility, i.e., the person’s overall happiness or satisfaction. Applying this concept to
trade, every person tries to maximize his/her benefits in a trade.

We define a few terms in economics that might be useful in solving our original puzzle. The demand for a good is how much of that good consumers would buy at various prices. Similarly, the supply for a good is how much of that good producers would produce at various prices. Demand for a good can be influenced by several factors such as population, availability and price of related goods, and whether the good is a necessity or a luxury. Similarly, supply for a good can be influenced by several factors such as production costs, number of suppliers, availability and price of related goods, and conditions of production. The graph of price versus quantity demanded by customers is called demand curve. Similarly, the graph of price versus quantity supplied by a producer is called supply curve. Typically, the quantity and price are plotted horizontally and vertically, respectively.

The law of demand states that when the price of a good increases, the quantity demanded decreases and vice versa. Similarly, law of supply states that when the price of a good increases, the quantity supplied increases and vice versa. When the quantity demanded is greater than the quantity supplied, there is a shortage. On the other hand, when the quantity supplied is greater than the quantity demanded, there is a surplus. The point at which the quantity demanded is equal to the quantity supplied is called the equilibrium point. In a free market (a free market is a market where the prices of goods and services are determined solely by the consumers and producers and not by governments or other organizations), the prices reach the equilibrium point where each person who wants to sell a good at its current price can find another person who wants to buy the good. A market at the equilibrium point is beneficial to everyone and is good for the society.

Let’s now understand how exactly profits and losses (loss is a negative profit) are computed using the information described above. Suppose there is a company that manufactures pens. Let the quantity demanded per month, quantity supplied per month, and their prices per unit be as shown in Table 56. Then, the demand and the supply curves for the company-produced pens are depicted in Figure 57. Observe that the point O in the figure is the equilibrium point.

Suppose the company manufactures 20,000 pens. The company’s cost of producing each pen is 3 units of currency (point B in Figure 57) and a consumer is willing to pay 7 units (point A in Figure 57). Therefore, there is a profit of 4 units per pen and the total profits the company makes is $20,000 \times 4 = 80,000$ units per month. In general, profits can be computed and maximized in a variety of ways depending on variegated factors. For the sake of simplicity, we have considered here the most common way of computing profits.

Profit = Total revenues – Total costs

Having equipped with some fundamentals of trade and economics, we can now try to solve our original drunkard’s dilemma puzzle.
### Solution

The chicken village has lots of chickens and the rice village has a lot of rice. Chicken will be produced in large quantities in the chicken village everyday and rice will be produced in large quantities in the rice village everyday. There is a lot of demand for rice in the chicken village and a lot of demand for chicken in the rice village. From the **law of demand**, when there is more demand, the prices of the goods or products rise. Hence, people in the chicken village will be willing to offer more than 1 chicken for a unit of rice or some other good that is in high demand. Similarly, the people in the rice village will be willing to offer more than 1 unit of rice for a single chicken or any other good that is in high demand.

Suppose \( r_r, c_r, r_c, c_c \) be the prices of 1 unit of rice in the rice village, 1 chicken in the rice village, 1 unit of rice in the chicken village, and 1 chicken in the chicken village, respectively. For simplicity, we assume that all chickens are identical. Then, from the given problem, the initial conditions are:

\[
c_r = 2r_r \quad \text{(in the rice village),}
\]
\[
r_c = 2c_c \quad \text{(in the chicken village).}
\]

The drunkard can be considered as a company that supplies the good for the desired customers i.e., rice for the chicken village and chicken for the rice village. Though the drunkard does not produce chickens or rice, the drunkard’s action is equivalent to a company producing the high demand chickens in the rice village and vice versa. The drunkard is able to do this via **transportation** by transporting chickens and rice from a region of high production and low demand to a region of low production and high demand.

Let \( b_r \) and \( b_c \) be the prices of 1 bottle of beer in the rice and chicken villages, respectively. Then, from the given problem,

\[
c_r = r_r + b_r \quad \text{(in the rice village),}
\]
\[
r_c = c_c + b_c \quad \text{(in the chicken village).}
\]
From the initial village conditions and beer conditions, we have

\[ b_r = r_r \quad (\text{in the rice village}), \]
\[ b_c = c_c \quad (\text{in the chicken village}). \]

All the formulas described above are true initially. However, the formulas keep changing continuously based on the values (or prices) of rice, chicken, and beer which in turn get affected because of the demand and supply of rice, chicken, and beer in the two villages.

As per the \textit{law of supply}, when the price of a good increases, the quantity supplied increases and vice versa. When the people of the chicken village are ready to trade many chickens for a single unit of rice, it means the price of rice is higher than that of chicken in the chicken village. Similarly, when the people of the rice village are ready to trade many units of rice for a single chicken, it means the price of chicken is higher than that of rice in the rice village. According to the law of supply, two things can happen: \( a \) people of the chicken village might grow rice in addition to chicken and the people of the rice village might grow chicken in addition to rice, and/or \( b \) more people might follow the drunkard’s idea of transporting chickens and rice between the two villages.

\[ \text{[Reaching the equilibrium point.]} \] Due to laws of demand and supply as described above, the percentage of rice in the chicken village increases from 0\% to 50\% and the percentage of chicken in the rice village increases from 0\% to 50\%. That is

\[ c_r = r_r \quad (\text{in the rice village}), \]
\[ r_c = c_c \quad (\text{in the chicken village}). \]

This is the \textit{equilibrium point} where people who grow chicken in the rice village and people who grow rice in the chicken village are neither profited or under a loss. However, people who transport chickens and rice between the two villages are under a loss. At this equilibrium point, the drunkard who brings a chicken to an alcohol shop in the rice village gets only a beer. He does not get back a unit of rice as change. Similarly, the drunkard who brings a single unit of rice to an alcohol shop in the chicken village gets only a beer. He does not get back a chicken as change. This means that at equilibrium point, transportation has no advantages or benefits. Several drunkards would be wasting their time in transporting products that are not in demand and hence would not gain any profits. In fact, they would be under a loss because their service of transportation would not add any value, it would simply consume their time and energy without giving any benefits. Hence, it is quite possible that many drunkards would quit transporting no-demand products between the two villages at the equilibrium point.

Suppose the percentage of chicken in the rice village and rice in the chicken village reduces due to some unknown reasons. Then, the demand for the corresponding products increases automatically. As per the analysis described above, more people will be involved in producing the desired products and the society again moves towards the equilibrium and nobody is profited and nobody is under a loss. Therefore, it is possible that the market or society keeps oscillating between various points on the demand and supply curves trying to reach the equilibrium point i.e., the point of
intersection of the demand and the supply curves.

**[Profit/loss vs. mutual benefit.]** Let’s now discuss the differences between the two perspectives: profit/loss and mutual benefit theory. When a producer (e.g.: organization/company/person) produces something for the people, the producer might incur either profits or losses. Typically, the profits and losses are defined for a producer and not for a consumer. We can say that the producer incurred profits/losses but it would not be meaningful to say that the consumer incurred profits/losses. Hence, in the most generic sense, it would be incorrect to say that in a trade between a producer and a consumer, the producer incurred profits (or losses) and the consumer incurred losses (or profits). In this puzzle, initially, the drunkard had huge profits because he/she supplied the high-demand products to the villagers.

In the future, when more drunkards try to transport the high-demand products to a village, the demand for these products automatically reduces and hence the drunkards might not be benefited. In fact, drunkards incur losses at the equilibrium point if they travel between the two villages. The values of the chickens and the rice units will be the same in both the villages. Hence, the transportation of more goods will not add any value and will only incur losses.

The puzzle is about the barter system. Typically, in a barter, the two parties of the trade can be considered as producers and are benefited in different ways. However, the benefits might be of varying degrees. In this puzzle, the drunkard is benefited and the villagers are also benefited.

Both the profit/loss and mutual benefit perspectives can be used to describe the drunkard’s scenario. However, mutual benefit theory would be more correct in the sense that it beautifully captures the essence of the scenario as the scenario involves the barter system. Both the drunkard and the alcohol shop owners (or villagers) are benefited in different ways.

**Problems**

1. Suppose, in United States, 1 Dollar = $x$ Euros and in Europe, 1 Euro = $x$ Dollars, for some $x > 0$. What happens then?

**References**

The origins of trade is explained in [Sehgal, 2015]. The types of currencies is described in [Thornton, 2016]. Concepts of demand, supply, profits, losses, equilibrium point, and competition are concisely explained in [Frakt and Piper, 2014]. Excellent books on trade, business, entrepreneurship, and economics are [Bauman and Klein, 2010] (and [Bauman and Klein, 2011]) and [Schiff and Lockman, 1985]. Bitcoin was introduced by Satoshi Nakamoto [Nakamoto, 2008].
Uncountability

Problem

Prove that there are an infinite types of infinity.

Solution

What exactly is infinity? Infinity is the measure of some quantity that continuously increases without bounds or limits. For example, the length of the real number line is infinite and the number of points on a finite-length line segment is infinite. Are there really infinite types of infinity? Similarly, are there more than one type of zeros or ones or any specific real number, too? How do we benefit from knowing the mathematical fact that there are an infinite types of infinity? Anyways, let’s now delve deep in understanding different types of infinity.

Infinity of natural numbers ($\mathbb{N}_0$)

The notion of finite and infinite can be made mathematically precise by using it on sizes of sets. A set is a collection of related items. If $A$ is a set, then the number of elements in $A$ is denoted as $|A|$. Two sets $A$ and $B$ have the same size, denoted by $|A| = |B|$, iff there is a one-to-one correspondence between the two sets. We say that a function $f : A \rightarrow B$ is one-to-one correspondence if every element of $A$ can be uniquely mapped to an element of $B$ and every element of $B$ can be uniquely mapped to an element of $A$.

Consider a simple problem. You are given a bag full of apples and another bag full of bananas, how do you check which fruit is in greater number without using more than 1 bit of information? Note that 1 bit can uniquely identify $2^1 = 2$ states and hence you are not allowed to count more than 1. Assume that you are present in a room alone with no access to counting machines.

The simple problem above can be solved using one-to-one correspondence. In the first step, pick up 1 apple and 1 banana from the respective bags and discard them. In the second step, pick up 1 apple and 1 banana from the respective bags and discard them. Continue this process till one of the bags is empty. If both bags are empty, then they have an equal number of the two fruits. On the other hand, if one bag is empty and another bag is nonempty, then the fruit in the nonempty bag is greater in number than the other.

A set is finite iff it is an empty set or if it can be put into a one-to-one correspondence with natural numbers $\{1, 2, 3, \ldots, n\}$ for some natural number $n$. A set is countably infinite iff it can be put into a one-to-one correspondence with the set of natural numbers $\mathbb{N}$, i.e., $\{1, 2, 3, \ldots\}$. A set is countable iff it is finite or countably infinite. This process is called counting as humans possibly used their fingers to count items long
time ago. Let $\aleph_0$ (called aleph null) denote the first type of infinity humans learned, which is the infinity represented by the set of natural numbers. That is, $|\mathbb{N}| = \text{infinity of natural numbers} = \aleph_0$.

An infinite set $A$ is countable, i.e., $|A| = \aleph_0$, iff there is a one-to-one correspondence between the set of natural numbers $\mathbb{N}$ and $A$ as shown in Table 28.

<table>
<thead>
<tr>
<th>$\mathbb{N}$</th>
<th>$A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>← −−−−→ first element of $A$</td>
</tr>
<tr>
<td>2</td>
<td>← −−−−→ second element of $A$</td>
</tr>
<tr>
<td>3</td>
<td>← −−−−→ third element of $A$</td>
</tr>
<tr>
<td>·</td>
<td>···</td>
</tr>
<tr>
<td>$n$</td>
<td>← −−−−→ $n$th element of $A$</td>
</tr>
</tbody>
</table>

Table 28: One-to-one correspondence between $\mathbb{N}$ and set $A$.

Table 29: Top: Incorrect mapping between $\mathbb{N}$ and $\mathbb{N}^{\text{even}}$. Bottom: One-to-one correspondence between $\mathbb{N}$ and $\mathbb{N}^{\text{even}}$.

<table>
<thead>
<tr>
<th>$\mathbb{N}$</th>
<th>1 2 3 4 5 6 7 ··· $n$ ···</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{N}^{\text{even}}$</td>
<td>2 4 6 ···</td>
</tr>
</tbody>
</table>

A set $A$ is called a subset of set $B$ if each element of $A$ is an element of $B$. A subset $A$ of set $B$ is called a proper subset of set $B$ if $A$ is a subset of $B$ and $B$ contains at least one element that is not in $A$.

[Even natural numbers.] For simplicity, let’s consider the set of positive even numbers denoted as $\mathbb{N}^{\text{even}}$, which is a proper subset of $\mathbb{N}$. Let’s compare the sizes of $\mathbb{N}$ and $\mathbb{N}^{\text{even}}$.

There is no one-to-one correspondence between $\mathbb{N}$ and $\mathbb{N}^{\text{even}}$ as depicted in the top part of Figure 29. Right? Wrong! How? This is because a one-to-one mapping between the two sets is shown in the bottom part of the figure. Our aim must be to check if there is any kind of mapping between two infinite sets that leads to one-to-one correspondence. We should not conclude that there is no one-to-one correspondence between two infinite sets simply because we failed to find one. It is important to note that showing that there is no one-to-one correspondence between two infinite sets is relatively a harder problem compared with showing that there is a one-to-one correspondence. This example shows that the number of natural numbers is same as the number of even natural numbers even though even natural numbers is a proper subset of the set of natural numbers. Hence, $|\mathbb{N}^{\text{even}}| = \aleph_0$.

[Integers.] Let’s consider the set of integers $\mathbb{Z}$, which is a superset of $\mathbb{N}$. Let’s compare the sizes of $\mathbb{N}$ and $\mathbb{Z}$. Clearly, there is a one-to-one correspondence between $\mathbb{N}$ and $\mathbb{Z}$ as shown in Table 30.

[Positive rationals.] A positive rational is a number of the form $\frac{p}{q}$, where both $p$ and $q$ are natural numbers. Let’s consider the set of positive rationals $\mathbb{Q}^+$. Let’s compare the sizes of $\mathbb{N}$ and $\mathbb{Q}^+$.

The positive rational numbers can be neatly written on a 2-D grid as shown in Figure 58. The entry at the $i$th row and $j$th column is the positive rational $\frac{i}{j}$.

We cannot count these numbers row-by-row because we will never finish counting the first row to reach the first element of the second row. Similarly, we cannot count these numbers column-by-column because we will never finish counting the first column to reach the first element of the second column. How should we proceed then?
The idea is to count the elements diagonal-by-diagonal as shown in Figure 58 and Table 31.

Notice that while counting we did not consider rationals $\frac{2}{2}, \frac{3}{3}, \frac{4}{4}$ etc. Why? Because a rational number has infinite number of representations, e.g. $\frac{1}{2} = \frac{2}{4} = \frac{3}{6} = \frac{4}{8} = \cdots$ and $\frac{1}{2} = \frac{2}{4} = \frac{3}{6} = \frac{4}{8} = \cdots$. As we like to count each positive rational exactly once, we eliminate all representations that are not in their simplest form i.e. representations where the numerator and denominator have a common factor greater than 1.

Figure 58 and Table 31 clearly show that there is a one-to-one correspondence between $\mathbb{N}$ and $\mathbb{Q}^+$ and hence $|\mathbb{Q}^+| = \aleph_0$.

*Countably infinite sets.* We can show using the approach discussed above that the number of even integers is $\aleph_0$, the number of square numbers is $\aleph_0$, the number of prime numbers is $\aleph_0$, the number of multiples of 3 is $\aleph_0$, and the number of rationals is $\aleph_0$. In general,

A proper subset of a countably infinite set is countable.

In summary, we have

$$|\mathbb{N}| = |\mathbb{Z}| = |\mathbb{Z}_{\text{even}}| = |\mathbb{Z}_{\text{squares}}| = |\mathbb{Z}_{\text{primes}}| = |\mathbb{Z}_{\text{multiples of 3}}| = |\mathbb{Q}| = \aleph_0$$
Infinity of real numbers \((\aleph_1)\)

[Cantor's proof using diagonalization.] We prove using contradiction that the real numbers in the half-open interval \((0, 1]\) is greater than the number of natural numbers. Suppose the real numbers in the range \((0..1]\) are countable as shown in the left part of Table 32.

We now construct a number that does not appear in the list. Let’s construct a new number \(d = 0.d_1d_2d_3\ldots d_n\ldots\) as follows:

\[
d_n = \begin{cases} 
1 & \text{if } a_{nn} \neq 1, \\
2 & \text{if } a_{nn} = 1. 
\end{cases}
\]

If we apply the rule above to the right part of Table 33 we will get the new number as \(d = 0.12112\ldots\) and this number is not present in the list. The new number \(d\) is different in the 1st decimal position from the 1st number on the list. The new number \(d\) is different in the 2nd decimal position from the 2nd number on the list. In general, for each natural number \(n\), the constructed real number \(d\) differs in the \(n\)th decimal position from the \(n\)th number on the list. So, the constructed number is different than every number on the list. This implies that the newly constructed number \(d \in (0, 1]\) is not on the list. Contradiction!

[Cantor’s proof using nested intervals.] We prove using contradiction that the number of real numbers in the open interval \((0, 1)\) is greater than the number of natural numbers.
Let's assume that the real numbers in the range \((0, 1)\) are countable. Let all distinct real numbers in \((0, 1)\) be listed as \(r_1, r_2, r_3, \ldots\) We will prove that there is a real number in \((0, 1)\) that is not present in the given sequence. To find a number in \((0, 1)\) that is not in the given sequence, we construct two sequences \(a_1, a_2, a_3, \ldots\) and \(b_1, b_2, b_3, \ldots\) of real numbers as follows.

Find the first two real numbers in the sequence \(r_1, r_2, r_3, \ldots\) that are contained in the open interval \((0, 1)\). Let \(a_1\) and \(b_1\) be the smaller and larger of these two numbers, respectively. Then define \(I_1 = [a_1, b_1]\) and note that \(I_1 \subset [0, 1]\). Similarly, find the first two real numbers in the sequence \(r_1, r_2, r_3, \ldots\) that are contained in the open interval \((a_1, b_1)\). Let \(a_2\) and \(b_2\) be the smaller and larger of these two numbers, respectively. Then define \(I_2 = [a_2, b_2]\) and note that \(I_2 \subset I_1\). Continue this process and define \(I_3, I_4, I_5, \ldots\) In general,

\[
I_i = [a_i, b_i] \text{ for all } i
\]

Two cases arise. The number of intervals can be finite or infinite.

Suppose the number of intervals is finite, i.e., say the last interval is \(I_\ell\), where \(\ell\) is finite. This means that every real number in the open interval \((a_\ell, b_\ell)\) has not been listed in the given sequence of real numbers. Contradiction!

Suppose the number of intervals is infinite. We see that

\[
\begin{align*}
  a_1 < a_2 < a_3 < \cdots < 1 & \quad \text{(increasing sequence)} \\
  b_1 > b_2 > b_3 > \cdots > 0 & \quad \text{(decreasing sequence)} \\
  I_1 \supset I_2 \supset I_3 \supset \cdots & \quad \text{(superset relation)}
\end{align*}
\]

Observe that \(a_1, a_2, a_3, \ldots\) sequence is strictly increasing and bounded above by any \(b_k\). Similarly, \(b_1, b_2, b_3, \ldots\) sequence is strictly decreasing and bounded below by any \(a_k\). As bounded monotonic sequences always have limits, let's define those limits as

\[
a_\infty = \lim_{i \to \infty} a_i \text{ and } b_\infty = \lim_{i \to \infty} b_i. \text{ Then, } I_\infty = [a_\infty, b_\infty].
\]

We know that \(a_1 < b_1, a_2 < b_2, \ldots\). In general,

\[
[a_i < b_i \text{ for all } i] \implies a_\infty \leq b_\infty
\]

We assume without loss of generality that that the given sequence is \((r_1 = a_1), (r_2 = b_1), (r_3 = a_2), (r_4 = b_2), \ldots\). It is easy to see that \((a_1, b_1)\) excludes \([r_1, r_2]\), \((a_2, b_2)\) excludes \([r_1, r_2, r_3, r_4]\), and so on. In general,

\[
(a_i, b_i) \text{ excludes } [r_1, r_2, \ldots, r_{2^i-1}, r_{2^i}] \text{ for all } i.
\]

This implies that \(r_1 \notin (a_1, b_1), r_2 \notin (a_2, b_2), \text{ and so on. In general,}

\[
r_i \notin (a_i, b_i) \text{ for all } i.
\]

We now show that a real number \(s \in I_\infty\) is not present in the given sequence \(r_1, r_2, r_3, \ldots\). We see that \(I_\infty \subset (a_i, b_i)\) for all \(i\). As \(s \in I_\infty\) and \(I_\infty \subset (a_i, b_i)\), we have \(s \in (a_i, b_i)\) for all \(i\). We also know that \(r_i \notin (a_i, b_i)\) for all \(i\). So, \(s \neq r_i, s \neq r_2, s \neq r_3, \ldots\). Hence, \(s\) is a real number in \(I_\infty\) that is not present in the sequence \(r_1, r_2, r_3, \ldots\) Contradiction!

[ Wenner's proof using Cauchy sequences for rational numbers. ] This proof is based on proof by contradiction using Cauchy sequences for rational numbers.

A sequence \(\{a_n\}\) has a limit equal to \(\lim_{n \to \infty} a_n = L\), if for all \(\epsilon > 0\), there exists

131
natural number \( N \) such that for all indices \( n > N, |a_n - L| < \epsilon \). The sequence \( \{a_n\} \) is called \textit{convergent} and is said to converge to \( L \) if \( L \) is finite. The sequence \( \{a_n\} \) is called \textit{divergent} if there is no limit or if the limit is infinite.

A sequence \( \{a_n\} \) of real numbers is called a \textit{Cauchy sequence} if for every \( \epsilon > 0 \), there is a natural number \( N \) such that for all natural numbers \( m, n > N \) we have \( |a_m - a_n| < \epsilon \). This means that given any positive distance \( \epsilon \), all except a finite number of numbers from the sequence are less than \( \epsilon \) from each other. It can be shown that all Cauchy sequences are convergent and all convergent sequences are Cauchy sequences. If the terms of a Cauchy sequence are made up of rational numbers only, then it is called a \textit{Cauchy sequence of rational numbers}.

Some examples of Cauchy sequences on rational numbers with their limits are:

- \( \lim_{n \to \infty} \frac{n^2}{n+1} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{5}{4} + \frac{6}{5} + \frac{7}{6} + \cdots \)
- \( \lim_{n \to \infty} \frac{n^2}{n+1} = \frac{2}{1^2} + \frac{3}{2^2} + \cdots \)
- \( \lim_{n \to \infty} \frac{n^2}{n+1} = \frac{4}{1^2} + \frac{5}{2^2} + \frac{6}{3^2} + \cdots \)
- \( \lim_{n \to \infty} \frac{n^2}{n+1} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \cdots \)

- \( \lim_{n \to \infty} \frac{n^2}{n+1} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \frac{1}{6^2} + \cdots \)

Cauchy sequences of rational numbers can be defined for exponential function, logarithmic function, trigonometric functions, inverse trigonometric functions, hyperbolic functions, and inverse hyperbolic functions, all using Maclaurin series.

A beautiful observation on Cauchy sequences on rational numbers is the following: The two Cauchy sequences on rational numbers \( \{a_n\} \) and \( \{b_n\} \) are equivalent if and only if \( \lim_{n \to \infty} a_n = \lim_{n \to \infty} b_n \). In other words, \( \{a_n\} \) and \( \{b_n\} \) are equivalent if and only if \( a_n - b_n \) converges to 0. Hence, \textit{real numbers are the equivalence classes of the Cauchy sequences on rational numbers}.

Let \( C \) be the collection of all Cauchy sequences on rational numbers; \( \{a_n\} \) and \( \{b_n\} \) be Cauchy sequences on rational numbers; and \( \langle a_n \rangle \) denote the equivalence class of \( C \). Then, \( \mathbb{R} \) is the set of all equivalence class of Cauchy sequences on rational numbers, i.e.,

\[ \mathbb{R} = \{ \langle a_n \rangle \mid \{a_n\} \in C \}. \]

We assume that \( \mathbb{R} \) is countable/listable as a set of all equivalence classes of Cauchy sequences on natural numbers, i.e., \( \mathbb{R} = \{ \langle a^{(1)}_n \rangle, \langle a^{(2)}_n \rangle, \langle a^{(3)}_n \rangle, \langle a^{(4)}_n \rangle, \langle a^{(5)}_n \rangle, \ldots \} \). We will define an equivalence class \( \langle b_n \rangle \) such that \( \langle b_n \rangle \) is not equivalent to any of \( \langle a^{(1)}_n \rangle \) or \( \langle a^{(2)}_n \rangle \) or \( \langle a^{(3)}_n \rangle \) etc. This leads to a contradiction, which automatically implies that \( \mathbb{R} \) is uncountable.

Let's define \( \{N_1, N_2, N_3, N_4, N_5, \ldots \} \) inductively as follows:

\begin{align*}
\text{[Base case]} & \quad \text{Select } N_1 \text{ such that for all } m, n \geq N_1, |\langle a^{(1)}_n \rangle - \langle a^{(1)}_m \rangle | < \frac{1}{2^0}. \\
\text{[Recursive case]} & \quad \text{Select } N_k \text{ for } k > 1 \text{ such that } \\
& \quad N_k > N_{k-1} \text{ and for all } m, n \geq N_k, |\langle a^{(k)}_n \rangle - \langle a^{(k)}_m \rangle | < \frac{1}{2^{k+2}}
\end{align*}

(19)

132
Observe that \(\{N_k\}\) is an increasing sequence of natural numbers and hence for all \(k \in \mathbb{N}, N_k \geq k\).

Let's define \([b_1, b_2, b_3, b_4, b_5, \ldots]\) inductively as follows:

**Base case.** Select a rational number \(b_1\) such that \(|b_1 - a^{(1)}_{N_1}| \geq \frac{1}{2^3}\).

**Recursive case.** Select a rational number \(b_k\) for \(k > 1\) such that

\[
|b_{k-1} - b_k| < \frac{1}{2^{3k}} \quad \text{and} \quad |b_k - a^{(k)}_{N_k}| \geq \frac{1}{2^{3k+1}} \quad (20)
\]

Is such a choice of \(\{b_k\}\) possible? Yes! For example, define \(b_k\) in the following way:

\[
b_k = \begin{cases} 
  b_{k-1} + \frac{1}{2^{3k}} & \text{if } b_{k-1} \geq a^{(k)}_{N_k}, \\
  b_{k-1} - \frac{1}{2^{3k}} & \text{if } b_{k-1} < a^{(k)}_{N_k}.
\end{cases}
\]

We now show that the defined sequence \(\{b_n\}\) is a Cauchy sequence on rational numbers, i.e., \(\{b_n\} \in C\). This implies that \(\langle b_n \rangle\) belongs to \(\mathbb{R}\).

Fix \(\epsilon > 0\). Choose a natural number \(N\) dependent on \(\epsilon\) such that \(\frac{1}{2^N} < \epsilon\). Then, for all \(n \geq m \geq N\) we have

\[
|b_m - b_n| = \left| \sum_{i=m+1}^{n} (b_{i-1} - b_i) \right| \leq \sum_{i=m+1}^{n} |b_{i-1} - b_i| \quad (|x + y| \leq |x| + |y|)
\]

\[
< \sum_{i=m+1}^{n} \frac{1}{2^{3i}} \quad \text{(recursive case of sequence \(\{b_n\}\))}
\]

\[
< \frac{1}{2^{3N}} < \epsilon
\]

\(\implies\) \(\{b_n\}\) is a Cauchy sequence on rational numbers

\(\implies\) \(\{b_n\} \in C \implies \{b_n\} \in \mathbb{R}\)

We now show that the constructed sequence \(\{b_n\}\) is not equivalent to \(\{a^{(1)}_n\}\), is not equivalent to \(\{a^{(2)}_n\}\), is not equivalent to \(\{a^{(3)}_n\}\), so on up to infinity. This automatically implies that \(\{b_n\}\) is not present in the list of \(\mathbb{R}\).

For any \(k \in \mathbb{N}\) and \(n \geq N_k \geq k\), we have

\[
|b_n - a^{(k)}_n| = |(b_n - b_k) + (b_k - a^{(k)}_{N_k}) + (a^{(k)}_{N_k} - a^{(k)}_n)|
\]

\[
\geq |b_k - a^{(k)}_{N_k}| - |b_n - b_k| - |a^{(k)}_{N_k} - a^{(k)}_n|
\]

\[
\implies |y| = |x + y + z - x - z| \leq |x + y + z| + |x| + |z| \quad (\because |x + y + z| \geq |y| - |x - z|)
\]

\[
> \frac{1}{2^{3k+1}} - \sum_{i=k+1}^{n} |b_{i-1} - b_i| - \frac{1}{2^{3k+2}} \quad \text{(using Equations 20 and 19)}
\]

\[
\geq \frac{1}{2^{3k+2}} - \sum_{i=k+1}^{n} |b_{i-1} - b_i| \quad (|x + y| \leq |x| + |y|)
\]

\[
> \frac{1}{2^{3k+2}} - \sum_{i=k+1}^{n} \frac{1}{2^{3i}} \quad \text{(using Equation 20)}
\]

\[
> \frac{1}{2^{3k+2}} - \sum_{i=k+1}^{\infty} \frac{1}{2^{3i}} = \frac{1}{2^{3k+2}} - \frac{1}{7} \cdot \frac{1}{2^{3k}} = \frac{3}{28} \cdot \frac{1}{2^{3k}}
\]

133
the tails of \( \{b_n\} \) and \( \{a_n^{(k)}\} \) sequences differ by a positive distance

\( \Rightarrow \) \( \{b_n\} \) is not equivalent to \( \{a_n^{(k)}\} \) for any \( k \in \mathbb{N} \) as listed in \( \mathbb{R} \)

\( \Rightarrow \) \( \langle b_n \rangle \not\in \mathbb{R} \)

How is it possible that \( \langle b_n \rangle \) is simultaneously present and absent in \( \mathbb{R} \)? This is a clear contradiction! So, our original assumption that \( \mathbb{R} \) is countable/listable is incorrect. This implies \( \mathbb{R} \) is uncountable.

[Notes.] Why is function \( 2^{3k} \) (and its sister functions \( 2^{3k+1} \) and \( 2^{3k+2} \)) chosen in Equations 19 and 20? Can we choose say \( 2^k \) or \( 2^{k+1} \) and their sister functions? No! Because, with these functions, we will not be able to show that \( \langle b_n \rangle \in \mathbb{R} \). In fact, we could choose any function of the form \( 2^{pk} \), where \( p \geq 3 \). Selecting \( p = 3 \) would be the simplest of such functions.

Observe that unlike the diagonalization proof we cannot compare a term of the \( \{b_n\} \) sequence with the corresponding term of the \( \{a_n^{(k)}\} \) sequence for any \( k \in \mathbb{N} \). Why? Because all the sequences in the entire equivalence class of \( \langle a_n^{(k)} \rangle \) differ with each other in some terms. So, we say that the equivalence classes of two given sequences are completely different if their tails do not match i.e., their limiting values are different.

[Levy’s proof using supremum.] What we are going to present now is probably the most confusing proof among all proofs to show that the set of real numbers is uncountable. Hence, follow the proof very closely and read every line multiple times. Like every other proof for this problem, this proof also uses contradiction.

We assume that the set of reals is countable. That is let \( \mathbb{R} = \{r_1, r_2, r_3, \ldots\} \). We will need several notations and definitions to show that the set of reals cannot be listed/counted using contradiction.

Define function \( f : \mathbb{R} \to \mathbb{R}^+ \) as

\[
f(r_n) = \frac{1}{2^n} \quad \text{for all } n \in \mathbb{N}
\]

Then, (i) \( f(r_n) > 0 \) for all \( n \in \mathbb{N} \), i.e., function value of every real number is positive, and (ii) \( \sum_{a \in B} f(a) \leq 1 \) for all finite-sized set \( B \subseteq \mathbb{R} \), i.e., the sum of function values for every finite-sized subset of \( \mathbb{R} \) is at most 1.

If \( A \) is a nonempty set of real numbers that is bounded above, then there exists a number we call supremum of set \( A \), denoted by \( S(A) \), such that (i) \( S(A) \geq a \) for all \( a \) in \( A \), i.e., \( S(A) \) is the upper bound of \( A \), and (ii) for each \( \epsilon > 0 \), there exists a number \( a \) in \( A \) such that \( a > S(A) - \epsilon \), i.e., \( S(A) \) is the least such upper bound.

If \( A \) is a nonempty set of real numbers, then define

\[
m(A) = S \left( \left\{ \sum_{b \in B} f(b), \text{ where } B \text{ is a finite-sized subset of } A \right\} \right)
\]

which implies that

\[
0 \leq m(\text{any subset of } \mathbb{R}) \leq 1 \quad (21)
\]

using the definition of the function \( f \).

Let’s define the set

\[
X = \{a \in \mathbb{R} \text{ such that } a < m((-\infty, a))\}
\]

134
Then, (i) \( X \) is nonempty because \( X \) must include 0 due to Equation \( \text{[21]} \) and (ii) \( X \) is bounded above by 1 due to Equation \( \text{[21]} \). Hence, the set \( X \) must have a supremum.

Let’s set

\[ \epsilon = f(S(X)) \]

Then, (i) \( \epsilon > 0 \) using the definition of function \( f \), and (ii) there exists \( x \) in \( X \) such that \( x > S(X) - f(S(X)) \) using the definition of supremum on set \( X \).

We make the following three observations: (i) Observation 1: \( S(X) < x + f(S(X)) \), (ii) Observation 2: \( x < m(-\infty, x) \), using the definition of \( X \), and (iii) Observation 3: \( x \leq S(X) \) because \( x \) is in \( X \) and using the definition of supremum.

Suppose we want to compute

\[ m((-\infty, x + f(S(X)))) \]

\[ = m((-\infty, x) \cup [x + f(S(X))]) \]

Observe that (i) \((-\infty, x)\) does not contain \( S(X) \) because \( x \leq S(X) \) from observation 3 and (ii) \([x, x + f(S(X))]\) contains \( S(X) \) because \( x \leq S(X) < x + f(S(X)) \) from observations 1 and 3. Therefore,

\[ m((-\infty, x + f(S(X)))) \]

\[ \geq m((-\infty, x)) + f(S(X)) \quad \text{(by the definition of } m) \]

\[ > x + f(S(X)) \quad \text{(using observation 2)} \]

This means that \( x + f(S(X)) \) is in \( X \), by using the definition of \( f \). From observation 1, we have \( x + f(S(X)) > S(X) \). How can \( x + f(S(X)) \) be in the set \( X \) and then be greater than the supremum of the set \( X \)? This contradicts the fact that \( S(X) \) is the supremum of \( X \). Therefore, the initial assumption is false and hence \( \mathbb{R} \) is uncountable.

[Hilbert’s hotel.] This is a beautiful thought experiment to show many aspects of countable infinity. Imagine a no-vacancy fully-booked infinite-room hotel in heaven with rooms numbered 1, 2, 3, so on up to infinity. Assume that each room is occupied by a person. Let’s check whether it is possible to accommodate new guests or not in the following scenarios.

\[ n \in \mathbb{N} \] guests arrive.] Possible!

Move person currently in room \( i \) to room \( i + n \) for all \( i \in \mathbb{N} \).

Move guest \( j \) to room \( j \) for all \( j \in \{1, 2, 3, \ldots, n\} \).

[Infinitely many guests arrive.] Possible!

Move person currently in room \( i \) to room \( 2i \) for all \( i \in \mathbb{N} \).

Move guest \( j \) to room \( 2j - 1 \) for all \( j \in \mathbb{N} \).

[Infinitely many buses, each containing infinitely many guests arrive.] Possible! Let

\[ R(j, k) = \frac{(j + k)^2 + (j - k) + 2}{2} \quad \text{(22)} \]

Move person currently in room \( i \) to room \( R(0, i) \) for all \( i \in \mathbb{N} \).

Move person \( k \) of bus \( j \) to room \( R(j, k) \) for all \( j, k \in \mathbb{N} \).

What is the intuition behind this idea? We can imagine that the hotel rooms are arranged in a triangle, as shown in Figure 59. First, the existing occupants of the hotel are moved to rooms in column 0. Then all the guests from \( j \)th bus are moved to
rooms in column \( j \) for every \( j \in \mathbb{N} \). This triangular mapping is shown in Equation [22]

\[
R(j, k, \ell, m) = 2^j \cdot 3^k \cdot 5^\ell \cdot 7^m
\]

(23)

Move person currently in room \( i \) to room \( R(0, 0, 0, i) \) for all \( i \in \mathbb{N} \).
Move person \( m \) in floor \( \ell \) of bus \( k \) at entrance \( j \) to room \( R(j, k, \ell, m) \) for all \( j, k, \ell, m \in \mathbb{N} \).

What is the intuition behind this idea? Every natural number greater than 1 can be uniquely represented as a product of primes using the unique prime factorization theorem. The four variables \( j, k, \ell, m \) become the exponents of the first four prime numbers and hence a 4-tuple of these variables map both the existing as well as the new guests to unique rooms. However, this method leaves several rooms empty, e.g.: 2, 3, and 5. (If we want a method that leaves no room empty we can use the higher-dimensional generalization of the triangular method.) We can use this method to solve other variants, too. But again, the approach might leave out some rooms empty.

[Infinitely many guests, each having infinitely long name arrive.] Impossible! This is because infinitely long sequences of names can be put in a one-to-one correspondence with real numbers but real numbers cannot be put in a one-to-one correspondence with the natural numbers.

Infinite types of infinity \((\aleph_0, \aleph_1, \aleph_2, \aleph_3, \ldots)\)

We now prove that there are an infinite types of infinity in two steps.

[Step 1. If \( A \) is a set, then \( |A| < |\mathcal{P}(A)| \).] Let’s prove this step using contradiction.
Suppose \( A \) is a set and \( |A| \geq |\mathcal{P}(A)| \). Note that \( \mathcal{P}(A) \) must contain all elements of \( A \). Hence \( |A| > |\mathcal{P}(A)| \) is impossible. So let’s suppose \( |A| = |\mathcal{P}(A)| \) for any set \( A \). This implies that there is a one-to-one correspondence \( f : A \rightarrow \mathcal{P}(A) \) between \( A \) and \( \mathcal{P}(A) \). If element \( a \) belongs to set \( A \), then \( f(a) \) represents the subset of \( A \) in \( \mathcal{P}(A) \) that corresponds to element \( a \).

Now consider the set \( Y \), where,
\[
Y = \{ \text{element } a \text{ in } A \text{ such that } a \text{ is not in } f(a) \}
\]
Clearly, \( Y \) is a subset of set \( A \). This means that \( Y \) must be present in \( \mathcal{P}(A) \). As there is a one-to-one correspondence between \( A \) and \( \mathcal{P}(A) \), there must be an element \( X \) in \( A \) that corresponds to \( Y \) in \( \mathcal{P}(A) \), as shown in Figure [60]. That is,
\[
\text{There is an element } X \text{ in } A \text{ such that } f(X) = Y
\]

Now let’s answer this simple question. Does \( X \) belong to \( Y \)? Two cases arise:
Figure 60: If \( f : A \to \mathcal{P}(A) \) is a one-to-one correspondence and \( Y \) belongs to \( \mathcal{P}(A) \), then there is an element \( X \) in \( A \) such that \( f \) maps \( X \) to \( Y \).

- \([X \notin Y.]\) If \( X \) does not belong to \( Y \), then \( X \) belongs to \( Y \) by definition of \( Y \).
- \([X \in Y.]\) If \( X \) belongs to \( Y \), then \( X \) does not belong to \( Y \) by definition of \( Y \).

This is a clear contradiction! Therefore, our supposition is wrong. Thus \(|A| < |\mathcal{P}(A)|\).

**Step 2.** \( N_0 < N_1 < N_2 < N_3 < \cdots \) We now prove that there an infinite types of infinity. Substitute \( A = \mathbb{N} \) in Step 1 to get \(|\mathbb{N}| < |\mathcal{P}(\mathbb{N})|\). Similarly, substitute \( A = \mathcal{P}(\mathbb{N}) \) in Step 1 to get \(|\mathcal{P}(\mathbb{N})| < |\mathcal{P}(\mathcal{P}(\mathbb{N}))|\). Continue this process till infinity to get

\[
|\mathbb{N}| < |\mathcal{P}(\mathbb{N})| < |\mathcal{P}(\mathcal{P}(\mathbb{N}))| < |\mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N})))| < \cdots
\]

From the previous section we know that \(|\mathbb{R}| = |\mathcal{P}(\mathbb{N})| = \aleph_1\). Let’s consider \(|\mathcal{P}(\mathcal{P}(\mathbb{N}))| = N_2\), \(|\mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N})))| = N_3\), and so on. Thus, we have

\[
N_0 < N_1 < N_2 < N_3 < \cdots
\]

There are an infinite types of infinity.

**References**

Georg Cantor was the first person to show using diagonalization proof \([\text{Cantor, 1891}]\) and nested intervals proof \([\text{Cantor, 1874}]\) that the number of real numbers represents a bigger infinity compared with the the number of natural numbers. The convoluted proof based on Cauchy sequences on rational numbers is from \([\text{Wenner, 1969}]\). To know more about infinite sequences and series, refer to the book \([\text{Bourchtein and Bourchtein, 2022}]\). The highly complicated analytical proof is from \([\text{Levy, 2009}]\). Multiple proofs of uncountability of real numbers are given in \([\text{Knapp and Silva, 2012}]\). Hilbert’s Hotel is introduced by David Hilbert and popularized by George Gamow \([\text{Gamow, 1988}]\).
Prisoners and Hats

Problem

10 prisoners are standing in a single line, all facing one direction. Each prisoner can see only the prisoners in front of him (higher-numbered prisoners). An executioner comes and places a hat on everyone’s head. The color of the hat is any of the three colors: red, green, or blue. Prisoners cannot see the color of their own hat. The executioner goes to the 1st prisoner and asks him his hat color. The prisoner can respond with any of: “red”, “green” or “blue”. The answer is audible to everyone. If his answer matches his hat color, he survives. If his answer does not match his hat color, he dies. The execution happens only after all prisoners have guessed their hat colors. The executioner then proceeds to the 2nd prisoner, and asks him his hat color. This process continues until all the prisoners have been questioned.

One day is given to the prisoners to come up with a strategic plan for maximizing the number of lives saved. What is the strategy? How many prisoners can be saved?

Solution

Solution: \([n \text{ prisoners}, \text{ line}, k \text{ colors}, \text{ sequential}]\)

There are several hat puzzles that require strategies and it seems that using modular arithmetic is one of the best ways to solve such puzzles.

[Alternate-saving strategy (non-optimal).] A common solution to the puzzle is as follows. Let’s denote the 10 prisoners by \(P_1, P_2, \ldots, P_{10}\) and colors red, blue, and green by letters \(r, b,\) and \(g\), respectively.

Starting from the first prisoner, every alternate prisoner shouts the hat color of the prisoner in his front. When there are 10 prisoners in the line, prisoner \(P_1\) shouts the hat color of \(P_2\). The color shouted by \(P_1\) may or may not match his hat color and hence he may or may not survive. Prisoner \(P_2\) shouts the hat color shouted by \(P_1\) and survives. Next prisoner \(P_3\) shouts the hat color of \(P_4\). Prisoner \(P_4\) shouts the hat color shouted by \(P_3\) and survives. This process of alternate-saving continues. When there are odd number of prisoners, the last prisoner shouts a random color. In this strategy, a minimum of 5 prisoners will be saved. Let the real hat colors of the prisoners \(P_i\) to \(P_{10}\) be \([r, b, r, g, b, r, r, g, b, g]\). Then Table \(34\) shows the application of the alternate-saving strategy, saving 5 prisoners.

![Table 34](image)

**Number of prisoners saved among 10 prisoners \(\geq 5\)**

**Number of prisoners saved among \(n\) prisoners \(\geq \left\lfloor \frac{n}{2} \right\rfloor\)**

[Modular arithmetic strategy (optimal).] We use modular arithmetic to solve the
puzzle. Let’s define some variables that we will use to describe the strategy. Let \( h_i \) be the actual hat color number of \( P_i \); \( x_i \) be \( P_i \)'s guess of his hat color number \( h_i \); \( F_i \) be the sum of all actual hat color numbers in front of \( P_i \) i.e., \( F_i = h_{i+1} + h_{i+2} + \cdots + h_{10} \); \( B_i \) be the sum of all actual hat numbers at the back of \( P_i \) till \( P_2 \) i.e., \( B_i = h_2 + h_3 + \cdots + h_{i-1} \); \( a \mod b \) be the remainder when a whole number \( a \) is divided by a whole number \( b \).

Let the real hat colors of the prisoners \( P_1 \) to \( P_{10} \) be \([r, b, r, g, r, r, g, b, r, g, g] \). We code the colors \( r, g, b \) with numbers \( 0, 1, 2 \), respectively. Table 35 shows the application of the modular arithmetic strategy, saving 9 out of 10 prisoners. The strategy has two steps:

1. **Plan of the first prisoner \( P_1 \),** and
2. **Plan of the remaining prisoners.**

### Table 34: Alternate-saving strategy for 10 prisoners.

<table>
<thead>
<tr>
<th>Prisoner</th>
<th>( P_1 )</th>
<th>( P_2 )</th>
<th>( P_3 )</th>
<th>( P_4 )</th>
<th>( P_5 )</th>
<th>( P_6 )</th>
<th>( P_7 )</th>
<th>( P_8 )</th>
<th>( P_9 )</th>
<th>( P_{10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual hat color</td>
<td>( r )</td>
<td>( b )</td>
<td>( r )</td>
<td>( g )</td>
<td>( b )</td>
<td>( r )</td>
<td>( g )</td>
<td>( b )</td>
<td>( g )</td>
<td></td>
</tr>
<tr>
<td>Guessed hat color</td>
<td>( b )</td>
<td>( b )</td>
<td>( g )</td>
<td>( g )</td>
<td>( r )</td>
<td>( r )</td>
<td>( g )</td>
<td>( g )</td>
<td>( g )</td>
<td>( g )</td>
</tr>
<tr>
<td>Survives</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 35: Modular arithmetic strategy for 10 prisoners

#### Step 1. Plan of the first prisoner \( P_1 \).

The first prisoner \( P_1 \) takes the modulo of the sum of all actual hat color numbers in front of him with the number of colors and he shouts this number. In other words, \( P_1 \) shouts \( x_1 \), where \( x_1 \) is found by the equation

\[
x_1 = F_1 \mod 3.
\]

Why does \( P_1 \) shout \( F_1 \mod 3 \)? Prisoner \( P_1 \) instead of guessing his hat color number, he gives the prisoners in front of him information that might be useful for them to guess their hat color numbers easily.

In our example, we see that \( x_1 = F_1 \mod 3 = 7 \mod 3 = 1 \). Hence, \( P_1 \) shouts 1 (or green). Though green is not his hat color, this number gives information to the remaining prisoners. If the hats are put on the prisoners randomly, then the probability of \( P_1 \)'s survival is \( \frac{1}{3} \). On the other hand, if hats are put on the prisoners arbitrarily, then the executioner can make sure that \( P_1 \) dies.

#### Step 2. Plan of the remaining prisoners.

We know that \( F_1 \) is the sum of all hat color numbers from \( P_2 \) to \( P_{10} \). We transform \( F_1 \) to get

\[
F_1 = h_2 + h_3 + \cdots + h_{10} = (B_i + h_i + F_i).
\]

for all \( i \in [2, 10] \)
Using information given by \( P_1 \) and the formula above, we have
\[
x_i = F_i \mod 3 = (B_i + h_i + F_i) \mod 3.
\]
for all \( i \in [2, 10] \)

An intuition for the formula is shown below

Each prisoner \( P_i \), where \( i \in [2, 10] \), already knows his \( F_i \), which is the sum of all actual hat color numbers of prisoners in front of him. Prisoner \( P_i \) can find \( B_i \) by adding the guessed hat color numbers of the prisoners at his back till \( x_2 \). With this knowledge, each prisoner \( P_i \) can guess correctly his hat color number \( x_i \) using the formula
\[
(B_i + x_i + F_i) \mod 3 = x_1.
\]
for all \( i \in [2, 10] \).

Let’s apply this plan to our example. For the 2nd prisoner \( P_2 \), we know that \( F_2 = 5 \) and \( B_2 = 0 \) (because while computing \( B_i \)'s we do not consider the first prisoner). Substituting the values in the equation \((B_2 + x_2 + F_2) \mod 3 = x_1\), we get \( x_2 = 2 \) (blue). Similarly, for the 3rd prisoner \( P_3 \), we know that \( F_3 = 5 \) and \( B_3 = 2 \). Substituting the values in the equation \((B_3 + x_3 + F_3) \mod 3 = x_1\), we get \( x_3 = 0 \) (red). This process continues till prisoner \( P_{10} \). Refer to \textsc{FinitePrisonersSequential} for the strategy for \( n \) prisoners and \( k \) colors. Thus,

Number of prisoners saved among 10 prisoners \( \geq 9 \)
Number of prisoners saved among \( n \) prisoners \( \geq n - 1 \)

\textbf{FINITEPRISONERSSEQUENTIAL}(\( n, k \))

\textbf{Input:} Number of prisoners and hat colors are \( n \) and \( k \), respectively.
\textbf{Output:} Strategy to maximize the number of lives saved.
1. \( F_1 \leftarrow h_2 + h_3 + \cdots + h_n \); \( x_1 \leftarrow F_1 \mod k \)
2. \( P_1 \) shouts his hat color as \( x_1 \)
3. \textbf{for} \( i \leftarrow 2 \) \textbf{to} \( n \) \textbf{do}
4. \hspace{1em} \( B_i = x_2 + x_3 + \cdots + x_{i-1} \)
5. \hspace{1em} \( F_i = h_{i+1} + h_{i+2} + \cdots + h_n \)
6. \hspace{1em} \( x_i \) is computed from \((B_i + x_i + F_i) \mod k = F_1 \mod k \)
7. \hspace{1em} \( P_i \) shouts his hat color as \( x_i \)

\textbf{Variant: [\( n \) prisoners, circle, \( k \) colors, simultaneous]}

Let the number of prisoners and hat colors be \( n \) and \( k \), respectively. The prisoners are shown beforehand all \( k \) different colored hats that will be used for the execution. The prisoners are made to stand in a big circle. The executioner places hats on everyone’s
head. Each prisoner gets an opportunity see the hat colors of all other prisoners but not the color of his own hat. The prisoners are not allowed to communicate with each other. At a specified time, all prisoners must guess their hat colors simultaneously. If a prisoner’s guess is incorrect, he will be shot dead. One day is given to the people to come up with a plan for maximizing the number of lives saved. What strategy can maximize the number of lives saved?

**[Solution.]** The problem is solved in two parts: (1) We solve the problem with \(n\) prisoners and \(n\) colors. (2) We use the solution from the first part to solve the problem with \(n\) prisoners and \(k\) colors.

**[Part 1. \(n\) prisoners with \(n\) colors.]** Let there be \(n\) prisoners \(P_0\) to \(P_{n-1}\). Let \(h_i\) be the actual hat color number of \(P_i\); \(x_i\) be guessed hat color number of \(P_i\); \(S\) be the sum of all hat color numbers i.e., \(S = h_0 + h_1 + \cdots + h_{n-1}\); and \(S_i\) be the sum of all other hat numbers except \(h_i\) i.e., \(S = S_i + h_i\).

Each prisoner guesses his hat color \(x_i\) as

\[
x_i = (i - S_i) \mod n \quad \text{for all } i \in [0, n-1]
\]

We know from the definition of modulus (or mod or \(\%\)) that

\[
S \mod n = j \quad \text{for some } j \in [0, n-1]
\]

\[
\implies (S_i + h_j) \mod n = j \quad \text{for some } j \in [0, n-1]
\]

\[
\implies h_j \mod n = (j - S_j) \quad \text{for some } j \in [0, n-1]
\]

Observe that when all prisoners \(P_1\) to \(P_{n-1}\) guess their hat color numbers according to the formula \(x_j = (i - S_i) \mod n\), there is exactly one prisoner \(P_j\) whose guessed hat number matches his hat color i.e., \(x_j = h_j\). That prisoner \(P_j\) alone will be right and the rest of the prisoners will be wrong.

**[Part 2. \(n\) prisoners with \(k\) colors.]** Select \(\left\lfloor \frac{n}{k} \right\rfloor \times k\) prisoners. The remaining prisoners can guess randomly. Divide the selected prisoners into \(\left\lfloor \frac{n}{k} \right\rfloor\) groups each consisting of \(k\) prisoners. Regard each group as a subproblem with \(k\) prisoners and \(k\) colors. We already know how to solve each such subproblem using the solution given in the first part described above. As we can save exactly one prisoner in each group, we can save a total of \(\left\lfloor \frac{n}{k} \right\rfloor\) from the selected prisoners. **FINITEPRISONERSSIMULTANEOUS** is the algorithm for the strategy.

**Number of prisoners saved is in the range** \(\left[ \left\lfloor \frac{n}{k} \right\rfloor, n - \left\lfloor \frac{n}{k} \right\rfloor (k - 1) \right]\).

**Variant: [Infinite prisoners, line, \(k\) colors, sequential]**

Let the number of prisoners and hat colors be countably infinite and \(k\), respectively. The prisoners are shown beforehand all \(k\) different colored hats that will be used for the execution. The prisoners are made to stand in a line facing positive infinity. The executioner places hats on everyone’s head. Each prisoner \(P_i\) can see the hat colors of \(P_{i+1}, P_{i+2}, \ldots\) but cannot see the hat colors of \(P_1, P_2, \ldots, P_{i-1}, P_i\). The prisoners
**FinitePrisonersSimultaneous**($n, k$)

**Input:** Number of people $n$ and number of colored hats $k$.
**Output:** Strategy to save exactly $\left\lfloor \frac{n}{k} \right\rfloor$ prisoners.

1. **if** $k > n$ **then**
2. **else**
3. make $\left\lfloor \frac{n}{k} \right\rfloor$ pairwise disjoint groups of $k$ prisoners
4. **parallel for** group ← 1 to $\left\lfloor \frac{n}{k} \right\rfloor$ do
5. **FinitePrisonersSimultaneousGroup**(group, $k$)

**FinitePrisonersSimultaneousGroup**(group, $n$)

**Input:** Number of people in the group and colored hats are both equal to $n$.
**Output:** Save exactly one life in group group.

1. **parallel for** $i$ ← 0 to $n - 1$ do
2. $S_i$ ← sum of the hat numbers of everyone in the group as seen by $P_i$
3. person $i$ guesses his hat color as $x_i = (i - S_i) \mod n$

are not allowed to communicate with each other. The prisoners must guess their hat colors one-by-one starting from the first prisoner. Each prisoner’s voice is audible to other prisoners. Prisoners who guess incorrectly are shot dead. One day is given to the people to come up with a plan. What is the strategy to minimize the number of deaths?

**Solution (optimal).** In this solution, we make sure that at most one prisoner gets executed and all the remaining infinite number of prisoners survive. The solution uses the axiom of choice and hence only guarantees the existence of a strategy without explicitly providing a strategy.

The solution is presented in four steps: (1) Divide the hat sequences into groups. (2) Use a choice function on the groups. (3) Plan of the first prisoner $P_1$. (4) Plan of the remaining prisoners.

**Step 1. Divide the hat sequences into groups.** A hat color is in the range $[0, k - 1]$. A hat configuration/sequence is defined as an arrangement of hats for the prisoners. Each arrangement is a $k$-ary sequence of digits. The given arrangement of hats is called the given hat sequence. For $n$ prisoners, the total number of hat sequences is $k^n$. For an infinite number of prisoners, the number of hat sequences is infinite. For example, when $k = 5$, a possible hat sequence is $20013143021341431\ldots$.

Two hat sequences are related when they are equal after a certain position. For example, the following two 5-ary hat sequences $A$ and $B$ are related because the boxed suffix digits starting from 14... are the same.

$$A : 203204103240421040331404220231\ldots$$

$$B : 0403200102343223242221404220231\ldots$$
The related relation satisfies transitivity. If sequences $A$ and $B$ are related and sequences $B$ and $C$ are related, then $A$ and $C$ are related.

$B : 040320010234322342221404$ 220231...

$C : 304020343332440234203221$ 220231...

Each $k$-ary hat sequence belongs to precisely one group. All the related sequences are collected into a single group. The related relation is also called the equivalence relation and the sequence groups are also called equivalence classes. We divide the entire set of hat sequences into various groups (or equivalence classes) of related (or equivalent) sequences.

[Step 2. Use a choice function on the groups.] Axiom of choice says that given a collection of sets, we can form a new set consisting of one element from each of the input sets. If the collection consists of a finite number of sets, then choosing a representative element from each set is simple. However, if the collection consists of an infinite number of sets, then constructing a choice function $f$ that chooses an element from each of the sets is nontrivial. Hence, we assume that such a function exists without constructing it.

$$f(\text{set 1, set 2, } \ldots) \rightarrow \{\text{an element from set 1, an element from set 2, } \ldots\}$$

We apply the axiom of choice to our problem as follows. In this problem, we have sets of sequences. We assume that there exists a choice function $f$ that selects a representative $k$-ary sequence from each of the sequence groups. The prisoners meet and agree upon a choice function.

[Step 3. Plan of the first prisoner $P_1$.] Let $S$ denote the given hat sequence, $s_i$ denote the actual hat color of prisoner $P_i$ in $S$, and $x_i$ denote the guessed hat color by $P_i$. Then $P_i$ sees $S : x_1 \ x_2 \ \ldots \ \ x_i \ | \ \ \ s_{i+1} \ s_{i+2} \ s_{i+3} \ \ldots$

$i$ hats guessed visible hats

Every prisoner $P_i$ identifies the group $G$ to which the sequence $S$ belongs. The prisoner can also know the unique representative sequence $R$ of the group $G$ by using an agreed-upon choice function. Let $r_i$ denote the hat color of prisoner $P_i$ in $R$. We define $F_i$ (respectively, $B_i$) as the sum of differences between the hat colors in the given sequence $S$ and the chosen sequence $R$ in front of (respectively, at the back of) prisoner $P_i$. Then

$$F_i = (s_{i+1} - r_{i+1}) + (s_{i+2} - r_{i+2}) + \cdots = \sum_{j=i+1}^{\infty} (s_j - r_j) \quad \text{for } i \geq 1$$

$$B_i = (x_{i-1} - r_{i-1}) + (x_{i-2} - r_{i-2}) + \cdots + (x_2 - r_2) = \sum_{j=2}^{i-1} (x_j - r_j) \quad \text{for } i \geq 2$$

The sequences $S$ and $R$ differ in only a finite number of places because $S$ and $R$ are related. Hence, $F_i$ is finite.

Prisoner $P_1$ sees the following hat sequence in front of him and guesses $x_1$

$$P_1 \text{ sees } S : \ x_1 \ | \ s_2 \ s_3 \ s_4 \ \ldots$$

$$P_1 \text{ guesses } x_1 = F_1 \mod k$$

143
The first prisoner’s guess is majorly to give the prisoners in front of him information about the differences of their hat numbers in the given sequence $S$ with their corresponding hat numbers in the representative sequence $R$.

**Step 4. Plan of the remaining prisoners.**] Prisoner $P_i$ sees the following hat sequence in front of him and guesses $x_i$ using the information from the previously guessed numbers

$$P_i \text{ sees } S : x_1 \ x_2 \ldots \ x_i \ s_{i+1} \ s_{i+2} \ s_{i+3} \ldots$$

$P_i$ guesses $x_i$ solving $(B_i + (x_i - r_i) + F_i) \mod k = x_i$ for $i \geq 2$

The formula is very similar to that we used for $n$ prisoners version, however, the hat numbers there is replaced by the difference of the hat numbers between $S$ and $R$.

For example, for $k = 5$ colors, suppose

$$S = 023420322120130124233414321444\ldots$$
$$R = 234222140420130124233414321444\ldots$$

then Table 36 describes the strategy on an example in which only one prisoner gets executed and the remaining prisoners survive. INFINITEPRISONERSSEQUENTIAL represents the generic algorithm for the strategy.

<table>
<thead>
<tr>
<th>Prisoner $P_i$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
<th>$P_5$</th>
<th>$P_6$</th>
<th>$P_7$</th>
<th>$P_8$</th>
<th>$P_9$</th>
<th>$P_{10}$</th>
<th>$P_{11}$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual hat color number $s_i$</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>Representative hat color number $r_i$</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>Sum of differences at the back $B_i$</td>
<td>$-$</td>
<td>$-$</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>Sum of differences at the front $F_i$</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>Guessed hat color number $x_i$</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>Survives</td>
<td>$\chi$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

Table 36: Strategy for infinite prisoners on an example. The values are modulo 5.

**Variant: [Infinite prisoners, line, $k$ colors, simultaneous]**

Let the number of prisoners and hat colors be countably infinite and $k$, respectively. The prisoners are shown beforehand all $k$ different colored hats that will be used for the execution. The prisoners are made to stand in a line facing positive infinity. The executioner places hats on everyone’s head. Each prisoner $P_i$ can see the hat colors of $P_{i+1}, P_{i+2}, \ldots$ but cannot see the hat colors of $P_1, P_2, \ldots, P_{i-1}, P_i$. The prisoners are not allowed to communicate with each other. The prisoners must guess their hat colors simultaneously. Prisoners who guess incorrectly are shot dead. One day is given to the people to come up with a plan. What is the strategy to minimize the number of deaths?
[Solution (optimal).] In this solution, we make sure that only a finite number of prisoners die and all the remaining infinite number of prisoners survive. The solution uses the axiom of choice and hence only guarantees the existence of a strategy without explicitly providing a strategy.

The solution is presented in three steps: (1) Divide the hat sequences into groups. (2) Use a choice function on the groups. (3) Plan of the prisoners. The first two steps are exactly the same from the previous solution. Hence, we explain only the third step here.

[Step 3. Plan of the prisoners.] We inherit the notations from the previous solution. Each prisoner \( P_i \) looks at the hat sequence in front of him. He constructs the given hat sequence \( S \) as follows.

\[
P_i \text{ constructs } S : \begin{array}{c}
\text{? ? ... ? } \\
\text{i hats not visible} \\
\text{visible hats}
\end{array} \begin{array}{c}
S_{i+1} \ S_{i+2} \ S_{i+3} \ldots \\
\text{visible hats}
\end{array}
\]

\( P_{14} \) constructs \( S : \begin{array}{c}
\text{????????????????} \\
\text{14 hats not visible} \\
\text{visible hats}
\end{array} \begin{array}{c}
2342221404220231 \ldots \\
\text{(Example)}
\end{array}
\]

Every prisoner identifies the group \( G \) to which the sequence \( S \) belongs. Every prisoner can also know the representative sequence \( R \) of the group \( G \). The prisoner \( P_i \) guesses the hat color he was wearing in the representative sequence \( R \), i.e., \( r_i \).

\[
P_i \text{ guesses } r_i.
\]

The given hat sequence \( S \) and the representative hat sequence \( R \) are related as they belong to the same group. This means that the given and the representative sequences match after some position that is at a finite distance from the start of the prisoners’ line. Hence, all prisoners after this position will be correct in their guesses and prisoners before this position may or may not be correct. Hence, in this strategy, only a finite number of prisoners die and an infinite number of prisoners survive. INFINITEPRISONERSSEQUENTIAL algorithm gives the strategy.

Number of prisoners executed is finite.

Problems

1. [Variant: \( n \) prisoners, circle, \( k \) colors, simultaneous, pass.] Let the number of prisoners and hat colors be \( n \) and \( k \), respectively. The prisoners are shown beforehand all \( k \) different colored hats that will be used for the execution. The prisoners are made to stand in a big circle. The executioner places hats on everyone’s head. Each prisoner gets an opportunity see the hat colors of all other prisoners but not the color of his own hat. The prisoners are not allowed to communicate with each other. At a specified time, all prisoners must guess their hat colors or say “pass”, simultaneously. If at least one prisoner guesses incorrectly, all prisoners will be shot dead. If no prisoner guesses incorrectly at least one prisoner guesses correctly, all prisoners will be set free. One day is given to the people to come up with
### Infinite Prisoners Sequential \((k)\)

**Input:** Number of prisoners and hat colors are infinite and \(k\), respectively.

**Output:** Strategy to minimize the number of lives executed.

1. \(P_1\) constructs \(S\) and knows \(R\)
2. \(F_1 \leftarrow \sum_{j=2}^{\infty} (s_j - r_j); x_1 \leftarrow F_1 \mod k\)
3. \(P_1\) shouts his hat color as \(x_1\)
4. for \(i \leftarrow 2\) to \(\infty\) do
   5. \(P_i\) constructs \(S\) and knows \(R\)
   6. \(F_i = \sum_{j=i+1}^{\infty} (x_j - r_j); B_i = \sum_{j=2}^{i-1} (x_j - r_j)\)
   7. \(x_i\) is computed from \((B_i + (x_i - r_i) + F_i) \mod k = x_i\)
   8. \(P_i\) shouts his hat color as \(x_i\)

### Infinite Prisoners Simultaneous \((k)\)

**Input:** Number of prisoners and hat colors are infinite and \(k\), respectively.

**Output:** Strategy to minimize the number of lives executed.

1. parallel for \(i \leftarrow 1\) to \(\infty\) do
   2. \(P_i\) constructs \(S\) and knows \(R\)
   3. \(P_i\) shouts his hat color as \(r_i\)

---

...a plan. What is the strategy for their survival?

2. [Variant: Infinite colors.] Solve all the problems discussed with an infinite number of colors.

**References**

Solution to \([n\) prisoners, 2 colors, sequential] problem has appeared in [Winkler, 2002], Wu riddles forum [Wu, 2016], and [Zhou, 2008]. Solution to the \([n\) prisoners, \(k\) colors, simultaneous] problem is presented in [Hardin and Taylor, 2008]. According to Hardin and Taylor, solution to \([\infty\) prisoners, 2 colors, simultaneous] version was discovered by Yuval Gabay and Michael O’Connor. Refer to [Guo et al., 2006] for \([n\) prisoners, circle, \(k\) colors, simultaneous, pass] version.
Prisoners and Boxes

Problem

A 100 prisoners are about to be executed by a mad jailer based on the result of a game. The prisoners are given unique numbers from 1 to 100, one number per prisoner. The cruel jailer has placed the prisoner numbers in 100 boxes – one number inside a box, and the boxes are kept in a separate room. The boxes are uniquely numbered from 1 to 100. The 100 prisoners are let into the room one by one. Each prisoner can open up at most 50 boxes to find his number. He may or may not find his number in the boxes he opens. In any case, he must leave the room exactly as it was before he entered. There can be absolutely no form of communication between the prisoners. The aim of each prisoner is to find his number in at most 50 arbitrary boxes he opens. If all prisoners find their numbers in the boxes then they all survive. On the other hand, if at least one prisoner does not find his number then they all die. One day is given to the prisoners to meet and discuss a strategy to maximize the probability of their survival.

What strategy maximizes the probability of their survival?

Solution

The problem asks us to find a strategy to maximize the probability of survival of all prisoners. However, it is not clear which branches of mathematics other than probability theory might be helpful in attacking this problem. The problem is difficult to solve.

Let $P_i$ ($i \in [1,100]$) denote the $i$th prisoner whose number is $i$. Let $B_i$ denote the $i$th box. We like to compute the success probability i.e., the probability that all prisoners survive.

Basic strategies

[Same boxes strategy (non-optimal).] In this simple strategy, each prisoner opens the same 50 boxes $B_1$ to $B_{50}$, every time. In this case, the probability of success is 0 because the 50 boxes contain only 50 numbers and hence it is not possible for all 100 prisoners to find their numbers.

Success probability = 0.

[Random boxes strategy (non-optimal).] In this strategy, each prisoner opens 50 random boxes. Then, for $i \in [1,100]$

$$\text{Prob. that } P_i \text{ finds his number among 50 random boxes} = \frac{50}{100} = \frac{1}{2}$$
Prob. that $P_1$ to $P_{100}$ find their numbers among 50 random boxes they open

$$= \left( \frac{1}{2} \right)^{100} \approx 7.8886 \times 10^{-31}$$

Success probability $\approx 0.0000000000000000000000000000078886$.

**[Split boxes strategy (non-optimal).]** In this strategy, the first 50 prisoners $P_1$ to $P_{50}$ open the first 50 boxes $B_1$ to $B_{50}$ and the last 50 prisoners $P_{51}$ to $P_{100}$ open the last 50 boxes $B_{51}$ to $B_{100}$. All prisoners win if the first 50 prisoners find their names in the first 50 boxes (which implies that the last 50 prisoners find their names in the last 50 boxes). Then

- Prob. that $P_1$ finds his number among boxes $B_1$ to $B_{50} = \frac{50}{100}$
- Prob. that $P_2$ finds his number among boxes $B_1$ to $B_{50} = \frac{49}{99}$
- Prob. that $P_i, (i \in [1,50])$ finds his number among boxes $B_1$ to $B_{50} = \frac{(50 - i + 1)}{(100 - i + 1)}$
- Prob. that $P_{50}$ finds his number among boxes $B_1$ to $B_{50} = \frac{1}{51}$
- Prob. that prisoners $P_1$ to $P_{50}$ find their numbers among boxes $B_1$ to $B_{50}$

$$= \frac{50}{100} \times \frac{49}{99} \times \cdots \times \frac{1}{51} = \frac{(50!)^2}{100!} \approx 9.911653 \times 10^{-30}$$

Success probability $\approx 0.000000000000000000000000000009911653$.

The success probability of this strategy is approximately $12.6 \times$ better than that of random boxes strategy. It is still very low, however. Both the random boxes and split boxes strategies are commonly thought methods to solve the puzzle. The best strategy is in the realization that prisoners need not decide beforehand which boxes they will be opening. They can decide which boxes to open in the box room.

**Gál-Milterson-Skylum’s pointer boxes strategy (optimal)**

In this section, we present a mind-blowing optimal strategy based on combinatorics that has a success probability more than 30%. This probability is approximately $395288724065291741375858133648 \times$ better than that of the random boxes strategy. Impossible to believe that such a strategy exists, right?

There are many strategies where the prisoners decide dynamically which box to open. Initially $P_1$ can choose to open any of the 100 boxes. If the person fails to see his number on his first attempt, then the person can choose any of the remaining 99 boxes for each each of the 99 numbers the person has seen. If the person fails to see his number on this second attempt, then the person can choose any of the remaining 98 boxes for each of the 99×98 pairs of two numbers the person has seen. This process
continues. We have
#Choices for $P_i$ to open 1 box = 100
#Choices for $P_i$ to open 2 boxes = $100 \times 99^{99}$
#Choices for $P_i$ to open 3 boxes = $100 \times 99^{99} \times 98^{99\times98}$
#Choices for $P_i$ to open 50 boxes = $100 \times 99^{99} \times 98^{99\times98} \times 51^{99\times\cdots\times51}$

#Choices for $P_i$ to $P_{100}$ to open 50 boxes = \( \left(100 \times 99^{99} \times 98^{99\times98} \times 51^{99\times\cdots\times51}\right)^{100} \)

Oops! So we need to choose the best strategy among such gigantic number of possible strategies. Seems like an impossible task.

**Strategy.** The prisoner $P_i$, where $i \in [1, 100]$, opens box $B_i$. If $B_i$ contains the prisoner’s number $i$, then he stops, closes the box, and returns. On the other hand if the box contains some number $j$, where $j \neq i$, then the prisoner $P_i$ opens box $B_j$ to find his number. If $B_j$ contains the prisoner’s number $i$, then he stops, closes the box, and returns. If $B_j$ contains some other number $k$, where $k \neq i$, then the prisoner $P_i$ opens box $B_k$ to find his number. This process continues until either the prisoner $P_i$ finds his number $i$ or he has opened 50 boxes. At the end, the prisoner makes sure that all boxes are closed.

**Example.** For simplicity, let us assume that there are 10 prisoners and 10 boxes. Each prisoner can open at most 5 boxes and the rules are all the same. Let the boxes have prisoner numbers inside them as shown in Table 37.

<table>
<thead>
<tr>
<th>Box</th>
<th>Prisoner</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_1$</td>
<td>7</td>
</tr>
<tr>
<td>$B_2$</td>
<td>10</td>
</tr>
<tr>
<td>$B_3$</td>
<td>6</td>
</tr>
<tr>
<td>$B_4$</td>
<td>2</td>
</tr>
<tr>
<td>$B_5$</td>
<td>9</td>
</tr>
<tr>
<td>$B_6$</td>
<td>8</td>
</tr>
<tr>
<td>$B_7$</td>
<td>3</td>
</tr>
<tr>
<td>$B_8$</td>
<td>1</td>
</tr>
<tr>
<td>$B_9$</td>
<td>5</td>
</tr>
<tr>
<td>$B_{10}$</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 37: Boxes and the prisoner numbers they contain.

<table>
<thead>
<tr>
<th>Prisoner</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$[B_7, B_3, B_6, B_8]$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>$[B_2, B_{10}, B_4]$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>$[B_6, B_8, B_7, B_3]$</td>
</tr>
<tr>
<td>$P_4$</td>
<td>$[B_4, B_2, B_{10}]$</td>
</tr>
<tr>
<td>$P_5$</td>
<td>$[B_5, B_9]$</td>
</tr>
<tr>
<td>$P_6$</td>
<td>$[B_6, B_8, B_7, B_3]$</td>
</tr>
<tr>
<td>$P_7$</td>
<td>$[B_2, B_3, B_6, B_8]$</td>
</tr>
<tr>
<td>$P_8$</td>
<td>$[B_6, B_1, B_7, B_3, B_9]$</td>
</tr>
<tr>
<td>$P_9$</td>
<td>$[B_6, B_3]$</td>
</tr>
<tr>
<td>$P_{10}$</td>
<td>$[B_{10}, B_4, B_2]$</td>
</tr>
</tbody>
</table>

Table 38: Prisoners and their strategies.

Prisoner $P_1$ enters the box room and opens $B_1$ and finds number 7. As per the strategy, he opens box $B_7$ and finds 3. Then he opens $B_3$ to find 6. Continuing the process, he opens box $B_6$ and sees 8. Then he opens box $B_8$ for a surprise. He finds his number i.e., 1 and he is happy. He closes all the boxes and leaves the room. In a similar way, prisoner $P_2$ opens box $B_2$ and finds 10 inside. He opens $B_{10}$ to see 4. Opening box $B_4$ he finds his number and stops. Continuing in a similar fashion, other prisoners open the boxes as per the strategy as shown in Table 38.

It is clear from Table 38 that the set of box numbers opened by each prisoner forms a cycle or a loop. In fact, the strategies of prisoners $P_1, P_3, P_6, P_7,$ and $P_8$ forms the cycle
(1, 7, 3, 6, 8), the strategies of \( P_2, P_4, \) and \( P_{10} \) forms another cycle, and the strategies of \( P_5 \) and \( P_9 \) forms the third cycle. In total, the permutation of prisoner numbers in the 10 boxes can be written using cycles as

\[
\text{Permutation} = (1, 7, 3, 6, 8)(2, 10, 4)(5, 9)
\]

The core idea of the strategy is the concept of permutation cycles, mentioned above. All prisoners whose numbers are in a permutation cycle of length \( k \) open exactly \( k \) boxes. For example, \( P_1, P_3, P_6, P_7, \) and \( P_8 \) must open 5 boxes to find their numbers, \( P_2, P_4, \) and \( P_{10} \) must open 3 boxes to find their numbers, and \( P_3 \) and \( P_9 \) must open 2 boxes each.

The prisoners who find their numbers with this strategy will be those whose numbers occur in a permutation cycle of length at most 5. It is easy to see that for the above example, all prisoners will survive because each prisoner has opened at most 5 boxes to find his number. For an equally plausible other example, if the permutation of prisoner numbers can be represented with all cycles of length less than or equal to 5, the prisoners will survive.

[Success probability.] Now, the big question is: What is the probability that a random permutation of 10 numbers contains a cycle of length at most 5? This question is equivalent to asking the probability that a random permutation does not contain a cycle of length 6 or longer. To answer this question, let us first find the probability that a random permutation of 10 numbers having a 6-cycle (or cycle of length 6).

\[
\text{#Permutations of 10 numbers having a 6-cycle} = \text{#Ways to choose 6 numbers} \times \text{#Ways to arrange the 6 numbers in cyclic order} \times \text{#Ways the remaining 4 elements can be permuted}
\]

\[
= 10 \times 5! \times 4! = \frac{10!}{6! \times 4!} \times 5! \times 4! = \frac{1}{6} \times 10!
\]

Similarly, we can find the number of permutations of 10 numbers having a 7-cycle, so on up till a 10-cycle, respectively. This implies that a random permutation of 10 numbers has a 6-cycle with probability \( \frac{1}{6} \), a random permutation of 10 numbers has a 7-cycle with probability \( \frac{1}{7} \), continuing up till 10-cycle.

It is important to note that the reasoning above does not work for the computation of the number of permutations of 10 numbers having a 5-cycle or shorter because the permutation can have two 5-cycles. We now have

\[
\text{Prob. that a random permutation of 10 numbers contains a cycle of length} \geq 6 = \frac{1}{6} + \frac{1}{7} + \cdots + \frac{1}{10} = \frac{1627}{2520} \approx 0.645635.
\]

\[
\text{Prob. that a random permutation of 10 numbers contains a cycle of length at most 5} = 1 - \text{Prob. that a random permutation of 10 numbers contains a cycle of length} \geq 6
\]

\[
= 1 - \frac{1627}{2520} = \frac{893}{2520} \approx 0.354365.
\]

Therefore, the success probability for the 10 numbers example is more than 35%.
Extending the argument to the current puzzle involving 100 prisoners, we get

\[
\text{Probability that a random permutation of 100 numbers contains a cycle of length } \leq 50
\]

\[
= 1 - \left( \frac{1}{51} + \frac{1}{52} + \cdots + \frac{1}{100} \right)
\]

\[
\approx 0.311828 \approx 31.2\%.
\]

**Success probability \( \approx 31.2\% \).**

[**Generalization**.] Generalizing the problem for \( 2n \) prisoners and \( 2n \) boxes, we have

\[
\text{Success probability } = 1 - \left( \frac{1}{n + 1} + \frac{1}{n + 2} + \cdots + \frac{1}{2n} \right)
\]

\[
= 1 - \sum_{i=n+1}^{2n} \frac{1}{i}
\]

\[
\geq 1 - \int_{n}^{2n} \frac{1}{x} \, dx
\]

\[
= 1 - \left[ \ln(x) + \text{constant} \right]_{n}^{2n} = 1 - \ln 2 \approx 0.306853 \approx 30.7\%.
\]

**Success probability for \( 2n \) prisoners \( \geq 30\% \).**

Figure [61] gives the success probability for \( 2n \) prisoners when \( n \) varies. We see that for \( n \geq 1 \), the success probability decreases from 50% but always remains above 30%.

![Success probability graph](image)

**Figure 61:** Success probability of \( 2n \) prisoners for \( n \in [1, 30] \) for pointer-boxes strategy.

This has a profound implication. It means that even if there are a billion people and a billion boxes, prisoners can win with a probability strictly greater than 30%. It is unbelievable that such a great strategy exists to solve the puzzle.

**Problems**

1. **[Permutation cycle.]** The probability that a random permutation of \( 2n \) numbers
having a $k$-cycle, where $k \in [n + 1, 2n]$, is $1/k$. What is the probability when $k \in [1, n]$?

2. [Generalize.] Find the success probability when we have $n$ prisoners ($n$ can be even or odd) and each of them are allowed to open $k$ boxes where $k \in [1, n]$.

3. [Variant 1.] Let the puzzle be modified such that the prisoners win when none of the prisoners must find their numbers. What is the optimal strategy to solve this puzzle variant?

4. [Variant 2.] Let the puzzle be modified such that the prisoners win if strictly more than half of the prisoners find their numbers. What is the success probability for the pointer-boxes strategy for this puzzle variant?

References

The puzzle was first presented in [Gál and Miltersen, 2007]. Sven Skylum is credited to have developed the pointer boxes strategy. The optimality of the pointer boxes strategy was proved in [Curtin and Warshauer, 2006]. Generalization and a variant of the puzzle are considered in [Goyal and Saks, 2005] and [Avis and Broadbent, 2009], respectively. The problem has been discussed in [Stanley, 2013] too. Books such as [Beeler, 2015], [Brualdi, 2010], and [Bender and Williamson, 2006] give nice introductions to combinatorics.
Electrician and Wires

Problem

A hundred wires go over a very long distance in a straight underground tunnel. The tunnel is so long that it is difficult to determine the two ends of each of the wires. There is an electrician who has a battery and a light bulb. The battery has two electrical contacts (or polarities): + and −. The bulb will light up if two wires connect to the battery with appropriate polarity. The electrician can see the bulb light up even from the opposite side of the tunnel. Since the tunnel is very long, the electrician wants to minimize the number of trips between the two ends of the tunnel.

What strategy should the electrician use to identify the proper ends of the wires and minimizing the total number of trips?

Solution

Let’s understand how exactly a simple circuit is made using a battery, a light bulb and wire(s), as shown in Figure 62. A battery has two polarities: + and −. On the other hand, a bulb has no polarity. A wire is connected from − end of the battery to one of the electrical contacts of the bulb. Another wire is connected from + end of the battery to the remaining electrical contact of the bulb. At this point, electrons from − end of the battery travel through the closed circuit to the + end of the battery. The current in the loop makes the bulb glow.

![Figure 62: A simple circuit.](image)

It is important to observe that there is another type of bulb that has polarity. A polarity-based light bulb such as a diode, e.g. light-emitting diode (LED), has two ends: + and −. A battery can be connected to a polarity-based light bulb if we connect + end of the battery to the + terminal of the light bulb and similarly − end of the battery to the − end of the light bulb. Some of the solutions that we discuss use a polarity-based light bulb.

For simplicity, we assume that the tunnel goes from left to right. Without loss of generality, we assume that the electrician starts from the right end. Let the wires be labeled as \( w_1, w_2, \ldots, w_n \). For the given problem, \( n = 100 \).
Naïve solution (non-optimal)

At the left end of the tunnel, the electrician labels the \( n \) wires as \( w_1, w_2, \ldots, w_n \).

The electrician connects (or shorts) wires \( w_1 \) and \( w_2 \) to form group \( G \leftarrow \text{CONNECT}(w_1, w_2) \). He moves to the right end of the tunnel (trip 1). He identifies the group \( G \) of wires using the battery and the light bulb and labels each of the wires in the group as \( w_1 | w_2 \). He then moves to the left end of the tunnel (trip 2) and disconnects group \( G \) i.e., \( D \rightarrow G \) to form group \( G_n \). Hence, group \( G \) of the tunnel with the battery and a polarity-based light bulb (trip 1). He identifies and connects them to form group \( G \) at the left end of the tunnel, the electrician picks two wires, labels them as \( w_1 \) and \( w_2 \) to form group \( G \leftarrow \text{CONNECT}(w_1, w_2) \). He moves to the right end of the tunnel (trip 1). He identifies the group \( G \) of wires using the battery and the light bulb and labels each of the wires in the group as \( w_3 \) and \( w_4 \). He then moves to the left end of the tunnel (trip 2) and disconnects group \( G \). The electrician connects (or shorts) wires \( w_3 \) and \( w_4 \) to form group \( G \leftarrow \text{CONNECT}(w_3, w_4) \). He moves to the right end of the tunnel (trip 3). He identifies the group \( G \) of wires using the battery and the light bulb and labels each of the wires in the group as \( w_3 | w_4 \). He then moves to the left end of the tunnel (trip 4) and performs \( D \rightarrow G \).

Let \( k = \lfloor n/2 \rfloor - 1 \) when \( n \) is even and \( k = \lfloor n/2 \rfloor \) when \( n \) is odd. The electrician repeats this process a total of \( k \) times. At the end of this process (trip \( 2k \)), he would have identified \( k \) pairs of wires at the right end of the tunnel but he would not have identified the individual wires exactly. It is important to note that if \( n \) is even, the remaining unlabeled pair of wires is labeled as \( w_{n-1} | w_n \). On the other hand, if \( n \) is odd, the remaining unlabeled wire is labeled as \( w_n \).

The electrician now connects wires as: \( G_1 \leftarrow \text{CONNECT}(w_2, w_3), G_2 \leftarrow \text{CONNECT}(w_4, w_5) \), and so on till \( \text{CONNECT}(w_{2k}, w_{2k+1}) \). He moves to the right end of the tunnel (trip \( 2k+1 \)). Consider an example. Suppose the two wires that belong to group \( G_1 \) are labeled \( w_1 | w_2 \) and \( w_3 | w_4 \). Then, he can immediately know that the wires that form this group are \( w_2 \) and \( w_3 \). Similarly, he can identify the wires of all groups \( G_1, G_2, \ldots, G_k \), and the remaining wires using the battery and the light bulb. In this way, the electrician can correctly label all wires on the right end of the tunnel. Note that the battery and the light bulb will be present on the right end of the tunnel throughout the solution.

| Trips for \( n \geq 3 \) wires = \begin{cases} n - 1 & \text{if } n \text{ is even} \\ n & \text{if } n \text{ is odd} \end{cases} | . Trips for 100 wires = 99. |

Zig-zag pairing solution (non-optimal)

At the left end of the tunnel, the electrician picks two wires, labels them as \( w_1 \) and \( w_2 \) and connects them to form group \( G_L \leftarrow \text{CONNECT}(w_1, w_2) \). He moves to the right end of the tunnel with the battery and a polarity-based light bulb (trip 1). He identifies group \( G_L \) of wires and correctly labels the wires in the group \( G_L \) as \( w_1 \) and \( w_2 \).

The electrician now picks two wires, labels them as \( w_3 \) and \( w_4 \) and connects them to form group \( G_R \leftarrow \text{CONNECT}(w_3, w_4) \). He moves to the left end of the tunnel with the battery and the polarity-based light bulb (trip 2) and disconnects group \( G_L \) of wires i.e., \( D \rightarrow G_L \). He then identifies group \( G_R \) of wires and correctly labels the wires in the group \( G_R \) as \( w_3 \) and \( w_4 \).

This process is continued for \( \lfloor n/2 \rfloor \) trips. A pair of wires is correctly labeled in each trip. Hence, \( n \) wires can be correctly labeled with \( \lfloor n/2 \rfloor \) trips. Note that when \( n \) is odd, it is easy to label the only remaining wire.
Trips for \( n \geq 1 \) wires = \( \lceil \frac{n}{2} \rceil \). Trips for 100 wires = 50.

Requirements: A polarity-based light bulb.

**Divide-and-conquer solution (non-optimal)**

The electrician splits the \( n \) wires into two equal (or almost-equal) groups \( G_1 \) and \( G_2 \), and connects (or shorts) wires in \( G_1 \). The wires in \( G_1 \) and \( G_2 \) are labeled as \( G_1 \) and \( G_2 \), respectively. The electrician moves to the right end of the tunnel (trip 1) with a battery and a light bulb. He identifies the group \( G_1 \) of wires using the battery and the light bulb and labels each of the wires in the group as \( G_1 \). The wires that do not belong to group \( G_1 \) belong to group \( G_2 \) and are so labeled.

The electrician splits \( G_1 \) into equal (or almost equal) parts: \( G_{11} \) and \( G_{12} \). Similarly, \( G_2 \) is split into (almost) equal parts: \( G_{21} \) and \( G_{22} \). He shorts the groups \( G_{11} \) and \( G_{21} \) to form group \( G_+ \) i.e., \( G_+ \leftarrow \text{CONNECT}(G_{11}, G_{21}) \). The remaining wires belong to group \( G_- \). He then moves to the left end of the tunnel (trip 2) taking the battery and the light bulb. He first disconnects group \( G_1 \) of wires. Then, he can easily identify the group \( G_+ \) of wires using the battery and the light bulb. The wires that do not belong to \( G_+ \) belongs to \( G_- \). Wires that are labeled as \( G_1 \) and belong to \( G_+ \) (or \( G_- \)) will now be relabeled as \( G_{11} \) (or \( G_{12} \)), respectively. Similarly, the wires that are labeled as \( G_2 \) and belong to \( G_+ \) (or \( G_- \)) will now be relabeled as \( G_{21} \) (or \( G_{22} \)), respectively.

This process is continued. After the 1st trip, the wires are classified into \( 2^1 = 2 \) groups. After the 2nd trip, the wires are classified into \( 2^2 = 4 \) groups. Similarly, after the \( i \)th trip, the wires will be classified into \( 2^i \) groups. Each wire will go into a separate group and hence can be identified uniquely after at most \( \lceil \log_2 n \rceil \) trips.

Trips for \( n \geq 1 \) wires = \( \lceil \log_2 n \rceil \). Trips for 100 wires = 7.

**Improved divide-and-conquer solution (non-optimal)**

The electrician splits the \( n \) wires into three equal (or almost-equal) groups \( G_1 \), \( G_2 \), and \( G_3 \). He connects (or shorts) wires in \( G_1 \) and connects wires in \( G_2 \). The wires in each group are labeled with their group numbers. Groups \( G_1 \) and \( G_2 \) are connected to the positive and negative ends of the battery as shown in Figure 63. The electrician moves to the right end of the tunnel with the polarity-based light bulb (trip 1). He can easily identify the three groups using the polarity-based light bulb and continuity tests. The wires are then labeled with their group numbers.

The electrician splits each group \( G_i \), for \( i \in \{1, 2, 3\} \) into three equal (or almost equal) subgroups: \( G_{i1} \), \( G_{i2} \), and \( G_{i3} \). He shorts the three subgroups \( G_{11} \), \( G_{21} \), and \( G_{31} \) to form group \( G_+ \) i.e., \( G_+ \leftarrow \text{CONNECT}(G_{11}, G_{21}, G_{31}) \). The group \( G_+ \) is connected to the +ve terminal of the light bulb. Similarly, the three subgroups \( G_{12} \), \( G_{22} \), and \( G_{32} \) are connected to form \( G_- \) i.e., \( G_- \leftarrow \text{CONNECT}(G_{12}, G_{22}, G_{32}) \). The group \( G_- \) is connected to the –ve terminal of the light bulb. He then moves to the left end of the tunnel (trip 2).
The electrician disconnects all groups. He can now easily identify the wires that belong to groups $G_+$ and $G_-$ using the battery. Wires that are labeled as $G_i$, for $i = \{1, 2, 3\}$ and belong to $G_+$ (or $G_-$) will be relabeled as $G_{i1}$ (or $G_{i2}$), respectively. Wires that are labeled as $G_i$ and do not belong to either $G_+$ or $G_-$ will be relabeled as $G_{i3}$. In this way, a total of 9 subgroups can be identified in two trips.

This process is continued. After the 1st trip, the wires are classified into $3^1 = 3$ groups. After the 2nd trip, the wires are classified into $3^2 = 9$ groups. Similarly, after the $i$th trip, the wires will be classified into $3^i$ groups. Each wire will go into a separate group and hence can be identified uniquely after at most $\lceil \log_3 n \rceil$ trips.

Trips for $n \geq 1$ wires = $\lceil \log_3 n \rceil$. Trips for 100 wires = 5.

Requirements: A polarity-based light bulb.

**Snake-chain solution (optimal)**

Suppose there are 7 wires, as shown in Figure 64 (left). The electrician connects 3 pairs of wires and leaves an extra wire. The wires are labeled with the pair number. The electrician moves to the right end of the tunnel (trip 1) with a battery and a light bulb.

Figure 64: Snake-chain solution for (left) 7 wires and (right) 8 wires, respectively.
At the right end of the tunnel, the electrician can determine the connected pairs and the extra wire using a battery and a light bulb. The wires are connected as shown in the figure and they are labeled as follows. The extra wire is labeled as 4.1 and the wire connected to it is labeled as 3.2. The other wire of the third pair is labeled as 3.1 and the wire connected to it is labeled as 2.2. This process is continued until all wires are labeled. He then moves to the left end of the tunnel (trip 2) with the battery and the light bulb.

At the left end of the tunnel, the electrician can determine the connected groups using the battery and the light bulb. The extra wire is labeled as 4.1. The wire from pair 3 that is connected to the extra wire is labeled as 3.2. The other wire from pair 3 is labeled as 3.1. The wire from pair 2 that is connected to 3.1 is labeled as 2.2. This process is continued until all wires are labeled.

A similar solution works when we have 8 wires, as shown in Figure 64 (right). In this case, there are two extra/free wires. After two trips, at the left end of the tunnel, when the electrician determines that a free wire forms a closed circuit with a wire belonging to a pair, he labels the free wire as 4.1 and the other connected wire as 3.2. He labels the remaining free wire as 4.2. All other wires can be labeled easily. This solution can be generalized to any number of wires \( n \geq 3 \).

**Trips for \( n \geq 3 \) wires = 2.**

**Triangular number solution (optimal)**

We now describe the most beautiful solution for solving the puzzle. In this solution, each wire will be uniquely identified using a pair of numbers of the form \( a.b \), where \( a \) represents the group number and \( b \) represents the index number in the group. Suppose

\[
[1 + 2 + 3 + \cdots + (k-1) + k] + d = n,
\]

where, \( k \) is a natural number and \( d \in [0,k] \). Numbers of the form \( 1 + 2 + 3 + \cdots + m \) for some natural number \( m \) are called **triangular numbers**. The electrician splits the \( n \) wires into \( k + 1 \) groups: \( G_1, G_2, \ldots, G_{k+1} \). The group \( G_1 \) contains 1 wire, \( G_2 \) contains 2 wires, so on until \( G_k \) contains \( k \) wires. The last group \( G_{k+1} \) contains \( d \) wires, where \( d \leq k \). The wires in each of these groups is connected/shorted. Each wire is labeled with its group number. The electrician then moves to the right end of the tunnel (trip 1).

The electrician can determine the groups \( G_1, G_2, \ldots, G_{k+1} \) using a battery and a light bulb. A pictorial diagram of the solution for some wires is shown in Figure 65. The electrician labels the only wire in group \( G_1 \) as \( 1.1 \), the wires in group \( G_2 \) as \( 2.1, 2.2 \), so on, until the wires in group \( G_k \) are labeled as \( k.1, k.2, \ldots, k.k \). Finally, the wires in \( G_{k+1} \) are labeled as \( (k+1).1, (k+1).2, \ldots, (k+1).d \). There are multiple strategies to connect wires at the left end of the tunnel. We will describe one such strategy. First, the electrician connects wires with index 1 from groups \( G_1 \) to \( G_k \), leaving out the index 1 wire from group \( G_{k+1} \). This new group is called \( G_V \). He then connects wires of index \( i \in [2,k] \) that belong to groups \( G_i \) to \( G_{k+1} \). These new groups are called \( G_{V'} \), where
Figure 65: Triangular number solution for 9 wires.

\[ i \in [2, k]. \] This strategy works for any \( n \geq 3 \) number of wires. The electrician then moves to the left end of the tunnel (trip 2).

Consider the example given in the figure. Note that there are two groups, each of size 3. Still, electrician can uniquely identify the individual wires using continuity tests.

Trips for \( n \geq 3 \) wires = 2.

This solution is better than the snake-chain solution in the worst-case number of tests we need to perform to uniquely identify all the wires.

References

The problem, the snake-chain solution, and the triangular numbers solution were discussed in [Gardner, 1988].
Monty Hall Problem

Problem

Suppose you are on a game show and you are shown three doors: behind one of the
doors is a car and behind the other two doors are goats. The car is very special. It has
the ability to travel on road, in the air, and in the water. You desperately want to win
the car.

The game host asks you to choose one of the three doors. Once you have chosen a
door, the host opens another door revealing a goat. The host knows what is behind ev-
ery door and always chooses to open a door that conceals a goat. If both the remaining
doors have goats behind them, then one of the doors is opened at random. The host
then asks you whether you want to stick to your original chosen door or switch to the
remaining unopened door. You win whatever is behind the door you choose finally.
Should you stick to your original chosen door or switch to the remaining unopened
door?

Solution

The puzzle is based on an American television game show called *Let’s Make a Deal* and
is named after its creator and host Monty Hall. This is one of the most famous prob-
ability problems that has confused hundreds of mathematicians and statisticians.
Even Paul Erdos, who is considered one of the most prolific mathematicians of the
20th century, got the problem wrong and refused to accept the correct solution for
some time. So, let’s be very careful in our probabilistic journey.

It is important to note that the host never opens the door that hides the car. The
host always opens a goat-concealing door from the two remaining doors.

Common solution (incorrect)

Most people would like to stick to the original chosen door. Because they think that
it is equally likely the car is behind either of the two unopened doors. However, this
answer is incorrect. In further sections, we will look at several solutions which prove
that it is better to switch than to stick.

Logic solution

In this approach, we show through the fundamental meaning of probability why
switching is better than sticking. However, this approach is not mathematically rig-
orous.

Let’s suppose that you initially choose door 1. The probability of the car being
behind door 1 is \( \frac{1}{3} \) and the probability of the car being behind door 2 or door 3 is \( \frac{2}{3} \),
Suppose the host now opens door 3 that has a goat. The important observation here is that it did not happen by chance that the goat was behind the third door. The host knew very well that there was a goat behind the third door. Also, the host would never open door 1 knowing that you initially chose door 1. With the opening of door 3 which is a car-hiding door, the car-winning probabilities of the remaining doors change.

![Figure 66: The car-winning probability when you initially choose door 1. Left: Before the host opens door 3. Right: After the host opens door 3.](image)

A majority of people believe that the new car-winning probability of door 1 and door 2 increases to $\frac{1}{2}$ each. This is incorrect. The reasoning is as follows. Consider Figure 66. Given that you initially chose door 1, the host will never ever ever ever open door 1. The host only opens a goat-hiding door from one of doors 2 or 3. Hence, door 1 can be considered as an isolated unit and [door 2, door 3] can be considered as one big combined unit. When the goat-hiding door 3 is opened, the car-winning probability of door 1 is unaffected as it is an isolated unit and remains $\frac{1}{3}$. However, the car-winning probability of door 2 increases from $\frac{1}{3}$ to $\frac{2}{3}$ as the winning probability of [door 2, door 3] combined unit remains the same as $\frac{2}{3}$. We can show similar analysis for other cases of chosen and host-opened doors.

Switching is better than sticking because the car-winning probability with switching is $\frac{2}{3}$ and that with sticking is $\frac{1}{3}$.

There is another beautiful interpretation to the solution. Let’s suppose that you choose two doors from which the host opens one with a goat. It is easy to write an analysis to show that in such a situation, sticking is better than switching. Now, when you have to choose only one door, the answer gets reversed and in this scenario, switching is better than sticking.

**Exhaustive possibilities solution**

In this approach, we list all possibilities of decisions and results and then compute the probability from them. However, this approach is not mathematically rigorous.

Let’s suppose that you initially choose door 1. Table 39 summarizes the car-winning positions for all possible combinations of the host’s decisions. We see that the winning probability for sticking to chosen door is $\frac{1}{3}$, whereas the winning probability for switching door is $\frac{2}{3}$. Hence, switching is better than sticking.
### Decision tree solution

In this approach, we use a decision tree to showcase the conditional probability of the occurrence of one event given that another event has occurred. A *decision tree* is a fancy name for a tree that branches based on decision possibilities.

![Decision tree diagram]

Figure 67: Different possibilities and their probabilities through a decision tree when you initially choose door 1. Here, $P$ denotes the total probability.

<table>
<thead>
<tr>
<th>Car location</th>
<th>Host opens</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Door 1</td>
<td>Door 2</td>
<td>$\frac{1}{6}$</td>
</tr>
<tr>
<td>Door 1</td>
<td>Door 3</td>
<td>$\frac{1}{3}$</td>
</tr>
<tr>
<td>Door 2</td>
<td>Door 3</td>
<td>$\frac{1}{6}$</td>
</tr>
<tr>
<td>Door 3</td>
<td>Door 2</td>
<td>$\frac{1}{3}$</td>
</tr>
</tbody>
</table>

Table 40: Probabilities of possibilities when you initially choose door 1.

Let’s suppose that you initially choose door 1. Then, the decision tree from Figure 67 and Table 40 showcases the possibilities of all decisions taken and their probabilities. The car can be behind any door with probability of $\frac{1}{3}$. If the car is behind door 1, then the host can open either door 2 or door 3 with a total probability of $\frac{1}{6}$. On the other hand, if the car is behind door 2 or door 3, then the host can open door 3 or door 2, respectively, each with total probability of $\frac{1}{3}$.

Let’s compute the conditional probability of winning the car when you initially choose door 1 and the host opens, say, door 3.

Winning probability on switching = \[
\frac{\text{Host opens door 3 and the car is behind door 2}}{\text{Host opens door 3}} = \frac{\frac{1}{3} \times \frac{1}{3}}{\frac{1}{3} + \frac{1}{6}} = \frac{2}{3}
\]
Similarly, we can extend this argument to all other decision possibilities and arrive at the same result. Thus, switching is better than sticking.

Bayes’ theorem solution

In this approach, we use Bayes’ theorem to describe the conditional probability \( P(X_1, X_2, \ldots | Y_1, Y_2, \ldots) \) which is used to compute the probability of events \( X_1, X_2, \ldots \) occurring given that the other events \( Y_1, Y_2, \ldots \) have already occurred.

Suppose you choose door 1 and the host opens door 3. We want to compute the winning probability for switching. That is, we want to compute the probability that the car hides behind door 2. Let \( C_1, C_2, \) and \( C_3 \) denote the events that the car is present behind doors 1, 2, and 3, respectively. Let \( Y_1 \) denote the event that you initially choose door 1. Let \( H_3 \) denote the event that the host opens door 3. Then, we would like to compute the probability that the car hides behind door 2 i.e., \( P(C_2|Y_1, H_3) \).

The probability of the car being present behind any door is equally likely. So, \( P(C_1) = P(C_2) = P(C_3) = \frac{1}{3} \). Also, \( P(H_3|C_1, Y_1) = \frac{1}{2} \) as there are two equally likely choices (door 2 or 3) for the host to open; \( P(H_3|C_2, Y_1) = 1 \) as there is exactly one choice (door 3) for the host to open; and \( P(H_3|C_3, Y_1) = 0 \) as the host cannot open the car-concealing door.

\[
P(C_2|H_3, Y_1) = \frac{P(H_3|Y_1, C_2)P(Y_1, C_2)}{P(H_3, Y_1)} \quad (\because P(A|B, C)P(B, C) = P(B, C|A)P(A))
\]

\[
= \left( \frac{P(H_3|Y_1, C_2)P(Y_1, C_2)}{P(C_2)} \right) \frac{P(C_2)}{P(H_3, Y_1)} \quad (\because P(A, B|C)P(C) = P(A|B, C)P(B, C))
\]

\[
= \frac{P(Y_1|C_2)P(C_2)}{P(H_3|Y_1)} \quad (\because P(A, B|C)P(C) = P(A|B, C)P(B))
\]

\[
= \frac{\frac{1}{3} \cdot \frac{1}{3}}{\frac{1}{2} \cdot \frac{1}{3}} = \frac{2}{3}
\]

The winning probability for switching door is \( P(C_2|H_3, Y_1) = \frac{2}{3} \). So, switching is better than sticking.

Programming solution

In this approach, we simulate the game through programming. Though this method is not a mathematically rigorous proof, it gives a very good empirical evidence for the result.

The \textsc{MontyHallSimulation} algorithm simulates the Monty Hall game for 1 million trials. Figure 68 shows the simulation plot for 1,000 trials. We see that the number of wins from switching is almost double the number of wins from sticking for a fixed number of trials as shown by blue circle and red square points, respectively. The winning probability, computed as the number of wins over number of trials, for
MontyHallSimulation()

1. trials ← 1 million; wins ← 0
2. for i ← 1 to trials do
3. ⟨d₁, d₂, d₃⟩ ← RANDOMSHUFFLE([car, goat, goat]) // random permutation
4. c ← RANDOM(1, 2, 3) // chosen door
5. h ← HOSTOPENS(⟨d₁, d₂, d₃⟩, c) // door opened by the host
6. r ← {1, 2, 3} − {c, h} // remaining door
7. if dᵣ = car then wins ← wins + 1 // winning the car
8. print switching the door leads to count wins in trials trials
9. print winning probability from switching is \((100 \cdot \text{trials} \div \text{count})\) percent

HOSTOPENS(⟨d₁, d₂, d₃⟩, c)

1. option₁ ← (c + 1) mod 3; option₂ ← (c + 2) mod 3 // two door options
2. if dₖ = car then
3. return RANDOM(option₁, option₂) // chosen door has the car
4. else if d₉ = car then
5. return option₂ // option₁ door has the car
6. else if d₉ = car then
7. return option₁ // option₂ door has the car

switching the door is nearly \(2/3\) from the plot. Hence, switching is better than sticking.

Monty Hall simulation

![Monty Hall simulation graph](image)

Figure 68: The simulation plot for the Monty Hall game. The blue circle and red square denote the number of car-wins from switching and sticking, respectively, for a particular number of trials.
Gnedin’s game theory solution

In this approach, we use game theory for solving the puzzle. *Game theory* deals with the study of mathematical models for analyzing situations of conflict and/or cooperation between optimizing individuals who make rational decisions.

**[Game.]** The game can be modeled as a 2-player game played between you and the host. Your aim is to win the car. Let the door numbers be 1, 2, and 3. Then, the four moves of the game represented by the 4-tuple \([a, b, c, d]\) are:

1. The host chooses a door \(a \in \{1, 2, 3\}\) to hide the car.
2. You choose a door \(b \in \{1, 2, 3\}\). You do not know \(a\) but the host knows \(b\).
3. The host chooses a door different from the car-hiding door and the door chosen by you. The host offers you to either stick to your original chosen door \(b\) or switch to an unopened door \(c\).
4. You finally choose door \(d\) from doors \(b\) or \(c\). You win the car if door \(d\) hides the car (i.e. \(d = a\)). Otherwise, you win nothing.

For example, the 4-tuple \([2, 2, 1, 1]\) represents a game instance where the car is hidden behind door 2, you choose door 2, the host opens door 1, and you choose to switch to door 1. In this instance, you don’t win the car.

**[Game tree and strategies.]** A game tree is a directed graph representation of a game in which the nodes represent the positions and the edges represent the moves. Figure 69 shows the game tree for the Monty Hall problem. A player’s strategy is defined as a set of rules that defines the next move of the player. The strategies of the host and you are given in Table 41.

<table>
<thead>
<tr>
<th>Host’s strategy</th>
<th>Your strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st move</td>
<td>A random value for (b \in {1, 2, 3}) is chosen.</td>
</tr>
<tr>
<td>2nd move</td>
<td>The value of (d) depends on (b) and (c) specifying an action whether to stick or switch:</td>
</tr>
<tr>
<td></td>
<td>(c(a, b) = a) if (a \neq b)</td>
</tr>
<tr>
<td></td>
<td>(c(a, b) = a) if (a = b)</td>
</tr>
</tbody>
</table>

Table 41: Host’s strategy and your strategy.

The host’s strategy can be uniquely encoded with a pair of numbers. Suppose the host’s strategy is 12. It means that door 1 hides the car (i.e. \(a = 1\)) and door 2 is the switch door offered by the host when you initially choose door 1 (i.e. \(c(1,1) = 2\)). This implies \(c(1, 2) = 1\) and \(c(1, 3) = 1\).

Similarly, your strategy can be uniquely encoded with a 3-tuple. Suppose your strategy is 2sh. It means that you initially select door 2 (i.e. \(b = 2\)) and then in the final move choose to \(d(2, 1) = \text{switch}\) or \(d(2, 3) = \text{hold}\) (i.e. \(d(2, 1) = 1\) and \(d(2, 3) = 2\)) depending on constraints. In this example, the letter \(s\) denotes the switch action on the lower value of door number except 2, which is 1 and \(h\) denotes the hold (or stick or stay) action on the larger value of door number except 2, which is 3.

The host’s and your strategies are shown below:

Host’s strategies = \{12, 13, 21, 22, 31, 32\}
Your strategies = \{ss, sh, hs, hh, 2ss, 2sh, 2hs, 2hh, 3ss, 3sh, 3hs, 3hh\}

[Game matrix.] We can represent our 2-player game using a game matrix $M$ as shown in Table 42. In the game matrix, also called the payoff matrix, the rows and
columns represent the strategies of the host (or player 1) and you (or player 2), respectively. The matrix cell \( M(i, j) \) represents both the game instance and your payoff if the host follows strategy \( i \) and you follow strategy \( j \). The payoff is 1 if you win the prize and it is 0 if you don’t. For example, \( M(12, 2sh) = 1 \) represents the winning game instance when the host plays strategy 12 and you play strategy 2sh.

<table>
<thead>
<tr>
<th>Host's strategies</th>
<th>Always-hold</th>
<th>Your strategies</th>
<th>Always-switch</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1hh 2hh 3hh</td>
<td>Context-dependent</td>
<td>1ss 2ss 3ss</td>
</tr>
<tr>
<td>12</td>
<td>1 0 0</td>
<td>1 0 0 0 1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>13</td>
<td>1 0 0</td>
<td>0 0 0 1 1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>21</td>
<td>0 1 0</td>
<td>0 1 1 1 0 0</td>
<td>1 0 1</td>
</tr>
<tr>
<td>22</td>
<td>0 1 0</td>
<td>0 0 1 1 0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>31</td>
<td>0 0 1</td>
<td>1 1 1 0 0 1</td>
<td>1 1 0</td>
</tr>
<tr>
<td>32</td>
<td>0 0 1</td>
<td>1 1 0 0 1 1</td>
<td>1 1 0</td>
</tr>
</tbody>
</table>

Table 42: Your game/payoff matrix.

Please note that the number of winning or losing positions in the game tree as shown in Figure 69 is 24, whereas that in the game matrix is \( 6 \times 12 = 72 \). This is because a single leaf node of the game tree might be equivalent to multiple cells in the game matrix. For example, \([1, 2, 1, 1] \) from the game tree is simultaneously equivalent to \( M(12, 2ss), M(12, 2sh), M(13, 2ss), \text{ and } M(13, 2sh) \).

A basic or fundamental strategy is called a pure strategy. A complex collection of pure strategies that are chosen with different probabilistic proportions is called a mixed strategy. For example, 2sh is a pure strategy. On the other hand, the mixture of 2sh and 3ss strategies in probabilistic proportions of \( \frac{1}{4} \) and \( \frac{3}{4} \), respectively, is a mixed strategy.

\{1hh,2hh,3hh\} strategies represent always-hold strategies. Similarly, \{1ss,2ss,3ss\} strategies represent always-switch strategies and \{1hs,2hs,3hs,1sh,2sh,3sh\} strategies represent context-dependent strategies.

**[Payoffs for always-hold and always-switch strategies.]** Let’s suppose that the host follows a mixed strategy that combines the six pure strategies \{12, 13, 21, 22, 31, 32\} with equal probabilistic proportions (i.e. \( \frac{1}{6} \)). Let’s also suppose that you follow a mixed always-hold strategy that combines \{1hh,2hh,3hh\} strategies with equal probabilistic proportions (i.e. \( \frac{1}{3} \)). Then, Table 43 shows the mixed always-hold strategy and its payoff. Each cell value is obtained by multiplying the probabilities of its row and its column. The average overall payoff for the mixed strategy is the sum of the payoffs for all cells in the table. That is, your winning probability for the always-hold (or stick) strategy is \( 6 \times \frac{1}{18} = \frac{1}{3} \).

Similarly, the winning probability for the mixed always-switch strategy, as shown in Table 44, can be computed to be \( 12 \times \frac{1}{18} = \frac{2}{3} \). Finally, the winning probability for the mixed context-dependent strategies is \( 9 \times \frac{1}{18} = \frac{1}{2} \), which is unimportant to our current puzzle.

Thus, switching is better than sticking.
Problems

Solve the following variants of the original Monty Hall problem. The rules of the game follow the original game unless stated otherwise.

1. The host does not know what’s behind each of the doors. Suppose the host opens one of the two remaining doors randomly and finds a goat behind it. Should you stick or switch?

References

Similar versions of the problem have appeared before: Bertrand paradox in [Bertrand, 1889], three prisoners problem in [Gardner, 1959], and the three shell game in [Gardner, 1982].

The Monty Hall problem gets its name from a television game show Let’s Make a Deal hosted by Monty Hall. The problem was first posed in [Selvin, 1975]. It was posed again in [Savant, 1996], a reader of the Parade magazine. The problem was correctly answered by Marilyn vos Savant. vos Savant received nearly 10,000 letters in response to her answer. Among the responses that disagreed with her answer were many Ph.D. holders. This clearly demonstrates the unimaginably high counterintuitive-level of the puzzle.

Different solutions to the puzzle are summarized in Wikipedia. The game theory solution is presented in [Gnedin, 2011]. A good book on game theory is [Ferguson, 2014]. An decent book that wholly deals with the Monty Hall puzzle and its nearly 15 variants is [Rosenhouse, 2009]. Many nice probability puzzles can be found in [Mosteller, 1987]. A good text on probability theory with intuitions and applications is [Feller, 1968], [Feller, 1970]. Other good books on probability theory are [Bertsekas and Tsitsiklis, 2002], [Dekking et al., 2005], and [Tijms, 2012].
Poisoned Wine

Problem

A good king has a cellar with a thousand bottles of wine. A bad minister who had a deal with the neighboring king decides to kill the good king. He sends his most trustworthy servant to poison the wine bottles. The poison is so strong that even if diluted an unlimited number of times it still can kill a person. The king’s guards catch and kill the servant after the servant has poisoned only one bottle. The guards do know that exactly one bottle is poisoned but do not know which one. The guards also know that it takes at most a day for the poison to have its effect. The king decides to get some rats to taste the wines and identify the poisoned bottle.

What strategy can the king use to identify the poisoned bottle?

Solution

This puzzle belongs to a domain in mathematics called group testing. The problems in group testing have a common structure. We are given a set of items that is known to contain some defective items. Any subset of the items can be tested with a negative (i.e., all items are good) or positive (i.e., at least one of the items is defective) outcome. The aim is to optimize a specific parameter such as the number of tests and time taken.

Some example problems that belong to the domain of group testing are: [Blood testing.] Blood testing among a group of people for potential disease-causing organisms or chemicals (e.g.: AIDS). [Coin testing.] Identifying counterfeit coins (having different weight(s)) among a set of coins. [Bulb testing.] Identifying defective light bulbs. [Gas testing.] Identifying gas leakage points in a gas pipe.

Let’s formalize our problem. Let a strategy to solve our original problem of identifying the poisoned bottle(s) among a set of wine bottles be denoted by

\[ \text{Strategy}(n, d) = \{(r, k), t\} \]

where, \( n \) = number of wine bottles, \( d \in [0, n] \) = number of poisoned wine bottles, \( r \in [1, n] \) = total number of rats used in the tests, \( k \in [1, r] \) = maximum number of rats that might get killed, and \( t \) = time taken in days.

1 poisoned bottle

[Basic strategies.] Two simple strategies to identify a poisoned bottle among 1000 bottles are: (i) Strategy \((999, 1), 1\). We can use 999 rats simultaneously such that they drink wine from the first 999 different bottles. If a rat dies after a day, then we know the poisoned bottle. If none of the rats die, then the last bottle is poisoned. (ii)
A single rat is made to sip wine from the first 999 bottles on consecutive 999 days. If the rat dies after a few days, then we know the poisoned bottle. If the rat survives, then the last bottle is poisoned.

**Standard solution.** We use 10 rats to identify a poisoned bottle among 1000 bottles in 1 day. At most 9 rats might die. We number the wine bottles from 0 to 999. As $2^9 < 1000 \leq 2^{10}$ we need (according to information theory) and hence use a minimum of 10 digits to represent every wine bottle number in the form $b_9b_8\ldots b_0$. For example, bottle 0 is represented as 0000000000, bottle 1 is represented as 0000000001, bottle 999 is represented as 1111100111, and so on. Each bit corresponds to a rat. We number the rats from 0 to 9. The $i$th rat is made to sip wine from all bottles whose $i$th bit is set to 1, where $i \in [0, 9]$. In simple words, rat 0 is made to sip wine from all bottles whose rightmost bit ($b_0$) is set to 1 i.e., all odd-numbered bottles. Rat 1 is made to sip wine from all bottles whose second rightmost bit ($b_1$) is set to 1. And so on till rat 9 is made to sip wine from all bottles whose leftmost bit ($b_9$) is set to 1 i.e., bottles numbered from 512 to 999. All rats can be simultaneously made to drink wine so that the poisoned bottle can be identified in 1 day. For example, if rats 0, 1, 2, 4, 8, and 9 die, then the poisoned bottle number is $2^0 + 2^1 + 2^2 + 2^4 + 2^8 + 2^9 = 791$ or 1100010111.

Table 45 shows which rats sip from which bottles. In general by listing which rats die, making a binary number out of them, and converting to decimal, the poisoned bottle can be identified in a single day.

<table>
<thead>
<tr>
<th>Bottle</th>
<th>Rat</th>
<th>Decimal</th>
<th>Binary</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0000000000</td>
<td>× × × × × × × × ×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0000000001</td>
<td>× × × × × × × × ×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0000000010</td>
<td>× × × × × × × × ×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0000000011</td>
<td>× × × × × × × × ×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>791</td>
<td>791</td>
<td>1100010111</td>
<td>✓ ✓ × × × ✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>999</td>
<td>999</td>
<td>1111100111</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1023</td>
<td>1023</td>
<td>1111111111</td>
<td>✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 45: Identify the poisoned bottle among 1000. A set bit or ✓ represents that the corresponding rat tastes the wine. 1024 bottles are shown for completeness.

If there are exactly 1024 bottles, in the worst case all 10 rats die. If there are less than 1024 bottles and $\geq 512$ bottles, in the worst case 9 out of 10 rats die. This is because a maximum of only 9 bits can be set if the number of bottles are in the range $[512, 1023]$. Therefore,

$$
\text{Strategy}(1000, 1) = \{(1001, 1), (999, 1), (1, 1), (1, 1)\}
$$

$$
\text{Strategy}(n, 1) = \{(n-1, 1), (1, 1), (1, 1)\}, (\lfloor \log_2 n \rfloor, \lfloor \log_2 n \rfloor, 1)\}
$$

Suppose that it is equally likely for any bottle of wine to be poisoned, we want to
sort in decreasing order the rats according to the probability of their survival. We label the wine bottles from 0 to \(n - 1\). Table 45 shows the sipping of 10 rats. Due to the structure of the binary code, the survival probability of rat \(i\) is greater than or equal to that of \(j\) where \(i \leq j\). Let \(p_i\) be the probability of survival of rat \(i\). Then \(p_1 \geq p_2 \geq \cdots \geq p_{\lfloor \log_2 n \rfloor}\). It is left to the reader to find the exact survival probabilities of all the rats.

[Generalized binary search (non-optimal).] We have \(n\) bottles of wine and \(d = 1\) of them is poisoned. We use the core idea of binary representation for \(n\) bottles and testing them out. In this method, every bottle number is represented in the binary system, where, each digit is either 0 or 1. Each digit position corresponds to a rat. If a rat value is 0, then the rat does not sip wine. If a rat value is 1, then the rat sips wine immediately. If the bottle from which the rat sips wine is poisoned, then the rat dies after a day.

We want to minimize the number of days \(t\) given \(r\) rats allowing a maximum of \(k\) kills or deaths. Clearly there exists a tradeoff between \(n\), \(r\), \(k\), and \(t\) and we would like to find this tradeoff. First, we find the maximum number of bottles that can be tested in a single day using \(r\) rats and at most \(k\) deaths. Second, we find the minimum number of days to test all \(n\) bottles.

Let \(\binom{a}{b}\) represent the combination or selection of \(a\) elements taking \(b\) at a time. Then,

- Max. #bottles from which exactly 1 rat sips wine in 1 day = \(\binom{r}{1}\)
- Max. #bottles from which exactly \(i \leq k\) rats sip wine in 1 day = \(\binom{r}{i}\)
- Max. #bottles from which exactly \(k\) rats sip wine in 1 day = \(\binom{r}{k}\)
- Max. #bottles from which at least 1 rat sips wine in 1 day = \(\binom{r}{1} + \binom{r}{2} + \cdots + \binom{r}{k}\)
- Max. #bottles from which at least 1 rat sips wine = \(n - 1\)

Max. #days to identify the poisoned bottle in \(n\) bottles =\[
\left\lfloor \frac{(n-1)}{\binom{r}{1} + \binom{r}{2} + \cdots + \binom{r}{k}} \right\rfloor
\]

Note that no more than \(k\) rats can sip wine from any single bottle because the maximum number of deaths allowed is \(k\). We can identify a poisoned bottle (if it exists) among the first \(\binom{r}{1} + \binom{r}{2} + \cdots + \binom{r}{k}\) bottles after the first day using at most \(r\) rats and allowing a maximum of \(k\) deaths. If the first \(\binom{r}{1} + \binom{r}{2} + \cdots + \binom{r}{k}\) doesn’t contain a poisoned bottle, then we can identify a poisoned bottle (if it exists) among the second \(\binom{r}{1} + \binom{r}{2} + \cdots + \binom{r}{k}\) bottles after the second day using at most \(r\) rats and allowing a maximum of \(k\) deaths. The process is continued until we cover the first \(n - 1\) bottles. If none of the rats die during the testing of the first \(n - 1\) bottles, then the last bottle is poisoned. Thus, we have:

\[
\text{#Days-minimizing strategy}(n, 1) = \left\langle (r, k), \left\lfloor \frac{(n-1)}{\binom{r}{1} + \binom{r}{2} + \cdots + \binom{r}{k}} \right\rfloor \right\rangle
\]

We can use the core ideas explained above to obtain a #deaths-minimizing strategy when \(r\) and \(t\) are fixed; and a #rats-minimizing strategy when \(k\) and \(t\) are fixed.

Let’s consider a special case when \(k = r\). When \(k = r\), the sum \(\sum_{i=0}^{k} \binom{r}{i}\) reduces to
2^r - 1. Then, we have

\[
\text{Strategy}(n, 1) = \langle (r, r), \left\lfloor \frac{(n-1)}{2^r-1} \right\rfloor \rangle.
\]

\[
\text{Strategy}(1000, 1) = \langle (7, 6), 8 \rangle.
\]

Suppose we are given \( r \) and \( k \). Suppose rats numbered \( a_1, a_2, \ldots, a_m \) die after \( \ell \) days, then the poisoned bottle number (\( \in [0, n - 1] \)) is

\[
\text{Poisoned bottle} = (\ell - 1)(^rC_1 + ^rC_2 + \cdots + ^rC_k) + \left( 2^{a_1} + 2^{a_2} + \cdots + 2^{a_m} \right)
\]

Though the ideas we have used in this section are good, they do not lead to the optimal solution. We can indeed increase the number of bottles tested each day using \((t + 1)\)-ary search algorithm thereby reducing the total number of days to identify the poisoned bottle.

[\text{Gorlin’s } (t + 1)\text{-ary search (optimal).}] In this algorithm, every bottle number is represented in base-\((t + 1)\) system with \( \lceil \log_{t+1} n \rceil \) digits, where, each digit is in the range \([0, t]\). Each digit position corresponds to a rat and the digit value corresponds to the number of days after which the rat dies, if the bottle is poisoned. If a rat value is 0, then the rat does not sip wine. If a rat value is a positive integer \( \ell \leq t \), then the rat sips wine after \( \ell - 1 \) days. If the bottle from which the rat sips wine is poisoned, then the rat dies after \( \ell \) days.

We want to test \( n \) bottles for a poisoned wine bottle using \( r \) rats, with at most \( k \) rats dying and minimizing the number of days. The core idea is to find the maximum number of bottles that can be tested using \( r \) rats, with at most \( k \) dying, in \( t \) days as a function of \( t \). Then, choose the minimum value of \( t \) such the maximum number of bottles that can be tested is greater than or equal to \( n \).

Maximum number of wine bottles that can be tested with \( r \) rats, with at most \( k \) dying, in \( t \) days is same as the number of distinct \( r \)-digit numbers (possibly with leading zeros) in base-(\( t + 1 \)) number system such that at most \( k \) digits are non-zero. Then,

Max. #bottles that can be tested with exactly 0 rat dying = \(^rC_0 \cdot t^0\)

Max. #bottles that can be tested with exactly 1 rat dying = \(^rC_1 \cdot t^1\)

Max. #bottles that can be tested with exactly \( i \leq k \) rats dying = \(^rC_i \cdot t^i\)

Max. #bottles that can be tested with exactly \( k \) rats dying = \(^rC_k \cdot t^k\)

Max. #bottles that can be tested with at most \( k \) rats dying = \(^rC_0 \cdot t^0 + ^rC_1 \cdot t^1 + \cdots + ^rC_k \cdot t^k\)

Min. #days to test \( n \) bottles with at most \( k \) rats dying = \( \min_t \left\{ \sum_{i=0}^{k} ^rC_i \times t^i \geq n \right\} \)

\[
\text{#Days-minimizing strategy}(n, 1) = \langle (r, k), \min_t \left\{ \left( \sum_{i=0}^{k} ^rC_i \times t^i \right) \geq n \right\} \rangle
\]

We can use the core ideas explained above to obtain a #deaths-minimizing strategy when \( r \) and \( t \) are fixed; and a #rats-minimizing strategy when \( k \) and \( t \) are fixed.
Let's consider a special case when \( k = r \). When \( k = r \), the sum \( \sum_{i=0}^{k} \binom{r}{i} t^i \) reduces to \( (t+1)^r \). Table in Figure 70 (left) illustrates multiple strategies with different values of \( t \) and \( r \) for \( n = 1000 \).

<table>
<thead>
<tr>
<th>( t+1 )</th>
<th>( \log_{5+1} 10^3 )</th>
<th>Condition</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>( 2^3 &lt; 1000 \leq 2^{10} )</td>
<td>\langle (10, 9), 1 \rangle</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>( 3^3 &lt; 1000 \leq 3^7 )</td>
<td>\langle (7, 6), 2 \rangle</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>( 4^4 &lt; 1000 \leq 4^5 )</td>
<td>\langle (5, 4), 3 \rangle</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>( 5^4 &lt; 1000 \leq 5^5 )</td>
<td>\langle (5, 4), 4 \rangle</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>( 6^3 &lt; 1000 \leq 6^4 )</td>
<td>\langle (4, 3), 5 \rangle</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>( 7^3 &lt; 1000 \leq 7^4 )</td>
<td>\langle (4, 3), 6 \rangle</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>( 8^3 &lt; 1000 \leq 8^4 )</td>
<td>\langle (4, 3), 7 \rangle</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>( 9^3 &lt; 1000 \leq 9^4 )</td>
<td>\langle (4, 3), 8 \rangle</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>( 10^3 &lt; 1000 \leq 10^3 )</td>
<td>\langle (3, 3), 9 \rangle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bottle</th>
<th>Rat</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0 )</td>
<td>\langle 0, 0, 0, 0, 0, 0 \rangle</td>
</tr>
<tr>
<td>( 1 )</td>
<td>\langle 0, 0, 0, 0, 0, 1 \rangle</td>
</tr>
<tr>
<td>( 2 )</td>
<td>\langle 0, 0, 0, 0, 0, 2 \rangle</td>
</tr>
<tr>
<td>( 3 )</td>
<td>\langle 0, 0, 0, 0, 0, 3 \rangle</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>\ldots</td>
</tr>
<tr>
<td>( 791 )</td>
<td>\langle 1, 0, 0, 2, 0, 2 \rangle</td>
</tr>
<tr>
<td>( 999 )</td>
<td>\langle 1, 1, 0, 1, 0, 0 \rangle</td>
</tr>
</tbody>
</table>

Figure 70: Left: Strategies to identify a poisoned wine among 1000 bottles for different base-\( (t+1) \) systems. Right: Strategy to identify a poisoned wine among 1000 bottles in \( t = 2 \) days. If a rat value is 0, then the rat does not sip. If a rat value is 1, then the rat sips immediately and might die in 1 day. If the value is 2, the rat sips after 1 day and might die in 2 days.

Using binary search algorithm, we can find a poisoned bottle among \( n \) bottles in a single day using at most \( \lceil \log_2 n \rceil \) rats. Using \( (t+1) \)-ary search algorithm, we can find a poisoned bottle among \( n \) bottles in \( t \) days using at most \( \lceil \log_{t+1} n \rceil \) rats. Table in Figure 70 (right) shows a strategy using base-3 representation to identify a poisoned wine in 2 days among 1000 bottles.

| \( \text{Strategy}(n, 1) = \langle [\lceil \log_{t+1} n \rceil], [\lceil \log_{t+1} n \rceil] \rangle, t \rangle \). \( \text{Strategy}(1000, 1) = \langle (7, 6), 2 \rangle \) |

Observe the improvement of \( \langle (7, 6), 2 \rangle \) strategy using \( (t+1) \)-search method compared with the \( \langle (7, 6), 8 \rangle \) strategy using the generalized binary search algorithm.

We are given \( r \) and \( t \). Suppose rats numbered \( a_1, a_2, \ldots, a_m \) die after \( \ell_1, \ell_2, \ldots, \ell_m \), respectively, then the poisoned bottle number (\( \epsilon_1, n - 1 \)) is

\[
\text{Poisoned bottle} = \ell_1 \times t^{a_1} + \ell_2 \times t^{a_2} + \cdots + \ell_m \times t^{a_m}
\]

For example, consider \( n = 1000, d = 1, r = 7, \) and \( t = 2 \). Suppose rat 6 dies after 1 day and rats 3, 1, and 0 die after 2 days, then the poisoned bottle number is \( 1 \times 3^6 + 2 \times 3^3 + 2 \times 3^1 + 2 \times 3^0 = 791 \), as shown in Figure 70 (right).

\( d \) poisoned bottles

Let \( s \) denote the number of tests required to identify the poisoned bottles. Let a strategy be denoted by

\[
\text{Strategy}(n, d) = \langle (r, k), (s, t) \rangle
\]

172
[Basic strategies.] Suppose $d$ out of $n$ bottles are poisoned. Three simple strategies to identify $d$ poisoned bottles among $n$ bottles are:

(i) Complete parallel strategy $(n - 1, d, (n - 1, 1))$. We can use $n - 1$ rats simultaneously such that they drink wine from the first $n - 1$ different bottles. If $d$ rats die after a day, then we know the corresponding poisoned bottles. If $d - 1$ rats die after a day, then the corresponding bottles and the last bottle too are poisoned.

(ii) Serial strategy $(d, d, (n - 1, n - 1))$. Use $d$ rats to sip wine from the first $n - 1$ bottles on $n - 1$ consecutive days (use next rat only when one dies). It is straightforward to identify the poisoned bottles when $d$ or $d - 1$ rats die.

(iii) Repeated binary splitting strategy. Use the normal binary splitting algorithm $\text{BINARYSPLITTINGSEARCH}$ repeatedly for $d$ times.

$$\text{Strategies}(n, d) = \{(n - 1, d, (n - 1, 1)), (d, d, (n - 1, n - 1)), ((r, k), (s, t))\}$$

where, $r = k = s = d[\log(n)]$ and $t = s - \lceil \log(n) \rceil + 1$

---

**BINARYSPLITTINGSEARCH($S, n$)**

- **Input:** Set $S$ of $n$ wine bottles.
- **Output:** Identify a poisoned bottle in a contaminated set $S$ of $n$ wine bottles.

1. while $S$ is non-empty do
2.     test a subset $S'$ of $S$ with size $n' = \lceil n/2 \rceil$
3.     if $S'$ is pure then
4.         all bottles in $S'$ are good
5.     else if $S'$ is contaminated then
6.         $S \leftarrow S - S'$; $n \leftarrow n - n'$
7.     end if
8.     identify the single bottle in $S$ as poisoned

---

[Li’s $t$-stage search (near-optimal).] In this algorithm, there are $t$ stages. We assume that the bottles are labeled with their bottle numbers in the range $[0, n - 1]$. At stage 1, the $n$ bottles are arbitrarily divided into almost equal-sized $g_1 \geq d$ groups, each containing $m_1$ (or possibly $m_1 - 1$) bottles. We use $g_1$ rats to test these groups (one rat per group). A rat dies after a day if it sips wine from a contaminated group, i.e., a group with one of its bottles poisoned. The bottles from all contaminated groups are pooled together. At stage 2, the new pool of bottles is arbitrarily divided into almost equal-sized $g_2 \geq d$ groups, each containing $m_2$ (or possibly $m_2 - 1$) bottles. We use $g_2$ rats to test these groups and the process continues. In general, at stage $i \in [2, t]$, the bottles from all the contaminated groups of stage $i - 1$ are pooled together. The pool is arbitrarily divided into almost equal-sized $g_i \geq d$ groups, each containing $m_i$ (or possibly $m_i - 1$) bottles. We use $g_i$ rats to test these groups for purity or contamination. Finally, at stage $t$, $m_t$ is set to 1 and all poisoned bottles are identified.

Let’s analyze the values of the necessary parameters. We have

$$\text{Max. \#tests to identify the poisoned bottles} = g_1 + g_2 + \cdots + g_t$$
Max. #rats that might die = \( td \)
Max. #rats that survive = \( \max (g_1, g_2, \ldots, g_t) - d \)
Max. #rats required = \( \max (g_1, g_2, \ldots, g_t) + (t - 1)d \)

Let \( s \) denote the number of tests. Then

\[
\text{Strategy}(n, d) = \left\langle (r, k), (s, t) \right\rangle
\]

where, \( r = (\max (g_1, g_2, \ldots, g_t) + (t - 1)d), k = td, \) and \( s = (g_1 + g_2 + \cdots + g_t) \).

Suppose we want to minimize the number of tests keeping other parameters fixed. To simplify analysis, let’s assume that \( k_i \) is continuous. We have

\[
s = g_1 + g_2 + \cdots + g_t \leq \frac{n}{m_1} + d \left( \frac{m_1}{m_2} + \cdots + \frac{m_{t-2}}{m_{t-1}} + \frac{m_{t-1}}{m_t} \right)
\]

set \( m_i = \begin{cases} 
(n/d)^{t/(i-1)} & \text{if } i \in [1, t-1], \\
1 & \text{if } i = t,
\end{cases} \)

to get

\[
g_{i \in [1, t]} \leq d \left( \frac{n}{d} \right)^{1/t} \text{ and } s \leq td \left( \frac{n}{d} \right)^{1/t}
\]

To find the optimal value of \( s \), we differentiate \( s \) with respect to \( t \) and set the derivative to zero. Thus

\[
\frac{ds}{dt} = d \left( \frac{n}{d} \right)^{1/t} \left( 1 - t \ln(n/d) \right) / t^2 = 0 \implies t = \ln \left( \frac{n}{d} \right)
\]

Substituting \( t \) in the required equations, we have

\[
\text{#Tests-minimizing strategy}(n, d) = \left\langle (r, k), (s, t) \right\rangle
\]

where, \( r = (d (n/d)^{1/t} + (t - 1)d), k \leq d \ln(n/d), s \leq (e/ \log(e)) \cdot d \ln(n/d), \) and \( t = \ln(n/d) \).

**IStageSearch** \( (S, n, d) \)

**Input:** Set \( S \) of \( n \) wine bottles with at most \( d \) poisoned bottles.

**Output:** Identify at most \( d \) poisoned among \( n \) wine bottles in \( t \) days minimizing #tests.

1. **for** stage \( i \leftarrow 1 \) **to** \( t \) **do**
   2. divide \( S \) evenly into \( g_i \) groups of \( m_i \) (or \( m_{i-1} \)) bottles
   3. \( m_{i \in [1, t-1]} \leftarrow (n/d)^{(t-i)/t}; m_t \leftarrow 1 \)
   4. \( g_{i \in [1, t]} \leq d (n/d)^{1/t} \) test the \( g_i \) groups simultaneously
   5. \( S \leftarrow \) collect bottles from the contaminated groups
   6. set \( S \) contains at most \( d \) poisoned bottles

The strategy above needs to be fine-tuned for integral solutions through approximating with \( t = \left\lfloor \ln \left( \frac{n}{d} \right) \right\rfloor \) or \( \left\lceil \ln \left( \frac{n}{d} \right) \right\rceil \). The algorithm is near-optimal with respect to the number of tests in the sense that the number of tests required by the algorithm differs from the information theoretic bound of \( \left\lfloor \log^a C_d \right\rfloor \) by a constant factor or a constant.
[Hwang's generalized binary search (near-optimal).] Consider testing a group for poison. The advantage having a large group size is that if the test outcome is negative, then the bottles do not contain poison, which saves several further tests. On the other hand, the advantage having a small group size is that if the test outcome is positive, then only a small number of further tests are enough to identify the poisoned bottles. Hence, carefully choosing a proper group size is important and affects the goodness metrics of a strategy.

The core idea of Hwang's generalized binary search algorithm is in cleverly choosing a group size such that the probability of a test being positive is nearly half. The **GeneralizedBinarySearch** algorithm gives the strategy.

The number of tests $s$ required for $d \geq 2$ is $\lceil \log(\frac{n-d+1}{d}) \rceil + d - 1$. It is not clear how to parallelize this algorithm efficiently to minimize $t$ keeping the other parameters fixed. The computation of the values of $r$, $k$, and $t$ are left to the reader as an exercise.

**GeneralizedBinarySearch($S, n, d$)**

**Input:** Set $S$ of $n$ bottles with at most $d$ poisoned bottles.

**Output:** Identify at most $d$ poisoned bottles in a set $S$ of $n$ wine bottles minimizing #tests.

1. while $S$ is of size $n \geq 2d - 1$ do
2. set $q \leftarrow \lceil \log \left( \frac{(n-d+1)}{d} \right) \rceil$
3. test a subset $S'$ of $S$, with size $n' = 2^q$
4. if $S'$ is pure then
5. all bottles in $S'$ are good
6. $S \leftarrow S - S'$; $n \leftarrow n - n'$
7. else if $S'$ is contaminated then
8. use **BinarySplittingSearch** to identify 1 defective and, say, $x$ good bottles
9. $S \leftarrow S - \{identified bottles\}$
10. $n \leftarrow n - 1 - x$; $d \leftarrow d - 1$
11. test the remaining set $S$ of size $n \leq 2d - 2$ items individually

Unknown $d$ poisoned bottles

The solutions previously discussed can identify exactly or at most $d$ poisoned bottles among $n$ bottles. But, what if the number of poisoned bottles is unknown, as in most practical applications?

[Cheng-Du-Xu's zig-zag search (near-optimal).] The algorithm can identify any unknown number of poisoned bottles among a set of wine bottles. The core idea of the algorithm is as follows. Initially, we select the entire set of bottles as a subset to be tested. If a subset is tested to be pure, then the size of the subset to be tested in the next iteration is doubled. On the other hand, if a subset is contaminated, then the
size of the subset to be tested in the next iteration is halved. In general, the size of
the subset in the next iteration depends on test outcome in the current iteration.

The algorithm ZigZagSearch is so named because the size of a to-be-tested sub-
set might change in a zig-zag fashion. The maximum number of tests $s$ required is
$d \log(n/d) + 3d + 0.5 \log^2(d) + 1.5 \log(d) + 2$.

---

<table>
<thead>
<tr>
<th>ZigZagSearch$(S, n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Set $S$ of $n$ wine bottles.</td>
</tr>
<tr>
<td><strong>Output:</strong> Identify all the poisoned bottles in a set $S$ of $n$ wine bottles minimizing #tests.</td>
</tr>
<tr>
<td>1. $q \leftarrow \lceil \log(n) \rceil$</td>
</tr>
<tr>
<td>2. while $S$ is non-empty do</td>
</tr>
<tr>
<td>3. test a subset $S'$ of $S$, with size $n' = 2^q$</td>
</tr>
<tr>
<td>4. if $S'$ is pure then</td>
</tr>
<tr>
<td>5. all bottles in $S'$ are good</td>
</tr>
<tr>
<td>6. $S \leftarrow S - S'$; $n \leftarrow n - n'$; $q \leftarrow q + 1$</td>
</tr>
<tr>
<td>7. else if $S'$ is contaminated then</td>
</tr>
<tr>
<td>8. if $q &gt; 0$ then</td>
</tr>
<tr>
<td>9. use BinarySplittingSearch to identify a defective and, say, $x$ good bottles</td>
</tr>
<tr>
<td>10. $S \leftarrow S -$ [identified bottles]; $n \leftarrow n - 1 - x$; $q \leftarrow q - 1$</td>
</tr>
<tr>
<td>11. if $q = 0$ then</td>
</tr>
<tr>
<td>12. identify the single bottle in $S$ as poisoned</td>
</tr>
<tr>
<td>13. $S \leftarrow S - S'$; $n \leftarrow n - n'$</td>
</tr>
</tbody>
</table>

---

Problems

1. [Optimal algorithms.] Design optimal algorithms with respect to $r$, $k$, $s$, and $t$ to solve the problem.
2. [Analyze parameters.] Analyze the parameters $r$, $k$, $s$, and $t$ for those algorithms for which the parameter values were not estimated.
3. [Non-adaptive testing.] A strategy is called adaptive if future tests depend on the outcome of the previous tests. On the other hand, a strategy is called non-adaptive if all tests are known beforehand. All strategies we have seen till now are adaptive. Design optimal non-adaptive strategies to solve the problem.
4. [Fault-tolerance testing.] Due to several practical issues, tests are generally not infallible and might have a small percentage of failure. Design optimal strategies to identify the poisoned bottles even if some tests fail.
5. [Sick and immune rats.] Suppose that some rats are sick i.e., rats that would die from any wine and some rats are immune i.e., rats that would not die from any wine. What strategy can be used to identify the poisoned bottles?
References

Group testing was first studied in [Dorfman, 1943] during World War II for massive blood testing, who also gave a 2-stage algorithm to solve the problem. The $t$-stage and the generalized binary search near-optimal algorithms are given in [Li, 1962] and [Hwang, 1972], respectively, and nicely presented in [Du and Hwang, 2000]. The zig-zag algorithm is given in [Cheng et al., 2014]. Optimal $(t+1)$-ary search algorithm is by Andrey Gorlin (2016). A comprehensive book that covers the history, mathematical properties, algorithms, and applications of group testing is [Du and Hwang, 2000]. Variants of the puzzle are discussed in [Gardner, 2006], [Barr, 1982], and [Shasha, 2002]. Omitted proofs and analyses of the algorithms can be found in respective references. See [Anderson, 2018] for codes involving various bit operations.
Egg Dropping

Problem

There is a 100-floored building and we are given two identical eggs. We define threshold floor of a building as the highest floor in the building from and below which when the egg is dropped, the egg does not break, and above which when the egg is dropped, the egg breaks.

We want to find the threshold floor of the 100-floored building. The threshold floor of the building can be anything in the range $[0, 100]$. A threshold floor 0 means that when the egg is dropped from floor 1, the egg breaks, which in turn means that there is no threshold floor in the building.

What strategy finds the threshold floor minimizing the worst-case number of drops?

Solution

This is a popular interview problem and has appeared in the interviews of elite companies several times. Let’s nail the problem. We shall analyze all solutions starting from the simplest non-optimal to the toughest optimal solution. We call the highest floor from which the egg does not break as threshold floor. The threshold floor can be 0, which means there is no threshold floor. The threshold floor can be 100, which means even from the last floor the egg does not break.

2 eggs

[Linear search (non-optimal).] We drop an egg from the first floor. If it does not break, we drop it from the second floor. If it does not break, we drop it from the third floor and so on. If the egg does not break when it is dropped from the $i$th floor but breaks when it is dropped from the $(i+1)$th floor, it means that the threshold floor is $i$. In the worst case, the number of drops required to identify the threshold floor is 100. Note that we are not making use of the second egg.

Number of drops = 100.

[Binary search (non-optimal).] In this strategy, we do binary search using the first egg and linear search using the second egg. We drop the first egg from 50th floor. If it breaks, we use the second egg to do a linear search in the range of floors $[1, 49]$. If the first egg does not break, we drop it again from the 75th floor. If it breaks, we use the second egg to do a linear search in the range of floors $[51, 74]$. If the first egg does not break, we drop it again from 87th floor. This process continues. We make use of both
the eggs. In the worst case, the number of drops required to identify the threshold floor will be 1 + 49 = 50.

| Number of drops = 50 |

[Ternary search (non-optimal).] In this strategy, we do ternary search using the first egg and linear search using the second egg. We drop the first egg from the 33rd floor. If it breaks, we use the second egg to do a linear search in the range of floors [1, 32]. If the first egg does not break, we drop it again from the 66th floor. If it breaks, we use the second egg to do a linear search in the range of floors [34, 65]. If the first egg does not break, we drop it again from 66 + 11 = 77th floor. The process continues. In the worst case, the number of drops required to identify the threshold floor is 2 + 32 = 34.

| Number of drops = 34 |

[k-ary search (non-optimal).] We saw that the binary search (k = 2) reduced the number of drops compared to linear search. Ternary search (k = 3) further reduced the number of drops. We want to find the value of k that minimizes the number of drops. Once we know k, we can use k-ary search.

In this strategy, we do a k-ary search (divide the total number of floors into k parts) using the first egg and linear search using the second egg. Let the number of floors be n. We drop the first egg from \( p = \lfloor n/k \rfloor \)th floor. If it breaks, we use the second egg to do a linear search in the range of floors [1, \( p - 1 \)]. Else, we drop the same egg again at 2\( p \)th floor. If it breaks, we use the second egg to do a linear search in the range of floors \([p + 1, 2p - 1]\). If the first egg does not break, we drop it again from 3\( p \)th floor. The process continues.

Number of drops \( D \) required to identify the threshold floor is given by

| Number of drops \( (D) = \left( \frac{n}{k} \right) + k - 2. |

To minimize this expression, we need to differentiate \( D \) w.r.t \( k \) and equate it to zero. Then we get,

\[
\frac{dD}{dk} = \frac{d}{dk} \left( \frac{n}{k} + k - 2 \right) = \left( -\frac{n}{k^2} + 1 \right) = 0
\]

Simplifying, we get \( k = \sqrt{n} \). Substituting \( k \) in the equation for \( D \), we get

| Number of drops for n floors = \( 2\left\lfloor \sqrt{n} \right\rfloor - 2. |
| Number of drops for 100 floors = 18. |

[Konhauser-Velleman-Wagon’s decrement search (optimal).] In binary search, we divided the number of floors into two equal parts. In ternary search, we divided the number of floors into three equal parts. In k-ary search, we divided the number
of floors into \( k \) equal parts. These solutions are not optimal because they divide the number of floors into equal parts.

Optimality comes from dividing the number of floors into unequal parts. The parts must be such that the number of floors in the upper parts must be less than the number of floors in the lower parts. Why? Because, each time we drop an egg and if it does not break, we move up and the number of drops increases by 1. To make up for the lost drop and to maintain the same number of drops even when we move up, the number of floors in the upper part must be 1 less than the number of floors in the immediate below part. This means the number of floors in each part must be one less than its immediate below part.

As per the reasoning described above, if we drop the first egg from floor \( p \) and if it does not break, we must not drop the egg from floor \( 2p \), instead we must drop it from the floor \( p + (p - 1) \). On the other hand, if the egg breaks, we use the second egg to do a linear search in the range of floors \([1, p - 1]\) to find the threshold floor. If the first egg does not break for the second time, we must not drop the egg from \( 3p \), instead we must drop it from floor \( p + (p - 1) + (p - 2) \). Again, if the first egg breaks, we use the second egg to do a linear search in the range of floors \([p + 1, 2p - 2]\). The process continues till we cover all floors in the building. We are interested in the smallest value of \( p \) such that,

\[
p + (p - 1) + (p - 2) + \cdots + 1 \geq n \implies \frac{p(p + 1)}{2} \geq n
\]

| Number of drops for \( n \) floors = \( p = \lceil \sqrt{2n + 0.25} - 0.5 \rceil \). |
| Number of drops for 100 floors = 14. |

where \( \lceil \rceil \) represents the least integer or ceil function.

When \( n = 100 \), we drop the first egg from the following floors: 14, 14 + 13, 14 + 13 + 12, 14 + 13 + 12 + 11, and so on till 100 i.e., 14, 27, 39, 50, 60, 69, 77, 84, 90, 95, 99, and 100. If the first egg breaks, we use the second egg to do a linear scan in the required range to find the threshold floor. Refer to \textsc{DecrementSearch} for an algorithm that works for an arbitrary number of floors.

<table>
<thead>
<tr>
<th>\textsc{DecrementSearch}(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Number of floors ( n ).</td>
</tr>
<tr>
<td><strong>Output:</strong> Identify the threshold floor among ( n ) floors using 2 eggs.</td>
</tr>
<tr>
<td>1. ( p \leftarrow \lceil \sqrt{(2n + 0.25)} - 0.5 \rceil ); prevfloor ( \leftarrow 1 ); floor ( \leftarrow p ); drops ( \leftarrow 0 )</td>
</tr>
<tr>
<td>2. \textbf{while} floor ( \leq n ) \textbf{do}</td>
</tr>
<tr>
<td>3. \quad \text{drop the first egg from floor} floor; drops ( \leftarrow ) drops + 1</td>
</tr>
<tr>
<td>4. \quad \text{if} the first egg breaks \textbf{then}</td>
</tr>
<tr>
<td>5. \quad \mid \text{use \textsc{LinearSearch} in} \text{[prevfloor, floor - 1]} \text{to find the threshold floor}</td>
</tr>
<tr>
<td>6. \quad \text{else if} the first egg doesn’t break \textbf{then}</td>
</tr>
<tr>
<td>7. \quad \mid \text{prevfloor} \leftarrow \text{floor} + 1 ; p \leftarrow p - 1 ; \text{floor} \leftarrow \text{floor} + p</td>
</tr>
</tbody>
</table>

180
Generalization

We can generalize the problem in multiple ways:

- [Drops.] Given $n$ floors and $k$ eggs, minimize #drops $d$.
- [Floors.] Given $k$ eggs and $d$ drops, maximize #floors $n$.
- [Eggs.] Given $n$ floors and $d$ drops, minimize #eggs $k$.

We are interested in minimizing the #drops fixing the #floors and #eggs because this generalization represents real-life problems where we are given some fixed resources (like #floors and #eggs) and we are asked to optimize the execution time (like #drops).

[Sniedovich’s dynamic programming search (optimal).] We denote the minimum number of drops required to find the threshold floor in the consecutive $i$ floors using $j$ eggs as $drops[i, j]$. We would like to compute $drops[n, k]$, where $n$ is the number of floors in the building and $k$ is the number of eggs we can use.

We use a tremendously powerful algorithm design technique called dynamic programming to minimize #drops. The reader is recommended to understand the technique from a good algorithms textbook.

We compute $drops[n][k]$ using the following recurrence.

$$
drops[i, j] = \begin{cases} 
  i & \text{if } i \in [0, n], j = 1, \\
  i & \text{if } i \in [0, 1], j \in [2, k], \\
  1 + \min_{x \in [1, i]} \left\{ \max \left\{ drops[x - 1, j - 1], \ \\
  drops[i - x, j] \right\} \right\} & \text{if } i \in [2, n], j \in [2, k]. 
\end{cases}
$$

The base cases when $j = 1$ (1 egg) and $i = 1$ (1 floor) is straightforward. The recursion case for $i$ floors and $j$ eggs means the following. If we drop an egg from floor $x \in [1, i]$, there can be two cases: (i) the egg breaks, in which case we are left with $j - 1$ eggs and we need to test floors in the range $[1, i - 1]$. The number of drops required in this case is $1 + drops[x - 1, j - 1]$; and (ii) the egg does not break, in which case we are left with $j$ eggs and we need to test floors in the range $[x + 1, i]$. The number of drops required in this case is $1 + drops[i - x, j]$. As we need to compute the worst-case minimum number of drops, we take the minimum of all maximums by varying $x$ in the range $[1, i]$.

MINIMIZEDROPS-DP represents the dynamic programming algorithm for computing the minimum number of drops for $n$ floors given $k$ eggs. The left part of Figure 71 gives a pictorial representation of the DP algorithm and the right part gives the plot of $drops[n, k]$ for varying $n$. The algorithm computes only the minimum number of drops required to find the threshold floor and not the information about the threshold floor and which egg has to be dropped from which floor. The sequence of floors from which the eggs need to be dropped can be found by storing the optimal $x$ for every cell of the $drops$ dynamic programming table and using traceback. The reader can refer to standard algorithms textbooks to understand this concept.

The algorithm’s time complexity is $\Theta(n^2k)$ and space complexity is $\Theta(nk)$. The
MINIMIZEDROP-DP($n, k$)

**Input:** Number of floors $n$ and number of eggs $k$.

**Output:** Minimum number of drops $\text{drops}[n, k]$.

1. for $i \leftarrow 1$ to $n$ do
   
   2. $\text{drops}[i, 1] \leftarrow i$ // when there is only 1 egg

3. for $j \leftarrow 1$ to $k$ do

4. $\text{drops}[0, j] \leftarrow 0$; $\text{drops}[1, j] \leftarrow 1$ // when there are 0 or 1 floors

5. for $i \leftarrow 2$ to $n$ do

6. for $j \leftarrow 2$ to $k$ do

7. $\text{minimum} \leftarrow i$

8. for $x \leftarrow 1$ to $i$ do

9. $\text{maximum} \leftarrow \max(\text{drops}[x - 1, j - 1], \text{drops}[i - x, j])$

10. if maximum < $\text{minimum}$ then

11. $\text{minimum} \leftarrow \text{maximum}$

12. $\text{drops}[i, j] \leftarrow \text{minimum} + 1$

13. return $\text{drops}[n, k]$

---

**Figure 71:** Left: The core idea of the MINIMIZEDROP-DP algorithm. Right: Plot for $\text{drops}[n, k]$ when $n$ is varying from 1 to 30 and $k$ is fixed at 2, 3, or 10.

The algorithm’s runtime can be reduced to $O(nk \log n)$. We can find $\min_{x \in [1:i]}(\max(\text{drops}[x - 1, j - 1], \text{drops}[i - x, j]))$ in $O(\log n)$ time using binary search as follows. When $x$ is a variable, $\text{drops}[x - 1, j - 1]$ is an increasing function and $\text{drops}[i - x, j]$ is a decreasing function and hence $\max(\text{drops}[x - 1, j - 1], \text{drops}[i - x, j])$ will have a global minimum. This global minimum can be found using a variant of binary search. We can further reduce the algorithm’s runtime to $\Theta(nk)$. We store the optimal $x$ for every cell of the $\text{drops}$ DP table. When we want to find the optimal $x$ of a new cell, we make use of the optimal $x$ of its previous cells and search around its neighborhood.

[Boardman’s binary sequence solution (optimal).] We denote the maximum
number of floors in which we can find the threshold floor with at most $d$ drops and at most $k$ eggs by $floors[d, k]$. We compute $floors[d, k]$ using the following recurrence.

$$floors[i, j] = \begin{cases} 0 & \text{if } ij = 0, \\ floors[i - 1, j] + floors[i - 1, j - 1] + 1 & \text{if } ij > 0. \end{cases}$$

The base case when $i = 0$ (0 drops) or $j = 0$ (0 eggs) is straightforward. The recursion case for $i$ drops and $j$ eggs is found from the core concept as shown in Figure 72 (left). Let’s assume we have $i$ drops and $j$ eggs available. When we drop an egg from a floor that leads to the optimal solution, there are two cases: (i) the egg breaks, in which case we are left with $j - 1$ eggs and $i - 1$ drops; and (b) the egg does not break, in which case we are left with $j$ eggs and $i - 1$ drops. This means the term $floors[i, j]$ must be one greater than the sum of $floors[i - 1, j - 1]$ and $floors[i - 1, j]$. Hence, the recurrence. The plot of $floors[d, k]$ for varying $d$ is given in Figure 72 (right).

To find the number of drops, we have to compute $floors[i, j]$ for some known $i$ and $j$. The term can be computed efficiently if we find a direct formula for the recurrence of $floors[d, k]$. The direct formula for $floors[d, k]$ is found by mapping the method of finding a threshold floor in a building using $d$ drops and $k$ eggs to a binary sequence consisting of $d$ bits and at most $k$ 1s.

We map the method of finding the threshold floor in a building (using $d$ drops and $k$ eggs) to a binary sequence. An egg drop has two consequences: (i) the egg breaks, denoted as 1 or (ii) the egg does not break, denoted as 0. We use at most $d$ drops and find the threshold floor. The method to find the threshold floor can be written as a binary sequence i.e., a sequence of 0 and 1 bits. In this binary sequence, number of bits will be at most $d$ (as we can use at most $d$ drops) and bit-1 can appear at most $k$ times (as we can use at most $k$ eggs). For binary sequences of length $\ell$ ($< d$) that have
$k$ number of 1s, we append them with $(d - ℓ)$ 0s without loss of generality.

In this way, every floor in the building can be uniquely encoded with a binary sequence of $d$ bits having at most $k$ number of 1s. Figure 73 shows a decision tree for 3 drops and 2 eggs. The internal nodes represent the floor numbers from which we drop eggs for testing. The edge labels 0 and 1 represent whether the egg breaks or not. The leaf nodes represent the threshold floors. We see that the path length from root to leaf is at most 3 and in each path there are at most two 1s. The leaf node 0 represents that there is no threshold floor. Excluding leaf 0, there are a total of 6 leaves in the tree, which corresponds to 6 floors. Hence, $\text{floors}[3, 2] = 6$. In general, $\text{floors}[d, k]$ can be found by the following formula (How?):

$$\text{floors}[d, k] = dC_1 + dC_2 + dC_3 + \cdots + dC_k = \sum_{i=1}^{k} dC_i \leq 2^d,$$

(24)

where $dC_i$ represents the number of ways of choosing $i$ out of $d$ items.

We can compute $\text{floors}[d, k]$ in $O(k)$ time by using the identity $dC_{i+1} = dC_i \times (d-i)/(i+1)$ for all $i \geq 1$. We need to find a minimum value of $d$ such that $\text{floors}[d, k] \geq n$. This can be done in $O(n)$ time by scanning and checking. We can speed up the process to $O(\log n)$ using binary search as $\text{floors}[d, k]$ is an increasing function i.e., $\text{floors}[d, k] \leq \text{floors}[d + 1, k]$. Therefore, the total time complexity of finding the number of drops reduces to $O(k \log n)$.

**Problems**

1. Please write the pseudocode of the algorithm to compute $\text{drops}[n, k]$ in $\Theta(nk)$ time.
2. Define $\text{drops}[n, \infty]$ as the minimum number of drops required to identify the threshold floor among $n$ floors with an unlimited number of eggs. Compute $\text{drops}[n, \infty]$.
3. Let $\text{eggs}[n, d]$ represent the minimum number of eggs required to find the threshold floor in $n$ floors with $d$ drops. Write a dynamic programming recurrence for $\text{eggs}[n, d]$.  

184
References
A version of the puzzle and the decrement solution appeared in the book [Konhauser et al., 1996]. The dynamic programming solution is given in [Sniedovich, 2003]. The binary sequence solution is given in [Boardman, 2004] and [Wills, 2009]. A generating function solution is given in [Boardman, 2004] and the theory on generating functions is found in [Wilf, 2006].
Fox and Duck

Problem

There is a circular pond. A duck is at the center of the pond wanting to get to the boundary of the pond and fly away. However, a fox is waiting on the boundary of the pond to eat the duck. The fox cannot swim and the duck cannot fly from the water. The speed of the fox is 4 times the speed of the duck.

Is it possible for the duck to reach the boundary of the pond and escape from the cruel fox?

Solution

This puzzle is a living proof that when geometry and reasoning merge it makes for a fantastic mind-blower.

Shortest distance strategy (incorrect)

It is common to think that the duck can never escape from the fox as explained through the following strategy. Consider that the duck is at the center of the pond and the fox is at some point on the circumference of the pond. The duck moves along the radius directly opposite to the initial position of the fox as shown in Figure 74. In this strategy, the duck has to move a distance of \( r \) to reach the boundary and the fox has to cover half the circumference i.e., a distance of \( \pi r \) to catch the duck. By the time the duck covers a distance \( r \), the fox would have already covered a distance of \( 4r \). As \( 4r > \pi r \), the fox can easily catch the duck and it is impossible for the duck to escape.

In this strategy, the duck cannot escape from the fox. Just because the strategy did not work, it does not mean that there is no other strategy in which the duck can escape. Let’s analyze a strategy which proves that it is possible for the duck to escape from the fox.

Maximum lag strategy (non-optimal)

The fox can catch the duck when it stays on the duck’s radius. When the duck is circling near the edge of the pond, the fox can stay on the duck’s radius and hence can easily catch the duck. When the duck is circling near the center of the pond, it becomes very difficult for the fox to stay on the duck’s radius and hence the fox lags behind.

Till what (maximum) distance from the center of the pond when the duck circles, the fox will be barely able to stay on the duck’s radius? The speed of the fox is 4 times the speed of the duck. The circumference at a radius of \( \frac{r}{4} \) is exactly 4 times smaller than the circumference of the pond. Hence, when the duck circles at a radius of \( \frac{r}{4} \), the...
fox is just able to stay on the duck’s radius. But, when the duck circles at a radius less than $\frac{r}{4}$, the fox will lag behind.

To summarize: (i) If the duck circles at a radius $\geq \frac{r}{4}$, then the fox stays on the duck’s radius. (ii) On the other hand, if the duck circles at a radius $< \frac{r}{4}$, then the fox lags behind.

**[Strategy.]** The duck circles at a radius $\ell$ ($< \frac{r}{4}$) from the center of the pond so that the fox lags behind, as shown in Figure 75. The duck swims long enough to gain a $180^\circ$ lag i.e., the center of the pond is in between the fox and the duck. At this point, the duck swims straight to the edge of the pond which is at a distance $(r - \ell)$. The fox will not be able to catch up with the duck. The duck reaches the pond and flies away before the fox can attack the duck.

For the above strategy to work, what must be the value of $\ell$? We find it as follows. Let $\ell = \frac{r}{4} - d$, where $d > 0$. Assuming that the duck is on the circle of radius $\ell$ and the fox is lagging behind by $180^\circ$, we compute

Distance the duck has to travel $= r - \ell = \frac{3r}{4} + d$

Distance the fox has to travel $= \pi r$

For the duck to survive, it has to reach the pond before the fox attacks it. As the speed of the fox is 4 times the speed of the duck, 4 times the distance covered by the duck must be smaller than half the circumference. That is,

$$4 \left( \frac{3r}{4} + d \right) < \pi r \implies d < \left( \frac{\pi - 3}{4} \right) r$$

So, the range of values $d$ and $\ell$ can take are

$$0 < d < \left( \frac{\pi - 3}{4} \right) r \implies \left( \frac{4 - \pi}{4} \right) r < \ell < \frac{r}{4}$$
The duck moves to the circle of radius $\ell$, where $(4 - \pi)\frac{r}{4} < \ell < \frac{r}{4}$, and swims round and round such that the fox lags behind the duck by 180°. Then, the duck swims to the edge of the pond along the radius (opposite to fox) and the fox will not be able to catch the duck.

![Figure 75: Maximum lag strategy. The duck can escape from the fox but this is not the optimal strategy.](image)

[Maximum speed of the fox.] What is the maximum speed of the fox such that the duck can escape from the fox?

Let’s say that the fox runs at a speed $x$ times the duck’s speed. Then we write the range of $\ell$ as

$$\left(\frac{x - \pi}{x}\right) r < \ell < \frac{r}{x}$$

When the speed of the fox increases, the lower bound of $\ell$ increases. This means the range of $\ell$ goes on decreasing and at the minimum speed of the fox such that the duck can never escape, the lower bound of $\ell$ equals its upper bound. When the lower bound merges with upper bound, there exists no $\ell$. Hence, the duck cannot escape at this speed. That is

$$\left(\frac{x - \pi}{x}\right) = \frac{r}{x} \implies x - \pi = 1 \implies x = \pi + 1 \implies x \approx 4.1416$$

With the maximum lag strategy described above, the fox cannot catch the duck if the fox’s speed is strictly less than $(\pi + 1)$ times the duck’s speed.

The strategy solves the puzzle but we can do much better. The next strategy improvises on the current strategy.
Maximum lag and right direction strategy (optimal)

We generalize the puzzle and give the best possible strategy. Let the speed of the fox be \( x \) times the speed of the duck. We will find the maximum speed of the fox such that the duck can still escape. If that maximum speed is greater than \((\pi + 1)\), we will have a better strategy than the previous strategy.

![Diagram](image.png)

Figure 76: Maximum lag and right direction strategy. (a) Top-left: Reach the circle of safety, (b) Top-right: Multiple possible paths, (c) Bottom-left: Choose a particular path, and (d) Reach the boundary of the pond.
Let’s call the circle of radius $\ell = \frac{r}{x} - \epsilon$, where $\epsilon > 0$, as the circle of safety. The mind-blowing strategy we are going to discuss in further paragraphs consists of two steps:

1. **[Reach the circle of safety.]** The duck creates the maximum lag in the best possible way and reaches the circle of safety.
2. **[Reach the boundary of the pond.]** The duck chooses the right direction to reach the boundary of the pond in a straight line.

**[Step 1. Reach the circle of safety.]** The duck starts from the center and swims outward as shown in Figure 76(a), such that the center of the pond $O$ is always between the duck and the fox i.e., the three points are collinear. The duck completes the semicircle $OQA$ of radius $\frac{r}{(2x)}$ and reaches point $A$ on the circle of safety while the fox runs for the duck. It is easy to see that $A$ is at a distance of $\left(\frac{r}{x} - \epsilon\right)$ from $O$.

In the previous strategy we said that the duck swims straight from the center of the pond to the circle of safely and then circles around so that the center of the pond comes between the duck and the fox. However, in this strategy the duck swims in a semicircle of radius $\frac{r}{(2x)}$ and reaches the circle of safety in the fastest possible way.

**[Step 2. Reach the boundary of the pond.]** The duck moves tangentially to the radius of the circle to the opposite side of the fox and reaches the pond, as shown in Figure 76(d). The fox has to cover a distance more than half the circumference of the pond. The duck reaches the pond and flies away before the fox can catch it. We see in the further paragraphs why the method works and how we arrive at this method.

In the previous strategy, when the duck reaches the circle of safety it swims straight to the boundary of the pond (see Figure 75). The duck swims in a direction that minimizes the time to reach the shore. However, in this strategy, the duck swims in a direction that maximizes the distance between itself and the fox when it reaches the shore.

When the duck reaches the circle of safety, it can swim in a straight line in any of the infinite directions. Four possible directions are shown as paths 1 to 4 in Figure 76(b). We already did choose path 1 in the previous strategy. We don’t choose path 4 because that path takes the duck inside the circle of safety and leads in another exit which would result in a different solution as if the duck exited from that other side. Now we are left with path 2 and path 3. We see that path 2 is a generic path that covers path 1 and path 3 as its special cases. Hence we choose path 2 and the duck swims through $AC$ as shown in Figure 76(c).

Let’s write down the observations from Figure 76(c). Radians will be used as a unit of all angles. We see that $OB = OC = r; OA = \frac{r}{x}; \angle BOC = \theta; \angle BAC = \alpha; OB' = r \sin \theta; DB = \pi r; BC = r \theta; DBC = \pi r + r \theta; AC = a = \sqrt{(BC)^2 + (AB')^2} = \sqrt{(r \sin \theta)^2 + (r \cos \theta - (\frac{r}{x}))^2} = \sqrt{r^2 - \left(\frac{2r}{x}\right) \cos \theta + (\frac{r}{x})^2}$; and $\angle BAC = \alpha = \sin^{-1}\left(\frac{(BC)}{a}\right) = \sin^{-1}\left(\frac{r}{a}\right) \sin \theta$.

From our observations, we need to find $\alpha$ and the maximum value of $x$ so that the duck can still escape. We solve this in three parts. First, we write $x$ as a function of $\theta$, say $x = f(\theta)$. Second, we find the optimal value of $x$ (from its corresponding $\theta$ value) by taking the derivative of $f$ with respect to $\theta$ and equating it to zero i.e., by solving...
\[ \frac{dx}{d\theta} = f'(\theta) = 0. \] Third and finally, we compute \( \alpha \) corresponding to the optimal value of \( x \).

**[Part 1. Write \( x \) as a function of \( \theta \).]** For the strategy to work, the time taken for the fox to travel \( DBC \) must be greater than the time taken for the duck to travel \( BC \). At the limit, the times taken by both fox and the duck will be the same. Thus,

\[ x \cdot AC = DBC \]

\[ \implies xa = r(\pi + \theta) \quad \text{(substitute)} \]
\[ \implies x^2a^2 = r^2(\pi + \theta)^2 \quad \text{(square)} \]
\[ \implies x^2 \left( r^2 - \frac{(2r^2)}{x} \cos \theta + \left( \frac{r}{x} \right)^2 \right) = r^2(\pi + \theta)^2 \quad \text{(substitute for \( a \))} \]
\[ \implies x^2 - 2x \cos \theta + 1 = (\pi + \theta)^2 \quad \text{(simplify)} \]
\[ \implies (x - \cos \theta)^2 + (1 - \cos^2 \theta) = (\pi + \theta)^2 \quad \text{(add and subtract \( \cos^2 \theta \) and simplify)} \]
\[ \implies (x - \cos \theta)^2 + \sin^2 \theta = (\pi + \theta)^2 \quad \text{(use \( \sin^2 \theta + \cos^2 \theta = 1 \))} \]
\[ \implies x = \cos \theta + \sqrt{(\pi + \theta)^2 - \sin^2 \theta} \quad \text{(simplify)} \]

**[Part 2. Compute the optimal value of \( x \).]** We compute the optimal value of \( x \) by differentiating the equation above and equating it to zero. That is

\[ \frac{dx}{d\theta} = -\sin \theta + \frac{(\pi + \theta) - \sin \theta \cos \theta}{(\pi + \theta)^2 - \sin^2 \theta} = 0 \]
\[ \implies \sin \theta \left( (\pi + \theta)^2 - \sin^2 \theta \right) = (\pi + \theta) - \sin \theta \cos \theta \quad \text{(simplify)} \]
\[ \implies (\pi + \theta)^2 (1 - \sin^2 \theta) - 2(\pi + \theta) \sin \theta \cos \theta + \left( \sin^2 \theta + \cos^2 \theta \right) \sin^2 \theta = 0 \quad \text{(square, simplify)} \]
\[ \implies (\pi + \theta)^2 \cos^2 \theta - 2(\pi + \theta) \sin \theta \cos \theta + \sin^2 \theta = 0 \quad \text{(\( \sin^2 \theta + \cos^2 \theta = 1 \))} \]
\[ \implies (\pi + \theta)^2 \cot^2 \theta - 2(\pi + \theta) \cot \theta + 1 = 0 \quad \text{(divide by \( \sin^2 \theta \))} \]
\[ \implies ((\pi + \theta) \cot \theta - 1)^2 = 0 \quad \text{(square formula)} \]
\[ \implies \tan \theta = \pi + \theta \quad \text{(simplify)} \]
\[ \implies \theta = 1.3518168 \ldots \quad \text{(use a software)} \]
\[ \implies x = 4.6033388 \ldots \quad \text{(use \( x \)'s equation)} \]

**[Part 3. Compute \( \alpha \) corresponding to the optimal value of \( x \).]** We have

\[ \alpha = \sin^{-1} \left( \frac{r \sin \theta}{a} \right) \]
\[ \implies \alpha = \sin^{-1} \left( \frac{rx \sin \theta}{r(\pi + \theta)} \right) \quad \left( a = AC = \frac{DBC}{x} = \frac{r(\pi + \theta)}{x} \right) \]
\[ \implies \alpha = \sin^{-1} \left( \frac{x \sin \theta}{\tan \theta} \right) \quad \text{(\( \tan \theta = \pi + \theta \))} \]
\[ \implies \alpha = \sin^{-1} (x \cos \theta) \quad \text{(simplify)} \]
\[ \implies \alpha = \sin^{-1} (4.60333889 \times \cos 1.35181689) \quad \text{(substitute)} \]
\[ \implies \alpha = 90^\circ \quad \text{(use a software)} \]
This means our path 3 is the actual path that the duck must follow. The duck’s path is shown in Figure 76(d).

With the maximum lag and right direction strategy, the fox cannot catch the duck if the fox’s speed is strictly less than 4.603339 times the duck’s speed.

Problems
1. For every strategy discussed above, compute the time taken by the duck to reach the shore.

References
The puzzle has been discussed in [Guy, 1961], [O’Beirne, 1961], [Schuurman and Loder, 1974], [Gardner, 1989], [Mongan et al., 2007], [Genetics, 2016], and [Moiseeva, 2011]. The present puzzle and several other chasing and escaping puzzles are beautifully explained in Paul J. Nahin’s excellent book [Nahin, 2012]. For trigonometry, refer to Ron Larson’s nice book [Larson, 2022] and for calculus, refer to the excellent book by George B. Thomas et al. [Thomas et al., 2018].
Pirates and Gold Coins

Problem

A pirate ship has captured a treasure of 100 gold coins. The treasure has to be split among 10 pirates. The most powerful pirate proposes a way to split the coins among them. All pirates vote for the proposal, including the proposer. If at least half of the pirates agree for the proposal, the coins will be divided according to the proposal. If at least half of the pirates do not agree for the proposal, then the pirate will be killed and second most powerful pirate proposes a way to split the coins. This process continues till the least powerful pirate. The pirates would strive hard to survive, kill others, and grab as much coins as possible.

What must be the strategy of the most powerful pirate to maximize his share?

Solution

Let the pirates be denoted by $P_1, P_2, \ldots$, where $P_1$ is more powerful than $P_2$ who is more powerful than $P_3$ and so on.

[Fewer pirates.] Suppose there is only one pirate $P_1$. Then, $P_1$ keeps all 100 coins to himself. Suppose there are two pirates $P_1$ and $P_2$. Then, the more powerful pirate $P_1$ keeps all the 100 coins to himself and the other pirate $P_2$ gets nothing. As the proposer $P_1$ votes for himself and it is half of the total votes, the proposal will be implemented.

Suppose there are three pirates $P_1, P_2,$ and $P_3$. If $P_1$ was not present, the problem would have been reduced to a two pirate problem according to which $P_2$ would get 100 and $P_3$ nothing. With this knowledge, $P_1$ will bribe $P_3$ for the minimum number of gold coins and $P_3$ will vote in favor of $P_1$. $P_2$ gets nothing. The proposal (or distribution of coins) would be: 99 for $P_1$, 0 for $P_2$, and 1 for $P_3$. Suppose there are four pirates $P_1, P_2, P_3,$ and $P_4$. The proposal is similar as in the case of three pirates. Here, pirate $P_1$ bribes $P_3$ for 1 gold coin and makes the proposal: 99 for $P_1$, 0 for $P_2$, 1 for $P_3$, and 0 for $P_4$. We continue this process till 10 pirates.

We see the pattern. The first (or the most powerful) pirate gets the lion’s share of the gold coins and others get 0 and 1 coins alternatively. In fact, the odd numbered pirates (except $P_1$) get 1 coin each and the even numbered pirates get nothing. The remaining coins will be taken by $P_1$, the proposer.

When there are 10 pirates, the pirate $P_1$ bribes four pirates $P_3, P_5, P_7, P_9$ for 1 gold coin each and makes the proposal: 96 for $P_1$, 1 each for $P_3, P_5, P_7, P_9$, and 0 for $P_2, P_4, P_6, P_8, P_{10}$.

Gold distribution for 10 pirates $[P_1, \ldots, P_{10}]$ is $[96, 0, 1, 0, 1, 0, 1, 0, 1, 0]$. 

193
More pirates. Gold distribution strategies among different number of pirates are given in Table 46. The pattern of distribution of gold coins continues till 200 pirates.

<table>
<thead>
<tr>
<th>k</th>
<th>P₁</th>
<th>P₂</th>
<th>P₃</th>
<th>P₄</th>
<th>P₅</th>
<th>P₆</th>
<th>..</th>
<th>P₁₉₉</th>
<th>P₂₀₀</th>
<th>P₂₀₁</th>
<th>P₂₀₂</th>
<th>P₂₀₃</th>
<th>P₂₀₄</th>
<th>..</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>99</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>99</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>98</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>98</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>..</td>
</tr>
<tr>
<td>199</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>..</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>..</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>201</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>..</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>202</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>..</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>203</td>
<td>✗</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>..</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>204</td>
<td>✗</td>
<td>✗</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>..</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

Table 46: Strategies of distribution of 100 gold coins among different number of pirates. The symbol ✗ denotes that the pirate will be killed.

Suppose there are 201 pirates. Pirate $P_1$ votes for himself and gets 100 votes from others. He bribes the odd numbered pirates from $P_3$ to $P_{201}$ for one gold coin each and keeps himself nothing. Though he gains nothing, he at least survives because of the 101 votes.

Suppose there are 202 pirates. Pirate $P_1$ votes for himself and gets 100 votes from other 101 pirates who would get nothing under $P_2$’s proposal, who include $P_3, P_5, \ldots, P_{201}$ and $P_2$. $P_1$ bribes 100 of those 101 pirates, which can be done in any of 101 ways. Though $P_1$ gains nothing, he at least survives because of 101 votes. Note that only one of the 101 ways of $P_1$’s proposal is shown for 202 pirates in Table 46.

Suppose there are 203 pirates. Pirate $P_1$ votes for himself and gets 100 votes from other 101 pirates who would definitely get nothing under $P_2$’s proposal. (There is one more pirate who does not get anything under $P_2$’s proposal but it is not known beforehand who that pirate will be.) Though $P_1$ can bribe 100 of the 101 pirates, which can be done in any of 101 ways, $P_1$ definitely cannot get a majority as he can only get a maximum of 101 votes. Hence, $P_1$ will be killed, the problem reduces to that of 202 pirates and $P_2$ makes the proposal.

In a similar way, we can solve the puzzle when there are more than 203 pirates.

**Generalization**

If there are $n$ gold coins and $k$ pirates then a pirate strategy is given in the PIRATES-STRATEGY algorithm.
**PIRATESSTRATEGY**(\(n, k\))

**Input:** Number of gold coins is \(n\) and the number of pirates is \(k\).

**Output:** The pirate strategy proposed by \(P_{\text{max}}(k-2n-1, 1)\).

1. **for** \(i \leftarrow 1\) **to** \(k - (2n + 2)\) **do**
2. \(P_i \leftarrow \mathcal{X}\) \hspace{1cm} // pirates who get killed
3. \(\text{temp} \leftarrow [(k - 1)/2]\)
4. \(P_i \leftarrow n - \text{temp}; i \leftarrow i + 1\) \hspace{1cm} // pirate who proposes
5. **for** \(j \leftarrow 1\) **to** \(\text{temp}\) **do**
6. \(P_i \leftarrow 0; i \leftarrow i + 1\) \hspace{1cm} // pirates who get nothing
7. \(P_i \leftarrow 1; i \leftarrow i + 1\) \hspace{1cm} // pirates who get a gold coin and hence vote
8. **if** \(i = k\) **then**
9. \(P_i \leftarrow 0\) \hspace{1cm} // pirate who gets nothing
10. **return** \([P_1, P_2, \ldots, P_k]\)

**Variant: Hungry demons and a sleeping man**

A village has many hungry demons. They all see a sleeping man. If a demon eats the sleeping man, the demon falls asleep. If another demon eats the sleeping demon, the eater demon falls asleep. This process of demon eating demon continues.

All demons are extremely intelligent. Hence their first priority is to be alive (i.e., not to be eaten by other demons) and then the second priority is to eat any sleeping being. In summary, a demon eats a sleeping being only if it is confident that it will not be eaten even after it falls asleep. What happens eventually?

[Answer.] If there are odd number of demons, one demon eats the sleeping man and nothing else happens. In constrast, if there are even number of demons, nothing happens. Why?

**Take-home lessons**

We used the following small-instances problem-solving strategy to solve the puzzle:

1. Solve small instances of a given puzzle.
2. Identify a pattern among the solutions.
3. Generalize the solution.

Small instances are easier to solve than large instances. Searching for a gold ring in a small field is easier than searching for a gold ring in a big field. Therefore, we should aim to solve small instances of the given problem and then try to find a pattern in the solutions to solve larger instances.

Science has often been considered as the study of patterns in nature. In the most general sense, a pattern is any kind of regularity. They are the generalizations of different but related instances. Patterns are everywhere. Mathematical formulas are the algebraic patterns that show the relation between certain numbers; mathematical functions are the geometric patterns that describe the relation between independent and dependent variables; algorithms are the step-by-step procedure patterns to solve...
computational problems; algorithm design techniques are the problem-solving strategy patterns that can be used to design algorithms; design patterns are the generic software planning patterns used in software design; predictions of the future (e.g.: weather forecasting, planetary motion, etc) are based on the generalized patterns of observed events in some prediction models (such as statistics, machine learning, probability theory, and theories of physics); medical diagnosis uses disease or illness patterns to determine the disease or illness of a person based on symptoms; clothes and buildings are decorated with artistic patterns some of which are based on mathematics; and finally, there exists a physics theory that propounds that an $i$th dimensional universe resides on the event horizon of an $(i + 1)$th dimensional universe and this happens recursively for all $i$.

Once we have identified patterns in small instances, we generalize them to formulas, functions, algorithms, and principles. This powerful problem-solving strategy can be used to solve a wide variety of problems.

**References**

The puzzle and its solution appear in [Stewart, 1999].
Camels and Bananas

Problem

A camel has to transport bananas from a market to an oasis. The distance between the market and the oasis is 1000 kilometers. There are 3000 bananas at the market. The camel can carry up to 1000 bananas at a time. For every kilometer it walks, it eats a banana. The camel rider wants to transfer a maximum number of bananas to oasis with the conditions mentioned above. What is the best strategy for the camel rider?

Solution

Solution (optimal)

The capacity of the camel is 1000 bananas. Let the market and oasis be denoted by $A$ and $B$, respectively. Then the distance between $A$ and $B$ is 1000 km. We use the notation $A \rightarrow B$ to represent the journey from point $A$ to point $B$. Initially, we solve a simpler version of the problem with only 1000 bananas. Then we increase the complexity of the problem and the solution.

[1000 bananas.] There are 1000 bananas at point $A$. When the camel travels from $A$ to $B$ and eats a banana for every kilometer, there would be no bananas left when the camel reaches $B$. Table 47 gives the journey details of the camel. If the distance between $A$ and $B$ was less than 1000 km, then we could have transferred non-zero number of bananas to $B$.

<table>
<thead>
<tr>
<th>Step</th>
<th>Trip</th>
<th>Distance</th>
<th>Bananas at $A$</th>
<th>Bananas at $B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>–</td>
<td>–</td>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$A \rightarrow B$</td>
<td>1000</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 47: Camel’s journey when there are 1000 bananas at $A$.

[2000 bananas.] Suppose we have 2000 bananas at point $A$. The camel cannot carry all 2000 bananas at once. It can carry at most 1000 bananas at a time. This means the camel has to make multiple trips. The camel transfers all 2000 bananas from point $A$ to an intermediate point $X$ in more than one trip such that there are 1000 bananas at point $X$. From $X$, the camel can carry all 1000 bananas to point $B$ in one shot. However, there is a penalty to be paid in the journey between $A$ and $X$ because
the camel has to make three trips to carry 2000 bananas from A to X: (i) $A \rightarrow X$ to carry the first 1000 bananas. (ii) $X \rightarrow A$ to return. (iii) $A \rightarrow X$ to carry the remaining 1000 bananas.

$$A \longleftrightarrow X \longrightarrow B$$

The lengths $AX$ and $XB$ can be found as follows. We know that the camel ate 1000 bananas during the three trips of $AX$. Hence,

$$3 \cdot AX = 1000 \implies AX = 333.33 \implies XB = 1000 - AX = 666.67$$

Initially, there are 2000 bananas at A. Distances $AX = 333^{1\over 3}$ km and $XB = 666^{2\over 3}$ km. The camel starts from A with 1000 bananas and reaches X with $666^{2\over 3}$ bananas. It returns from X with $333^{1\over 3}$ bananas and reaches A. The bananas were simply used as fuel to return to A. Again, the camel starts from A with 1000 bananas and reaches X with $666^{2\over 3}$ bananas. After three trips ($A \rightarrow X, X \rightarrow A, A \rightarrow X$) there will be 1000 bananas at X. Now the camel starts from X with 1000 bananas and reaches B with $333^{1\over 3}$ bananas. Table 48 summarizes the camel’s journey.

<table>
<thead>
<tr>
<th>Step</th>
<th>Trip</th>
<th>Distance</th>
<th>Bananas at A</th>
<th>X</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>2000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$A \rightarrow X$</td>
<td>$333^{1\over 3}$</td>
<td>1000</td>
<td>$666^{2\over 3}$</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$X \rightarrow A$</td>
<td>$333^{1\over 3}$</td>
<td>1000</td>
<td>$333^{1\over 3}$</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>$A \rightarrow X$</td>
<td>$333^{1\over 3}$</td>
<td>0</td>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>$X \rightarrow B$</td>
<td>$666^{2\over 3}$</td>
<td>0</td>
<td>0</td>
<td>$333^{1\over 3}$</td>
</tr>
</tbody>
</table>

Table 48: Camel’s journey when there are 2000 bananas at A.

[3000 bananas.] Suppose there are 3000 bananas at point A. The camel cannot carry all 3000 bananas in three trips. So the camel transfers all 3000 bananas from point A to an intermediate point X in five trips such that there are 2000 bananas at point X. From X, the camel can carry all 2000 bananas to point B as explained in the previous section. Camel’s journey from A to X: (i) $A \rightarrow X$ to carry first 1000 bananas. (ii) $X \rightarrow A$ to return. (iii) $A \rightarrow X$ to carry second 1000 bananas. (iv) $X \rightarrow A$ to return. (v) $A \rightarrow X$ to carry third and final 1000 bananas.

$$A \longleftrightarrow X \longleftrightarrow Y \longrightarrow B$$

The lengths $AX, XY,$ and $YB$ can be found as follows. We know that the camel eats 1000 bananas during the five trips of $AX$. Hence,

$$5 \cdot AX = 1000 \implies AX = 200 \text{ and } 3 \cdot XY = 1000 \implies XY = 333.33$$

$$\implies YB = 1000 - AX - XY = 466.67$$

Initially, there are 3000 bananas at A. After five trips between A and X there will be 2000 bananas at X. $AX = 200$ km. After three trips between X and Y there will be
1000 bananas at $Y$. $XY = 333\frac{1}{3}$ km. After the final trip between $Y$ and $B$ there will be $533\frac{1}{3}$ bananas at $B$. $XB = 466\frac{2}{3}$ km. Table 49 summarizes the camel’s journey.

Table 49: Camel’s journey when there are 3000 bananas at $A$.

<table>
<thead>
<tr>
<th>Step</th>
<th>Trip</th>
<th>Distance</th>
<th>Bananas at</th>
<th>A</th>
<th>X</th>
<th>Y</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>–</td>
<td>–</td>
<td>3000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$A \rightarrow X$</td>
<td>200</td>
<td>2000</td>
<td>800</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$X \rightarrow A$</td>
<td>200</td>
<td>2000</td>
<td>600</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>$A \rightarrow X$</td>
<td>200</td>
<td>1000</td>
<td>1400</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>$X \rightarrow A$</td>
<td>200</td>
<td>1000</td>
<td>1200</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>$A \rightarrow X$</td>
<td>200</td>
<td>0</td>
<td>2000</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>$X \rightarrow Y$</td>
<td>$333\frac{1}{3}$</td>
<td>0</td>
<td>1000</td>
<td>$666\frac{2}{3}$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>$Y \rightarrow X$</td>
<td>$333\frac{1}{3}$</td>
<td>0</td>
<td>1000</td>
<td>$333\frac{1}{3}$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>$X \rightarrow Y$</td>
<td>$333\frac{1}{3}$</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>$Y \rightarrow B$</td>
<td>$466\frac{2}{3}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$533\frac{1}{3}$</td>
<td>0</td>
</tr>
</tbody>
</table>

[Generalization.] There are $n$ bananas at point $A$. The camel’s capacity is $c$ i.e., the camel can carry at most $c$ bananas at once. The camel has to transfer a maximum number of bananas to point $B$. The distance between $A$ and $B$ is $d$ kilometers. The camel eats a banana for every kilometer it walks. Let $f(n, c, d)$ represent #bananas transferred to $B$. Then $f(n, c, d)$ can be computed using the recurrence:

$$f(n, c, d) = \begin{cases} 
0 & \text{if } d > n \text{ or } d < 0, \\
 n - d & \text{if } c \geq n \geq d, \\
 f\left(n - r, c, d - \frac{r}{|\left\lfloor \frac{n}{c} \right\rfloor - 1|}\right) & \text{otherwise}, 
\end{cases}$$

(25)

where $r = n - c\left(\left\lfloor \frac{n}{c} \right\rfloor - 1\right)$.

[Base case.] If $d$ is greater than $n$ then the camel cannot transfer any bananas to $B$. In fact the camel can only walk for $n$ kms and it will halt at a distance of $d - n$ kms from the destination. If $c$ is greater than or equal to $n$ and $n$ is greater than or equal to $d$, then the camel can start with $n$ bananas and safety transfer $(n - d)$ bananas to the destination.

[Recursion case.] If the base case conditions are not met then we use the recursion case which is used to transfer bananas from one milestone to another. The recursion case is hard to understand. Consider #bananas at the source which is $n$. We want to transfer as many bananas as possible to the next milestone. We transfer $n - r$ bananas, where $r = n - c\left(\left\lfloor \frac{n}{c} \right\rfloor - 1\right)$, to the next milestone. For example, if there were $n = 3000$ bananas and $c = 1000$, then $r = 1000$. This means we transfer $n - r = 2000$ bananas to the next milestone. If there were $n = 2847$ bananas and $c = 1000$, then $r = 847$. This means we transfer $n - r = 2000$ bananas to the next milestone. In this case $r = 847.$
Number of trips required to transfer \( n \) bananas at the source to \( n - r \) bananas at a milestone is \((2\lfloor n/c \rfloor - 1)\). The camel eats \( r \) bananas in \((2\lfloor n/c \rfloor - 1)\) trips which implies that the distance traveled by the camel to the first milestone is \((2\lfloor \frac{n}{c} \rfloor - 1)\). Hence, the distance \( d \) reduces to \((d - \frac{r}{\lfloor \frac{n}{c} \rfloor - 1})\), which is the distance between the first milestone and the final destination. The number of trips to reach the second milestone from the first is \((2\lfloor \frac{n}{c} \rfloor - 3)\). This process continues.

The recurrence relation can be solved as follows. Note that \( n - r = c\lfloor \frac{n}{c} \rfloor - 1\). This means that after the first application of the recursion case \#bananas remaining will be a multiple of \( c \). From here, \( r = c \) for all future iterations of the recursive case. Therefore

\[
\begin{align*}
f(n, c, d) &= f \left(n - r, c, d - \frac{r}{2\lfloor n/c \rfloor - 1}\right) \\
&= f \left(c\lfloor \frac{n}{c} \rfloor - 1, c, d - \left(\frac{n - c\lfloor \frac{n}{c} \rfloor - 1}{2\lfloor n/c \rfloor - 1}\right)\right) \\
&= f \left(c\lfloor \frac{n}{c} \rfloor - 2, c, d - \left(\frac{n - c\lfloor \frac{n}{c} \rfloor - 1}{2\lfloor n/c \rfloor - 1}\right) - \left(\frac{c}{2\lfloor n/c \rfloor - 3}\right)\right) \\
&= f \left(c, c, d - \left(\frac{n - c\lfloor \frac{n}{c} \rfloor - 1}{2\lfloor n/c \rfloor - 1}\right) - \left(\frac{c}{2\lfloor n/c \rfloor - 3}\right)\right) \\
&= c - \left(\frac{n - c\lfloor \frac{n}{c} \rfloor - 1}{2\lfloor n/c \rfloor - 1}\right) - \left(\frac{c}{2\lfloor n/c \rfloor - 3}\right) \cdots - \left(\frac{c}{3}\right) \\
&= \left(\frac{n - c\lfloor \frac{n}{c} \rfloor - 1}{2\lfloor n/c \rfloor - 1}\right) + c \left(1 + \frac{1}{3} + \cdots + \frac{1}{2\lfloor n/c \rfloor - 3}\right) - d \\
&= \left(\frac{n - ck}{2k + 1}\right) + c \sum_{i=1}^{k} \frac{1}{2i - 1} - d \quad (k = \lfloor \frac{n}{c} \rfloor - 1)
\end{align*}
\]

We have \( n \) bananas that needs to be transferred across a distance of \( d \) kms. The camel’s capacity is \( c \) bananas. Then

\[
\text{#Bananas transferred} = \left(\frac{n - ck}{2k + 1}\right) + c \sum_{i=1}^{k} \frac{1}{2i - 1} - d \quad \text{, where, } k = \lfloor \frac{n}{c} \rfloor - 1
\]

**Problems**

1. There are \( p \) camels with capacities \( c_1, c_2, \ldots, c_p \), respectively and they eat \( b_1, b_2, \ldots, b_p \) bananas per km, respectively. Initially there are \( n \) bananas at the market and the distance between market and oasis is \( d \) kms. What is the strategy to maximize the number of bananas transferred to oasis?
References

A related problem called the jeep problem existed during World War II. [Fine, 1947a] first solved the problem in 1947. The problem and the solution was later generalized in [Phipps, 1947]. Variants of the puzzle have been discussed in [Alway, 1957]. The puzzle has also been discussed in Louis A. Graham’s legendary puzzle book [Graham, 2012].
River Crossing

Problem

Three married couples must cross a river. They have a boat that holds at most two people at a time. The husbands are extremely jealous. No husband allows his wife to be on the same bank as another man unless he himself is there. How can the three couples safely cross the river satisfying the constraints and minimizing the number of trips?

Solution

This is a typical example of a class of puzzles called river crossing puzzles. The most common approach used to solve puzzles in this class is trial-and-error. The problem with trial-and-error is that it is not a systematic and organized method for solving a problem. If we are given another complicated instance of the puzzle, we need to struggle to solve that instance. Even if we succeed, we will have to struggle to solve the 3rd instance, 4th instance, and so on up to infinity. On the other hand, if we design a systematic and organized method for solving the problem, we can use that approach to solve an infinite instances of similar puzzles.

State diagrams can be used to solve river crossing puzzles systematically.

State diagram

*State diagram* is a very powerful approach to solve a decent class of puzzles. A state diagram is a graph consisting of vertices and edges, where a vertex represents a *state* or configuration in the process and an edge denotes a *transition* between two states. We show the versatility of the method by using it to solve a variety of puzzles. A state diagram can be represented as a tree or a directed acyclic graph or a generic graph (that can contain cycles) depending on the requirements of the problem or solver. We represent state diagrams as *directed acyclic graphs* in our presentations. A state diagram has a single *start state* and may have zero or more *final states*. A path from the starting state to the final state without visiting an already visited vertex is called a *simple solution* and a solution that minimizes some cost (e.g.: the number of edges or the total edge cost) is called an *optimal solution*. If there is no path from the start state to any of the final states, then the problem has no solution. If there are paths from the start state to multiple final states, then there are multiple simple solutions and some of them are optimal.

The constraints of the given problem define the state transitions. A feasible state in which a constraint is not satisfied is called an *invalid state*. A plausible state that cannot be reached from the starting state is called an *unreachable state*. A state
that is not already visited in the exploration of the state diagram is called an *unvisited state*. In our presentations start and final states are shown in light green color, unreachable states are not shown, already visited states will not be visited again, intermediate states are shown in light blue color, and invalid states are shown in light red color. If a state diagram is huge we will not show the invalid states.

**Puzzle: Farmer, wolf, goat, and cabbage**

A farmer went to a market and purchased a wolf, a goat, and a cabbage. There is a river he has to cross to bring his purchases home. He has a boat that he can row to carry at most two objects, including himself. In the absence of the farmer, the wolf eats the goat or the goat eats the cabbage. How can the farmer carry himself and all his purchases to the other side of the river safely?

![State diagram for farmer, wolf, goat, and cabbage puzzle.](image)

**Figure 77: State diagram for farmer, wolf, goat, and cabbage puzzle.**

<table>
<thead>
<tr>
<th>Concept</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start/final states</td>
<td>Starting state: [fwgc</td>
</tr>
<tr>
<td>Invalid states</td>
<td>[wg</td>
</tr>
<tr>
<td>State transitions</td>
<td>If #objects on farmer’s side is $k$, then at most $k$ state transitions</td>
</tr>
</tbody>
</table>

**Figure 78: Details of the state diagram for the farmer, wolf, goat, and cabbage puzzle.**

**[Solution.]** The configuration of the entire system can be represented as a state. A state is denoted by a box with left and right chambers that represent the left and right sides of the river, respectively. The four objects ($f =$ farmer, $w =$ wolf, $g =$ goat, $c =$ cabbage) are placed in one of the two chambers. The boat is assumed to be present on the farmer’s side. It is sufficient to show the objects present on the left bank only,
from which the objects present on the right bank can be deduced. However, for ease of reading and understanding, we show the objects on both sides of the river. The starting state is \( [f \ wgc \ | \ ] \) and the final state is \( [\ | \ f\ wgc] \). The state transitions are as follows. If a state has \( k \) objects on the farmer’s side including the farmer, then there are at most \( k \) transitions from that state. This is because the farmer can go to the other river bank alone or with any of the remaining \( k - 1 \) objects. Any state where the wolf eats the goat or the goat eats the cabbage is an invalid state. The state diagram is given in Figure 77 and the details related to the diagram, including the two optimal solutions, are given in Figure 78.

There are two optimal solutions to this puzzle (see Figure 77). Optimal solution takes 7 trips.

[Generalization.] The farmer has to transport \( n \) objects to the other side of the river with some constraints. The boat can carry \( b \) items along from the farmer. How can the farmer transport all \( n \) objects? We can use state diagrams to find all optimal solutions to the puzzle. When \( n \) increases, the size of the state diagram increases tremendously. Some instances of the puzzle might be impossible to solve. This impossibility result can be shown using graphs as follows. Create a graph such that each vertex is an object (not including the farmer) and each edge between two objects represents a conflict such that they cannot be left alone without supervision. We call this graph as the Alcuin graph. A vertex cover in a graph is a set of vertices such that every edge in the graph is incident on a vertex in that set. A minimum vertex cover in a graph is a vertex cover with minimum size. Let the size of the minimum vertex cover for the Alcuin graph be \( m \). Let \( b_{\text{min}} \) be the minimum number of items that can be carried on the boat for the puzzle. Then \( m \leq b_{\text{min}} \leq m + 1 \). If \( b < b_{\text{min}} \), then that instance of the puzzle cannot be solved.

Puzzle: Missionaries and cannibals

Three missionaries and three cannibals must cross a river. They have a boat that can carry at most two people. At each bank, the number of cannibals can never be more than the number of missionaries. In case the cannibals outnumber the missionaries, the cannibals will eat the missionaries. How can all the missionaries and cannibals cross the river safely minimizing the number of trips?

[Solution.] In this puzzle, a state is denoted by a box with left and right chambers that represent the left and right sides of the river, respectively. The two types of people \( \{m = \text{missionary, } c = \text{cannibal}\} \) are placed in both the chambers such that the total number of missionaries is three and the total number of cannibals is three. An underline in a chamber represents the boat location. The starting state is \( [3m \ 3c \ | \ 0m \ 0c] \) and the final state is \( [0m \ 0c \ | \ 3m \ 3c] \). In the state transitions, \( 1m, 2m, 1c, 2c, \) or \( 1m \ 1c \) can cross the river. If cannibals outnumber missionaries in any bank of any state, then that state is an invalid state and we do not show such invalid states in the state diagram. The state diagram is given in Figure 79 and the details related to the
**Figure 79: State diagram for missionaries and cannibals puzzle.**

<table>
<thead>
<tr>
<th>Concept</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start/final states</td>
<td>Starting state: $[3m\ 3c \</td>
</tr>
<tr>
<td>Invalid states</td>
<td>$[1m\ 2c \</td>
</tr>
<tr>
<td>State transitions</td>
<td>Boat can be either on the left bank or on the right bank. There are at most 5 state transitions from any state. Transitions can be $1m$, $2m$, $1c$, $2c$, or $1m\ 1c$.</td>
</tr>
</tbody>
</table>

Figure 80: Details of the state diagram for the missionaries and cannibals puzzle.

There are four optimal solutions to this puzzle (see Figure 79). Optimal solution takes 11 trips.

**Original puzzle: Jealous husbands**

[Solution.] This puzzle is similar to missionaries and cannibals puzzle, but is slightly more complicated due to the numbering of the three couples. In this puzzle, a state is denoted by a box with left and right chambers that represent the left and right sides of the river, respectively. The two types of people ($h = \text{husband}, w = \text{wife}$) are placed in both the chambers. An underline in a chamber represents the boat location.

The starting state is $[h_1\ h_2\ h_3\ w_1\ w_2\ w_3]$ and the final state is $[\ | h_1\ h_2\ h_3\ w_1\ w_2\ w_3]$. In the state transitions, $h_1 w_1$, $h_2 w_2$, $h_3 w_3$, $w_1$, $w_2$, $w_3$, $w_1 w_2$, $w_1 w_3$, or $w_2 w_3$ can cross the
river. So there are at most 9 possible state transitions from a state. If at any time, a woman is in the company of a man unless her own husband is present, then that state is an invalid state and we do not show such invalid states in the state diagram. The state diagram is given in Figure 81.

| There are 486 optimal solutions to this puzzle (see Figure 81).  
| Optimal solution takes 11 trips. |

### Puzzle: Burning ropes

You are stranded in a jungle and you want to cook food. You have the necessary ingredients such as firewood, pot, water, food items, lighter, etc to cook food. The food must be cooked for exactly 45 minutes. Unfortunately your clock is not working and there is no other standard way to measure time such as using mobiles, laptops, Internet, etc. You realize that you have two ropes and each rope burns for exactly 1 hour. But each rope burns nonuniformly along its length. Assume that you can burn the ropes from one or both of its ends. How do you measure exactly 45 minutes using the two ropes and a lighter?

**[Solution.]** In this puzzle, a state is denoted by a box with upper and lower chambers with some information related to ropes as follows. In the upper chamber, term $i$ represents that rope $i$ has none of its ends burning, term $\overline{i}$ represents that rope $i$ has one of its ends burning, and term $\overline{\overline{i}}$ represents that rope $i$ has both of its ends burning. The value below a term represents the length of time remaining for that term. A state transition can happen in any of the 8 possible ways:

1: burn rope 1 from one end until the rope burns out completely,
1: burn rope 1 from both ends until the rope burns out completely,
2: burn rope 2 from one end until the rope burns out completely,
2: burn rope 2 from both ends until the rope burns out completely,
12: burn rope 1 from one end and rope 2 from one end until one of the ropes burns out completely,
12: burn rope 1 from one end and rope 2 from both ends until one of the ropes burns out completely,
12: burn rope 1 from both ends and rope 2 from one end until one of the ropes burns out completely, and
12: burn rope 1 from both ends and rope 2 from both ends until one of the ropes burns out completely.

We also add the time taken for each state transition as the edge cost. Observe that at least one rope is completely burned out in every state transition. It is also important to note that there can be multiple (i.e., more than 1) edges between two states. However, in a regular graph there is at most one edge between any two vertices.

The state diagram is given in Figure 82 and its details are given in Figure 83. The starting and the final states are shown in the diagrams. Our objective is to generate all possible optimal paths from the starting state to the final state such that the path cost (i.e., sum of the edge costs in the path) is 45 min and the path has the least number of edges.
Start/final states
Starting state: \([h_1 h_2 h_3 w_1 w_2 w_3]\) final state: \([h_1 h_2 h_3 w_1 h_2 w_3]\)

State transitions
At most 9 state transitions: \(h_1 w_1, h_2 w_2, h_3 w_3, w_1, w_2, w_3, w_1 w_2, w_1 w_3, w_2 w_3\)

Figure 81: State diagram for the jealous husbands puzzle.
**Mathematical and Algorithmic Puzzles**

**Figure 82:** State diagram for the burning ropes puzzle.

**Figure 83:** Details of the state diagram for the burning ropes puzzle.

There are two optimal solutions to this puzzle (see Figure 82).
Optimal solution takes two steps.

**Puzzle: Colored octopuses**

There are three types of octopuses: 5 red, 4 blue, and 3 green. When two octopuses of different colors meet, they both change to the third color. What is the minimum number of meets required for all octopuses to acquire the same color?

**[State diagram solution.]** We can use state diagrams to solve the puzzle. In this puzzle, a state is denoted by a box with left, middle, and right chambers that represent the number of red, blue, and green octopuses, respectively. A state transition can happen in any of the three ways:

- If \( b \geq 1 \) and \( c \geq 1 \), then \([a, b, c] \rightarrow [a + 2, b - 1, c - 1]\) (more red octopuses)
- If \( a \geq 1 \) and \( c \geq 1 \), then \([a, b, c] \rightarrow [a - 1, b + 2, c - 1]\) (more blue octopuses)
- If \( a \geq 1 \) and \( b \geq 1 \), then \([a, b, c] \rightarrow [a - 1, b - 1, c + 2]\) (more green octopuses)

We see that each of these transitions does not change the total number of octopuses.
That is, the total number of octopuses remains $5 + 4 + 3 = 12$ after every transition. The starting state is $[5, 4, 3]$ and the final state is any of $[[12, 0, 0], [0, 12, 0], [0, 0, 12]]$. Because the total number of octopuses is fixed, we can use barycentric coordinate system to show the transition of states as shown in Figure 84. Our aim is starting from $[5, 4, 3]$ and following the blue arrows in the figure to reach one of the corner vertices. It is easy to that it is not possible to go from starting state to any of the corner vertices. For this reason,

There is no solution to this puzzle (see Figure 84).

[Modular arithmetic solution.] We can solve the puzzle using modular arithmetic, specifically congruence modulo 3.

Starting state vector $= [5, 4, 3] = [2, 1, 0] \pmod{3}$
We can find the next state of a transition by adding one of the following vectors:

Red transition vector $= [2, -1, -1] = [2, 2, 2] \pmod{3}$
Blue transition vector $= [-1, 2, -1] = [2, 2, 2] \pmod{3}$
Green transition vector $= [-1, -1, 2] = [2, 2, 2] \pmod{3}$

In congruence modulo 3, the transitions can be written as shown in Figure 85. This shows that the visited states will be $[2, 1, 0] \pmod{3}$ or $[1, 0, 2] \pmod{3}$ or $[0, 2, 1] \pmod{3}$.

The final state in congruence modulo 3 is one of the following vectors:

Final state vector $= [12, 0, 0] = [0, 0, 0] \pmod{3}$
Final state vector $= [0, 12, 0] = [0, 0, 0] \pmod{3}$
Final state vector $= [0, 0, 12] = [0, 0, 0] \pmod{3}$

Figure 85: Modular arithmetic approach.

We see that we will never be able to reach $[0, 0, 0] \pmod{3}$ i.e., the final state. Hence,

There is no solution to this puzzle.

[**System of linear Diophantine equations solution.**] The state diagram solution is computationally expensive. The modular arithmetic solution though cannot be applied to arbitrary instances of the puzzle. The most elegant and powerful approach of solving this puzzle is using the nonnegative integer solutions to a system of linear Diophantine equations. This approach works on arbitrary instances of the puzzle and hence is the most generic solution to the puzzle.

The solution to the puzzle exists if there is a solution in nonnegative integers to the following equation:

$$\text{starting vector} + x \cdot \text{red transition vector} + y \cdot \text{blue transition vector} + z \cdot \text{green transition vector} = \text{final vector}$$

For example, the final state $[12, 0, 0]$ can be reached from the starting state $[5, 4, 3]$ if there are nonnegative integers $x$, $y$, and $z$ such that

$$[5, 4, 3] + x \cdot [2, -1, -1] + y \cdot [-1, 2, -1] + z \cdot [-1, -1, 2] = [12, 0, 0]$$

The vector equation above can be rewritten as a system of linear Diophantine equations:

$$5 + 2x - y - z = 12$$
$$4 - x + 2y - z = 0$$
$$3 - x - y + 2z = 0$$

210
Taking constants to the right side and representing the system as a matrix vector product we have

\[ A \cdot X = B, \text{ where } A = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}, \quad X = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad B = \begin{bmatrix} 7 \\ -4 \\ -3 \end{bmatrix} \]

We can use an online linear Diophantine equation solver to show that the system above cannot be solved in nonnegative integers. Similarly, we write a system of linear Diophantine equations for the final state \([0, 12, 0]\) and for the final state \([0, 0, 12]\) separately and reach the conclusion that all these three systems do not have solutions in nonnegative integers. For this reason,

There is no solution to this puzzle.

**Take-home lessons**

**[State diagram representations.]** State diagrams can be represented using recursion trees, or directed acyclic graphs (DAGs), or graphs (that can contain cycles). What are the pros and cons of different representations? Recursion trees are ideal to clearly show all possible solutions, including non-optimal solutions, if visited states are also depicted in the diagram. Such trees are easy to read and understand. However, visualizing such trees are computationally expensive. DAGs are compact representations of state diagrams if we are interested in generating all possible optimal solutions (e.g.: refer to Figures 77, 79, 81, 82, 130). The visited states are not shown in DAGs and hence DAGs are not good representations of state diagrams for visualizing all non-optimal solutions. Constructing DAGs are computationally less expensive compared with constructing recursion trees. Graphs are ideal representations of state diagrams if we want to look at all the states and all state transitions. Graph representation is not preferred if visualizing solutions is the objective.

**[State diagram applications.]** State diagrams are used as the visual representations of a class of computational models called finite state machines or deterministic finite automata, which are applied in ATMs, vending machines, ticket machines, traffic signal systems, calculators, digital watches, automatic doors, elevators, washing machines, and even search engines. Diagrams very similar to state diagrams are used as the visual representations of the most intuitive and generic computational models called Turing machines, which define computation as a sequence of changing configurations or states. Thoughts, experiences, life, and more generally universe itself might be giant computations and such computations might be representable through diagrams similar to state diagrams.

**Problems**

1. **[Modified farmer puzzle.]** A farmer has to transport a wolf, a goat, a rabbit, and a cabbage. There is a river he has to cross to bring his purchases home. He has a boat that he can row to carry at most two objects, including himself. In the absence
of the farmer, the wolf eats the goat and the rabbit, or the goat and the rabbit eat the cabbage. How can the farmer carry himself and all his purchases to the other side of the river safely?

(Answer: Impossible. Hint: There is a better approach than constructing state diagrams. Show that the number of extra seats in the boat is smaller than the size of the minimum vertex cover for the Alcuin graph.)

2. [15 puzzle.] 15 puzzle is a mechanical sliding puzzle having a $4 \times 4$ grid containing 15 tiles numbered from 1 to 15, leaving one unoccupied tile position. The starting and ending states of the grid are shown in the left and right parts of Figure 86. In every step, a horizontally/vertically adjacent cell is moved to occupy the unoccupied tile position. The goal of the puzzle is to place the tiles in the sorted order in the minimum number of steps.

(Answer: Impossible. Hint: There is a better approach than constructing state diagrams. Show that the parity of the target permutation is different from the parity of the source permutation.)

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12 \\
13 & 14 & 15 & 16
\end{array}
\quad
\begin{array}{cccc}
15 & 14 & 13 & 12 \\
11 & 10 & 9 & 8 \\
7 & 6 & 5 & 4 \\
3 & 2 & 1 & 1
\end{array}
\]

Figure 86: 15 puzzle: (left) starting configuration and (right) ending configuration

References

The earliest known reference to river crossing puzzles is Alcuin’s Latin book Propositiones ad Acuendos Juvenes from 9th century [Hadley and Singmaster, 1992]. The farmer, wolf, goat, and cabbage puzzle, missionaries and cannibals, and jealous husbands puzzle are simple variants of puzzles from that book. The river crossing puzzles were solved using adjacency matrix representation of graph in [Schwartz, 1961] and using state diagrams in [Fraley et al., 1966]. The relationship between the minimum number of extra places on the boat $b_{\text{min}}$ and the minimum vertex cover $m$ is given in [Csorba et al., 2012]. A variant of colored octopuses puzzle first appeared in Kvant magazine (1985) and later in [Tao, 2006]. The bridge crossing puzzle first appeared in [Levmore and Cook, 2003]. Its generalization for $n$ people with arbitrary times and capacity of 2 is given in [Rote, 2002] and [Backhouse, 2011]. The generalization to bridge capacity $c$ is given in [Backhouse and Truong, 2015].
Sand Timers

Problem

You are stranded in a jungle and you want to cook food. You have the necessary ingredients such as firewood, pot, water, food items, lighter, etc to cook food. The food must be cooked for exactly 15 minutes. Unfortunately your clock is not working and there is no other standard way to measure time such as using mobiles, laptops, Internet, etc. You realize that you have two sand timers: one for 11 minutes and another for 7 minutes. How do you measure exactly 15 minutes using the two sand timers minimizing the number of flips?

Solution

This is another cute puzzle which can be solved systematically using state diagrams.

State diagram solution (without using epsilon transitions)

We can use state diagrams to solve the puzzle. In this puzzle, a state is denoted by a box with left and right chambers that represent the amount of time remaining in the 11-min and 7-min sand timers, respectively. A state transition can happen in at most six ways.

- F(11), E(11): Flip the 11-min sand timer and wait for the 11-min sand timer to empty
- F(11), E(7): Flip the 11-min sand timer and wait for the 7-min sand timer to empty
- F(7), E(11): Flip the 7-min sand timer and wait for the 11-min sand timer to empty
- F(7), E(7): Flip the 7-min sand timer and wait for the 7-min sand timer to empty
- F(11, 7), E(11): Flip both sand timers and wait for the 11-min sand timer to empty
- F(11, 7), E(7): Flip both sand timers and wait for the 7-min sand timer to empty

Suppose the current state is \([a, b]\), then the state transitions can be formally represented as follows:

\[
\begin{align*}
\bullet \ [a, b] & \xrightarrow{F(11), E(11)} [0, \max(0, a + b - 11)] \quad (#\text{flips} = 1, \text{transition time} = (11 - a)) \\
\bullet \ [a, b] & \xrightarrow{F(11), E(7)} [\max(0, 11 - a - b), 0] \quad (#\text{flips} = 1, \text{transition time} = b) \\
\bullet \ [a, b] & \xrightarrow{F(7), E(11)} [0, \max(0, 7 - b - a)] \quad (#\text{flips} = 1, \text{transition time} = a) \\
\bullet \ [a, b] & \xrightarrow{F(7), E(7)} [\max(0, a + b - 7), 0] \quad (#\text{flips} = 1, \text{transition time} = (7 - b)) \\
\bullet \ [a, b] & \xrightarrow{F(11, 7), E(11)} [0, \max(0, a - b - 4)] \quad (#\text{flips} = 2, \text{transition time} = (11 - a)) \\
\bullet \ [a, b] & \xrightarrow{F(11, 7), E(7)} [\max(0, 4 - a + b), 0] \quad (#\text{flips} = 2, \text{transition time} = (7 - b)) \\
\end{align*}
\]

Observe that each state transition makes at least one flip and has a nonzero transition time, where transition time is the time taken to empty sand timer(s). Because the time taken for each transition is nonzero we say that the solution does not make use of epsilon transitions.
As per the transitions defined above, it is possible to have multiple directed edges (or transitions) from a state $A$ to a state $B$. For example,

- $[0, 0] \xrightarrow{F(11), E(11)} [0, 0]$ (#flips = 2, transition time = 11)
- $[0, 0] \xrightarrow{F(11), E(11)} [0, 0]$ (#flips = 1, transition time = 11)
- $[0, 0] \xrightarrow{F(7), E(7)} [0, 0]$ (#flips = 1, transition time = 7)

If there are two transitions from state $A$ to state $B$ that have the same transition time but different number of flips, then we can eliminate that transition that has more number of flips. In the example above, we can eliminate the first transition.

Is $[0, 0]$ the starting state in this puzzle? Not necessarily. There is no specific starting state and final state in this puzzle. Any state can be the starting state and any state can be the final state as long as the path cost between the two states is 15 minutes and the number of flips in the path is minimized.

We use a generic graph instead of a directed acyclic graph (DAG) to visualize the state diagram for this puzzle. As there is no specific starting state, we will need to show the paths from all states. If we use a DAG to visualize paths from all states, the resulting diagram might get extremely huge and might look ugly. Hence, we will use a generic graph to represent the state diagram, as shown in Figure 87. To make the diagram less cluttered we use the following shorthand notations: $e_1$: $F(11), E(11)$; $e_2$: $F(11), E(7)$; $e_3$: $F(7), E(11)$; $e_4$: $F(7), E(7)$; $e_5$: $F(11, 7), E(11)$; and $e_6$: $F(11, 7), E(7)$.

A solution is any path in the state diagram that has a path cost of the time to be measured (i.e., 15 in this puzzle). Any solution, i.e., a path between two states $A$ and $B$, is associated with two values: the total number of flips made in that solution and the wait time of the solution, i.e., the time required to reach the state $A$ from the starting state. A waittime-optimal solution is a solution that minimizes the wait time (i.e., 0 in this puzzle) among all solutions and minimizes the total number of flips (i.e., 4 in this puzzle) among all solutions that minimizes the waittime. A flips-optimal solution is a solution that minimizes the total number of flips (i.e., 2 in this puzzle) among all solutions and minimizes the wait time (i.e., 7 in this puzzle) among all solutions that minimizes the number of flips.

For example, four (among many other) solutions to the puzzle are as follows:

1. $[0, 0] \xrightarrow{F(11, 7), E(7)} [4, 0] \xrightarrow{F(7), E(11)} [0, 3] \xrightarrow{F(11, 7), E(7)} [7, 0]$
   (time = 7 + 4 + 4 = 15, wait time = 0, #flips = 3 + 1 + 1 = 5)
2. $[0, 0] \xrightarrow{F(11, 7), E(7)} [4, 0] \xrightarrow{F(7), E(11)} [0, 3] \xrightarrow{F(7), E(7)} [0, 0]$ (waittime-optimal solution)
   (time = 7 + 4 + 4 = 15, wait time = 0, #flips = 2 + 1 + 1 = 4)
3. $[0, 3] \xrightarrow{F(7), E(7)} [0, 0] \xrightarrow{F(11), E(11)} [0, 0]$
   (time = 4 + 11 = 15, wait time = 11, #flips = 1 + 1 = 2)
4. $[4, 0] \xrightarrow{F(7), E(11)} [0, 3] \xrightarrow{F(11), E(11)} [0, 0]$
   (flips-optimal solution)
   (time = 4 + 11 = 15, wait time = 7, #flips = 1 + 1 = 2)

The first two solutions have zero wait time (as they start from the starting state) but the number of flips is not minimum. On the other hand, the last two solutions have the minimum number of flips but have nonzero wait time. In the last two solutions, we have not shown how to reach the states $[10, 0]$ and $[4, 0]$ from the starting
Figure 87: State diagram for the sand timers puzzle without using epsilon transitions.

state [0, 0] in time 23 and 7, respectively. The second solution given above is waittime-optimal because it has the minimum wait time, and the number of flips is minimized among all solutions that have the minimum wait time. The last solution given above is flips-optimal because it has the minimum number of flips, and the wait time is minimized among all solutions that have the minimum number of flips.

There is one optimal solution of each category to this puzzle (see Figure 87). Waittime-optimal solution requires 4 flips. Flips-optimal solution requires 2 flips and has a wait time of 7.
State diagram solution (using epsilon transitions)

In the state diagram without using state transitions solution, every transition has at least one flip. If we define state transitions in a different way we possibly could do improvise on the wait time and/or the number of flips required for the solutions.

Figure 88: State diagram for the sand timers puzzle using epsilon transitions.

In this solution, a state transition can happen in at most four ways:
- F(11): Flip the 11-min sand timer
- F(7) Flip the 7-min sand timer
- E(11): Empty the 11-min sand timer
- E(7): Empty the 7-min sand timer
Suppose the current state is \([a, b]\), then the state transitions can be formally represented as follows:

- \([a, b] \xrightarrow{F(1)} [(11-a), b]\) \hspace{1cm} (#flips = 1, transition time = 0)
- \([a, b] \xrightarrow{F(7)} [a, (7-b)]\) \hspace{1cm} (#flips = 1, transition time = 0)
- If \(a > 0\), then \([a, b] \xrightarrow{E(1)} [0, \max(0, b-a)]\) \hspace{1cm} (#flips = 0, transition time = \(a\))
- If \(b > 0\), then \([a, b] \xrightarrow{E(7)} [\max(0, a-b), 0]\) \hspace{1cm} (#flips = 0, transition time = \(b\))

Several concepts and ideas can be inherited from the previous solution (that does not use epsilon transitions) including the elimination of redundant directed edges between two vertices, using a generic graph instead of a DAG to represent the state diagram, and wait-time- and flips-optimal solutions.

For example, six (among many other) solutions to the puzzle are as follows:

1. \([0, 0] \xrightarrow{F(1)} [11, 0] \xrightarrow{F(7)} [11, 7] \xrightarrow{E(7)} [4, 0] \xrightarrow{F(7)} [4, 7] \xrightarrow{E(1)} [0, 3] \xrightarrow{F(1)} [11, 3] \xrightarrow{F(7)} [11, 4] \xrightarrow{E(7)} [7, 0]\)
   
   (time = 0 + 0 + 7 + 0 + 4 + 0 + 0 + 4 = 15, wait time = 0, #flips = 1 + 1 + 0 + 1 + 0 + 1 + 0 + 1 = 5, #edges = 8)

2. \([0, 0] \xrightarrow{F(1)} [11, 0] \xrightarrow{F(7)} [11, 7] \xrightarrow{E(7)} [4, 0] \xrightarrow{F(7)} [4, 7] \xrightarrow{E(1)} [0, 3] \xrightarrow{F(7)} [0, 4] \xrightarrow{E(7)} [0, 0]\)
   
   (time = 0 + 0 + 7 + 0 + 4 + 0 + 4 = 15, wait time = 0, \textbf{(waittime-optimal solution)}
   
   #flips = 1 + 1 + 0 + 1 + 0 + 1 + 0 = 4, #edges = 7)

3. \([0, 4] \xrightarrow{E(7)} [0, 0] \xrightarrow{F(1)} [11, 0] \xrightarrow{E(1)} [0, 0]\)
   
   (time = 4 + 0 + 11 = 15, wait time = 11, #flips = 0 + 1 + 0 = 1, #edges = 3)

4. \([4, 7] \xrightarrow{E(1)} [0, 3] \xrightarrow{F(1)} [11, 3] \xrightarrow{E(7)} [8, 0] \xrightarrow{E(1)} [0, 0]\)
   
   (time = 4 + 0 + 3 + 8 = 15, wait time = 7, \textbf{(flips-optimal solution)}
   
   #flips = 0 + 1 + 0 + 1 + 0 = 1, #edges = 4)

5. \([4, 0] \xrightarrow{E(1)} [0, 0] \xrightarrow{F(1)} [11, 0] \xrightarrow{E(1)} [0, 0]\)
   
   (time = 4 + 0 + 11 = 15, wait time = 7, \textbf{(flips-optimal solution)}
   
   #flips = 0 + 1 + 0 + 1 = 1, #edges = 3)

6. \([4, 7] \xrightarrow{E(1)} [0, 3] \xrightarrow{F(1)} [11, 3] \xrightarrow{E(1)} [0, 0]\)
   
   (time = 4 + 0 + 11 = 15, wait time = 7, \textbf{(flips-optimal solution)}
   
   #flips = 0 + 1 + 0 = 1, #edges = 3)

The state diagram is given in Figure 88.

There are one waitime-optimal and three flips-optimal solutions (see Figure 88). Waitime-optimal solution requires 4 flips. Flips-optimal solution requires 1 flip and has a wait time of 7.

Let’s compare optimal solutions from the previous approach (without using epsilon transitions) with that of this approach (using epsilon transitions). The waittime-optimal solution from both approaches are the same, i.e., has 4 flips and the order of flips and emptying is the same. However, the flips-optimal solutions from both approaches are completely different in the following ways. First, the optimal number of flips is 2 using the previous approach whereas it is 1 using the current approach. Second, the number of flips-optimal solutions is 1 using the previous approach whereas it is 3 using the current approach. For this reason, the current approach is better than the previous approach. Hence, defining state transitions in a proper way is so important to obtain the optimal solutions.
Algorithm to compute all flips-optimal solutions for state diagram method (using epsilon transitions)

Computing waittime-optimal solutions is easy. How? To compute waittime-optimal solutions, we run Dijkstra’s single-source shortest-path algorithm with the starting node \((0,0)\) as the source vertex and stopping when the path length equals the time to be measured.

In comparison, computing flips-optimal solutions is complicated. How? As per the definition, flips-optimal solutions consist of all paths whose path cost is the time to be measured, which minimize the number of flips (among all paths with the same path cost), and which minimize the waittime among all those paths that minimize the number of flips. Computing flips-optimal solutions is a version of the shortest path problem that is extremely unique. The types of shortest-path problems one learns in standard undergraduate and graduate courses on computer science are as follows:

- One-to-one (1 vertex is a source and 1 vertex is a target)
- One-to-many (1 vertex is a source and all vertices are targets)
- Many-to-one (All vertices are sources and 1 vertex is a target)
- Many-to-many (All vertices are sources and all vertices are targets)

Observe that the problem of computing the flips-optimal solutions does not fit into any of the above standard types. *The aesthetic beauty of the problem is due to the fact that this is a shortest path problem with no specific source vertex.* This implies that the source vertex of our shortest path problem can be any of the infinitely possible states. But how is it possible that the number of states is infinite? Isn’t it finite because each state is of the form \((\ell, r)\), where \(\ell\) and \(r\) are the amounts of time left in the left and right sand timers, respectively? Good question. The answer depends on the specific problem. If we are computing waittime-optimal solutions then we can define a state to be of the form \((\ell, r)\). In comparison, if we are computing flips-optimal solutions, then it would be easier to solve the problem if we define a state to be of the form \(((\ell, r), t)\), where \(t\) denotes the cost of the path (i.e., total time taken) from \((0,0)\) to \((\ell, r)\).

The algorithm to compute all flips-optimal solutions for the state diagram method (using epsilon transitions) consists of the following steps:

**[Stage 1. Check solvability.]** Suppose \(L, R\) denote the capacities of the left and right sand timers in minutes, respectively and \(T\) denote the amount of time to be measured. It can be proved that the puzzle is solvable if and only if \(T\) is divisible by \(\text{GCD}(L, R)\).

**[Stage 2. Construct the directed graph.]** In this stage, given the capacities \((L, R)\) of the left and right sand timers in minutes, we construct a directed graph \(G\) with vertices and edges using the *breadth-first search* as shown in the CREATEGRAPH algorithm. The graph \(G\) is represented as an adjacency-list and implemented as a hash map.

Each vertex \(u\) has the following attributes:

- \(u.status\): form \((\ell, r)\) to denote the time left in the left and right sand timers.
- \(u.waittime\): minimum time required to reach vertex \(u\) from the starting vertex.
- \(u.sequence\): sequence of vertices starting from \((0,0)\) to this vertex.
**CREATEGRAPH**(*L, R*)

**Input:** *L, R* ∈ ℤ are the amount of time of the left and right sand timers.
**Output:** Directed graph.
1. Create *G*, i.e., graph based on adjacency-list implemented as a hash map
2. Create a hash set to store the visited nodes
3. Create a starting node *u* with *u*.status ← (0, 0) and *u*.waittime ← 0
4. Create a queue *Q* and initialize it with the starting node *u*
5. **while** *Q* is nonempty **do**
6.   *u* ← *Q*.Pop()
7.   Set (*ℓ*, *r*) ← *u*.status, *t* ← *u*.waittime, and mark *u* as visited
8.   // **Transition 1.** Flip left sand timer .................................
9.   Create *u*’s neighbor vertex *v*; Set *v*.status ← (*L* − *ℓ*, *r*)
10.  **UPDATE**\textsc{NEIGHBOR}\textsc{VERTEX}(*Q, u, v, t*)
11.  Insert into graph *G* the directed edge *e* = (*u*, *v*) such that
12.      *e*.action ← Flip(*L*), *e*.flips ← 1, *e*.time ← 0
13.  // **Transition 2.** Flip right sand timer ...............................  
14.  Create *u*’s neighbor vertex *v*; Set *v*.status ← (*ℓ*, *R* − *r*)
15.  **UPDATE**\textsc{NEIGHBOR}\textsc{VERTEX}(*Q, u, v, t*)
16.  Insert into graph *G* the directed edge *e* = (*u*, *v*) such that
17.      *e*.action ← Flip(*R*), *e*.flips ← 1, *e*.time ← 0
18.  // **Transition 3.** Empty left sand timer ............................... 
19.  **if** *ℓ* > 0 **then**
20.     Create *u*’s neighbor vertex *v*; Set *v*.status ← (0, max(0, *r* − *ℓ*))
21.     **UPDATE**\textsc{NEIGHBOR}\textsc{VERTEX}(*Q, u, v, t + *ℓ*)
22.     Insert into graph *G* the directed edge *e* = (*u*, *v*) such that
23.      *e*.action ← Empty(*L*), *e*.flips ← 0, *e*.time ← *ℓ*
24.  // **Transition 4.** Empty right sand timer ............................... 
25.  **if** *r* > 0 **then**
26.      Create *u*’s neighbor vertex *v*; Set *v*.status ← (max(0, *ℓ* − *r*), 0)
27.      **UPDATE**\textsc{NEIGHBOR}\textsc{VERTEX}(*Q, u, v, t + *r*)
28.      Insert into graph *G* the directed edge *e* = (*u*, *v*) such that
29.      *e*.action ← Empty(*R*), *e*.flips ← 0, *e*.time ← *r*
30.  **return** *G*

**UPDATE**\textsc{NEIGHBOR}\textsc{VERTEX}(*Q, u, v, t*)

1. **if** *v* is unvisited **or** *t* < *v*.waittime **then**
3.     **if** *v* is unvisited **then** *Q*.Push(*v*)

Each edge *e* has the following attributes:
- *e*.action: one of the four transitions: Flip(*L*), Flip(*R*), Empty(*L*), Empty(*R*).
- *e*.flips: #flips in the transition: either 1 or 0.
e.time: transition time: either 0 or p (if a sand timer having p time left is emptied).

We create the graph using the breadth-first search algorithm. As per the algorithm, the graph is built in concentric circles using the queue Q, i.e., starting from the vertex (0, 0), all vertices are visited that are at a 1-edge distance, then all vertices are visited that are at a 2-edge distance, and so on until all vertices are visited. In this process, the attributes of all vertices and edges are populated.

[Stage 3. Compute all flips-optimal solutions.] In this stage, we will compute all flips-optimal solutions using the COMPUTEFLIPSOPTIMALSOLUTIONS algorithm.

[Data structures.] We use a few data structures to efficiently solve this problem.

The core concept of this algorithm is paths. So, first, we define a path object. Each path p has the following attributes:

- p.source: status of the source vertex a of this path.
- p.destination: status of the destination vertex b of this path.
- p.time: total time taken in this path.
- p.flips: #flips in this path.
- p.waittime: same as a.waittime; shortest time to reach vertex a from vertex 0.
- p.sequence: sequence of vertices in this path.

We then use a min-heap H that stores paths. In the min-heap H, the paths are ordered by the number of flips first and then by waittime. In other words, if $p_1$ and $p_2$ are two paths in min-heap H, then the two paths can be compared as follows:

\[
\begin{cases} 
    p_1 < p_2 & \text{if } p_1.\text{flips} < p_2.\text{flips} \text{ or } (p_1.\text{flips} = p_2.\text{flips} \text{ and } p_1.\text{waittime} < p_2.\text{waittime}), \\
    p_1 = p_2 & \text{if } p_1.\text{flips} = p_2.\text{flips} \text{ and } p_1.\text{waittime} = p_2.\text{waittime}, \\
    p_1 > p_2 & \text{otherwise.}
\end{cases}
\]

Second, we define a state corresponding to a path. Each state s corresponding to a path p has the following attributes:

- s.status: same as p.destination.
- s.time: same as p.time.

We then define a hash map called stateflips that maps a state with the minimum number of flips to reach that state.

[Algorithm.] We are now ready to present the algorithm, which is a complicated variant of the all-source best-first search algorithm.

[Step 1. Initialize.] Initially, for each vertex u in the graph G, we define a path p that both start and end at vertex u, insert this path p into the min-heap H, define a state s corresponding to path p, and add its entry into stateflips. We maintain three variables related to the desired results:

- flips_min: #flips of the flips-optimal solutions.
- waittime_min: waittime of the flips-optimal solutions.
- solutions: list of all flips-optimal solutions.

[Step 2. Run a variant of all-source best-first search.] While the min-heap H is nonempty, do the following. Remove a path from H with the highest priority (minimize #flips first, waittime second) and call this path as p. If $p.\text{flips} > \text{flips}_\text{min}$ or if $p.\text{flips} \leq \text{flips}_\text{min}$ and $p.\text{time} = \text{targettime}$, then go to Step 3. On the other hand, if
**Input:** Graph $G$ and the time to be measured $\text{targettime}$.  
**Output:** All flips-optimal solutions.

1. // Create a min-heap $H$ with paths; The heap is ordered based on flips first and wait time second ..........................
2. // Create a hash map $\text{stateflips}$ that maps the visited states with the minimum number of flips required to reach those states ............
3. for each vertex $u$ in directed graph $G$ do
   4. Create a path $p$ corresponding to vertex $u$ as:
      5. $(p.\text{flips}, p.\text{waittime}, p.\text{source}, p.\text{destination}, p.\text{time}, p.\text{sequence}) \leftarrow (0, u.\text{waittime}, u.\text{status}, u.\text{status}, 0, u.\text{status})$
      6. $H.$Insert($p$)
   7. Create a state $s \leftarrow [p.\text{destination}, p.\text{time}]$; $\text{stateflips}[s] \leftarrow 0$
8. // All flips-optimal solutions are stored in $\text{solutions}$ ................
9. $\text{flips}_\text{min} \leftarrow \infty$; $\text{waittime}_\text{min} \leftarrow \infty$; $\text{solutions} \leftarrow \left[\right]$
10. while heap $H$ is nonempty do
    11. $p \leftarrow H.$DeleteMin()
    12. // Stop: All flips-optimal solutions have been explored ..........
    13. if $p.\text{flips} > \text{flips}_\text{min}$ then break
    14. else if $p.\text{time} = \text{targettime}$ then
       15. // Stop: All flips-optimal solutions have been explored ......
       16. if $p.\text{waittime} > \text{waittime}_\text{min}$ then break
       17. $\text{flips}_\text{min} \leftarrow p.\text{flips}; \text{waittime}_\text{min} \leftarrow p.\text{waittime}; \text{solutions}.\text{Insert}(p.\text{sequence})$
    18. else if $p.\text{time} < \text{targettime}$ then
       19. Let $u$ be the vertex such that
          20. $u.\text{status} \leftarrow p.\text{destination}$ and $u.\text{waittime} \leftarrow p.\text{waittime}$
       21. for each neighbor vertex $v$ of vertex $u$ in graph $G$ do
          22. Let $e = (u, v)$ be the directed edge in graph $G$
             23. Create a path $p'$ corresponding to vertex $v$ as:
                24. $p'.\text{flips} \leftarrow p.\text{flips} + e.\text{flips}$
                25. $p'.\text{waittime} \leftarrow p.\text{waittime}$
                26. $p'.\text{source} \leftarrow p.\text{source}$
                27. $p'.\text{destination} \leftarrow v.\text{status}$
                28. $p'.\text{time} \leftarrow p.\text{time} + e.\text{time}$
                29. $p'.\text{sequence} \leftarrow \text{Concatenate}(p.\text{sequence}, [v.\text{status} : p'.\text{time}])$
             30. Create a state $s' \leftarrow [p'.\text{destination}, p'.\text{time}]$
             31. if $p'.\text{time} \leq \text{targettime}$ and ($s'$ not in $\text{stateflips}$ or $p'.\text{flips} \leq \text{stateflips}[s']$) then
                32. $H.$Insert($p'$)
                33. $\text{stateflips}[s'] \leftarrow p'.\text{flips}$
       21. return $\text{solutions}$

221
If \( p.\text{flips} \leq \text{flips}_{\text{min}} \) and \( p.\text{time} < \text{target time} \), then we need to expand the path in multiple ways using the neighboring vertices of \( p.\text{destination} \). Let \( p.\text{source} \) be the vertex \( q \) and \( p.\text{destination} \) be the vertex \( u \). Then for each neighboring vertex \( v \) of \( u \), we consider the path from \( q \) to \( v \) via \( u \) as path \( p' \) and get all its attributes as shown in Figure 89, which covers the core logic of the entire algorithm. We define a state \( s' \) that corresponds to path \( p' \) using \( p'.\text{destination} \) and \( p'.\text{time} \). If \( p'.\text{time} \) is greater than \( \text{target time} \), we do not care for this path \( p' \) and hence we do not add the path \( p' \) to the min-heap \( H \). Similarly, if \( s' \) is present in \( \text{state flips} \) and \( p'.\text{flips} > \text{state flips}[s'] \), then it means that \( p' \) is a path which requires more number of flips than another path that reaches the same destination with the same total time, so we do not care for this path \( p' \) and hence we do not add the path \( p' \) to the min-heap \( H \). We add the path \( p' \) to the min-heap \( H \) in every other case and also update \( \text{state flips}[s'] \) with \( p'.\text{flips} \).

When the min-heap \( H \) is empty, we return the list of all flips-optimal solutions stored in \( \text{solutions} \).

\[ p'.\text{flips} = \text{#flips in } qu + \text{#flips in } uv \]
\[ p'.\text{waittime} = p.\text{waittime} = \text{total time of } 0q \]
\[ p'.\text{source} = p.\text{source} = \text{starting vertex of path } p' \text{ is } q \]
\[ p'.\text{destination} = \text{ending vertex of path } p' \text{ is } v \]
\[ p'.\text{time} = \text{total time of } qu + \text{time of } uv \]
\[ p'.\text{sequence} = \text{sequence of vertices from } q \text{ till } v, \text{ via } u \]

Figure 89: Expansion of path \( p \) to path \( p' \).

[Step 3. Terminate or accept solutions.] If \( p.\text{flips} > \text{flips}_{\text{min}} \), then it means that every path that has been stored in the min-heap \( H \) has greater \#flips than \( \text{flips}_{\text{min}} \). This implies that none of those paths can be flips-optimal solutions and hence we stop and return the \( \text{solutions} \) that have been explored till that point. On the other hand, if \( p.\text{flips} \leq \text{flips}_{\text{min}} \) and \( p.\text{time} = \text{target time} \), then we check if \( p.\text{waittime} > \text{waittime}_{\text{min}} \). If \( p.\text{waittime} > \text{waittime}_{\text{min}} \), then it means that every path that has been stored in the min-heap \( H \) has greater waittime than \( \text{waittime}_{\text{min}} \). This implies that none of those paths can be flips-optimal solutions and hence we stop and return the \( \text{solutions} \) that have been explored till that point. In contrast, if \( p.\text{waittime} \leq \text{waittime}_{\text{min}} \), then we have found a flips-optimal solution and hence we add that path \( p.\text{path} \) to \( \text{solutions} \).

[Example.] We will show how the algorithm computes all flips-optimal solutions by applying it on a simple example. Suppose we want to measure 4 minutes using two sand timers of capacity 5 and 3 minutes. The execution of the algorithm for this example is shown in Figure 50. We first insert all paths that start from a vertex and
end at the same vertex into a min-heap – these paths have 0 flips and 0 time but can have nonzero waittime. We then keep delete-min’ing and inserting paths into the min-heap as per the algorithm. The figure shows the step-by-step order in which the paths are removed from the heap. We are interested in flips-optimal solutions i.e., the paths that can measure 4 minutes but which have minimum #flips first and then minimum waittime second among all solutions that measure 4 minutes. There are three flips-optimal solutions in this example and they are removed from the min-heap in steps 31, 32, and 35. We use the notation [(a, b) : c] to mean the vertex (a, b) is reached after c time measured from a defined source. When we delete-min a path from the min-heap that has either more #flips or more waittime than a flips-optimal solution, then we stop the execution. In this example, we encounter such a path in step 37 and hence we stop executing the algorithm.

**Problems**

1. [*Linear Diophantine equation solution.*] It seems like the sand timers puzzle can be solved using the solutions to a linear Diophantine equation. Design an algorithm, if it exists, to the generalized n sand timers puzzle that uses the solutions to a linear Diophantine equation \(a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = c\), where natural number c is the time to be measured, and natural numbers \(a_1, a_2, \ldots, a_n\) are the capacities of the sand timers.

**References**

The sand timers puzzle has appeared in [Gardner, 2020] and [Wells, 1993].
<table>
<thead>
<tr>
<th>Step</th>
<th>#Flips</th>
<th>Waittime</th>
<th>Time</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[0,0 : 0]</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[0,3 : 0]</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[5,0 : 0]</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[5,3 : 0]</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>E(3) [(0,3 : 0)] [0,0 : 3]</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>E(3) [(5,3 : 0)] [2,0 : 3]</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>[2,3 : 0]</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>[3,0 : 0]</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>E(5) [(2,3 : 0)] [(0,1) : 2]</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>E(3) [(2,3 : 0)] [(0,0) : 3]</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>E(5) [(3,0 : 0)] [(0,0) : 3]</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>E(5) [(2,3 : 0)] E(5) [(0,1) : 2] E(3) [(0,0) : 3]</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>[2,0 : 0]</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>[3,3 : 0]</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>E(5) [(2,0 : 0)] [(0,0) : 2]</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>E(3) [(3,3 : 0)] [(0,0) : 3]</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>[5,2 : 0]</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>[5,1 : 0]</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>[0,2 : 0]</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>5</td>
<td>2</td>
<td>E(3) [(5,2 : 0)] [(3,0) : 2]</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>E(3) [(5,1 : 0)] [(4,0) : 1]</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>5</td>
<td>2</td>
<td>E(3) [(0,2) : 0] [(0,0) : 2]</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>[(0,1) : 0]</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>E(3) [(0,1) : 0] [(0,0) : 1]</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>[(1,3) : 0]</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>[(4,3) : 0]</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>[(1,0) : 0]</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>[(4,0) : 0]</td>
</tr>
<tr>
<td>29</td>
<td>0</td>
<td>6</td>
<td>3</td>
<td>E(3) [(4,3) : 0] [1,0 : 3]</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>[1,0 : 0] E(5) [(0,0) : 1]</td>
</tr>
<tr>
<td>31</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>E(5) [(0,0) : 4]</td>
</tr>
</tbody>
</table>

(Path to source: [(0,0) : 0] F(5) [(5,0) : 0] F(3) [(5,3) : 0] F(4) [(2,0) : 3] F(3) [(2,3) : 3] F(5) [(0,1) : 5] F(5) [(5,1) : 5] [4,0 : 6] F(3) [4,3 : 6])

| 32   | 0      | 6        | 4    | E(3) [(4,3) : 0] E(5) [(1,0) : 3] E(5) [(0,0) : 4] |

(Path to source: [(0,0) : 0] F(5) [(5,0) : 0] F(3) [(5,3) : 0] F(4) [(2,0) : 3] F(3) [(2,3) : 3] F(5) [(0,1) : 5] F(5) [(5,1) : 5] [4,0 : 6] F(3) [4,3 : 6])

| 33   | 0      | 6        | 1    | E(5) [(1,3) : 0] [(0,2) : 1] |
| 34   | 0      | 6        | 3    | E(3) [(1,3) : 0] [(0,0) : 3] |
| 35   | 0      | 6        | 4    | E(5) [(4,3) : 0] [(0,0) : 4] |

(Path to source: [(0,0) : 0] F(5) [(5,0) : 0] F(3) [(5,3) : 0] F(4) [(2,0) : 3] F(3) [(2,3) : 3] F(5) [(0,1) : 5] F(5) [(5,1) : 5] [4,0 : 6] F(3) [4,3 : 6])

| 36   | 0      | 6        | 3    | E(3) [(1,3) : 0] [(0,2) : 1] E(3) [(0,0) : 3] |
| 37   | 1      | 0        | 3    | E(3) [(5,3) : 0] F(3) [(2,0) : 3] F(3) [(2,3) : 3] |

Table 50: Flips-optimal solutions to measure 4 min using (5,3) min sand timers.
Water Pouring

Problem

There are two empty containers: a 3-liter container and a 5-liter container. There is a fresh pond with water that can be used to either fill up or empty a container. The three types of operations we can perform are: filling up a container, emptying a container, and transferring water from one container to other. How can we measure exactly 4 liters of water using the two containers and the pond minimizing the total number of operations?

Solution

Water pouring puzzles are also called 
jug pouring or liquid transfer or decanting puzzles. They are entertaining and challenging. Such puzzles can teach us a variety of algorithmic design techniques.

Trial-and-error solution

Many people can answer the puzzle using a trial-and-error solution. Figure 90 gives a non-optimal solution to measure 4 liters in 8 steps. Figure 91 gives the optimal solution to measure 4 liters in just 6 steps.

Here is the optimal solution. Initially, both the 3-liter and the 5-liter containers are empty. In the first step, we fill the 5-liter container with water. In the second step, we transfer water from the 5-liter container to the 3-liter container until the 3-liter container is full. Following the steps as shown in Figure 91, we find a container with exactly 4 liters of water. We find from the table that after 6th step, we will have 4 liters in the 5-liter container. Thus the problem is optimally solved in 6 steps.

\[
\begin{array}{c|cccccccc}
\text{Step} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
\text{3-liter} & 0 & 3 & 0 & 3 & 1 & 1 & 0 & 3 & 0 \\
\text{5-liter} & 0 & 0 & 3 & 3 & 5 & 0 & 1 & 1 & 4 \\
\end{array}
\]

Figure 90: Non-optimal solution.

\[
\begin{array}{c|ccccccc}
\text{Step} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
\text{3-liter} & 0 & 0 & 3 & 0 & 2 & 2 & 3 \\
\text{5-liter} & 0 & 5 & 2 & 2 & 0 & 5 & 4 \\
\end{array}
\]

Figure 91: Optimal solution.

State diagram solution

We can use state diagrams to solve the puzzle.

In this puzzle, a state is denoted by a box with left and right chambers that represent the amount of water in the 5-liter and 3-liter containers, respectively. A state transition can happen in at most three ways. (i) Fill: filling a container, (ii) Empty: emptying a container, and (iii) Transfer: transferring from one container to another till the former becomes empty or the latter becomes full. The starting state is [0, 0].
Figure 92: State diagram for water pouring using a pond puzzle.

Figure 93: Graphical diagram for water pouring using a pond puzzle.
and the final state is \([4, -]\) because if we measure 4 liters it has to be in the 5-liter container and we do not care for the amount of water in the 3-liter container. The state diagram is given in Figure 92 and its details are given in Figure 94.

There is one optimal solution to this puzzle (see Figure 92).
Optimal solution takes 6 operations.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start/final states</td>
<td>Starting state: ([0, 0]), Final state: ([4, -])</td>
</tr>
<tr>
<td>State transitions</td>
<td>There are at most 4 state transitions</td>
</tr>
</tbody>
</table>

Figure 94: Details of the state diagram for water pouring using a pond puzzle.

We can also construct a graph as shown in Figure 93 to depict the states. The states are the vertices of a rectangular grid with the bottom-left vertex at \([0, 0]\) and the top-right vertex at \([5, 3]\). The vertical and horizontal axes represent the amount of water in 5-liter and 3-liter containers, respectively. A directed edge from one vertex to another vertex represents a state transition in the direction of the arrow. An edge with arrows at both ends represents that the state transition is possible in both directions. Observe that the graph has only the border vertices of the rectangular grid and has no internal vertices such as \([2, 1]\) and \([4, 2]\) because these states are not possible given the constraints of the puzzle. The graph contains only those vertices where one of the coordinate values is either empty (i.e., 0) or full (i.e., 3 or 5).

The objective is to start from state \([0, 0]\) and reach \([4, -]\) minimizing the number of edges in the path. An optimal solution is shown in the diagram using red edges.

[Breadth-first search.] The general puzzle can be solved by constructing the graph on-the-fly, executing the breadth-first search (BFS) algorithm, and finding the shortest path to a node in which the sum of contents in all of the containers equals \(k\).

Let \(V\) and \(E\) denote the sets of nodes (or vertices) and edges in the graph, respectively. Let \(b\) denote the branching factor and \(d\) denote the distance between the source and the destination nodes. We have \(|V| \leq (c_1 + 1)(c_2 + 1) \cdots (c_n + 1)\), \(|E| = O(|V|)\), and \(b \leq 3\). Then, the space and time complexity of the BFS algorithm is \(O(b^{d+1}) = O(3^d)\).

Averbach-Chien’s Diophantine equation solution (2 containers, optimal)

What does the following equation mean to you?

\[
3x + 5y = 4 \quad (26)
\]

Did you know that this 1-inch long equation leads to a beautiful and elegant mathematical strategy to solve the puzzle? In this section, we will understand how to solve the puzzle using equations like the above, called Diophantine equations, in which there are more unknowns than the number of equations.

[Core idea.] Let’s understand the hidden relationship between Equation 26 and our puzzle. Let \(x, y\) be integers to denote the number of times the two containers are filled or emptied. The absolute value \(|x|\) denotes the net number of times the 3-liter
container is completely filled from the pond (if \( x > 0 \)) or completely emptied (if \( x < 0 \)). Similarly, the absolute value \( |y| \) denotes the net number of times the 5-liter container is completely filled from the pond (if \( y > 0 \)) or completely emptied (if \( y < 0 \)). Then, \( 3x + 5y \) denotes the net number of liters that is removed from the pond. As we would like to measure exactly 4 liters of water, we set \( 3x + 5y = 4 \). This equation is called a linear Diophantine equation in two unknowns.

[Problem-solving.] We need to check if the puzzle is solvable (or the desired amount measurable) before attempting to find the solution. The reason for this step is that if there is no solution to the puzzle, we don’t want to spend an unlimited amount of time searching for a solution and wasting our infinitely precious lives. Hence, before trying to measure 4 liters of water, we must be sure that we can in fact measure 4 liters of water. A quantity of 4 liters is measurable if after a sequence of feasible steps we can get 4 liters of water in a container.

The puzzle has a solution if its corresponding equation \( 3x + 5y = 4 \) has a solution in integers. From the theory of linear Diophantine equations, an equation \( ax + by = k \) has an infinite number of integer solutions if \( k \) is divisible by \( \text{GCD}(a, b) \), where \( \text{GCD} \) denotes the greatest common divisor. As 4 is divisible by \( \text{GCD}(3, 5) = 1 \), the puzzle is solvable and it has an infinite number of solutions.

There are an infinite number of integral solutions to Equation 26:

\[
\begin{align*}
3 \times (-7) + 5 \times (+5) &= 4 \\
3 \times (-2) + 5 \times (+2) &= 4 \\
3 \times (+3) + 5 \times (-1) &= 4 \\
3 \times (+8) + 5 \times (-4) &= 4
\end{align*}
\]

As we want to minimize the number of steps to measure 4 liters, we are interested in only those solutions that reduce the number of times the containers are filled and/or emptied. Hence, we select only those values of \( x \) and \( y \) that are as close to 0 as possible. So we select two solutions: \( x = -2 \) and \( y = +2 \) and \( x = +3 \) and \( y = -1 \). We have

\[
\begin{align*}
3 \times (-2) + 5 \times (+2) &= 4 \\
3 \times (+3) + 5 \times (-1) &= 4
\end{align*}
\]

These two equations denote two solutions to our puzzle that are possible candidates for an optimal solution. Let’s first understand how these equations can be used to design algorithms for solving our puzzle. Then we will investigate on which of these two solutions might be optimal.

Consider the equation \( 3 \times (-2) + 5 \times (+2) = 4 \). Here, \( x = -2 \) and \( y = +2 \) means that we completely fill the 5-liter container twice and empty the 3-liter container twice. But how can we fill the 5-liter container twice? Simple. Whenever we fill the 5-liter container, we transfer the water to the 3-liter container as many times as necessary. We can only fill the 5-liter container when it is completely empty. Whenever the 3-liter container is full, we empty it. Similarly, the equation \( 3 \times (+3) + 5 \times (-1) = 4 \) can be interpreted as a solution to the puzzle.
WATERPOURING gives an algorithm for solving the puzzle. The pseudocode for the algorithm covers the ideas we have already discussed and is self-explanatory. The algorithm corresponding to the equation \(3 \times (-2) + 5 \times (+2) = 4\) is invoked as WATERPOURING(5, 3, 4) and the algorithm corresponding to the equation \(3 \times (+3) + 5 \times (-1) = 4\) is invoked as WATERPOURING(3, 5, 4). Both the algorithms can be used to measure 4 liters of water.

One of the two runs: WATERPOURING(5, 3, 4) or WATERPOURING(3, 5, 4) is optimal i.e., requires the minimum number of steps to measure 4 liters. We can find the optimal solution by running both WATERPOURING(5, 3, 4) and WATERPOURING(3, 5, 4) and selecting the run (and its corresponding equation) that has the least number of steps.

Can’t we analytically solve the problem to find the optimal solution? It is not clear how to find the optimal solution analytically. Maybe we can use modular arithmetic to find the optimal solution and also the optimal number of steps analytically.

[Generalization.] Let’s first formalize the generalized problem statement for two containers. We are given two containers of integer capacities \(a\) and \(b\) liters such that \(1 \leq a \leq b\). We want to measure integer \(c\) liters such that \(1 \leq c \leq b\). WATERPOURING-2CONTAINERS gives an optimal algorithm to solve the problem, which uses Diophantine analysis as follows. This optimal solution corresponds to the integer solution \((x^{\text{opt}}, y^{\text{opt}})\) of the linear Diophantine equation

\[
ax + by = c
\]

such that \(a, b, c \in \mathbb{N}\) and \(\max(a, c) \leq b\) (27) that minimizes the number of steps in measuring integer \(c\) liters. We present the algorithm in multiple steps as follows.

[Step 1. Find a solution \((x^*, y^*)\) to Equation [27].] We use the extended Euclidean algorithm to check the solvability of puzzle; and find a solution if the puzzle is solvable. The extended Euclidean algorithm takes nonnegative integers \(a\) and \(b\) and returns \(\gcd(a, b)\), and a solution \((x^\dagger, y^\dagger)\) that satisfies the equation

\[
ax + by = \gcd(a, b)
\]

If \(c\) is not divisible by \(\gcd(a, b)\), then the puzzle is not solvable. If \(c\) is divisible by \(\gcd(a, b)\), then there are an infinite number of solutions to Equation [27]. Suppose \((x^*, y^*)\) be one of the infinitely possible solutions to Equation [27]. Then

\[
(x^*, y^*) = \left(\frac{c}{\gcd(a, b)} \cdot x^\dagger, \frac{c}{\gcd(a, b)} \cdot y^\dagger\right)
\]

[Step 2. Find two candidate optimal solutions to Equation [27].] If Equation [27] is solvable, then there are an infinite number of solutions. Once we know a single solution \((x^*, y^*)\), we can compute all solutions to Equation [27] using the equation

\[
(x, y) = \left(x^* - \frac{kb}{\gcd(a, b)}, y^* + \frac{ka}{\gcd(a, b)}\right)
\]

where \(k\) is any integer.

In this puzzle, we do not really care for all infinite solutions. We are more interested in the two candidate optimal solutions \((x^\prime, y^\prime)\) and \((x^\prime\prime, y^\prime\prime)\). We can find these two
**WaterPouring-2Containers**\((a,b,c)\)

**Input:** Natural numbers \(a, b, c\) such that \(\max(a,c) \leq b\).

**Output:** Optimal algorithm to measure \(c\) liters using the two containers.

1. **// Step 1. Find a solution to Equation 27** ...............................
2. \((d,y^*,x^*) \leftarrow \text{ExtendedEuclideanAlgorithm}(b,a)\)
3. **if** \(c\) is not divisible by \(d\) **then**
4. | Solution does not exist; **return**
5. \((x^*,y^*) \leftarrow \left(\frac{y^*}{d} \cdot x^*, \frac{x^*}{d} \cdot y^*\right)\)
6. **// Step 2. Find two candidate optimal solutions to Equation 27** .......
7. \(((x',y'),(x'',y'')) \leftarrow \text{Find2CandidateOptimalSolutions}(a,b,d,x^*,y^*)\)
8. **// Step 3. Find the optimal strategy measure \(c\) liters** ..............
9. Let strategy \(\mathcal{A}\) denote WaterPouring\((a,b,c)\), which corresponds to \((x',y')\)
10. Let strategy \(\mathcal{B}\) denote WaterPouring\((b,a,c)\), which corresponds to \((x'',y'')\)
11. Execute both strategies \(\mathcal{A}\) and \(\mathcal{B}\)
12. Optimal strategy is one of \(\mathcal{A}\) and \(\mathcal{B}\) that minimizes the number of steps

---

**ExtendedEuclideanAlgorithm**\((a,b)\)

**Input:** Two integers \(a \geq b \geq 0\).

**Output:** Finds \(d = \text{GCD}(a,b)\) and an integer solution to \(ax + by = d\).

1. \(x_2 \leftarrow 1; x_1 \leftarrow 0; y_2 \leftarrow 0; y_1 \leftarrow 1\)
2. **while** \(b > 0\) **do**
3. | \(q \leftarrow \left\lfloor \frac{a}{b} \right\rfloor\); \(r \leftarrow a - qb; x \leftarrow x_2 - qx_1; y \leftarrow y_2 - qy_1\)
4. | \(a \leftarrow b; b \leftarrow r; x_2 \leftarrow x_1; x_1 \leftarrow x; y_2 \leftarrow y_1; y_1 \leftarrow y\)
5. \(d \leftarrow a; x \leftarrow x_2; y \leftarrow y_2\)
6. **return** \((d,x,y)\)

---

**Find2CandidateOptimalSolutions**\((a,b,d,x^*,y^*)\)

**Input:** \(1 \leq a \leq b, d = \text{GCD}(a,b)\), solution \((x^*,y^*)\) to solve Eq. 27

**Output:** Finds two candidate optimal solutions to solve the equation.

1. \(\Delta x \leftarrow \frac{x^*}{d}; \Delta y \leftarrow \frac{y^*}{d}; k' \leftarrow \left\lfloor \frac{x^*}{\Delta x} \right\rfloor; k'' \leftarrow \left\lfloor \frac{y^*}{\Delta y} \right\rfloor\)
2. \((x',y') \leftarrow (x^* - k' \cdot \Delta x, y^* + k' \cdot \Delta y)\)
3. \((x'',y'') \leftarrow (x^* - k'' \cdot \Delta x, y^* + k'' \cdot \Delta y)\)
4. **if** \(x' < 0\) **then**
5. | **SWAP**\(((x',y'),(x'',y''))\) // to make sure \(x'\) is nonnegative
6. **return** \(((x',y'),(x'',y''))\)

Candidate optimal solutions by setting proper values of \(k\) in Equation 28 such that \(x' \geq 0\) and \(x'' < 0\).

**Step 3. Find the optimal strategy to measure \(c\) liters.** The solution \((x',y')\), where \(x' \geq 0\), corresponds to the water pouring strategy WaterPouring\((a,b,c)\). Similarly, the solution \((x'',y'')\), where \(x'' < 0\), corresponds to the water pouring strategy
**WaterPouring** $(f, t, c)$

**Input:** Containers $f$ for filling, $t$ for emptying. $f$ means “from” and $t$ means “to”. We require that $c \leq \max(f, t)$.

All variables: $f, t, c, \text{from}, \text{to}, \text{step}$ are global.

**Output:** Algorithm to measure $c$ liters. Returns the number of steps.

1. **Initialize();** FILL()
2. while $\text{from} \neq c$ and $\text{to} \neq c$ do
3.   TRANSFER()
4.   if $\text{from} = c$ or $\text{to} = c$ then break
5.   if $\text{from} = 0$ then FILL()
6.   if $\text{to} = t$ then EMPTY()
7. return step \hspace{1cm} // number of steps

**Initialize()**

1. from $\leftarrow 0$; to $\leftarrow 0$; step $\leftarrow 0$
2. print step: (from, to)

**Fill()**

1. from $\leftarrow f$; step $\leftarrow step + 1$
2. print step: (from, to): Fill $f$

**Empty()**

1. to $\leftarrow 0$; step $\leftarrow step + 1$
2. print step: (from, to): Empty $t$

**Transfer()**

1. move $\leftarrow \min(\text{from}, t - \text{to})$
2. from $\leftarrow \text{from} - \text{move}$; to $\leftarrow \text{to} + \text{move}$; step $\leftarrow step + 1$
3. print step: (from, to): Move move

**WaterPouring** $(b, a, c)$.

One of these two candidate strategies is optimal (i.e., has the minimum number of steps). We will have to execute both the strategies **WaterPouring** $(a, b, c)$ and **WaterPouring** $(b, a, c)$ and choose that strategy (or solution) that optimizes the number of steps (i.e., fill, empty, and transfer operations).

[Important remark.] Let’s call the solution $(x^{\text{opt}}, y^{\text{opt}})$ that corresponds to the optimal strategy. So, $(x^{\text{opt}}, y^{\text{opt}})$ has to be one of $(x’, y’)$ and $(x’’, y’’).$ It is not clear how to compute $(x^{\text{opt}}, y^{\text{opt}})$ without actually executing the strategies corresponding to $(x’, y’)$ and $(x’’, y’’).$ One might intuitively think that the solution that minimizes $(|x| + |y|)$ has to be optimal, but it is not true.

**Shieh-Tsai’s linear Diophantine equation solution** $(n$ containers, non-optimal$)$

There are $n$ containers of integer capacities such that $1 < a_1 < a_2 < a_3 < \cdots < a_n$. Initially, all these containers are empty. There is a pond that can be used to either fill up or empty a container. We need to measure exactly $c$ liters of water from the given containers.
**WaterPouring-nContainers**\(a[1 \ldots n], c\)

**Input:** Naturals \(1 < a_1 < a_2 < \cdots < a_n\), \(1 \leq k \leq a[1] + a[2] + \cdots + a[n]\) are global

**Variables:** \(s[1 \ldots n]\), \(x[1 \ldots n]\), \(v[1 \ldots n]\), \(t\), \(\ell\) are global

**Output:** Strategy to measure \(c\) liters.

1. Compute a solution \(x[1 \ldots n]\) \(\quad// (x_1^*, x_2^*, \ldots, x_n^*)\)
2. **initialize(); sequence()**
3. **print** Number of steps = \(t\)

**initialize()**

1. Create a state array \(s[1 \ldots n] \leftarrow [0, \ldots, 0]\)
2. Create the array \(v[1 \ldots n] \leftarrow x[1 \ldots n]\); \(t \leftarrow 0\)
3. Set \(\ell \leftarrow \min_i(a[i] + a[n - 1] + \cdots + a[n - i + 1] \geq c)\)
4. **print** \(v: s[1 \ldots n]\)

**sequence()**

1. for \(i \leftarrow 1\) to \(n\) do
2. | if \(s[i] = 0\) and \(v[i] > 0\) then **fill(i)**
3. while there exists \(i\) such that \(v[i] < 0\) do
4. | find the first \(j\) such that \(s[j] > 0\) and \(v[j] \geq 0\)
5. **transfer(j, i)**
6. if \(s[j] = 0\) and \(v[j] > 0\) then **fill(j)**
7. if \(s[i] = a[i]\) then **empty(i)**
8. while there exists \(i\) such that \(v[i] > 0\) do
9. | find the first \(j \geq n - \ell\) such that \(s[j] \neq a[j]\) and \(v[j] = 0\)
10. **transfer(i, j)**
11. if \(s[i] = 0\) then **fill(i)**

**fill(i)**

1. \(s[i] \leftarrow a[i]; v[i] \leftarrow v[i] - 1; t \leftarrow t + 1\)
2. **print** \(v: s[1 \ldots n]\): Fill \(i\)

**empty(j)**

1. \(s[j] \leftarrow 0; v[i] \leftarrow v[i] + 1; t \leftarrow t + 1\)
2. **print** \(v: s[1 \ldots n]\): Empty \(j\)

**transfer(i, j)**

1. \(m \leftarrow \min(s[i], a[j] - s[j])\)
2. \(s[i] \leftarrow s[i] - m; s[j] \leftarrow s[j] + m; t \leftarrow t + 1\)
3. **print** \(v: s[1 \ldots n]\): Transfer \(i\) to \(j\)

Figure 95: Algorithm to solve the water pouring puzzle for \(n\) containers.
A quantity of natural number $c$ liters is \textit{measurable} if after a sequence of feasible steps, we can get the total amount of water in all the containers equal to $c$. An amount of $c$ liters can be measured if and only if $c$ is at most the sum of capacities of all the containers and $c$ is a multiple of the GCD of all container capacities.

If there exists a solution for a given $c$, we can always find integral values of $x_1, x_2, \ldots, x_n$ in the linear Diophantine equation in $n$ unknowns:

$$a_1 x_1 + a_2 x_2 + a_3 x_3 + \cdots + a_n x_n = c$$

Solving this linear Diophantine equation for infinite solutions (if solvable) is given in Appendix. Suppose we obtain a solution $(x_1^*, x_2^*, \ldots, x_n^*)$ from the algorithm given in Appendix, then the strategy to measure $c$ liters is given in WATERPOURING-$n$CONTAINERS algorithm from Figure 95.

**Linear Diophantine equation solution ($n$ containers, optimal)**

The optimal algorithm to solve this generalized puzzle is not known.

Suppose P denotes the class of computational problems that can be solved by deterministic algorithms in polynomial-time and NP denotes the class of computational problems that can be solved by nondeterministic algorithms in polynomial time. Unless P = NP, there exists no polynomial-time algorithm which can solve the $n$ containers puzzle optimally. In fact, unless P = NP, there exists no polynomial-time algorithm which can solve the $n$ containers puzzle within any fixed constant approximation factor.

**Variant: Water pouring without using a pond**

There is an 8-liter container full of water. We are given two empty containers of 3 liters and 5 liters. We do not have a pond to fill or empty water. That means the only operation we can use is transferring water from one container to other. How can we measure exactly 4 liters of water using the three containers minimizing the total number of operations?

[State diagram solution.] In this puzzle, a state is denoted by a box with three values that represent the amount of water in the 8-liter, 5-liter, and 3-liter containers, respectively. A state transition can happen in only one way: transferring from one container to another till the former becomes empty or the latter becomes full. The starting state is $[8, 0, 0]$, and the final state is $[4, -\cdot, -\cdot]$ or $[-\cdot, 4, -\cdot]$ because if we measure 4 liters it has to be in the 8-liter or 5-liter containers and we do not care for the amount of water in the other containers. The state diagram is given in Figure 96 and its details are given in Figure 98.

There is one optimal solution to this puzzle (Figure 92).
Optimal solution takes 6 operations.

We can also construct a graph as shown in Figure 97 to depict the states. The states are the vertices in a \textit{barycentric coordinate system}, where the location of a vertex is specified by reference to a simplex (i.e., the generalization of a triangle). The
Figure 96: State diagram for water pouring without using a pond puzzle.

Figure 97: Barycentric coordinate system for the water pouring puzzle without using pond puzzle.
barycentric coordinates in this puzzle is a 3-tuple \((a, b, c)\) such that \(a + b + c = 8\), whose endpoints will be \([8, 0, 0]\), \([0, 8, 0]\), and \([0, 0, 8]\). There are 55 states in the barycentric diagram. However, only those vertices that are present on the border/boundary of the yellow-colored rectangular region with corners \([8, 0, 0]\), \([3, 5, 0]\), \([0, 5, 3]\), and \([5, 0, 3]\) are feasible states. The objective is to start from state \([8, 0, 0]\) and reach \([4, -4, -]\) or \([-, 4, -]\) minimizing the number of edges in the path. An optimal solution is shown in the diagram using red edges.

**Problems**

1. Design an efficient optimal algorithm for solving the \(n\)-containers puzzle.
2. Design an efficient optimal algorithm for solving the \(n\)-containers puzzle when the container capacities are real numbers.

**References**

The breadth-first search (BFS) and other shortest path graph algorithms are discussed in algorithm textbooks given in Bibliography. The BFS solution for solving the puzzle is given in [Bellman et al., 1970], [Murray-Lasso, 2003], and [Reba and Shier, 2014]. Double ended queue and other data structures are covered in the data structure texbooks given in Bibliography.

To learn more about Diophantine equations, consult books such as [Niven et al., 2008], [Schrijver, 1998], and [Hardy et al., 2008]. The linear Diophantine equation solution for solving the 2-containers puzzle can be traced back to [Averbach and Chein, 1980]. The lower/upper bounds for the number of steps in the solution are given in [Boldi et al., 2002], [Pfaff and Tran, 2005], and [Shieh and Tsai, 2008]. The linear Diophantine equation solution for solving the \(n\)-containers puzzle and the non-existence of polynomial time optimal algorithms (unless P=NP) are given in [Shieh and Tsai, 2008]. The extended Euclidean algorithm is given in [Menezes et al., 2018].
Circle of Death

Problem

A king has captured a thousand of his enemy soldiers, which includes the enemy king. The king decides to kill the prisoners sparing the life of exactly one prisoner using the rules as follows. The thousand prisoners are made to stand in a circle and are given sequential numbers from 1 to 1000. Starting from the first prisoner, every second prisoner (i.e., 2, 4, 6, ...) is killed. The process is continued until only one prisoner remains, who is then freed. The enemy king is at position 465. Everyone is anxious to know if the enemy king would be set free in this process.

Does the enemy king survive?

Solution

This is one of the classic mathematical and computational problems, famously called Josephus problem, usually found in discrete mathematics and computer science textbooks. This puzzle is probably the greatest pedagogical example to teach the analysis of algorithms.

[Problem statement.] Let’s formalize the problem. Suppose the prisoners numbered 1, 2, 3, …, n are standing in a circle in the same order. Let \( n \) be the number of prisoners in the circle, every \( k \)th prisoner in clockwise direction is killed in every step, and \( j \) be the step/iteration number. Let

\[
C(n, k, j) = \text{Prisoner who is killed at the } j\text{th timestep when there are } n \text{ prisoners in a circle and every } k \text{th prisoner is killed starting from the first prisoner.}
\]

\[
C(n, k) = C(n, k, n) = \text{Last prisoner selected for variables } n \text{ and } k.
\]

where, \( n \) and \( k \) are independent variables and \( j \) is upper bounded by \( n \).

The puzzle asks us to verify whether \( C(1000, 2) \) is 465. If \( C(1000, 2) = 465 \), then the enemy king will not be killed and will go free. On the other hand, if \( C(1000, 2) \neq 465 \), then the enemy king will be killed. The plain simple answer to the puzzle is

\[
C(1000, 2) \neq 465. \text{ Hence, the enemy king will be killed.}
\]

Interestingly, \( C(1000, 2, 999) = 465 \) which implies that the enemy king is the last person to be killed. The king wanted to keep his enemy king’s hope to live long enough only to be squashed at the end.

In the algorithms that follow, we aim to compute \( C(n, k, j) \) for a given \( n, k, \) and \( j \).

[Types of algorithms.] There are several algorithms for solving this puzzle and they can be classified into two types.
[Type 1. Algorithms based on data structures.] These algorithms make use of data structures such as arrays, linked lists, and balanced search trees. The data stored in these structures are the prisoner numbers. Hence, all these algorithms occupy space of $\Omega(n)$. As all these algorithms make use of Insert, Delete and Search functions the time complexity of these algorithms must be a minimum of $n$ i.e., $\Omega(n)$.

These algorithms have an interesting property: when they compute $C(n, k, j)$ they also compute as intermediate results $C(n, k, 1), C(n, k, 2), \ldots, C(n, k, j - 1)$. So these algorithms will compute $C(n, k, 1..n)$ aka the Josephus permutation when they compute $C(n, k)$.

[Type 2. Algorithms based on recurrences.] It is beautiful to see that there can be so many recurrences that compute the exact same result. Recurrences can be implemented both recursively and iteratively (aka non-recursively). When recurrences (with constant amount of history) are implemented iteratively the extra space required is $\Theta(1)$ usually. Hence, non-recursive recurrence-based algorithms are more space-efficient compared with data structures based algorithms. Furthermore, recurrence-based algorithms typically have much better time complexity compared with data structure based algorithms as the recurrence-based algorithms use sophisticated mathematical properties of the problem. In summary, recurrences are the best ways to solve this puzzle, i.e., to compute $C(n, k, j)$. See Table 51.

These algorithms when they compute $C(n, k, j)$, they do not compute $C(n, k, 1), C(n, k, 2), \ldots, C(n, k, j - 1)$ as intermediate results. So if one wants to compute the Josephus permutation, they need to compute $C(n, k, j)$ for each value of $j$ from 1 to $n$ and time taken for this process can be evaluated by taking the summation of time for computing $C(n, k, j)$ over all values of $j$.

**Type 1. Algorithms based on data structures**

[Circular Boolean array algorithm.] We can definitely use the pen-and-paper method and cross out every second prisoner from the first prisoner. The only prisoner remaining would be set free. This is a good approach when the number of prisoners is small. When the number of prisoners is large, we cannot use a pen and a paper. However, we will use a similar approach using circular Boolean array.

A Boolean array can be used to compute $C(n, k, j)$. We use congruence modulo $n$ in a 0-indexed Boolean array to simulate a circle. Create an array $A[0..n-1]$ and initialize it to all ones. The assignment $A[i] = 1$ means that prisoner $(i+1)$ is in the circle and the assignment $A[i] = 0$ means that the prisoner $(i+1)$ is not in the circle. We eliminate all prisoners in $n$ iterations. In each iteration $m \in \{1, 2, 3, \ldots, j\}$, we skip $(k-1)$ ones in the cyclical order and then set the next $k$th one to zero (i.e., prisoner at that location or index will be killed). The order in which the indices of the array are set to zero is the order in which the prisoners are killed. Figure 99 gives an algorithm to compute $C(n, k, j)$.

\[
\langle \text{Time, Space} \rangle \text{ complexity to compute } C(n, k, j) \text{ is } \langle ?, \Theta(n) \rangle.
\]

[Circular array algorithm.] An array can be used to compute $C(n, k, j)$ for a given
n, k, and j. We use congruence modulo n in a 0-indexed array to simulate a circle. Create an array A such that $A[i] = i + 1$ for $i \in \{0, 1, 2, \ldots, n - 1\}$. We eliminate the prisoners in n iterations. In each iteration $m \in \{1, 2, 3, \ldots, j - 1\}$, we delete the entry at the kth position from the current location (i.e., prisoner at that location or index will be killed) and update the current location. The number at the kth position from the current location is C(n, k, j). The order in which the indices of the array are deleted is the order in which the prisoners are killed. Figure 100 gives an algorithm to compute C(n, k, j).

\[
\text{〈Time, Space〉 complexity to compute } C(n, k) \text{ is } \langle O(jn), \Theta(n) \rangle.
\]

[Circularly linked list algorithm.] A circularly linked list data structure can be used to compute C(n, k, j) for a given n, k, and j. We assume that circularly linked list is implemented using a singly linked list such that the tail node points back to the head node. First, we create a circularly linked list with numbers from 1 to n, where n points back to 1. In this list, we start from the first node and delete every kth node until the circularly linked list is empty. Figure 101 gives an algorithm to compute C(n, k, j).

We assume that the circularly linked list is implemented using singly linked list. The time complexity of InsertLast, FirstNode, Next, GetValue, and Delete functions in the
**Algorithm**

**CIRCLE-OF-DEATH-BOOLEAN-ARRAY**($n, k, j$)

- **Input:** $n$ prisoners, every $k$th prisoner is killed, $j$th prisoner to be executed.
- **Output:** $C(n, k, j)$.

1. array $A[0, 1, 2, \ldots, (n-1)] \leftarrow [1, 1, 1, \ldots, 1]$; $i \leftarrow 0$
2. for $m \leftarrow 1$ to $j$
   3. for $\ell \leftarrow 1$ to $k-1$
      4. while $A[i] = 0$ do $i \leftarrow (i + 1) \bmod n$
      5. $i \leftarrow (i + 1) \bmod n$
   6. while $A[i] = 0$ do $i \leftarrow (i + 1) \bmod n$
   7. $A[i] \leftarrow 0$
   8. $i \leftarrow (i + 1) \bmod n$ // prisoner $(i + 1)$ is killed
3. return $C(n, k, j)$

**CIRCLE-OF-DEATH-ARRAY**($n, k, j$)

- **Input:** $n$ prisoners, every $k$th prisoner is killed, $j$th prisoner to be executed.
- **Output:** $C(n, k, j)$.

1. array $A[0, 1, 2, \ldots, (n-1)] \leftarrow [1, 2, 3, \ldots, n]$; $i \leftarrow 0$; $size \leftarrow n$
2. for $m \leftarrow 1$ to $j-1$
   3. $i \leftarrow (i + k - 1) \bmod size$
   4. for $\ell \leftarrow i$ to $size-2$ do $A[\ell] \leftarrow A[\ell + 1]$
   5. $size \leftarrow size - 1$
   6. $i \leftarrow (i + k - 1) \bmod size$
   7. $C(n, k, j) \leftarrow A[i]$
3. return $C(n, k, j)$

**CIRCLE-OF-DEATH-CLL**($n, k, j$)

- **Input:** $n$ prisoners, every $k$th prisoner is killed, $j$th prisoner to be executed.
- **Output:** $C(n, k, j)$.

1. create an empty CircularlyLinkedList $L$
2. for $i \leftarrow 1$ to $n$ do $L$.InsertLast($i$)
3. pointer $\leftarrow L$.FirstNode()
4. for $m \leftarrow 1$ to $j$
   5. for $\ell \leftarrow k-1$ do $pointer \leftarrow pointer$.Next()
   6. $C(n, k, m) \leftarrow pointer$.GetValue()
   7. $pointer$.Delete()
3. return $C(n, k, j)$

Figure 99: Algorithm to solve the circle of death puzzle using a Boolean array.

Figure 100: Algorithm to solve the circle of death puzzle using an array.

Figure 101: Algorithm to solve the circle of death puzzle using a circularly linked list.

The algorithm are constant. Hence,
Initially, the Θ the Figure 106 gives the algorithm to compute \( r_0 \) variable to is \((n, k, j)\). We also maintain a circularly doubly linked list of node contains the number of active leaves in the subtree rooted at that non-leaf node. Each tree \( T \) to denote \( s \) number of consecutive prisoner numbers (except possibly for the last tree); in the last tree \( T_{\left\lfloor \frac{n}{k} \right\rfloor} \) and the number of such dead leaves will be \( s \cdot \left\lceil \frac{n}{s} \right\rceil - n \). A non-leaf node contains the number of active leaves in the subtree rooted at that non-leaf node. We also maintain a circularly doubly linked list of \( \left\lceil \frac{n}{s} \right\rceil \) nodes such that the \( i \)th node of the list points to the root of the \( i \)th tree \( T_i \). This helps in fast access of numbers in succeeding and preceding groups. The total time required to create the initial data structure is \( \Theta(n) \). Figure 102 shows the initial data structure for \( n = 9 \) and \( k = 5 \). Following the definition, \( s = 4 \) and \( \left\lceil \frac{n}{s} \right\rceil = 3 \). Hence, we have three trees each with four
Initial data structure for $n = 9$ (number of active leaf nodes), $k = 5$, and hence $s = 4$ (i.e., largest power of 2 not greater than $k$).

After deleting the 5th entry, i.e., $C(9, 5, 1) = 5$, the total active leaves in the two adjacent trees $T_2$ and $T_3$ are equal to 4, so we merge $T_2$ and $T_3$.

After combining the two adjacent trees, the combined tree $T_2$ now has four active leaves. The process of deleting the 5th entries can continue.

After deleting the 5th entries four times, i.e., $C(9, 5, 2) = 1$, $C(9, 5, 3) = 7$, $C(9, 5, 4) = 4$, and $C(9, 5, 5) = 3$, the total active leaves in $T_1$ and $T_2$ is 4, so we merge them.

Figure 102: First few steps of Lloyd’s algorithm for solving the circle of death puzzle.
After combining the two adjacent trees, the combined tree $T_1$ now has four active leaves. The process of deleting the 5th entries can continue.

After deleting the 5th entries four times, i.e., $C(9, 5, 6) = 6$, $C(9, 5, 7) = 9$, $C(9, 5, 8) = 2$, and $C(9, 5, 9) = 8$, we are done with the algorithm.

Figure 103: Last few steps of Lloyd’s algorithm for solving the circle of death puzzle.

leaves, and the number of dead leaves is $(s \cdot \lceil \frac{n}{s} \rceil - n) = 4 \cdot 3 - 9 = 3$.

We compute $C(n, k, j)$ after $j$ iterations. In iteration $i \in \{1, 2, 3, \ldots, j\}$, we delete the $k$th active leaf from the current node pointer in three stages:

[Stage 1. Find the next number.] Find the next active leaf node, i.e., $C(n, k, i)$. We can find the next active leaf node to be marked by finding the tree in which it occurs and then descending the tree to the actual leaf node. An important point to note is that the next active leaf node to be marked occurs in either next or next-to-next tree from the tree associated with the last active leaf node that was marked. The exact tree where the next active leaf node will be found depends on the number of active leaf nodes in those trees. Searching the tree and descending the tree to identify the active leaf node to be marked requires $\Theta(\log s)$ time.

[Stage 2. Mark the number as dead.] Store the leaf node that represents $C(n, k, i)$ found in the previous stage and mark it as dead. Marking a leaf node as dead does not mean deleting the node. The node will still be there but the contents might now represent an unused number like $-1$ or 0. When we mark a leaf node as dead (i.e., an active leaf becomes a dead leaf), we need to update (i.e., decrement by 1) the counts of all ancestor nodes of that marked node. This is because that particular active node turned into a dead node and hence the number of active nodes contained in the subtrees rooted at any ancestor node must be decremented by 1. This takes $\Theta(\log s)$ time.

[Stage 3. Combine trees if required.] If the total number of active leaves in two
consecutive trees is \( s \), then replace the two trees with a perfect binary tree, which is a combination of those two trees. We need this stage of merging to make sure that the number of trees searched in Stage 1 to locate the next active leaf node to be marked is at most two. Merging two consecutive trees and replacing the two consecutive trees by the merged tree requires \( \Theta(s) \) time. As the number of such merge operations is \( \left\lceil \frac{n}{s} \right\rceil - 1 \), the total time required for all merge operations is \( \Theta(n) \).

Figures [102 and 103] show steps for all three stages for the example of \( n = 9 \) and \( k = 5 \).

The total time complexity of the algorithm is

\[
\text{Time} = \text{Initialization time} + \#\text{iterations} \times (\text{Stage 1 time} + \text{Stage 2 time}) \\
+ \#\text{merge operations} \times \text{Stage 3 time}
\]

\[
= \Theta(n) + j \cdot \Theta(\log s) + \left\lceil \frac{n}{s} \right\rceil - 1 \cdot \Theta(s)
\]

\[
= \Theta(n + j \log s) = \Theta(n + j \log k)
\]

[Case 2. \( k > n \).] The original discoverer of this algorithm explained the algorithm for only Case 1, i.e., when \( n \geq k \). If we apply the algorithm explained in Case 1 to Case 2, the time complexity of the algorithm for Case 2 will turn out to be \( \Theta(k + j \log k) \) (How?) and its space complexity will be \( \Theta(k) \). In this section, we present a simple and efficient approach to adapt Case 1 algorithm to Case 2.

Let \( s \) be the largest power of 2 not greater than \( n \), i.e., \( s = 2^{\left\lfloor \log_2 n \right\rfloor} \). The modified definition of \( s \) is the biggest change required to get an efficient algorithm in this case. That’s it. All the other details of the algorithm in Case 1 can be ported to Case 2. There is one more slight change that will be required in Stage 1 as how to find the next number in \( \Theta(\log s) \) time. We leave it to the reader to figure this out. These modifications lead to an algorithm with time complexity \( \Theta(n + j \log n) \) (How?).

So, the algorithm’s complexity for both cases is as follows.

\[\langle\text{Time, Space}\rangle\text{ complexity to compute } C(n, k, j) \text{ is } \langle\Theta(n + j \cdot \log(\min(n, k))), \Theta(n)\rangle.\]

**Type 2. Algorithms based on recurrences**

[Knuth’s algorithm based on inverse permutation.] This algorithm uses the concept of inverse permutation to compute \( C(n, k, j) \). The algorithm can be considered as a hybrid of Type 1 and Type 2 as it includes data structures and a recurrence. The algorithm consists of three stages.

[Stage 1. Compute the \( b \) array from \( n \) and \( k \).] We compute the inverse permutation \([b_1, b_2, b_3, \ldots, b_n]\) using the following recurrence:

\[
b_i = \begin{cases} 
(k - 1) \mod n & \text{if } i = 1, \\
(b_{i-1} + k - 1) \mod (n - i + 1) & \text{if } i > 1.
\end{cases}
\]

(30)

The recurrence given above leads to one kind of inverse permutation but there are other kinds of inverse permutations as well. We use this specific inverse permutation \( b \) to compute \( C(n, k, 1 \ldots n) \).
**CircleOfDeath-Knuth-InversePermutation**($n,k$)

**Input:** $n$ prisoners in a circle, every $k$th person is killed.

**Output:** $C(n,k,1...n)$

1. // Stage 1. Compute the inverse table $b[1...n]$ from $n$ and $k$ ...........
2. $b[1] \leftarrow (k-1) \mod n$
3. **for** $i \leftarrow 2$ **to** $n$ **do** $b[i] \leftarrow (b[i-1] + k - 1) \mod (n - i + 1)$
4. // Stage 2. Compute $a[1...n]$ from $b[1...n]$ using exactly one of the
   following two algorithms ..............................................
5. $a[1...n] \leftarrow$ Knuth-Stage2-Slow($b[1...n]$)
6. $a[1...n] \leftarrow$ Knuth-Stage2-Fast($b[1...n]$)
7. // Stage 3. Compute $C(n,k,1...n)$ from $a[1...n]$ .........................
8. **for** $i \leftarrow 1$ **to** $n$ **do** $C(n,k,a_i) \leftarrow i$
9. **return** $C(n,k,1...n)$

**Knuth-Stage2-Slow($b[1...n]$)**

1. Create $x[0...n]$ and $a[1...n]$
2. $x[0] \leftarrow 0$
3. **for** $i \leftarrow n$ **to** 1 **do**
4. \[ \ell \leftarrow 0 \]
5. **for** $\ell \leftarrow 1$ **to** $b_i$ **do** $x[i] \leftarrow x[\ell]; x[\ell] \leftarrow i$
6. $\ell \leftarrow 0$
7. **for** $i \leftarrow 1$ **to** $n$ **do** $a[i] \leftarrow x[\ell]; \ell \leftarrow x[\ell]$
8. **return** $a[1...n]$

**Knuth-Stage2-Fast($b[1...n]$)**

1. **for** $i \leftarrow 1$ **to** $n$ **do** $B[i] \leftarrow \langle b[i], i \rangle$
2. $a[1...n] \leftarrow$ Knuth-Stage2-MergeSort($B[1...n]$)
3. Create array $a[1...n]$
4. **for** $i \leftarrow 1$ **to** $n$ **do** $\langle x,y \rangle \leftarrow a[i]; a[i] \leftarrow y$
5. **return** $a[1...n]$

**Knuth-Stage2-MergeSort($B[\ell...h]$)**

1. **if** $\ell = h$ **then** **return** $[B[\ell]]$
2. **else if** $\ell < h$ **then**
3. \[ m \leftarrow \left\lfloor \frac{h-\ell}{2} \right\rfloor + \ell \]
4. $L[1...n_L] \leftarrow$ Knuth-Stage2-MergeSort($B[\ell...m]$)
5. $R[1...n_R] \leftarrow$ Knuth-Stage2-MergeSort($B[m+1...h]$)
6. $A[1...(n_L+n_R)] \leftarrow$ Knuth-Stage2-Merge($L[1...n_L], R[1...n_R]$)
7. **return** $A[1...(n_L+n_R)]$

Figure 104: Algorithm to solve the circle of death puzzle using Recurrences 30,31
For example, the $b$ array when $n = 9$ and $k = 5$ is $[4, 0, 4, 2, 1, 1, 2, 0, 0]$.

**Stage 2. Compute the $a$ array from the $b$ array.** We can use any of the two algorithms to compute the $a$ array – one using a $\Theta(n^2)$ Knuth-Stage2-Slow algorithm or the other $\Theta(n \log n)$ Knuth-Stage2-Fast algorithm. Let’s focus on the fast algorithm which is based on a logic very similar to that behind the merge sort algorithm.

It is possible to show that

$$
\langle b_1, 1 \rangle \star \langle b_2, 2 \rangle \star \langle b_3, 3 \rangle \star \cdots \star \langle b_n, n \rangle = \langle 0, a_1 \rangle \langle 0, a_2 \rangle \langle 0, a_3 \rangle \cdots \langle 0, a_n \rangle
$$

where $\star$ is an associative binary operator, called the **merge operator**, defined as:

$$
(p, q) S_1 \star (p', q') S_2 = \begin{cases} 
(p, q) (S_1 \star ((p' - p, q') S_2)) & \text{if } p \leq p', \\
(p', q') ((p - p' - 1, q) S_1) \star S_2) & \text{if } p > p'.
\end{cases}
$$

(31)

where, $S_1$ and $S_2$ are concatenated strings of ordered pairs (of the form $\langle x, y \rangle$). The time taken to merge two strings consisting of $n_1$ and $n_2$ ordered pairs is $\Theta(n_1 + n_2)$.

Let $B_i = \langle b_i, i \rangle$ and $A_i = \langle 0, a_i \rangle$ for $i \in [1, n]$. It is easy to see that constructing $B$ given $b$ array and constructing $a$ from $A$ array is straightforward. The merge operation previously described can be written as

$$
B_1 \star B_2 \star B_3 \star \cdots \star B_n = A_1 A_2 A_3 \cdots A_n
$$

We can compute $A_1 A_2 A_3 \cdots A_n$ from $B_1 \star B_2 \star B_3 \star \cdots \star B_n$ using divide-and-conquer algorithm design technique in $\Theta(n \log n)$ time. This algorithm, named Knuth-Stage2-MergeSort, is almost identical to merge sort.
For example, given $b = \{4, 0, 4, 2, 1, 1, 2, 0, 0\}$ we can compute $a = \{2, 8, 5, 4, 1, 6, 3, 9, 7\}$ as shown in Figure 105.

Figure 105: Computing the $a = \{2, 8, 5, 4, 1, 6, 3, 9, 7\}$ array from $b = \{4, 0, 4, 2, 1, 1, 2, 0, 0\}$.

**Stage 3. Compute $C(n, k, 1...n)$ from the a array.** What is the intuition behind $a_i$? It means that $a_i$th person to be killed is the person numbered $i$. In other words, the person numbered $i$ is killed at the $a_i$th timestep/iteration. More formally, $C(n, k, a_i) \leftarrow i$ for each $i \in [1, n]$.

For example, given $a = \{2, 8, 5, 4, 1, 6, 3, 9, 7\}$, we compute $C(9, 5, 1...9) = \{5, 1, 7, 4, 3, 6, 9, 2, 8\}$.

**[Slow.]** $\langle$Time, Space$\rangle$ complexity to compute $C(n, k, j)$ is $\Theta\left(n^2\right), \Theta\left(n\right)$.

**[Fast.]** $\langle$Time, Space$\rangle$ complexity to compute $C(n, k, j)$ is $\Theta\left(n \log n\right), \Theta\left(n\right)$.

**[Takakazu-Euler-Tait’s algorithm.]** The algorithm uses the recurrence $T(n, k, j)$ which is a number in the range $\{0, 1, 2, \ldots, n - 1\}$. Figure 107 gives an algorithm to compute $C(n, k, j)$.

\[
T(n, k, j) = \begin{cases} 
(k - 1) \mod n & \text{if } j = 1, \\
(T(n - 1, k, j - 1) + k) \mod n & \text{if } j > 1.
\end{cases}
\]

(32)

\[
C(n, k, j) = T(n, k, j) + 1
\]

(33)

For example, let’s compute $C(9, 5, 8)$. Let’s assume that $\%$ operator is same as mod operator. The term $T(9, 5, 8)$ can be computed as

\[
T(9, 5, 8) = [(T(8, 5, 7) + 5) \mod 9]
\]

246
\[
\begin{align*}
&= (((T(7,5,6) + 5) \% 8) + 5) \% 9 \\
&= (((T(6,5,5) + 5) \% 7) + 5) \% 8 \\
&= (((T(5,5,4) + 5) \% 6) + 5) \% 7 \\
&= (((T(4,5,3) + 5) \% 5) + 5) \% 7 \\
&= (((T(3,5,2) + 5) \% 4) + 5) \% 7 \\
&= (((T(2,5,1) + 5) \% 3) + 5) \% 7 \\
&= (((4 \% 2) + 5) \% 3) + 5 \\
&= 1
\end{align*}
\]

and hence \(C(9,5,8) = T(9,5,8) + 1 = 2\). We can also compute \(C(9,5,8)\) using the pseudocode logic to get \([T_1, T_2, \ldots, T_8] = [1, 3, 4, 4, 3, 1, 6, 2]\) which implies \(C(9,5,8) = 2\).

\[\langle\text{Time, Space}\rangle\text{ complexity to compute } C(n,k,j) = \langle\Theta(j), \Theta(1)\rangle.\]

[Herstein-Kaplansky’s algorithm.] The algorithm uses the recurrence \(K(n,k,j)\) which is a number in the range \([1,2,3,\ldots,n]\). Figure \[108\] gives an algorithm to compute \(C(n,k,j)\).

\[
K(n,k,j) = \begin{cases} 
  j + (k - 1) \text{ mod } n & \text{if } j = 1, \\
  j + ((K(n - 1,k,j - 1) + k) \text{ mod } n) & \text{if } j > 1. 
\end{cases}
\]

(34)

\[
C(n,k,j) = K(n,k,j)
\]

(35)

\[\langle\text{Time, Space}\rangle\text{ complexity to compute } C(n,k,j) = \langle\Theta(j), \Theta(1)\rangle.\]

[Schubert’s algorithm.] This algorithm uses an exponentially-fast growing sequence to compute \(C(n,k,j)\) for a given \(n, k \geq 2\), and \(j \in [1,2,3,\ldots,n]\). First, define the sequence \(S^{(n,k,j)}\) dependent on the three parameters \(n, k, j\) as

\[
S^{(n,k,j)} = [S_1, S_2, S_3, \ldots\text{] where } S_i = \begin{cases} 
  k(n - j) + 1 & \text{if } i = 1, \\
  \left\lceil \frac{k}{k - 1} \times S_{i-1} \right\rceil & \text{if } i > 1. 
\end{cases}
\]

(36)

Then define the least integer \(\ell\) such that \(S_{\ell - 1} < nk + 1 < S_{\ell}\). Finally, \(C(n,k,j)\) can now be computed as \(C(n,k,j) = nk + 1 - S_{\ell - 1}\). Figure \[109\] gives an algorithm to compute \(C(n,k,j)\). For example, let’s compute \(C(9,5,8)\). We see that \([S_1, S_2, \ldots, S_{10}] = [6, 8, 10, 13, 17, 22, 28, 35, 44, 55]\). As \(44 < 9 \times 5 + 1 < 55\), we compute \(C(9,5,8) = 9 \times 5 + 1 - 44 = 2\).

\[\langle\text{Time, Space}\rangle\text{ complexity to compute } C(n,k,j) = \langle\Theta\left(\log^2(P)\right)(nk), \Theta(1)\rangle.\]

[Busche’s algorithm.] This algorithm uses an exponentially-fast growing sequence to compute \(C(n,k,j)\) for a given \(n, k \geq 2\), and \(j \in [1,2,3,\ldots,n]\). First, define the sequence \(B^{(n,k,j)}\) dependent on the three parameters \(n, k, j\) as

\[
B^{(n,k,j)} = [B_1, B_2, B_3, \ldots\text{] where } B_i = \begin{cases} 
  1 & \text{if } i = 1, \\
  \left\lceil \frac{k}{k - 1} \times (B_{i-1} + n - j) \right\rceil & \text{if } i > 1. 
\end{cases}
\]

(37)
### CircleOfDeath-BalancedSearchTree\((n, k, j)\)

**Input:** \(n\) prisoners, every \(k\)th prisoner is killed, \(j\)th prisoner to be executed.

**Output:** \(C(n, k, j)\).

1. create an empty BalancedSearchTree \(T\)
2. for \(i \leftarrow 1 \text{ to } n\) do \(T\).Insert\((i)\)
3. \(\text{rank} \leftarrow 0\)
4. for \(i \leftarrow 1 \text{ to } j\) do
   5. \(\text{rank} \leftarrow (\text{rank} - 1 + k) \mod (n - i + 1)\)
   6. \(C(n, k, i) \leftarrow T\).SmallestElement\((\text{rank} + 1)\)
   7. \(T\).Delete\((C(n, k, i))\)
8. return \(C(n, k, j)\)

Figure 106: Algorithm to solve the circle of death puzzle using binary search tree.

### CircleOfDeath-TakakazuEulerTait\((n, k, j)\)

**Input:** \(n\) prisoners, every \(k\)th prisoner is killed, \(j\)th prisoner to be executed.

**Output:** \(C(n, k, j)\).

1. \(\ell \leftarrow n - j + 1\)
2. \(T \leftarrow (((k - 1) \mod \ell) + 1\)
3. for \(i \leftarrow \ell + 1 \text{ to } n\) do
   4. \(T \leftarrow 1 + ((T + k - 1) \mod i)\)
4. \(C(n, k, j) \leftarrow T\)
5. return \(C(n, k, j)\)

Figure 107: Algorithm to solve the circle of death puzzle using Recurrence 32.

### CircleOfDeath-HersteinKaplansky\((n, k, j)\)

**Input:** \(n\) prisoners, every \(k\)th prisoner is killed, \(j\)th prisoner to be executed.

**Output:** \(C(n, k, j)\).

1. \(\ell \leftarrow n - j + 1\)
2. \(K \leftarrow (((k - 1) \mod \ell) + n - \ell + 1\)
3. for \(i \leftarrow \ell + 1 \text{ to } n\) do
   4. \(K \leftarrow (n - i + 1) + ((K + k - n + i - 2) \mod i)\)
4. \(C(n, k, j) \leftarrow K\)
5. return \(C(n, k, j)\)

Figure 108: Algorithm to solve the circle of death puzzle using Recurrence 34.

### CircleOfDeath-Schubert\((n, k, j)\)

**Input:** \(n\) prisoners, every \(k\)th prisoner is killed, \(j\)th prisoner to be executed.

**Output:** \(C(n, k, j)\).

1. \(S_{\text{curr}} \leftarrow k(n - j) + 1; S_{\text{prev}} \leftarrow S_{\text{curr}}\)
2. while \(S_{\text{curr}} \leq nk + 1\) do
   3. \(S_{\text{prev}} \leftarrow S_{\text{curr}}; S_{\text{curr}} \leftarrow \left\lceil \frac{k}{k - 1} \times S_{\text{curr}} \right\rceil\)
4. \(C(n, k, j) \leftarrow nk + 1 - S_{\text{prev}}\)
5. return \(C(n, k, j)\)

Figure 109: Algorithm to solve the circle of death puzzle using Recurrence 36.
Then define the least integer \( \ell \) such that \( B_{\ell-1} < kj + 1 < B_\ell \). Finally, \( C(n, k, j) \) can now be computed as \( C(n, k, j) = kj + 1 - B_{\ell-1} \). Figure \( \mathbf{110} \) gives an algorithm to compute \( C(n, k, j) \).

For example, let’s compute \( C(9, 5, 8) \). We see that \([B_1, B_2, \ldots, B_{10}] = [1, 3, 5, 8, 12, 17, 23, 30, 39, 50] \). As \( 39 < 5 \times 8 + 1 < 50 \), we compute \( C(9, 5, 8) = 5 \times 8 + 1 - 39 = 2 \).

\[
\langle \text{Time, Space} \rangle \text{ complexity to compute } C(n, k, j) \text{ is } \langle ?, \Theta(1) \rangle.
\]

**[Knuth’s algorithm.]** This algorithm uses a decreasing sequence to compute \( C(n, k, j) \) for a given \( n, k, \) and \( j \in \{1, 2, 3, \ldots, n\} \). First, define the sequence \( K^{(n,k,j)} \) dependent on the three parameters \( n, k, \) and \( j \) as

\[
K^{(n,k,j)} = [K_1, K_2, K_3, \ldots] \text{ where } K_i = \begin{cases} 
kj & \text{if } i = 1, \\
\frac{k(K_{i-1}-n)-1}{k-1} & \text{if } i > 1.
\end{cases}
\] (38)

Then define the least integer \( \ell \) such that \( K_\ell \leq n \). Finally, \( C(n, k, j) \) can now be computed as \( C(n, k, j) = K_\ell \). Figure \( \mathbf{111} \) gives an algorithm to compute \( C(n, k, j) \).

For example, let’s compute \( C(9, 5, 8) \). We see that \([K_1, K_2, \ldots, K_9] = [40, 38, 36, 33, 29, 24, 18, 11, 2] \). As \( 2 \leq 9 \), we compute \( C(9, 5, 8) = 2 \).

\[
\langle \text{Time, Space} \rangle \text{ complexity to compute } C(n, k, j) \text{ is } \langle ?, \Theta(1) \rangle.
\]

**[Uchiyama’s algorithm.]** This algorithm uses three inter-dependent sequences \( T, U^*, \) and \( U \) to compute \( C(n, k, j) \) for a given \( n \geq 2, k \geq 2, \) and \( j \in \{1, 2, 3, \ldots, n\} \). The \( i \)th term of the three sequences are defined as follows.

\[
T_i = \begin{cases} 
n - j & \text{if } i = 1 \text{ and } j < n, \\
1 & \text{if } i = 1 \text{ and } j = n, \\
\left\lfloor \frac{k(T_{i-1}+1)-U_{i-1}}{k-1} \right\rfloor & \text{if } i > 1.
\end{cases}
\] (39)

\[
U_i^* = \begin{cases} 
T_{i+1} & \text{if } i = 1 \text{ and } j < n \text{ and } k \mod (T_1 + 1) = 0, \\
k \mod (T_1 + 1) & \text{if } i = 1 \text{ and } j < n \text{ and } k \mod (T_1 + 1) \neq 0, \\
1 & \text{if } i = 1 \text{ and } j = n \text{ and } k \mod 2 = 0, \\
2 & \text{if } i = 1 \text{ and } j = n \text{ and } k \mod 2 \neq 0, \\
U_{i-1} + (k-1)(T_i + 1) - k(T_{i-1} + 1) & \text{if } i > 1.
\end{cases}
\] (40)

\[
U_i = \begin{cases} 
U_i^* & \text{if } i = 1, \\
T_{i+1} & \text{if } i > 1 \text{ and } U_i^* \mod (T_1 + 1) = 0, \\
U_i^* \mod (T_1 + 1) & \text{if } i > 1 \text{ and } U_i^* \mod (T_1 + 1) \neq 0.
\end{cases}
\] (41)

Then define the least integer \( \ell \) such that \( T_\ell \geq n \). Finally \( C(n, k, j) \) can now be computed as \( C(n, k, j) = U_{\ell-1} + k(n - T_{\ell-1} - 1) \). Figure \( \mathbf{112} \) gives an algorithm to compute \( C(n, k, j) \).

\[
\langle \text{Time, Space} \rangle \text{ complexity to compute } C(n, k) \text{ is } \langle ?, \Theta(1) \rangle.
\]
**CIRCLEOFDEATH-BUSCHE(n, k, j)**

**Input:** n prisoners, every kth prisoner is killed, jth prisoner to be executed.
**Output:** C(n, k, j).
1. B<sub>curr</sub> ← 1; B<sub>prev</sub> ← B<sub>curr</sub>
2. **while** B<sub>curr</sub> ≤ kj + 1 **do**
   3. B<sub>prev</sub> ← B<sub>curr</sub>; B<sub>curr</sub> ← ⌈ k − 1 × (B<sub>curr</sub> + n − j) ⌉
4. C(n, k, j) ← kj + 1 − B<sub>prev</sub>
5. **return** C(n, k, j)

Figure 110: Algorithm to solve the circle of death puzzle using Recurrence 37.

**CIRCLEOFDEATH-KNUTH(n, k, j)**

**Input:** n prisoners, every kth prisoner is killed, jth prisoner to be executed.
**Output:** C(n, k, j).
1. K ← kj
2. **while** K > n **do** K ← ⌊ k(k−n)−1 k−1 ⌋
3. C(n, k, j) ← K
4. **return** C(n, k, j)

Figure 111: Algorithm to solve the circle of death puzzle using Recurrence 38.

**CIRCLEOFDEATH-UCHIYAMA(n, k, j)**

**Input:** n prisoners, every kth prisoner is killed, jth prisoner to be executed.
**Output:** C(n, k, j).
1. **if** j < n **then**
   2. T<sub>curr</sub> ← n − j
   3. **if** U* = 0 **then** U* ← T<sub>curr</sub> + 1
   4. **else if** U* ≠ 0 **then** U* ← k mod (T<sub>curr</sub> + 1)
5. **else if** j = n **then**
   6. T<sub>curr</sub> ← 1; U* ← 1 + (k mod 2)
7. U<sub>curr</sub> ← U*; U<sub>prev</sub> ← U<sub>curr</sub>; T<sub>prev</sub> ← T<sub>curr</sub>
8. **while** T<sub>curr</sub> < n **do**
   9. T<sub>prev</sub> ← T<sub>curr</sub>; U<sub>prev</sub> ← U<sub>curr</sub>
   10. T<sub>curr</sub> ← ⌊ k(T<sub>prev+1</sub>−U<sub>curr</sub>) k−1 ⌋
   11. U* ← U<sub>curr</sub> + k(T<sub>curr</sub> − T<sub>prev</sub>) − (T<sub>curr</sub> + 1)
   12. U<sub>curr</sub> ← U* mod (T<sub>curr</sub> + 1)
13. **if** U<sub>curr</sub> = 0 **then** U<sub>curr</sub> ← T<sub>curr</sub> + 1
14. C(n, k, j) ← U<sub>prev</sub> + k(n − T<sub>prev</sub> − 1)
15. **return** C(n, k, j)

Figure 112: Algorithm to solve the circle of death puzzle using Recurrences 39, 40, 41.

[Halbeisen-Hungerbühler’s algorithm.] This algorithm uses two interdependent
sequences to compute \( C(n, k, j) \) for a given \( n, k \geq 2 \), and \( j \in \{1, 2, 3, \ldots, n\} \) as follows:

\[
A^{(n,k,j)} = [A_1, A_2, A_3, \ldots] \quad \text{where} \quad A_i = \begin{cases} 
(n - j) + \left\lfloor \frac{n-j+1}{k-1} \right\rfloor & \text{if } i = 1, \\
\left\lfloor \frac{kA_{i-1} - B_{i-1}}{k-1} \right\rfloor & \text{if } i > 1.
\end{cases}
\] (42)

\[
B^{(n,k,j)} = [B_1, B_2, B_3, \ldots] \quad \text{where} \quad B_i = \begin{cases} 
\left\lfloor \frac{n-j+1}{k-1} \right\rfloor k - 1 \mod (n - j + \left\lfloor \frac{n-j+1}{k-1} \right\rfloor) & \text{if } i = 1, \\
(B_{i-1} - A_{i-1}) \mod (k - 1) & \text{if } i > 1.
\end{cases}
\] (43)

Then define the least integer \( \ell \) such that \( \ell > 1 \) and \( A_\ell > n \). Then, \( C(n, k, j) \) is computed as \( C(n, k, j) = (n - A_{\ell-1})k + B_{\ell-1} + 1 \). It is important to note that the original discoverers of this algorithm did not define \( C(n, k, j) \) when \( \ell = 1 \) and \( A_\ell > n \); we can use Takakazu-Euler-Tait’s algorithm in that case. It can be shown that for \( M = 500, n \in [1, M], k \in [2, M], \) and \( j \in [1, n] \), more than 99% of the combinations of \( n, k, j \) satisfy the condition (there exists a least integer \( \ell \) such that \( \ell > 1 \) and \( A_\ell > n \)) of the algorithm; when \( M \) increases this success probability keeps increasing. Figure 113 gives an algorithm to compute \( C(n, k, j) \).

For example, let’s compute \( C(9, 5, 8) \). We see that \([A_1, A_2, \ldots, A_8] = [2, 3, 4, 5, 6, 7, 9, 12]\) and \([B_1, B_2, \ldots, B_8] = [0, 2, 3, 3, 2, 0, 1, 0]\). As \( 2 \leq 9 \), we compute \( C(9, 5, 8) = 2 \). As \( 8 \) is the least integer such that \( A_8 > 9 \), we compute \( C(9, 5, 8) = (9 - A_7) \times 5 + B_7 + 1 = 2 \).

[**Burde-Thériault’s algorithm.**] This algorithm works only for \( k \geq 2 \). Suppose \( \ell = n - j + 1 \). If \( k \geq n \), then we use Takakazu-Euler-Tait’s algorithm to compute \( C(n, k, j) \). On the other hand, if \( k < n \), then we follow the logic as described below.

First, we compute \( f_0, n_0, \) and \( B_0 \) as follows:

\[
f_0 = \begin{cases} 
((-\ell) \mod (k-1)) + 1 & \text{if } k \leq \ell, \\
C(k, k, k - \ell + 1) \text{ using Takakazu-Euler-Tait’s algorithm} & \text{if } k > \ell.
\end{cases}
\] (44)

\[
n_0 = \begin{cases} 
\left\lfloor \frac{f_0 + k(-1)}{k-1} \right\rfloor & \text{if } k \leq \ell, \\
\frac{k}{k} & \text{if } k > \ell.
\end{cases}
\] (45)

\[
B_0 = f_0.
\] (46)

If \( n < n_0 \), then \( C(n, k, j) = k \).

If \( n \geq n_0 \), then we compute \( f_i, n_i, \) and \( B_i \) for \( i > 0 \) as follows:

\[
f_i = ((B_{i-1} - n_i - 1) \mod (k-1)) + 1 - B_{i-1},
\] (47)

\[
n_i = \left\lfloor \frac{kn_{i-1} + f_i}{k-1} \right\rfloor,
\] (48)

\[
B_i = B_{i-1} + f_i.
\] (49)

Then, define the least integer \( p \) such that \( n_{p-1} \leq n < n_p \). Finally, \( C(n, k, j) \) can be computed as \( C(n, k, j) = B_{p-1} + k(n - n_{p-1}) \).

\[<\text{Time, Space}>\text{ complexity to compute } C(n, k, j) \text{ is } \Theta(n), n \leq k + O(\log \frac{n}{\log n}) \cdot \Theta(1) \text{, where } n_0 \text{ is from Equation 45.}\]
The algorithm uses the recurrence
\[ Gelgi’s algorithm. \]
\[ Booth’s algorithm. \]

\[ Gelgi’s algorithm. \] The algorithm uses the recurrence \( G(n, k) \) which is a number in the range \( \{0, 1, 2, \ldots, n - 1\} \). Figure 116 gives both recursive and nonrecursive algo-
CIRCLE OF DEATH—BURDE-ThÉRIAULT\((n, k, j)\)

**Input:** \(n\) prisoners, every \(k\)th person is killed, \(j\)th prisoner to be executed

**Output:** \(C(n, k, j)\).

1. \(\ell \leftarrow n - j + 1\)
2. **if** \(k = 1\) **then** \(B \leftarrow j\)
3. **else** if \(n \leq k\) **then**
   5. \(B \leftarrow CIRCLE OF DEATH-TAKAKAZU-EULER-TAIT(n, k, j)\)
4. **else if** \(n \geq k\) **then**
   3. \(B \leftarrow j\)
   4. **else**
   6. // If \(n \leq k\), then use Burde-Thériault’s logic ..............
   8. // Compute \(f_0\) and \(n_0\) ...........................
   11. **if** \(\ell < k\) **then**
   12. \(f \leftarrow CIRCLE OF DEATH-TAKAKAZU-EULER-TAIT(k, k - \ell + 1)\)
   13. \(n_{\text{curr}} \leftarrow k\)
   14. **else**
   16. **if** \(n < n_{\text{curr}}\) **then**
   17. \(B \leftarrow kj\)
   18. **else**
   20. \(n_{\text{prev}} \leftarrow n_{\text{curr}}\)
   21. \(B_{\text{curr}} \leftarrow f; B_{\text{prev}} \leftarrow B_{\text{curr}}\)
   22. \(f \leftarrow (B_{\text{curr}} - n_{\text{curr}} - 1) \mod (k - 1) + 1 - B_{\text{curr}}\)
   23. **while** \(n_{\text{curr}} \leq n\) **do**
   24. \(n_{\text{prev}} \leftarrow n_{\text{curr}}; n_{\text{curr}} \leftarrow \left\lfloor \frac{n_{\text{curr}} \times k + f}{k - 1} \right\rfloor\)
   25. \(B_{\text{prev}} \leftarrow B_{\text{curr}}; B_{\text{curr}} \leftarrow B_{\text{curr}} + f\)
   26. \(f \leftarrow ((B_{\text{curr}} - n_{\text{curr}} - 1) \mod (k - 1)) + 1 - B_{\text{curr}}\)
   27. \(B \leftarrow B_{\text{prev}} + (n - n_{\text{prev}})k\)
28. \(C(n, k, j) \leftarrow B\)
29. **return** \(C(n, k, j)\)

Figure 115: Algorithm to solve the circle of death puzzle using Recurrences \[44-49\]

Given algorithms to compute \(C(n, k)\) when \(k \geq 2\).

\[
G(n, k) = \begin{cases} 
1 & n = 0, \\
(G(n - 1, k) + k) \mod n & n \in [1, k - 1], \\
S + \left\lfloor \frac{S}{k - 1} \right\rfloor \cdot \begin{cases} S \geq 0 & S < 0 
\end{cases} + n \cdot \begin{cases} S \geq 0 & S < 0 
\end{cases} & n \geq k.
\end{cases}
\]  

\(C(n, k) = G(n, k) + 1\)  

(51)  

(52)
Here, \([L]\) represents the Iverson bracket for the logical expression \(L\). The term is 1 if the logical expression \(L\) is true and the term is 0 if the logical expression \(L\) is false.

\[
\langle \text{Time, Space}\rangle \text{ complexity to compute } C(n, k) \text{ is }
\Theta(n) \cdot \begin{cases} n \leq k & + O\left(k + \log\left(\frac{n}{k}\right)\right) \cdot n > k, \\ n \leq k & + \Theta\left(1 + \frac{n}{k}\right) \cdot \begin{cases} n \leq k & + \Theta\left(\log\left(\frac{n}{k}\right)\right) \cdot n > k \end{cases} \end{cases}
\]

**Complexity**

We implement all algorithms in Python (Google Colab). We provide the results of two experiments. In experiment 1 as shown in Figure , we fix \(k = 2^9\) and \(j = n\) and increase \(n\) in the range \([2^1, 2^{15}]\) in powers of 2. In experiment 2, we fix \(n = 2^{10}\) and \(j = n\) and increase \(k\) in the range \([2^1, 2^{20}]\) in powers of 2. We see that Burde-Thériault’s algorithm runs the fastest in both experiments.

**References**

The idea of the puzzle was introduced by Flavius Josephus in the book *The Jewish War* in 75 AD. [Augenstein and Tenenbaum, 1977] show how to solve the puzzle using different data structures such as arrays, circularly linked lists, and binary search trees. H. Schubert’s algorithm (1895) and E. Busche’s algorithm (1896) are very nicely presented by [Uchiyama, 2003]. According to Uchiyama, Takakazu-Euler-Tait’s algorithm is from Seki Takakazu (1683), Leonhard Euler (1776), and P. G. Tait (1900). The complicated recurrence of Uchiyama’s algorithm is given in [Uchiyama, 2003]. Knuth’s algorithm and Herstein-Kaplansky’s algorithm are nicely presented in [Wilson, 1979], who also explains Takakazu-Euler-Tait’s algorithm and Busche’s algorithm (Wilson calls it Ahrens algorithm but its discoverer is Busche). Jakobczyk’s algorithm is given in [Jakobczyk, 1973], Lloyd’s algorithm is given in [Lloyd, 1983], Gelgi’s algorithm is given in [Gelgi, 2002], Halbeisen-Hungerbühler’s algorithm is given in [Halbeisen and Hungerbühler, 1997], and Booth’s algorithm is given in [Booth, 1975]. [Ahrens, 1918] gives a detailed account of the Josephus problem. Schubert’s and Busche’s algorithm coincide with each other when \(j = n\) and this result is rediscovered in [Odlyzko and Wilf, 1991].
**CIRCLEOFDEATH-GELGI(n, k)**

**Input:** \( n \) prisoners, every \( k \)th prisoner is killed.

**Output:** \( C(n, k) \).

1. \( C(n, k) \leftarrow \text{GELGI}(n, k) + 1 \)
2. \textbf{return} \( C(n, k) \)

**GELGI(n, k)**

**Output:** An index in the range \([0, n - 1]\).

1. \textbf{if} \( n \leq 1 \) \textbf{then return} 0
2. \textbf{else if} \( n < k \) \textbf{then}
3. \hspace{1em} \( G \leftarrow \text{CIRCLEOFDEATH-TAKAKAZUEULERTAIT}(n, k, n) - 1 \)
4. \textbf{else if} \( n \geq k \) \textbf{then}
5. \hspace{1em} \( j \leftarrow \text{GELGI}(n - \left\lfloor \frac{n}{k} \right\rfloor, k) \)
6. \hspace{1em} \( T \leftarrow j - (n \mod k) \)
7. \hspace{1em} \( G \leftarrow T + \left\lfloor \frac{T}{k-1} \right\rfloor \)
8. \hspace{1em} \textbf{if} \( T < 0 \) \textbf{then} \( G \leftarrow T + n \)
9. \textbf{return} \( G \)

**CIRCLEOFDEATH-GELGI-NONRECURSIVE(n, k)**

**Input:** \( n \) prisoners, every \( k \)th prisoner is killed.

**Output:** \( C(n, k) \).

1. Stack \( S \leftarrow [\ ]; n_{\text{curr}} \leftarrow n \)
2. \textbf{while} \( n_{\text{curr}} \geq k \) \textbf{do}
3. \hspace{1em} \( S.\text{Push}(n_{\text{curr}}); n_{\text{curr}} \leftarrow n_{\text{curr}} - \left\lfloor \frac{n_{\text{curr}}}{k} \right\rfloor \)
4. \textbf{if} \( n_{\text{curr}} < k \) \textbf{and} \( n_{\text{curr}} \geq 1 \) \textbf{then}
5. \hspace{1em} \( S.\text{Push}(n_{\text{curr}}); n_{\text{curr}} \leftarrow n_{\text{curr}} - 1 \)
6. \( G_{\text{prev}} \leftarrow 0; G_{\text{curr}} \leftarrow G_{\text{prev}} \)
7. \textbf{while} Stack \( S \) is not empty \textbf{do}
8. \hspace{1em} \( n_{\text{curr}} \leftarrow S.\text{Pop}() \)
9. \hspace{1em} \textbf{if} \( n_{\text{curr}} = 1 \) \textbf{then} \( G_{\text{curr}} \leftarrow 1 \)
10. \hspace{1em} \textbf{else if} \( n_{\text{curr}} < k \) \textbf{then}
11. \hspace{2em} \( G_{\text{curr}} \leftarrow \text{CIRCLEOFDEATH-TAKAKAZUEULERTAIT}(n_{\text{curr}}, k, n_{\text{curr}}) - 1 \)
12. \hspace{1em} \textbf{else if} \( n_{\text{curr}} \geq k \) \textbf{then}
13. \hspace{2em} \( T \leftarrow G_{\text{prev}} - (n_{\text{curr}} \mod k) \)
14. \hspace{2em} \( G_{\text{curr}} \leftarrow T + \left\lfloor \frac{T}{k-1} \right\rfloor \)
15. \hspace{2em} \textbf{if} \( T < 0 \) \textbf{then} \( G_{\text{curr}} \leftarrow T + n_{\text{curr}} \)
16. \hspace{1em} \( G_{\text{prev}} \leftarrow G_{\text{curr}} \)
17. \( C(n, k) \leftarrow G_{\text{curr}} + 1 \)
18. \textbf{return} \( C(n, k) \)

Figure 116: Algorithm to solve the circle of death puzzle using Recurrence 51.
Figure 117: Performance evaluation of algorithms for solving the circle of death puzzle when $n$ is varying.

Figure 118: Performance evaluation of algorithms for solving the circle of death puzzle when $k$ is varying.
Coin Weighing

Problem
A king receives taxes in gold coins. The king finds out that a bad minister has cheated him by paying a bad gold coin as tax. The bad news is that the bad coin has made its way into the treasury and the bad coin looks identical to all remaining coins. The good news is that the bad coin has a different weight than others. What strategy should the king use to identify the defective coin using a two-sided weighing balance/scale and minimizing the number of weighings?

Solution
The puzzle belongs to a branch of mathematics called group testing. Using a two-sided weighing balance, we can compare two sets of coins. Suppose the two sets $A$ and $B$ have the same number of coins and they are placed on the left and right sides of the weighing balance, respectively. The result of the weighing tells if set $A$ is lighter than or heavier than or weights the same as set $B$. Let’s assume that there are $n \geq 1$ identical-looking coins are they are labeled as $1, 2, 3, \ldots, n$.

Variant: Find the lighter coin
Let’s consider a simpler variant before solving the original problem. Suppose we have $n$ coins and exactly one of them is lighter than the others. How do you identify the lighter coin using a two-sided weighing balance/scale and minimizing the number of weighings?

Let $S$ be the given set of $n$ coins, i.e., $S = \{1, 2, 3, \ldots, n\}$. Let sets $S_1$, $S_2$, and $S_3$ be the partition of set $S$. Let weight$(A)$ denote the sum of weights of all coins in the set $A$.

[Linear recursive algorithm (non-optimal).] Divide $S$ into three sets: $S_1$ contains the first coin, $S_2$ contains the second coin, and $S_3$ contains the remaining $n-2$ coins. Weigh the two sets $S_1$ and $S_2$. If weight$(S_1) < \text{weight}(S_2)$, then $S_1$ has the lighter coin. If weight$(S_1) > \text{weight}(S_2)$, then $S_2$ has the lighter coin. If weight$(S_1) = \text{weight}(S_2)$, then the lighter coin is present in $S_3$ and hence recursively identify the lighter coin in that set. If $n$ is odd and if the last weighing stays balanced, then the only remaining $n$th coin is the lighter coin. Figure 119 illustrates an example and FINDLIGHTER-LINEAR($\{1, 2, \ldots, n\}$) represents the algorithm. Let $W(n)$ denote the worst-case number of weighings by this algorithm. Then

$$W(n) = \begin{cases} 0 & \text{if } n = 0 \text{ or } n = 1, \\ W(n-2) + 2 & \text{if } n > 1. \end{cases}$$
\[ W(n) = \left\lfloor \frac{n}{2} \right\rfloor. \] For example, \( W(9) = 4. \)

Figure 119: Linear recursive algorithm for 9 coins.

Figure 120: Binary search algorithm for 9 coins.

Figure 121: Ternary search algorithm for 9 coins.

**[Binary search algorithm (non-optimal).]** Divide \( S \) into three sets: \( S_1 \) contains the first \( \left\lfloor \frac{n}{3} \right\rfloor \) coins, \( S_2 \) contains the second \( \left\lfloor \frac{n}{3} \right\rfloor \) coins, and \( S_3 \) contains the extra coin if \( n \) is odd. Weigh the two sets \( S_1 \) and \( S_2 \). If \( \text{weight}(S_1) < \text{weight}(S_2) \) (or \( \text{weight}(S_1) > \text{weight}(S_2) \)), then the lighter coin is present in the set \( S_1 \) (or \( S_2 \)) and hence recursively...
The lighter coin in that set. If weight$(S_1) =$ weight$(S_2)$, then the lighter coin is the only coin in set $S_3$. If a set is reached that contains a single coin, then that coin is lighter. Figure 120 illustrates an example and FINDLIGHTER-BINARY([1,2,...,n]).
represents the algorithm. Let $W(n)$ denote the worst-case number of weighings by this algorithm. Then

$$W(n) = \begin{cases} 0 & \text{if } n = 1, \\ W\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + 1 & \text{if } n > 1. \end{cases}$$

$W(n) = \lceil \log_2 n \rceil$. For example, $W(9) = 3$.

**Ternary search algorithm (optimal).** Divide $S$ into three sets: $S_1$ contains the first $\left\lfloor \frac{n}{3} \right\rfloor$ coins, $S_2$ contains the second $\left\lfloor \frac{n}{3} \right\rfloor$ coins, and $S_3$ contains the remaining $\left( n - 2 \left\lfloor \frac{n}{3} \right\rfloor \right)$ coins. Weigh the two sets $S_1$ and $S_2$. If $\text{weight}(S_1) < \text{weight}(S_2)$ (or $\text{weight}(S_1) > \text{weight}(S_2)$), then the lighter coin is present in the set $S_1$ (or set $S_2$ or set $S_3$, respectively) and hence recursively identify the lighter coin in that set. If a set is reached that contains a single coin, then that coin is lighter. Figure 120 illustrates an example and $\text{FINDLIGHTER-TERNARY}(\{1, 2, \ldots, n\})$ represents the algorithm. Let $W(n)$ denote the worst-case number of weighings by this algorithm. Then

$$W(n) = \begin{cases} n - 1 & \text{if } n = 1 \text{ or } n = 2, \\ W\left(\left\lceil \frac{n}{3} \right\rceil \right) + 1 & \text{if } n > 2. \end{cases}$$

$W(n) = \lceil \log_3 n \rceil$. For example, $W(9) = 2$.

**Original puzzle: Find the bad coin**

It is easy to see that when $n = 1$ or $n = 2$, it is algorithmically impossible to identify the bad coin.

**Fine-Manvel’s ternary search algorithm (optimal)**

We give this algorithm in three parts. The first two parts consider tiny variants of the puzzle. The third part makes use of the algorithms given for the first two parts to solve the original puzzle.

**Part 1. Find the bad coin in a set $S$ given that each coin is labeled “possibly lighter” (p.l.) or “possibly heavier” (p.h.) and given an extra good coin.**

Suppose each coin in the set $S$ is labeled p.l. or p.h. If a coin is labeled p.l., that coin can be a good coin or a lighter coin. The label p.h. can be defined similarly.

Divide the given set $S$ into three mutually disjoint sets: $S_1$ contains $\left\lfloor \frac{n}{3} \right\rfloor$ coins, $S_2$ contains $\left\lceil \frac{n}{3} \right\rceil$ coins, and $S_3$ contains the remaining $\left( n - 2 \left\lfloor \frac{n}{3} \right\rfloor \right)$ coins. The only constraint we need to make sure in the division is that sets $S_1$ and $S_2$ must contain the same number of p.l. coins (and hence the same number of p.h. coins). Weigh the two sets $S_1$ and $S_2$. If $\text{weight}(S_1) < \text{weight}(S_2)$, then the bad coin is present among the p.l. coins of $S_1$ or among the p.h. coins of $S_2$ and hence recursively identify the bad coin among
**FINDBAD-PLPH(S, good coin)**

**Input:** Set of coins having a bad coin; coins in set $S$ labeled as p.l. or p.h.

**Output:** The bad coin.

1. $n \leftarrow |S|$
2. if $n = 1$ then **return** the only coin in $S$
3. if $n = 2$ then
4. weigh one of the two coins and the extra good coin
5. **return** the selected coin if unbalanced; the remaining coin if balanced
6. $S_1 \leftarrow \left\lfloor \frac{n}{3} \right\rfloor$ coins of $S$; $S_2 \leftarrow$ another $\left\lfloor \frac{n}{3} \right\rfloor$ coins of $S$; $S_3 \leftarrow S - S_1 - S_2$
7. constraint: make sure the number of p.l. coins in $S_1$ is the same as that in $S_2$
8. weigh sets $S_1$ and $S_2$
9. if $\text{weight}(S_1) < \text{weight}(S_2)$ then
10. **return** FINDBAD-PLPH(p.l. coins of $S_1 \cup$ p.h. coins of $S_2$, good coin)
11. else if $\text{weight}(S_1) > \text{weight}(S_2)$ then
12. **return** FINDBAD-PLPH(p.h. coins of $S_1 \cup$ p.l. coins of $S_2$, good coin)
13. else if $\text{weight}(S_1) = \text{weight}(S_2)$ then
14. **return** FINDBAD-PLPH($S_3$, good coin)

**FINDBAD-GOOD(S, good coin)**

**Input:** Set of coins having a bad coin.

**Output:** The bad coin.

1. $n \leftarrow |S|$
2. if $n = 1$ then
3. weigh the only coin in $S$ and a good coin
4. **return** the coin with the result of the weighing
5. else if $n = 2$ then
6. weigh one of the two coins and the extra good coin
7. **return** the selected coin if weighing is unbalanced
8. weigh the remaining coin and the extra good coin
9. **return** the remaining coin
10. $S_1 \leftarrow$ first $\left\lfloor \frac{n}{3} \right\rfloor$ coins of $S$; $S_2 \leftarrow$ second $\left\lfloor \frac{n}{3} \right\rfloor$ coins of $S$; $S_3 \leftarrow S - S_1 - S_2$
11. weigh sets $S_1$ and $S_2$
12. if $\text{weight}(S_1) < \text{weight}(S_2)$ then
13. label every coin of $S_1$ (or $S_2$) as p.l. (or p.h., respectively)
14. **return** FINDBAD-PLPH($S_1 \cup S_2$, a coin from $S_3$)
15. else if $\text{weight}(S_1) > \text{weight}(S_2)$ then
16. label every coin of $S_1$ (or $S_2$) as p.h. (or p.l., respectively)
17. **return** FINDBAD-PLPH($S_1 \cup S_2$, a coin from $S_3$)
18. else if $\text{weight}(S_1) = \text{weight}(S_2)$ then
19. **return** FINDBAD-GOOD($S_3$, a coin from $S_1$ or $S_2$)
those coins. If weight\(S_1 > \text{weight}(S_2)\), then the bad coin is present among the p.h. coins of \(S_1\) or among the p.l. coins of \(S_2\) and hence recursively identify the bad coin among those coins. If weight\(S_1 = \text{weight}(S_2)\), then the bad coin is present in the set \(S_3\) and hence recursively identify the bad coin in that set. If a set is reached that contains a single coin, then that coin is either lighter or heavier based on its label. If a set is reached that contains two coins, then weigh one of the two coins and the extra good coin. If the scale is unbalanced, then the selected coin is the bad coin. If the scale is balanced, then the remaining coin is the bad coin. Figure 120 illustrates an example and FINDBAD-PLPH({1, 2, ..., \(n\)}) represents the algorithm. Let \(W(n)\) denote the worst-case number of weighings by this algorithm. Then

\[
W(n) = \begin{cases} 
  n-1 & \text{if } n = 1 \text{ or } n = 2, \\
  W\left(\left\lfloor \frac{n}{3} \right\rfloor \right) + 1 & \text{if } n > 2.
\end{cases}
\]

\(W(n) = \lceil \log_3 n \rceil\). For example, \(W(9) = 2\).

[Part 2. Find the bad coin given a good coin.] Divide the given set \(S\) into three sets: \(S_1\) contains \(\left\lfloor \frac{n}{3} \right\rfloor + 1\) coins, \(S_2\) contains \(\left\lfloor \frac{n}{3} \right\rfloor\) coins, and \(S_3\) contains the remaining \((n - 2 \left\lfloor \frac{n}{3} \right\rfloor - 1)\) coins. Weigh the two sets \(S_1\) and \((S_2 \cup \{\text{extra good coin}\})\). If weight\(S_1 < \text{weight}(S_2 \cup \{\text{extra good coin}\})\), then label each of the coins in \(S_1\) as p.l. and label each of the coins in \(S_2\) as p.h. and identify the bad coin present in \(S_1 \cup S_2\) using FINDBAD-PLPH algorithm and a good coin. If weight\(S_1 > \text{weight}(S_2 \cup \{\text{extra good coin}\})\), then label each of the coins in \(S_1\) as p.h. and label each of the coins in \(S_2\) as p.l. and identify the bad coin present in \(S_1 \cup S_2\) using FINDBAD-PLPH algorithm and a good coin. If weight\(S_1 = \text{weight}(S_2 \cup \{\text{extra good coin}\})\), then the bad coin is present in the set \(S_3\) and hence recursively find the bad coin in that set. If a set is reached that contains one/two coins, then it is straightforward to identify the bad coin making use of an extra good coin in one/two weighings, respectively. Figure 120 illustrates an example and FINDBAD-GOOD({1, 2, ..., \(n\)}) represents the algorithm. Let \(W(n)\) denote the worst-case number of weighings by this algorithm. FINDBAD-PLPH\((S_1 \cup S_2)\) with labels for the coins in sets \(S_1\) and \(S_2\) requires \(\lceil \log_3 (|S_1| + |S_2|) \rceil = \lceil \log_3 \left(2 \left\lfloor \frac{n}{3} \right\rfloor + 1\right)\rceil\) weighings from Part 1. FINDBAD-GOOD\((S_3)\) requires \(W\left(n - 2 \left\lfloor \frac{n}{3} \right\rfloor - 1\right)\) weighings. We have

\[
W(n) = \begin{cases} 
  n-1 & \text{if } n = 1 \text{ or } n = 2, \\
  W\left(n - 2 \left\lfloor \frac{n}{3} \right\rfloor - 1\right) + \lceil \log_3 \left(2 \left\lfloor \frac{n}{3} \right\rfloor + 1\right)\rceil & \text{if } n > 2.
\end{cases}
\]

\(W(n) = \lceil \log_3 (2n + 1) \rceil\). For example, \(W(13) = 3\).

[Part 3. Find the bad coin.] Divide the given set \(S\) into three sets: \(S_1\) contains \(\left\lfloor \frac{n}{3} \right\rfloor\) coins, \(S_2\) contains \(\left\lfloor \frac{n}{3} \right\rfloor\) coins, and \(S_3\) contains the remaining \((n - 2 \left\lfloor \frac{n}{3} \right\rfloor)\) coins. Weigh the two sets \(S_1\) and \(S_2\). If weight\(S_1 < \text{weight}(S_2)\), then label each of the coins in \(S_1\) as p.l. and label each of the coins in \(S_2\) as p.h. and identify the bad coin present in \(S_1 \cup S_2\)
**Algorithm.** $F$ represents the algorithm. Let $W \subseteq S$ find the bad coin in that set using the algorithm and a good coin from $S$, then label each of the coins in $S$ using $F$. A label is the same as the representation in the ternary number system. Each coin will get two labels: a *left (or clockwise) label* called $l$ and a *right (or anticlockwise) label* called $r$. The function $\text{GENERATETERNARYLABELS}(w,n)$ generates the following:

<p>| <strong>Input:</strong> | Set of coins having a bad coin. |</p>
<table>
<thead>
<tr>
<th><strong>Output:</strong></th>
<th>The bad coin.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $n \leftarrow</td>
<td>S</td>
</tr>
<tr>
<td>2. <strong>if</strong> $n = 1$ or $n = 2$ <strong>then</strong> there is no solution</td>
<td></td>
</tr>
<tr>
<td>3. $S_1 \leftarrow \text{first } \left\lceil \frac{n}{3} \right\rceil$ coins of $S$; $S_2 \leftarrow \text{second } \left\lceil \frac{n}{3} \right\rceil$ coins of $S$; $S_3 \leftarrow S - S_1 - S_2$</td>
<td></td>
</tr>
<tr>
<td>4. weigh sets $S_1$ and $S_2$</td>
<td></td>
</tr>
<tr>
<td><strong>if</strong> $\text{weight}(S_1) &lt; \text{weight}(S_2)$ <strong>then</strong></td>
<td></td>
</tr>
<tr>
<td>6. label every coin of $S_1$ (or $S_2$) as p.l. (or p.h., respectively)</td>
<td></td>
</tr>
<tr>
<td>7. return $F$INDBAD-PLPH($S_1 \cup S_2$, a coin from $S_3$)</td>
<td></td>
</tr>
<tr>
<td><strong>else if</strong> $\text{weight}(S_1) &gt; \text{weight}(S_2)$ <strong>then</strong></td>
<td></td>
</tr>
<tr>
<td>9. label every coin of $S_1$ (or $S_2$) as p.h. (or p.l., respectively)</td>
<td></td>
</tr>
<tr>
<td>10. return $F$INDBAD-PLPH($S_1 \cup S_2$, a coin from $S_3$)</td>
<td></td>
</tr>
<tr>
<td><strong>else if</strong> $\text{weight}(S_1) = \text{weight}(S_2)$ <strong>then</strong></td>
<td></td>
</tr>
<tr>
<td>12. return $F$INDBAD-GOOD($S_3$, a coin from $S_1$ or $S_2$)</td>
<td></td>
</tr>
</tbody>
</table>

Using $F$INDBAD-PLPH algorithm and a good coin from $S_3$. If $\text{weight}(S_1) > \text{weight}(S_2)$, then label each of the coins in $S_1$ as p.h. and label each of the coins in $S_2$ as p.l. and identify the bad coin present in $S_1 \cup S_2$ using $F$INDBAD-PLPH algorithm and a good coin from $S_3$. If $\text{weight}(S_1) = \text{weight}(S_2)$, then the bad coin is present in the set $S_3$ and find the bad coin in that set using $F$INDBAD-GOOD algorithm and a good coin (from $S_1$ or $S_2$). Figure [120] illustrates an example and $F$INDBAD-FINEMANVEL($\{1, 2, \ldots, n\}$) represents the algorithm. Let $W(n)$ denote the worst-case number of weighings by this algorithm. $F$INDBAD-PLPH($S_1 \cup S_2$) with labels for the coins in sets $S_1$ and $S_2$ requires $\lceil \log_3(|S_1| + |S_2|) \rceil = \left\lceil \log_3 \left(2 \frac{n}{3} \right) \right\rceil$ weighings from Part 1. $F$INDBAD-GOOD($S_3$) requires $\lceil \log_3(2|S_3| + 1) \rceil = \left\lceil \log_3 \left(2(n - 2 \frac{n}{3}) + 1 \right) \right\rceil$ weighings from Part 2. We have

$$W(n) = \begin{cases} \text{unsolvable} & \text{if } n = 1 \text{ or } n = 2, \\ \lceil \log_3(2n/3) \rceil + \lceil \log_3 (2n/3 + 1) \rceil & \text{if } n > 2. \end{cases}$$

$W(n) = \lceil \log_3(2n + 3) \rceil$ for $n \geq 3$. For example, $W(13) = 4$.

**Dyson’s ternary labels algorithm (optimal)**

Set $w = \lceil \log_3(2n + 3) \rceil$, which is the maximum number of weighings required to find the bad coin in a set of $n$ coins. We give this algorithm in two steps.

**Step 1. Generate ternary labels.** In this step, we generate two labels for each of the $n$ coins. A *label* is a $w$-sized string where each character comes from the set $\{0, 1, 2\}$. A label is the same as the representation in the ternary number system. Each coin will get two labels: a *left (or clockwise) label* called $l$ and a *right (or anticlockwise) label* called $r$. The function $\text{GENERATETERNARYLABELS}(w,n)$ generates
Figure 122: Generating $n_{\text{max}}$ number of left or clockwise ternary labels for $w = 2$, $w = 3$, and $w = 4$ to be used in Dyson's algorithm.

$\ell_{\text{label}}[1\ldots n]$ and $r_{\text{label}}[1\ldots n]$, i.e., a total of $2n$ labels.

Let $n_{\text{max}}(w) = \frac{3^w - 3}{2}$ for $w \geq 2$.

The following table gives the values of $n_{\text{max}}(w)$ for $w$ in the range $[2, 10]$.

<table>
<thead>
<tr>
<th>$w$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{\text{max}}(w)$</td>
<td>3</td>
<td>12</td>
<td>39</td>
<td>120</td>
<td>363</td>
<td>1092</td>
<td>3279</td>
<td>9840</td>
<td>29523</td>
</tr>
</tbody>
</table>

[Part 1. Generating left labels when $n = n_{\text{max}}(w)$] We explain the method for this part using a recursive method. Figure 122 shows how to derive left (or clockwise) labels for $n = 3$, $n = 12$, and $n = 39$.

Left labels for $n = 3$ (or $w = 2$) are $[01 \to 12 \to 20]$

Left labels for $n = 12$ (or $w = 3$) are generated using left labels for $n = 3$ (or $w = 2$) via recursion in the following way. Consider the left labels for $w = 2$, i.e., $[01, 12, 20]$. To get the left labels for $w = 3$, we consider the first third of the labels for $w = 2$, i.e., 01,
and append 0, 1, 2 to it to get \{010, 011, 012\}. We take each of these labels and cyclically rotate them twice. A cyclic rotation of a left label means replacing each of the digits in the label using the order 0 → 1 → 2 → 0. So, after considering cyclic rotations we will have \{010 → 121 → 202, 011 → 122 → 200, 012 → 120 → 201\}. We define a triad for a given \(w\) as a set of three labels \(\{0\ldots 0 1 \rightarrow 1 \ldots 1 2 \rightarrow 2 \ldots 20\}\). So, the triad for \(w = 3\) will be \{001 → 112 → 220\}. We have the following set including the triad.

| Left labels for \(n = 12\) (or \(w = 3\)) are \{010 → 121 → 202, 011 → 122 → 200, 012 → 120 → 201\} and triad \{001 → 112 → 220\} |

Left labels for \(n = 39\) (or \(w = 4\)) are generated using left labels for \(n = 9\) (or \(w = 3\)) via recursion in the following way. To get the left labels for \(w = 4\), we consider the first third of the labels for \(w = 3\), i.e., \{001, 010, 011, 012\}, and append 0, 1, 2 to it to get \{0010, 0011, 0012, 0100, 0101, 0102, 0110, 0111, 0112, 0120, 0121, 0122\}. We take each of these labels and cyclically rotate them twice. So, after considering cyclic rotations we will have \{0010 → 1121 → 2202, 0011 → 1122 → 2200, 0012 → 1120 → 2201, 0100 → 1211 → 2022, 0101 → 1212 → 2020, 0102 → 1210 → 2021, 0110 → 1221 → 2002, 0111 → 1220 → 2001, 0112 → 1201 → 2012, 0121 → 1202 → 2010, 0122 → 1200 → 2011\}. The triad for \(w = 4\) is \{0001 → 1112 → 2220\}. Including the triad we have

| Left labels for \(n = 39\) (or \(w = 4\)) are \{0010 → 1121 → 2202, 0011 → 1122 → 2200, 0012 → 1120 → 2201, 0100 → 1211 → 2022, 0101 → 1212 → 2020, 0102 → 1210 → 2021, 0110 → 1221 → 2002, 0111 → 1220 → 2001, 0112 → 1201 → 2012, 0121 → 1202 → 2010, 0122 → 1200 → 2011\} and triad \{0001 → 1112 → 2220\} |

Similarly, the left labels for \(w > 2\) can be recursively generated using that for \(w − 1\).

Note: We derive the left labels for \(w\) by taking the top third of the left labels for \(w − 1\) and then appending each label them by 0, 1, and 2. But how can we efficiently get the top third of an array of labels? There are at least three approaches we can follow: (i) Sort the array of labels and get the top third of the labels. We have used sorting in Figure 122 for simplicity and pedagogical reasons. (ii) Use any linear-time selection algorithm to find the top third labels of an array (the top third labels need not be sorted). (iii) Use bucketing technique. In this approach, we will have three buckets for labels starting from 0, 1, and 2. Let’s call them 0-bucket, 1-bucket, and 2-bucket, respectively. We know that the top third labels for \(w\) must start from 0. So, if we append all labels in the 0-bucket for \(w − 1\) with 0, 1, and 2, we get all-but-one labels of 0-bucket for \(w\). The remaining label that should be added to the 0-bucket comes from the triad. We can use cyclic rotations to compute 1-bucket and 2-bucket.

[Part 2. Generating left labels for \(n \in (n_{\max}(w − 1), n_{\max}(w))\).] Table 52 shows \(n\) left (or clockwise) labels for various values of \(n\). If we want to generate left labels for \(n \in (n_{\max}(w − 1), n_{\max}(w))\), say \(n = 3q + r\) (where \(r \in \{0, 1, 2\}\)), we assign \(3q\) left labels in \(q\) triples from the set of left labels for \(n_{\max}(w)\) and then assign \(r\) labels from triad using
the following.

\[
\text{Additional } r \text{ left labels } = \begin{cases} 
\frac{1 \ldots 12}{w-1} & \text{if } r = 1, \\
\frac{0 \ldots 01 \text{ and } 2 \ldots 20}{w-1} & \text{if } r = 2, \\
\text{None} & \text{if } r = 0.
\end{cases}
\]

<table>
<thead>
<tr>
<th>(w)</th>
<th>(n)</th>
<th>(n \mod 3)</th>
<th>Left labels</th>
<th>Triad labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td>–</td>
<td>01, 12, 20</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>010 \rightarrow 121 \rightarrow 202</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>010 \rightarrow 121 \rightarrow 202</td>
<td>001, 220</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>010 \rightarrow 121 \rightarrow 202, 011 \rightarrow 122 \rightarrow 200</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>010 \rightarrow 121 \rightarrow 202, 011 \rightarrow 122 \rightarrow 200</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>010 \rightarrow 121 \rightarrow 202, 011 \rightarrow 122 \rightarrow 200</td>
<td>001, 220</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>010 \rightarrow 121 \rightarrow 202, 011 \rightarrow 122 \rightarrow 200, 012 \rightarrow 120 \rightarrow 201</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>010 \rightarrow 121 \rightarrow 202, 011 \rightarrow 122 \rightarrow 200, 012 \rightarrow 120 \rightarrow 201</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>010 \rightarrow 121 \rightarrow 202, 011 \rightarrow 122 \rightarrow 200, 012 \rightarrow 120 \rightarrow 201</td>
<td>001, 220</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>010 \rightarrow 121 \rightarrow 202, 011 \rightarrow 122 \rightarrow 200, 012 \rightarrow 120 \rightarrow 201</td>
<td>001, 112, 220</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>1</td>
<td>0010 \rightarrow 1121 \rightarrow 2202, 0011 \rightarrow 1122 \rightarrow 2200, 0012 \rightarrow 1120 \rightarrow 2201, 0100 \rightarrow 1112</td>
<td>1112</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0010 \rightarrow 1121 \rightarrow 2202, 0011 \rightarrow 1122 \rightarrow 2200, 0012 \rightarrow 1120 \rightarrow 2201, 0100 \rightarrow 0001, 2220</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0010 \rightarrow 1121 \rightarrow 2202, 0011 \rightarrow 1122 \rightarrow 2200, 0012 \rightarrow 1120 \rightarrow 2201, 0100 \rightarrow 1112</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0010 \rightarrow 1121 \rightarrow 2202, 0011 \rightarrow 1122 \rightarrow 2200, 0012 \rightarrow 1120 \rightarrow 2201, 0100 \rightarrow 0001, 2220</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0010 \rightarrow 1121 \rightarrow 2202, 0011 \rightarrow 1122 \rightarrow 2200, 0012 \rightarrow 1120 \rightarrow 2201, 0100 \rightarrow 0001, 1112, 2220</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>39</td>
<td>0</td>
<td>0010 \rightarrow 1121 \rightarrow 2202, 0011 \rightarrow 1122 \rightarrow 2200, 0012 \rightarrow 1120 \rightarrow 2201, 0100 \rightarrow 0001, 1112, 2220</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 52: Generating left or clockwise ternary labels for \(w = 3\), \(w = 3\), and \(w = 4\).

For example, when \(n = 4\), we assign left labels from the triple 010 \rightarrow 121 \rightarrow 202 and another label 112 from the triad because \(4 \mod 3 = 1\). When \(n = 5\), we assign left labels from triple 010 \rightarrow 121 \rightarrow 202 and two labels 001 and 220 from the triad because \(5 \mod 3 = 2\). Continuing in this manner, we can assign left labels for all \(n \in (n_{\text{max}}(w-1), n_{\text{max}}(w))\).

[Part 3. Generating right labels from left labels.] Parts 1 and 2 showed how to generate \(n\) left labels for \(n \geq 3\). In this section, we show how to generate \(n\) right (or anticlockwise) labels for \(n \geq 3\). Every left label has a corresponding right label. Given a left label we can derive a right label as follows. Take each digit \(d\) of the left label and replace it by digit \((2 - d)\). In other words, replace 0 with 2, replace 2 with 0, and keep 1 as it is. For example, the right labels corresponding to left labels 000, 111, 2222,
012210112201 are 222, 111, 00000, 210012110021, respectively.

**FINDBAD-DYSON(S)**

**Input:** Set of $S$ coins, where $S = \{1, 2, \ldots, n\}$ having a bad coin with $n \geq 3$.

**Output:** The bad coin.

1. $n \leftarrow |S|; \ w \leftarrow \lceil \log_3 (2n) + 3 \rceil$
2. // Step 1. Generate left and right labels for the $n$ coins ............
3. $(\ell \text{label}[1 \ldots n], r\text{label}[1 \ldots n]) \leftarrow \text{GENERATETERNARYLABELS}(w, n)$
4. // Step 2. Find the bad coin using left and right labels ............
5. \text{return} IDENTIFYBADCOINUSINGLABELS($w, n, \ell \text{label}[1 \ldots n], r\text{label}[1 \ldots n]$)

**IDENTIFYBADCOINUSINGLABELS($w, n, \ell \text{label}[1 \ldots n], r\text{label}[1 \ldots n]$)**

1. Create an array $answer[1 \ldots w]$
2. \text{for} $j \leftarrow 1$ \text{ to } $w$ \text{ do}
3. Create dynamic arrays $S_1, S_2, S_3$ to mean left, right, and remaining sets
4. Left $S_1 \leftarrow [ ];$ Right $S_2 \leftarrow [ ];$ Remaining $S_3 \leftarrow [ ]$
5. \text{for} $i \leftarrow 1$ \text{ to } $n$ \text{ do}
6. \text{if} $\ell \text{label}[i, j] = 0$ \text{ then} $S_1.\text{Add}(i)$
7. \text{else if} $\ell \text{label}[i, j] = 1$ \text{ then} $S_3.\text{Add}(i)$
8. \text{else if} $\ell \text{label}[i, j] = 2$ \text{ then} $S_2.\text{Add}(i)$
9. // In the first weighing, $|S_1| = |S_2|$ because of the coin labels and after the first weighing we can identify a good coin ............
10. \text{if} $|S_1| > |S_2|$ \text{then} $S_2.\text{Add}(\text{goodcoin})$
11. \text{else if} $|S_1| < |S_2|$ \text{then} $S_1.\text{Add}(\text{goodcoin})$
12. Weigh $S_1$ and $S_2$ sets
13. \text{if} weight($S_1$) < weight($S_2$) \text{then}
14. \{} \text{answer}[j] \leftarrow 0; \text{if} \ j = 1 \text{ then} \text{goodcoin} \leftarrow S_3.\text{First}() \}\n15. \text{else if} weight($S_1$) = weight($S_2$) \text{then}
16. \{} \text{answer}[j] \leftarrow 1; \text{if} \ j = 1 \text{ then} \text{goodcoin} \leftarrow S_1.\text{First}() \}\n17. \text{else if} weight($S_1$) > weight($S_2$) \text{then}
18. \{} \text{answer}[j] \leftarrow 2; \text{if} \ j = 1 \text{ then} \text{goodcoin} \leftarrow S_3.\text{First}() \}\n19. \text{for} \ i \leftarrow 1 \text{ to } n \text{ do}
20. \text{if} \text{answer} = \ell \text{label}[i] \text{ then return} \text{lighter coin} \ i$
21. \text{if} \text{answer} = r\text{label}[i] \text{ then return} \text{heavier coin} \ i

[Step 2. Identify bad coin using ternary labels.] In this step, we identify the bad coin among $n$ coins using the ternary labels created in Step 1 and in $w$ weighings.

We know that the $i$th coin has left label $\ell \text{label}[i]$ for $i \in [1, n]$. In the $j$th weighing, where $j \in [1, w]$, we split the $n$ coins into three sets: left set $S_1$, right set $S_2$, and the remaining set $S_3$ and place the coins in the three sets as follows:

Left set $S_1 = \{ \text{coin } i \text{ if the } j\text{th digit in the left label of coin } i \text{ is } 0 \}$
Right set $S_2 = \{ \text{coin } i \text{ if the } j\text{th digit in the left label of coin } i \text{ is } 2 \}$
**Problem Statement:**

**Input:** $w$ weighings and $n$ coins.

**Output:** Left and right labels of size $w$ for each of the $n$ coins.

1. **if** $w = 2$ **then**
   
   ```
   \begin{align*}
   &\ell\text{label}[1, 2, 3] \leftarrow ([0, 1], [1, 2], [2, 0]) \quad \text{and} \quad r\text{label}[1, 2, 3] \leftarrow ([2, 1], [1, 0], [0, 2])
   \end{align*}
   ```

2. **else**

   ```
   \begin{align*}
   &\text{// Generate } n_{\text{max}}(w - 1) \text{ left labels each of size } (w - 1) \text{ ..............} \\
   &\ell\text{label}_\text{prev}, r\text{label}_\text{prev} \leftarrow \text{GENERATETERNARYLABELS}((w - 1), n_{\text{max}}(w - 1)) \\
   &\text{// Get the top third left labels of } \ell\text{label}_\text{prev} \text{ ..............} \\
   &m \leftarrow (1/3) \times n_{\text{max}}(w - 1) \quad \text{// } m \text{ is 1/3rd of size of } \ell\text{label}_\text{prev} \\
   &\text{topthird}[1...m] \leftarrow \text{Get the alphabetically first } m \text{ labels from } \ell\text{label}_\text{prev} \\
   &\text{// Create label array by appending digits 0, 1, 2 to the labels of topthird. Array label stores the initial } m \text{ left labels } \\
   &\text{Create arrays } \ell\text{label}[1...m], \text{temp}[1...w], \text{and the desired } \ell\text{label}[1...n] \\
   &\text{for } i \leftarrow 1 \text{ to } m \text{ do} \\
   &\quad \text{for } j \leftarrow 0 \text{ to } 2 \text{ do} \\
   &\quad\quad \{ \text{temp} \leftarrow \text{topthird}[i]; \text{temp.Add}(j); \text{label.Add}(\text{temp}) \} \\
   &\text{if } n = n_{\text{max}}(w) \text{ then } p \leftarrow \lfloor n/3 \rfloor - 1 \\
   &\text{else if } n \neq n_{\text{max}}(w) \text{ then } p \leftarrow \lfloor n/3 \rfloor \\
   &\text{// Derive triples of left labels by performing two cyclic permutations on labels from the label array } \\
   &\text{for } i \leftarrow 1 \text{ to } p \text{ do} \\
   &\quad \text{temp} \leftarrow \ell\text{label}[i]; \text{ell.Add}(\text{temp}) \\
   &\quad \text{for } j \leftarrow 1 \text{ to } w \text{ do} \text{temp}[j] \leftarrow (\text{temp}[j] + 1) \text{ mod } 3 \\
   &\quad \text{ell.Add}(\text{temp}) \\
   &\quad \text{for } j \leftarrow 1 \text{ to } w \text{ do} \text{temp}[j] \leftarrow (\text{temp}[j] + 1) \text{ mod } 3 \\
   &\quad \text{ell.Add}(\text{temp}) \\
   &\text{triad}[1, 2, 3] \leftarrow [0...01, 1...12, 2...20] \\
   &\text{// Add the 3 left labels of triad when } n = n_{\text{max}}(w) \text{ ..............} \\
   &\text{if } n = n_{\text{max}}(w) \text{ then} \\
   &\quad \{ \text{ell.Add(triad[1]); ell.Add(triad[2]); ell.Add(triad[3]) } \} \\
   &\text{// Add at most 2 left labels of triad when } n < n_{\text{max}}(w) \text{ ..............} \\
   &\text{else if } n \text{ mod } 3 = 1 \text{ then} \{ \text{ell.Add(triad[2]) } \} \\
   &\text{else if } n \text{ mod } 3 = 2 \text{ then} \{ \text{ell.Add(triad[1]); ell.Add(triad[3]) } \} \\
   &\text{// Derive } n \text{ right labels from } n \text{ left labels } \\
   &\text{Create an array } r\text{label}[1...n] \\
   &\text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
   &\quad \text{for } j \leftarrow 1 \text{ to } w \text{ do} \text{temp}[j] \leftarrow (2 - \ell\text{label}[i, j]) \\
   &\quad r\text{label}[i] \leftarrow \text{temp} \\
   &\text{return } (\ell\text{label}[1...n], r\text{label}[1...n])
   \end{align*}
   ```

**Algorithm:**

1. **if** $w = 2$ **then**
   
   ```
   \begin{align*}
   &\ell\text{label}[1, 2, 3] \leftarrow ([0, 1], [1, 2], [2, 0]) \quad \text{and} \quad r\text{label}[1, 2, 3] \leftarrow ([2, 1], [1, 0], [0, 2])
   \end{align*}
   ```

2. **else**

   ```
   \begin{align*}
   &\text{// Generate } n_{\text{max}}(w - 1) \text{ left labels each of size } (w - 1) \text{ ..............} \\
   &\ell\text{label}_\text{prev}, r\text{label}_\text{prev} \leftarrow \text{GENERATETERNARYLABELS}((w - 1), n_{\text{max}}(w - 1)) \\
   &\text{// Get the top third left labels of } \ell\text{label}_\text{prev} \text{ ..............} \\
   &m \leftarrow (1/3) \times n_{\text{max}}(w - 1) \quad \text{// } m \text{ is 1/3rd of size of } \ell\text{label}_\text{prev} \\
   &\text{topthird}[1...m] \leftarrow \text{Get the alphabetically first } m \text{ labels from } \ell\text{label}_\text{prev} \\
   &\text{// Create label array by appending digits 0, 1, 2 to the labels of topthird. Array label stores the initial } m \text{ left labels } \\
   &\text{Create arrays } \ell\text{label}[1...m], \text{temp}[1...w], \text{and the desired } \ell\text{label}[1...n] \\
   &\text{for } i \leftarrow 1 \text{ to } m \text{ do} \\
   &\quad \text{for } j \leftarrow 0 \text{ to } 2 \text{ do} \\
   &\quad\quad \{ \text{temp} \leftarrow \text{topthird}[i]; \text{temp.Add}(j); \text{label.Add}(\text{temp}) \} \\
   &\text{if } n = n_{\text{max}}(w) \text{ then } p \leftarrow \lfloor n/3 \rfloor - 1 \\
   &\text{else if } n \neq n_{\text{max}}(w) \text{ then } p \leftarrow \lfloor n/3 \rfloor \\
   &\text{// Derive triples of left labels by performing two cyclic permutations on labels from the label array } \\
   &\text{for } i \leftarrow 1 \text{ to } p \text{ do} \\
   &\quad \text{temp} \leftarrow \ell\text{label}[i]; \text{ell.Add}(\text{temp}) \\
   &\quad \text{for } j \leftarrow 1 \text{ to } w \text{ do} \text{temp}[j] \leftarrow (\text{temp}[j] + 1) \text{ mod } 3 \\
   &\quad \text{ell.Add}(\text{temp}) \\
   &\quad \text{for } j \leftarrow 1 \text{ to } w \text{ do} \text{temp}[j] \leftarrow (\text{temp}[j] + 1) \text{ mod } 3 \\
   &\quad \text{ell.Add}(\text{temp}) \\
   &\text{triad}[1, 2, 3] \leftarrow [0...01, 1...12, 2...20] \\
   &\text{// Add the 3 left labels of triad when } n = n_{\text{max}}(w) \text{ ..............} \\
   &\text{if } n = n_{\text{max}}(w) \text{ then} \\
   &\quad \{ \text{ell.Add(triad[1]); ell.Add(triad[2]); ell.Add(triad[3]) } \} \\
   &\text{// Add at most 2 left labels of triad when } n < n_{\text{max}}(w) \text{ ..............} \\
   &\text{else if } n \text{ mod } 3 = 1 \text{ then} \{ \text{ell.Add(triad[2]) } \} \\
   &\text{else if } n \text{ mod } 3 = 2 \text{ then} \{ \text{ell.Add(triad[1]); ell.Add(triad[3]) } \} \\
   &\text{// Derive } n \text{ right labels from } n \text{ left labels } \\
   &\text{Create an array } r\text{label}[1...n] \\
   &\text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
   &\quad \text{for } j \leftarrow 1 \text{ to } w \text{ do} \text{temp}[j] \leftarrow (2 - \ell\text{label}[i, j]) \\
   &\quad r\text{label}[i] \leftarrow \text{temp} \\
   &\text{return } (\ell\text{label}[1...n], r\text{label}[1...n])
   \end{align*}
   ```

3. **return** $(\ell\text{label}[1...n], r\text{label}[1...n])$
The image contains a table listing coin numbers and corresponding labels, as well as diagrams illustrating the identification of a bad coin using Dyson's algorithm for different numbers of coins (n = 12, 11, and 10). The table and diagrams are as follows:

<table>
<thead>
<tr>
<th>Coin number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Left label</strong></td>
<td>010</td>
<td>121</td>
<td>202</td>
<td>011</td>
<td>122</td>
<td>200</td>
<td>012</td>
<td>120</td>
<td>201</td>
<td>001</td>
<td>112</td>
<td>220</td>
</tr>
<tr>
<td><strong>Right label</strong></td>
<td>212</td>
<td>101</td>
<td>020</td>
<td>211</td>
<td>100</td>
<td>212</td>
<td>102</td>
<td>021</td>
<td>210</td>
<td>102</td>
<td>221</td>
<td>002</td>
</tr>
</tbody>
</table>

For n = 11:

- Left label: 010 121 202 011 122 200 012 120 201 001 112 220
- Right label: 212 101 020 211 100 022 210 102 021 221 002

For n = 10:

- Left label: 010 121 202 011 122 200 012 120 201 112
- Right label: 212 101 020 211 100 022 210 102 021 110

The diagrams on the right correspond to the first, second, and third weighings for each case, with symbols indicating the weighing order and decisions made at each step. The figure illustrates how Dyson's algorithm works to identify a bad coin among a given number of coins.

Figure 123: Identifying a bad coin using Dyson's algorithm when n is 12, 11, and 10. Expression $S_1 : S_2$ means weigh sets $S_1$ and $S_2$; symbols $g$ = good coin and $\times$ denotes impossible event. [Top:] Generate left and right labels. [Middle:] Order of weighings that denote which coins should be weighed in what order. [Bottom:] Decision trees for 12, 11, and 10 coins.
Remaining set \( S_3 = \{ \text{coin } i \text{ if the } j\text{th digit in the left label of coin } i \text{ is 1} \} \)

Before weighing left \( S_1 \) and right \( S_2 \) sets, we need to make sure that they have the same number of coins. There are three possibilities: \( |S_1| = |S_2|, |S_1| > |S_2|, \) and \( |S_1| < |S_2| \). If \( |S_1| = |S_2| \), there is nothing to be done. If \( |S_1| \neq |S_2| \), then it can be shown that the difference in the number of coins in the two sets is exactly 1. In that case, we add a good coin to the set that has one less coin to ultimately get \( |S_1| = |S_2| \). But how can we find a good coin? We will address this problem later.

In the \( j \)th weighing, where \( j \in [1, w] \), weigh the two sets \( S_1 \) and \( S_2 \) and note down the result of the weighing in \( answer[j] \) as below:

\[
answer[j] = \begin{cases} 
0 & \text{if weight}(S_1) < \text{weight}(S_2), \\
1 & \text{if weight}(S_1) = \text{weight}(S_2), \\
2 & \text{if weight}(S_1) > \text{weight}(S_2).
\end{cases}
\]

Note: In the first weighing, i.e., \( j = 1 \), \( |S_1| \) will always be equal to \( |S_2| \). In that case, if \( \text{weight}(S_1) = \text{weight}(S_2) \), then all coins in \( S_1 \) and \( S_2 \) are good coins. On the other hand, if \( \text{weight}(S_1) \neq \text{weight}(S_2) \), then all coins in \( S_3 \) are good coins. We can store this information to be later used whenever we need a good coin.

Finally, we would have the \( answer \) ternary label consisting of the result of \( w \) weighings. If \( answer \) matches the left label of coin \( i \), then we know that coin \( i \) is bad and it is lighter than other coins. On the other hand, if \( answer \) matches the right label of coin \( i \), then we know that coin \( i \) is bad and it is heavier than other coins.

[Examples.] We show how to find the bad coin in three cases: \( n = 12, n = 11, \) and \( n = 10 \). We generate the left and right labels for all \( n \) coins as shown in the top part of Figure 123. We then perform three different weighings using a defined set of coins (which is dependent on \( n \)) as shown in the middle part of Figure 123. When the number of coins in the two weighing sets are different (which is always the case when \( n \mod 3 \neq 0 \)), the set with one less coin will get an additional good coin that is found from the result of the first weighing. For instance, the third weighing when \( n = 10 \) requires a good coin, as shown in the middle part of Figure 123. All possibilities of these weighings are considered in the decision trees with branch factor 3 (is independent of \( n \)) and height 3 (is dependent on \( n \)) as shown in the bottom part of Figure 123. As can be seen from the diagram, a decision tree will always be symmetric in the sense that if lighter coin \( i \) representing a left label is at a distance of \( d \) from one end of the tree, then its counterpart – the heavier coin \( i \) will have a right label that will be at a distance of \( d \) from the other end of the decision tree.

Consider \( n = 10 \), i.e., the rightmost decision tree. Every internal node has three edges: 0 (top branch), 1 (middle branch), and 2 (bottom branch). The path 021 from the root leads to 09H. This means that the bad coin is 09 and it is heavy (H means heavy and L means light). This is true because we can see that 021 is the right label (meaning coin is heavy) of coin 9 when \( n = 10 \). The path 220 from the root leads to \( X \). This means that it is impossible to have this sequence of results of the three weighings when \( n = 10 \) and for the defined set of labelings.

270
\[ W(n) = \lfloor \log_3(2n + 3) \rfloor \text{ for } n \geq 3. \text{ For example, } W(13) = 4. \]

[**Wert’s cyclic rotation algorithm (optimal).**] Suppose \( S \) is a set of \( n \) coins stored as an array. We explain the algorithm in multiple steps.

[Step 1. Create groups and subgroups and compute weighing\(_{\text{initial}}\).] Represent \( n \) as one of the three cases: \( 3k + 3 \), \( 3k + 1 \), or \( 3k + 2 \). Divide the array \( S \) into three groups: \( S_1 \) contains \( k \) coins, \( S_2 \) contains the next \( k \) coins, and \( S_3 \) contains the next \( k \) coins. Divide each of the three groups \( S_i \), where \( i \in \{1, 2, 3\} \), as follows:

Divide each group \( S_i \), for \( i \in \{1, 2, 3\} \), of \( k \) coins into subgroups \( S_i^{(1)}, S_i^{(2)}, \ldots, S_i^{(\text{last})} \) where, \( S_i^{(1)} \) has \( 3^1 = 3 \) coins, \( S_i^{(2)} \) has next \( 3^2 = 9 \) coins, so on till the last subgroup \( S_i^{(\text{last})} \) has \( k \) minus previously assigned number of coins. So, the number of coins in the last subgroup \( S_i^{(\text{last})} \) need not be a power of 3.

We now add the remaining extra coins by creating one more subgroup in each group. We create and add subgroup \( S_i^{(0)} \) to group \( S_i \) for \( i \in \{1, 2, 3\} \) as follows:

\[
(S_i^{(0)}, S_2^{(0)}, S_3^{(0)}) = \begin{cases} 
(1 \text{ coin, 1 coin, 1 coin}) & \text{if } n = 3k + 3, \\
(0 \text{ coin, 0 coin, 1 coin}) & \text{if } n = 3k + 1, \\
(1 \text{ coin, 1 coin, 0 coin}) & \text{if } n = 3k + 2. 
\end{cases}
\]

Creation of groups and subgroups for all three cases is summarized in Table 53.

<table>
<thead>
<tr>
<th>( S_i )</th>
<th>( S_i^{(0)} )</th>
<th>( S_i^{(1)} )</th>
<th>( S_i^{(2)} )</th>
<th>( S_i^{(3)} )</th>
<th>( \cdots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_1 )</td>
<td>( 3^0 )</td>
<td>( 3^1 )</td>
<td>( 3^2 )</td>
<td>( 3^3 )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( S_2 )</td>
<td>( 3^0 )</td>
<td>( 3^1 )</td>
<td>( 3^2 )</td>
<td>( 3^3 )</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( S_3 )</td>
<td>( 3^0 )</td>
<td>( 3^1 )</td>
<td>( 3^2 )</td>
<td>( 3^3 )</td>
<td>( \cdots )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( S_i )</th>
<th>( S_i^{(0)} )</th>
<th>( S_i^{(1)} )</th>
<th>( S_i^{(2)} )</th>
<th>( S_i^{(3)} )</th>
<th>( \cdots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_1 )</td>
<td>( -3^1 )</td>
<td>( 3^2 )</td>
<td>( 3^3 )</td>
<td>( \cdots )</td>
<td></td>
</tr>
<tr>
<td>( S_2 )</td>
<td>( -3^1 )</td>
<td>( -3^2 )</td>
<td>( 3^3 )</td>
<td>( \cdots )</td>
<td></td>
</tr>
<tr>
<td>( S_3 )</td>
<td>( 3^0 )</td>
<td>( -3^1 )</td>
<td>( 3^2 )</td>
<td>( 3^3 )</td>
<td>( \cdots )</td>
</tr>
</tbody>
</table>

Case 1: \( n = 3k + 3 \)  
Case 2: \( n = 3k + 1 \)  
Case 3: \( n = 3k + 2 \)

Table 53: Groups and subgroups for three different cases.

Weigh sets \( S_1 \) and \( S_2 \). Store this weighing result in \( \text{weighing}_{\text{initial}} \):

\[
\text{Weighing result} = \begin{cases} 
-1 & \text{if } \text{weight}(S_1) < \text{weight}(S_2), \\
0 & \text{if } \text{weight}(S_1) = \text{weight}(S_2), \\
+1 & \text{if } \text{weight}(S_1) > \text{weight}(S_2). 
\end{cases}
\]

<table>
<thead>
<tr>
<th>( \text{weighing}_{\text{curr}} )</th>
<th>( \text{weighing}_{\text{initial}} )</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
<td>( S_2 ) has the heavier coin</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
<td>( S_1 ) has the lighter coin</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>( S_3 ) has the lighter coin</td>
</tr>
<tr>
<td>0</td>
<td>+1</td>
<td>( S_3 ) has the heavier coin</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>( S_1 ) has the heavier coin</td>
</tr>
<tr>
<td>+1</td>
<td>0</td>
<td>( S_2 ) has the lighter coin</td>
</tr>
</tbody>
</table>

Table 54: Six possibilities when \( \text{weighing}_{\text{curr}} \neq \text{weighing}_{\text{initial}} \).
**FINDBAD-WERT(S)**

**Input:** Set of coins stored in an array $S$.

**Output:** The bad coin.

1. // Step 1. Create groups/subgroups and compute $\text{weighing}_{\text{initial}}$ ............
2. $n \leftarrow |S|$; Represent $n$ as $3k + 3$ or $3k + 1$ or $3k + 2$
3. $S_1 \leftarrow$ first $k$ coins of $S$; $S_2 \leftarrow$ second $k$ coins of $S$; $S_3 \leftarrow$ third $k$ coins of $S$
4. for $i \leftarrow 1$ to $3$ do
5.    Divide $S_i$ into $S_i^{(1)}, S_i^{(2)}, \ldots, S_i^{(\text{last})}$ subgroups, where, $S_i^{(j)}$ contains $3^j$ coins
6.    Create subgroup $S_i^{(0)}$ as shown in Equation 53 and add it to group $S_i$
7. $\text{weighing}_{\text{initial}} \leftarrow \text{WEIGH}(S_1, S_2)$
8. // Step 2. Rotate last subgroups and compute $\text{weighing}_{\text{curr}}$ ............
9. while $|S_1| + |S_2| + |S_3| > 3$ do
10.    $(S_1^{(\text{last})}, S_2^{(\text{last})}, S_3^{(\text{last})}) \leftarrow (S_2^{(\text{last})}, S_3^{(\text{last})}, S_1^{(\text{last})})$
11.    $\text{weighing}_{\text{curr}} \leftarrow \text{WEIGH}(S_1, S_2)$
12. // If the bad coin is in one of the last subgroups ............
13. if $\text{weighing}_{\text{curr}} \neq \text{weighing}_{\text{initial}}$ then
14.    return FINDBAD-TERNARY($\text{weighing}_{\text{curr}}, \text{weighing}_{\text{initial}}$)
15. // If the bad coin is not in the last subgroups ............
16. else if $\text{weighing}_{\text{curr}} = \text{weighing}_{\text{initial}}$ then
17.    $\text{goodcoin} \leftarrow S_1^{(\text{last})}$ ; First()
18.    $S_1$.Remove($S_1^{(\text{last})}$); $S_2$.Remove($S_2^{(\text{last})}$); $S_3$.Remove($S_3^{(\text{last})}$)
19. // Step 3. Handle base cases ............
20. if $|S_1| + |S_2| + |S_3| = 3$ then
21.    $(S_1^{(\text{last})}, S_2^{(\text{last})}, S_3^{(\text{last})}) \leftarrow (S_2^{(\text{last})}, S_3^{(\text{last})}, S_1^{(\text{last})})$
22.    $\text{weighing}_{\text{curr}} \leftarrow \text{WEIGH}(S_1, S_2)$
23. return FINDBAD-TERNARY($\text{weighing}_{\text{curr}}, \text{weighing}_{\text{initial}}$)
24. if $|S_1| + |S_2| + |S_3| = 1$ then
25.    $\text{weighing}_{\text{curr}} \leftarrow \text{WEIGH}(S_3, \text{goodcoin})$
26. if $\text{weighing}_{\text{curr}} = -1$ then return lighter $S_3$
27. if $\text{weighing}_{\text{curr}} = +1$ then return heavier $S_3$
28. if $|S_1| + |S_2| + |S_3| = 2$ then
29.    $\text{weighing}_{\text{curr}} \leftarrow \text{WEIGH}(S_1, \text{goodcoin})$
30. if $\text{weighing}_{\text{initial}} = -1$ then
31.    if $\text{weighing}_{\text{curr}} = -1$ then return lighter $S_1$
32.    if $\text{weighing}_{\text{curr}} = +1$ then return heavier $S_2$
33. if $\text{weighing}_{\text{initial}} = +1$ then
34.    if $\text{weighing}_{\text{curr}} = -1$ then return lighter $S_2$
35.    if $\text{weighing}_{\text{curr}} = +1$ then return heavier $S_1$

Figure 124: Wert's algorithm to find the bad coin.
**WEIGH(S₁, S₂)**

1. if weight(S₁) < weight(S₂) then return -1
2. if weight(S₁) = weight(S₂) then return 0
3. if weight(S₁) > weight(S₂) then return +1

**FINDBAD-TERNARY(weighingcurr, weighinginitial)**

1. if weighingcurr = -1 then
   2. if weighinginitial = 0 then return heavier FINDHEAVIER-TERNARY(S₂(\text{last}))
   3. if weighinginitial = +1 then return lighter FINDLIGHTER-TERNARY(S₁(\text{last}))
   4. if weighingcurr = 0 then
      5. if weighinginitial = -1 then return lighter FINDLIGHTER-TERNARY(S₃(\text{last}))
      6. if weighinginitial = +1 then return heavier FINDHEAVIER-TERNARY(S₃(\text{last}))
   7. if weighingcurr = +1 then
      8. if weighinginitial = -1 then return heavier FINDHEAVIER-TERNARY(S₁(\text{last}))
      9. if weighinginitial = 0 then return lighter FINDLIGHTER-TERNARY(S₂(\text{last}))

Figure 125: Helper functions of Wert’s algorithm. The six possibilities when weighingcurr \neq weighinginitial are summarized in Table 54.

[Step 2. Cyclically rotate last subgroups and compute weighingcurr.] We first rotate the last subgroups by keeping the other subgroups as they are. That is, we transfer the last subgroup from S₂ to S₁, from S₃ to S₂, and from S₁ to S₃. The cyclic rotation of the last group is defined as follows:

(S₁(\text{last}), S₂(\text{last}), S₃(\text{last})) ← (S₂(\text{last}), S₃(\text{last}), S₁(\text{last})) i.e.,

S₂(\text{last}) becomes S₁(\text{last}), S₃(\text{last}) becomes S₂(\text{last}), S₁(\text{last}) becomes S₃(\text{last}) in parallel

Weigh sets S₁ and S₂ again. Store the result of this weighing in weighingcurr using Equation 54. Two subcases arise:

(i) [weighingcurr \neq weighinginitial.] This case is like the base case and if this case is encountered, the algorithm terminates immediately after this case executes. This case implies that the bad coin is in the last subgroups and hence must be in S₁(\text{last}) or S₂(\text{last}) or S₃(\text{last}). The six possibilities when weighingcurr \neq weighinginitial are covered in Table 54 and FINDBAD-TERNARY algorithm in Figure 125. Depending on specific possibility, we can find the bad coin among m coins and determine whether it is lighter or heavier using the FINDLIGHTER-TERNARY or FINDHEAVIER-TERNARY algorithm using ternary search, as discussed before, in \lfloor \log₃ m \rfloor weighings.

(ii) [weighingcurr = weighinginitial.] This implies that the bad coin is not in the last subgroups and hence must be one of the non-last subgroups. All the coins in the last subgroups are good coins and can be labeled as such for future use. We discard or remove the last subgroup from each of the three groups S₁, S₂, and S₃ for further processing. We repeat Step 2 until the total number of coins in S₁, S₂, S₃ is not greater than 3, i.e., until |S₁| + |S₂| + |S₃| ≤ 3.
Step 3. Handle base cases. At this point we know that $|S_1| + |S_2| + |S_3| \in \{1, 2, 3\}$. If $|S_1| + |S_2| + |S_3| = 3$, we have one coin in each of the three groups. In this case we rotate the three groups and weigh $S_1$ and $S_2$ and store this result in $\text{weighing}_{\text{curr}}$. Depending on the relation between $\text{weighing}_{\text{curr}}$ and $\text{weighing}_{\text{initial}}$ we can identify the bad coin and determine if it is lighter or heavier among the three coins. If $|S_1| + |S_2| + |S_3| = 1$, we have one coin in $S_3$ and none in the other two groups. We weigh this single coin in $S_3$ and a good coin already found in Step 2. We can then easily identify whether the coin in $S_3$ is lighter or heavier. If $|S_1| + |S_2| + |S_3| = 2$, we have one coin in each of $S_1$ and $S_2$ and none in $S_3$. We weigh $S_1$ and a good coin and store the result in $\text{weighing}_{\text{curr}}$. Depending on the relation between $\text{weighing}_{\text{curr}}$ and $\text{weighing}_{\text{initial}}$ we can identify the bad coin and determine if it is lighter or heavier among the two coins.

$$W(n) = \lceil \log_3(2n + 3) \rceil \text{ for } n \geq 3.$$

Problems

1. [Weighing balance : multiple weights.] You are given $n$ coins and a weighing balance. How do you find the weight of each coin minimizing the number of weighings assuming:
   (a) the unique weights of the coins $\{w_1, w_2, w_3, \ldots, w_{k\leq n}\}$ are known.
   (b) no other information related to coin weights.

2. [Digital weighing machine : multiple weights.] You are given $n$ coins and a digital weighing machine. How do you find the weight of each coin minimizing the number of weighings assuming:
   (a) the unique weights of the coins $\{w_1, w_2, w_3, \ldots, w_{k\leq n}\}$ are known.
   (b) no other information related to coin weights.

References

Many mathematicians have independently discovered the optimal algorithms for identifying the bad coin. The ternary search algorithm to find the bad coin is given in [Fine, 1947b] and beautifully presented in [Manvel, 1977]. The ternary system based algorithm to find the bad coin is in [Dyson, 1946] and described in [Shestopal, 1979], [Wetherfield, 2005], and [Routledge, 2006]. The algorithm based on cyclic rotations of last subgroups to find the bad coin is given in [Wert, 2022]. Note that Wert described his algorithm for a special case only; his algorithm has been generalized in this book. The non-adaptive ternary system algorithm to find the bad coin is in [Born et al., 2003]. When $k$ out of $n$ coins are lighter, then an algorithm to find the lighter coins in at most $\lceil \log_3 \binom{n}{c} \rceil + 1.5c + 9$ weighings, where $k < c$ for some integer $c$, is given in [Aigner and Li, 1997]. When $k$ out of $n$ coins are lighter, then an algorithm to find the lighter coins in at most $c \log_3 \binom{n}{k}$ weighings, where $c \leq \log_2 3 + \frac{1}{2}$, is given in [Wang and Du, 1997]. An encyclopedic survey of several coin weighing results is given in the best-written book on group testing [Du and Hwang, 2000].
Picture Hanging

Problem

We want to hang a picture with a string (or thread) connecting two points on the picture’s frame. The most common way to hang a picture using two nails is shown in Figure 126. In the figure, the string passes over the two nails. The string is assumed to be elastic and infinitely stretchable. If any of the two nails is removed, then the picture falls back on the other nail.

Is there a way to hang the picture such that if either nail comes out, the picture falls to the ground?

![Simple picture hanging with two nails.](image)

Figure 126: Simple picture hanging with two nails.

Solution

The puzzle belongs to the domains of geometry and algebra. Most people immediately start out to solve the puzzle using trial-and-error method. Though this approach might lead to the solution for two nails, it gets horrendously complicated to solve problem instances with more number of nails. We will see elegant algebraic solutions for this geometric puzzle.

Algebraic solution

[1 nail.] We introduce a simple idea that forms the basic building block on which the entire solution is built upon. Let a loop of the string around a nail as shown in Figure 127 be denoted by $a$. The left end (resp. right end) of the string is connected to the left point (resp. right point) on the picture frame. It is easy to see that with a loop like this the picture relies on the nail and does not fall.

Let $P_n$ be a sequence of letters (algebraic representation) to denote a geometric solution to the puzzle for $n$ nails. Then, for 1 nail, we have
[2 nails.] We build upon the basic building block to construct more complicated geometric figures. A loop over a nail in the reverse (i.e., anticlockwise) direction is denoted by $a^{-1}$. If we have a loop and a reverse loop on a nail then we denote it by $aa^{-1}$ (see Figure 127). When a string has a geometric shape of $aa^{-1}$, the picture will not stand on the nail and it falls to the ground. A loop and a loop inverse for a second nail is denoted by $b$ and $b^{-1}$, respectively. A solution for 2 nails is shown in Figure 128 and algebraically represented as

$$P_2 = aba^{-1}b^{-1}.$$

It is easy to see that if we remove the first nail, then $a$ and $a^{-1}$ disappears from $P_2$ leaving $P_2 = bb^{-1} = \phi$, which implies the picture falls. By a similar argument, when we remove the second nail, the picture falls.

[3 nails.] When we have a 3rd nail, the situation gets slightly complicated. The terms $c$ and $c^{-1}$ are used to denote a loop and a loop inverse for the 3rd nail. We can make use of $P_2$ to write $P_3$. We can use the terms: $(aba^{-1}b^{-1}), c$, and $c^{-1}$ to write $P_3$. We need to make sure that when the 3rd nail is removed, the picture falls. A solution for 3 nails is
\[ P_3 = (aba^{-1}b^{-1})c(bab^{-1}a^{-1})c^{-1}. \]

where, \((bab^{-1}a^{-1})\) is the inverse of \((aba^{-1}b^{-1})\) i.e., \((aba^{-1}b^{-1})^{-1} = (bab^{-1}a^{-1})\).

\[ n \text{ nails.} \] Let \(x_n\) denote the loop on the \(n^{th}\) nail and \(k_n \geq 1\) be the number of loops on the \(n^{th}\) nail i.e., \((x_n)^{k_n}\). Then \((x_n)^{-1}\) denotes the inverse loop on the \(n^{th}\) nail and \((x_n)^{-k_n}\) denotes \(k_n \geq 1\) number of inverse loops on the \(n^{th}\) nail. The inverse function for a sequence \((s_1 s_2 \cdots s_m)\) is defined as follows: \((s_1 s_2 \cdots s_m)^{-1} = (s_m^{-1} s_{m-1}^{-1} \cdots s_1)\).

Then, \(P_n\) for \(n\) nails can be computed as

\[
P_n = \begin{cases} 
(x_1)^{k_1} & \text{if } n = 1, \\
(P_{n-1}(x_n)^{k_n}P_{n-1}(x_n)^{-k_n}) & \text{if } n > 1.
\end{cases}
\]

The formula for \(P_n\) implies that we could have also written \(P_2\) as \(a^7b^{100}a^{-7}b^{-100}\). We can have any number of loops (or loop inverses) on a nail, but, typically we like to reduce the number of loops.

Let \(\ell_n\) denote the minimum length of the pattern \(P_n\) i.e., the length of \(P_n\) when \(k_i = 1\) for all \(i\). Then, we can write a recurrence for \(\ell_n\) and solve it as

\[
\ell_n = \begin{cases} 
1 & \text{if } n = 1, \\
2\ell_{n-1} + 2 & \text{if } n > 1.
\end{cases}
\]

Solving the recurrence, we get

\[
\ell_n = 3 \cdot 2^{n-1} - 2, \quad \text{for } n \geq 1
\]

Is it possible to find a better solution minimizing the number of turns of the string? That is, come up with a solution that minimizes the value of \(\ell_n\). Yes! In further section, we present a better solution.

**Algebraic solution using divide-and-conquer**

The difference between this solution and the previous solution becomes clear when we have at least 4 nails.

\[ 4 \text{ nails.} \] As per the previous solution, when we have 4 nails, the pattern \(P_4\) is computed as

\[ P_4 = (P_3)d(P_3)^{-1}d^{-1} \]

\[ = (aba^{-1}b^{-1}c bab^{-1}a^{-1}c^{-1})d(caba^{-1}b^{-1}c bab^{-1}a^{-1})d^{-1} \]

This pattern which does not use divide-and-conquer requires 22 loops and loop inverses. Instead, we could have grouped \(a\) and \(b\) together and grouped \(c\) and \(d\) together and then applied the pattern \(P_2\) for these two groups using divide-and-conquer. That is,

\[ P_4 = P_2(P_2(a,b), P_2(c,d)) \]

\[ = P_2(aba^{-1}b^{-1}, cdc^{-1}d^{-1}) \]

277
\[(aba^{-1}b^{-1})(cdc^{-1}d^{-1})(bab^{-1}a^{-1})(dcd^{-1}c^{-1})\]

This pattern that uses divide-and-conquer requires only 16 loops and loop inverses.

[n nails.] Generalizing, we have

\[
P_n(x_1, \ldots, x_n) = \begin{cases} \chi_{x_1}^{k_1} & \text{if } n = 1, \\ \chi_{x_1}^{k_1} x_2^{k_2} & \text{if } n = 2, \\ P_2(P_{\lfloor n/2 \rfloor}(x_1, \ldots, x_{\lfloor n/2 \rfloor}), P_{\lceil n/2 \rceil}(x_{\lceil n/2 \rceil}+1, \ldots, x_n)) & \text{if } n > 2. \end{cases}
\]

The minimum length \(\ell_n\) of \(P_n\) is computed as

\[
\ell_n = \begin{cases} 1 & \text{if } n = 1, \\ 2(\ell_{\lceil n/2 \rceil} + \ell_{\lfloor n/2 \rfloor}) & \text{if } n > 1. \end{cases}
\]

\(\ell_n \approx n^2 \quad \text{for } n \geq 1\)

Take-home lessons

[Systematic approach.] When we encounter this puzzle, it is tempting to solve the problem by trying tens of diagrams using the trial-and-error method. However, by resisting the temptation to solve every geometric problem using geometry and carefully mapping this geometric problem to algebraic problem, we were able to solve the problem without constructing a single complicated diagram. In fact, we can even solve a problem with a million nails requiring a trillion turns. Though it is hard for us to visualize the diagrams even a small number of nails, we have a solution that works beautifully.

Isaac Newton was able to decipher the pattern behind the confusing movements of the heavens using systematic and elegant mathematical techniques. Charles Darwin was able to decipher the pattern behind the creation of millions of species over millions of years using systematic and organized scientific experiments. Similarly, we were able to crack this seemingly haphazard puzzle using systematic and carefully planned mathematical technique. Systematic approaches of problem-solving yield good results.

Problems

1. [Subset of nails.] Develop a strategy such that the picture falls when any \(k\) out of \(n\) nails are removed.

References

The algebraic solution is given in [Pegg Jr., 2002]. The algebraic solution using divide-and-conquer is given by Chris Lusby Taylor [Pegg Jr., 2002]. The puzzle and the 2-nails solution are presented in [Winkler, 2007] and the \(n\)-nails solution is presented in [Demaine et al., 2014].
Bridge Crossing

Problem

Four friends must cross a dangerous bridge at night. They have a torch that will last for 12 minutes and it is impossible to cross the bridge without a torch. At most two people can cross the bridge at a time. The time taken to cross the bridge for the four friends are 1, 2, 4, and 5 minutes. When two people walk together the faster one waits for the slower one. How can all friends cross the bridge safely?

Solution

This is one of the famous algorithmic puzzles asked in technical interviews.

[Problem statement.] There are \( n \) friends \( \{1, 2, 3, \ldots, n\} \) who have to cross a bridge at night. We assume that the \( n \) friends are on the left side and they have to reach the right side. They have a torch and it is impossible to cross the bridge without a torch. The capacity of the bridge is \( c \), i.e., at most \( c \) number of people can cross the bridge at a time. The time taken to cross the bridge for the \( i \)th person is \( t_i \) minutes such that \( 0 < t_1 \leq t_2 \leq t_3 \leq \cdots \leq t_n \). When a set of people walk together the faster ones waits for the slowest one. How can all friends cross the bridge safely in the fastest time possible?

Greedy algorithm (non-optimal)

We can use the greedy approach to design an algorithm. Suppose the fastest person takes \( c - 1 \) other person to the opposite side and the fastest person comes back with the torch to the starting side alone. Then the total time required for everyone to cross the bridge is \( (f - 1)t_1 + \sum_{i=1}^{f-1} t_{(c-1)i+1} + t_n \), where \( f = \left\lceil \frac{n-1}{c-1} \right\rceil \). \textsc{BridgeCrossing-Greedy} gives the algorithm and it is not optimal.

| Crossing time is \( (f - 1)t_1 + \sum_{i=1}^{f-1} t_{(c-1)i+1} + t_n \ minutes, where \( f = \left\lceil \frac{n-1}{c-1} \right\rceil \). |

State diagram algorithm (optimal)

We solve the puzzle using state diagrams. For simplicity we show the state diagram for our original puzzle. However the approach can be extended to the generalized version of the puzzle.

In this puzzle, a state is denoted by a box with left and right chambers that represent the left and right sides of the bridge, respectively. The four friends, denoted by
their speeds 1, 2, 4 and 5 are placed in one of the two chambers. The torch is assumed to be present on the side marked with an underline. A state transition can happen in any of the 10 possible ways: 1, 2, 4, 5, 1 2, 1 4, 1 5, 2 4, 2 5, and 4 5. This is because at most two people can cross the bridge. We also add the time taken for each state transition as the edge cost in the state diagram. For example, the edge cost of the state transition 2 5 would be 5.

The state diagram is given in Figure 130 and its details are given in Figure 131. The starting and the final states are shown in the diagrams. Our objective is to generate all possible paths from the starting state to the final state such that the path cost (i.e., sum of the edge costs in the path) is minimized.

There are two optimal solutions to this puzzle (see Figure 130). Optimal solution takes 12 minutes.

Branch-and-bound algorithm (optimal)

We can solve the puzzle using the branch-and-bound algorithm design technique. BC-BRANCH&BOUND gives the branch-and-bound based algorithm. We define a couple of variables:

- leftset: set of persons on the left side of the bridge
- rightset: set of persons on the right side of the bridge
- trip: +1 if the next trip is forward and −1 if the next trip is backward
- currentpath: list/sequence of sets where each set is a trip; odd-numbered trips are forward trips and even-numbered trips are backward trips
- currentpathcost: crossing time for currentpath
- minpathcost: minimum crossing time of all feasible solutions (paths from root to leaves) that has been seen
- optimal solutions list: list of optimal solutions

Our algorithm like in the state diagram creates a state-space tree to explore states. We want to go from a state with leftset = \{1, 2, 3, . . . , n\} to a state with leftset = \{ \} in minimum crossing time. Initially, leftset = \{1, 2, 3, . . . , n\}, rightset = \{ \}, trip = +1
as the first trip is a forward trip, \( \text{currentpath} = [ ] \), \( \text{currentpathcost} = 0 \), \( \text{minpathcost} = \infty \), \( \text{optimalsolutionslist} = [ ] \). To derive the entire logic behind this branch-and-bound algorithm we need to consider three separate cases.

1. **Reject case.** Suppose we are at an arbitrary node/state of the recursion tree and that \( \text{currentpathcost} \) is strictly greater than \( \text{minpathcost} \). This case is obviously a reject case as the \( \text{currentpath} \) has a large crossing time than paths seen before. In such reject
**BC-BRANCH&BOUND([t_1, t_2, t_3, \ldots, t_n])**

**Input:** n crossing times 0 < t_1 ≤ t_2 ≤ t_3 ≤ \cdots ≤ t_n.

**Output:** Solution for n people crossing the bridge from left to right.

1. leftset ← \{1, 2, 3, \ldots, n\}; rightset ← \{\}; trip ← +1  // 1st trip is forward trip
2. currentpath ← []; currentpathcost ← 0; optimalsolutionslist ← [ ]
3. minpathcost ← \infty
4. BC-BRANCH&BOUND(leftset, rightset, currentpath, currentpathcost, trip)
5. print all solutions in optimalsolutionslist

**BC-BRANCH&BOUND(leftset, rightset, currentpath, currentpathcost, trip)**

**Input:** Local: Function arguments. Global: optimalsolutionslist.

**Output:** Optimal solutions added to optimalsolutionslist.

1. // Reject condition; will not lead to any optimal solution ............
2. if currentpathcost > minpathcost then return
3. // Leaf node of the recursion tree ........................................
4. if leftset is empty then
5. // Found a better solution than the current best solution ...........
6. if currentpathcost < minpathcost then
7. minpathcost ← currentpathcost; optimalsolutionslist ← [ ]
8. // Probably an optimal solution ...........................................
9. optimalsolutionslist.Insert(currentpath with its currentpathcost)
10. return
11. if trip = +1 then
12. if leftset has more than c people then
13. for each subset f of leftset of size in the range [2, c] do
14. choices ← choices.Insert(f)
15. else choices ← choices.Insert(leftset)
16. elseif trip = -1 then
17. for each 1-sized subset b of leftset do
18. choices ← choices.Insert(b)
19. // Loop through the choices and select one at a time ..............
20. for each item s in choices do
21. // Allocate resources and recurse ......................................
22. if trip = +1 then
23. { leftset ← leftset − s; rightset ← rightset ∪ s }
24. else if trip = -1 then
25. { leftset ← leftset ∪ s; rightset ← rightset − s }
26. currentpath ← currentpath.Insert(s)  // add trip s to currentpath
27. currentpathcost ← currentpathcost + t_{\text{max}}(s)  // add time t_{\text{max}}(s)
28. trip ← trip × -1  // next trip will switch between forward/backward
29. BC-BRANCH&BOUND(leftset, rightset, currentpath, currentpathcost, trip)

Figure 132: Optimal branch-and-bound algorithm to cross a bridge.
cases, we simply return to the parent node.

2. **Accept case.** Suppose we are at a leaf node of the recursion tree and it is not a reject case. Then there are two subcases:
   
   [(i) $\text{currentpathcost} < \text{minpathcost}$.] In this case, we have discovered a feasible solution that is better than all feasible solutions seen before. So we clear all contents of $\text{optimalsolutionslist}$, update $\text{minpathcost}$ to $\text{currentpathcost}$, and add the $\text{currentpath}$ to the $\text{optimalsolutionslist}$.
   
   [(ii) $\text{currentpathcost} = \text{minpathcost}$.] In this case, we add the $\text{currentpath}$ to $\text{optimalsolutionslist}$ as it has the same cost as that of the previous best feasible solutions.

3. **Normal case.** Suppose it is neither a reject case nor an accept case. We first try to find all possible choices/options for the child node. The number of choices decides the branch factor for the current node. There are two subcases.

   [(i) **Forward trip.**] If the next trip is a forward trip, i.e., $\text{trip} = +1$, then $\text{choices}$ will be the set of all possible subsets of $\text{leftset}$ of size in the range $[2, c]$ (or $\text{leftset}$ itself if the size of $\text{leftset}$ is not greater than $c$). For each possible choice (or forward trip) $s$, we subtract set $s$ from $\text{leftset}$ and add or take the union of set $s$ with $\text{rightset}$.

   [(ii) **Backward trip.**] If the next trip is a backward trip, i.e., $\text{trip} = -1$, then $\text{choices}$ will be the set of all possible 1-sized subsets of $\text{leftset}$ as we assume that only the fastest person on the right will return to the left side. For each possible choice (or backward trip) $s$, we subtract set $s$ from $\text{rightset}$ and add or take the union of set $s$ with $\text{leftset}$.

   For both subcases, for each possible choice $s$, we add the set $s$ to $\text{currentpath}$ and increment $\text{currentpathcost}$ with the maximum of the crossing times in the set $s$, i.e., $t_{\text{max}}$. We change the next trip status. Finally, we invoke BC-BRANCH&BOUND function for each of the child nodes recursively.

We have shown a recursive algorithm as recursive algorithms are easier to understand and generalize than their non-recursive counterparts.

**Sillke’s recursive algorithm ($c = 2$, optimal)**

We give a folklore recursive algorithm for solving the puzzle that works when the bridge capacity $c$ is 2.

**[Base case.]** There are three scenarios. If $n = 1$, then person $i$ goes from left to right in $t_1$ minutes. If $n = 2$, then persons $\{1, 2\}$ go from left to right in $t_2$ minutes. If $n = 3$, then persons $\{1, 2\}$ go from left to right in $t_2$ minutes, person 1 goes from right to left in $t_1$ minutes, and persons $\{1, 3\}$ go from left to right in $t_3$ minutes.

**[Recursive case.]** When $n \geq 4$, move people $n - 1$ and $n$ to the right side using one of the following two cases that minimizes crossing time and then recursively solve the puzzle for the remaining $n - 2$ people standing on the left side.

**[Case 1.]**

Persons $\{1, 2\}$ go from left to right in $t_2$ minutes.
Person 1 goes from right to left in $t_1$ minutes.
Persons $\{n - 1, n\}$ go from left to right in $t_n$ minutes.
Person 2 goes from right to left in $t_2$ minutes.
[Case 2.]

Persons \( \{1, n\} \) go from left to right in \( t_n \) minutes.
Person 1 goes from right to left in \( t_1 \) minutes.
Persons \( \{1, n - 1\} \) go from left to right in \( t_{n-1} \) minutes.
Person 1 goes from right to left in \( t_1 \) minutes.

Use one of the two cases above whichever leads to the minimum crossing time and then recursively solve the problem for the remaining \( n - 2 \) people \( \{1, 2, 3, \ldots, n - 2\} \). BC-SILLKE gives the nonrecursive version of the algorithm.

**BRIDGE-CROSSING-SILLKE([\( t_1, t_2, t_3, \ldots, t_n \)])**

**Input:** \( n \) crossing times \( 0 < t_1 \leq t_2 \leq t_3 \leq \cdots \leq t_n \).
**Output:** Optimal solution for \( n \) people crossing the bridge from left to right.

1. last ← \( n \)
2. while last ≥ 4 do
3.  if \( 2t_2 \leq t_1 + t_{last-1} \) then
4.    Persons \( \{1, 2\} \) go from left to right in \( t_2 \) minutes.
5.    Person 1 goes from right to left in \( t_1 \) minutes.
6.    Persons \( \{last - 1, last\} \) go from left to right in \( t_{last} \) minutes.
7.  else
8.  end if
9.    Persons \( \{1, last\} \) go from left to right in \( t_{last} \) minutes.
10.   Person 1 goes from right to left in \( t_1 \) minutes.
11.   Persons \( \{1, last - 1\} \) go from left to right in \( t_{last-1} \) minutes.
12.   Person 1 goes from right to left in \( t_1 \) minutes.
13.  end if
14.  last ← last − 2
15.  if last = 1 then
16.    Person 1 goes from left to right in \( t_1 \) minutes.
17.  else if last = 2 then
18.    Persons \( \{1, 2\} \) go from left to right in \( t_2 \) minutes.
19.  else if last = 3 then
20.    Persons \( \{1, 2\} \) go from left to right in \( t_2 \) minutes.
21.    Person 1 goes from right to left in \( t_1 \) minutes.
22.    Persons \( \{1, 3\} \) go from left to right in \( t_3 \) minutes.

Figure 133: Sillke’s optimal algorithm of crossing the bridge.

[Crossing time.] Let \( T_i \) be defined as the time taken by people \( \{1, 2, 3, \ldots, i\} \) to cross the bridge. Then

\[
T_i = \begin{cases} 
  t_i & \text{if } i = 1 \text{ or } i = 2, \\
  t_1 + t_2 + t_3 & \text{if } i = 3, \\
  t_1 + t_n + \min(2t_2, t_1 + t_{n-1}) + T_{i-2} & \text{if } i \geq 4. 
\end{cases}
\]
Backhouse’s dynamic programming algorithm (optimal)

BRIDECROSSING-BACKHOUSE in Figures 134 and 135 gives the extremely complicated algorithm based on dynamic programming to find an optimal crossing sequence for $n$ people and bridge capacity $c$. Interested readers can refer to the original paper by Backhouse for more details.

Time complexity to compute the optimal solution is $\Theta(n^2)$.

References

The history of the puzzle is unknown. Sillke’s algorithm is a folklore algorithm presented on Torsten Sillke’s web page [Sillke, 2022]. The most efficient algorithm known to solve the generalized version of the bridge crossing puzzle is presented in [Backhouse and Truong, 2015] and [Backhouse, 2011].
Figure 134: Backhouse’s bridge crossing DP algorithm. Remaining logic of the algorithm is continued in Figure 135.
BRIDECROSSING-BACKHOUSE(t[1...n], c)

25. Create an array leadpuretrips[1, 2, ..., maxpuretrips] ← [0, 0, ..., 0]
26. p ← 1; puretrips ← 0; i ← n; e ← 0
27. Create an arraylist of arraylists crossingsequence ← []

while i > c do
    j ← nomads[i, e]
    if j = 0 then
        leadpuretrips[p] ← n; puretrips ← puretrips + 1
        p ← p + 1; i ← i − c; e ← e + 1
    else
        nomadlist ← [1...j]; settlerlist ← [(i + 1 − (c − j))...i]
        forwardlist ← Concatenate(nomadlist, settlerlist)
        for k ← 1 to c do forwardlist[k] ← t[k]
        crossingsequence.Add(forwardlist)
        e ← e − (j − 1); i ← i − (c − j); backwardlist ← [t[j]]
        crossingsequence.Add(backwardlist)
    for k ← j − 1 downto 1 do
        forwardlist ← []
        for ℓ ← (leadpuretrips[p] − c + 1) to leadpuretrips[p] do
            forwardlist.Add(t[ℓ])
        crossingsequence.Add(forwardlist)
        backwardlist ← []; backwardlist.Add(t[k])
        crossingsequence.Add(backwardlist)
        p ← p − 1;
    while i ≠ 0 do
        j ← nomads[i, e]; forwardlist ← []
        for k ← 1 to i do forwardlist.Add(t[k])
        crossingsequence.Add(forwardlist)
        i ← j
    if j > 0 then
        e ← e − j + 1; backwardlist ← []; backwardlist.Add(t[k])
        crossingsequence.Add(backwardlist)
        j ← j − 1
    for k ← j to 1 do
        forwardlist ← []
        for ℓ ← leadpuretrips[p] − c + 1 to leadpuretrips[p] do
            forwardlist.Add(t[ℓ])
        crossingsequence.Add(forwardlist)
        p ← p − 1; backwardlist ← []; backwardlist.Add(t[k])
        crossingsequence.Add(backwardlist)
    return crossingsequence

Figure 135: Backhouse’s bridge crossing DP algorithm continued from Figure 134.
Horse Racing

Problem

There are 25 horses. There are only 5 race tracks and hence a maximum of 5 horses can participate in a race. Assume that there is no stop watch or timer and that there are no ties in the races. How do you select the 3 fastest horses minimizing the number of races?

Solution

This is a beautiful algorithmic puzzle commonly asked in technical interviews.

[Naïve solution (non-optimal).] Conduct 5 races, each race containing 5 horses, and from them select 15 fastest horses. Conduct 3 races, each race containing 5 horses, and select 9 fastest horses. Divide these 9 horses into 5 and 4 horses. Conduct a race for 5 horses and select 3 fastest horses among these 5 horses. There remains $3 + 4 = 7$ horses. Divide these 7 horses into 5 and 2 horses. Conduct a race for 5 horses and select 3 fastest horses among these 5 horses. There remains $3 + 2 = 5$ horses. There remains 5 horses. Conduct a race for 5 horses and select 3 fastest horses. These are the 3 fastest horses among 25 horses. Number of races = 11 (non-optimal).

[Standard solution (optimal).] Refer to Figure 136. Let the 25 horses be named as $h_{ij}$ for $i, j \in [1, 5]$. Conduct 5 races: race 1 with horses $h_{11} - h_{15}$, race 2 with horses $h_{21} - h_{25}$, race 3 with horses $h_{31} - h_{35}$, race 4 with horses $h_{41} - h_{45}$, and race 5 with horses $h_{51} - h_{55}$. Without loss of generality we can assume that the fastest horses in order of their ranks in race 1 are $h_{11}$ (fastest), $h_{12}$ (2nd fastest), $h_{13}$ (3rd fastest), $h_{14}$ (4th fastest), and $h_{15}$ (5th fastest), respectively. Similarly, the fastest horses in the order of their ranks in other races are $h_{21} - h_{25}$, $h_{31} - h_{35}$, $h_{41} - h_{45}$, and $h_{51} - h_{55}$. This means that the 3 fastest horses among 25 horses are definitely not $h_{14} - h_{54}$ and $h_{15} - h_{55}$ and hence they can be eliminated. After 5 races, 15 fastest horses will be selected.

Refer to Figure 137. We conduct a race among $h_{11} - h_{51}$. Without loss of generality we can assume that the fastest horses are $h_{11} - h_{31}$ in order of their ranks. This means that the 3 fastest horses cannot be $h_{41} - h_{43}$ and $h_{51} - h_{53}$. We are left with 9 fastest horses.

At this point we know with certainty that $h_{11}$ is the fastest horse among 25 horses. We also know that the 2nd fastest horse must be either $h_{21}$ or $h_{12}$. We do not yet know which are the 2nd and 3rd fastest horses, but, we know that they can never be among $h_{23}, h_{32}, h_{33}$. This leaves us with only 5 horses: $h_{21}, h_{12}, h_{31}, h_{32},$ and $h_{13}$. We conduct another race among the five horses $h_{21}, h_{12}, h_{31}, h_{32},$ and $h_{13}$ as shown in Figure 288.
The fastest and the 2nd fastest horses in this race will be the 2nd and the 3rd fastest horses among the 25 horses, respectively.

Number of races = 7 (optimal).

[Generalization.] There are $n$ horses. There are only $k$ number of race tracks and hence a maximum of $k$ horses can participate in a race. Assume that there is no stopwatch or timer and that there are no ties in the races. How do you select the $m$ fastest horses in the order of their ranks minimizing the number of races?

This problem is equivalent to the sorting problem. Suppose there are $n$ elements. We have access to a $k$-sorter that sorts at most $k$ elements in a single call to $k$-sorter. How do you select the $m$ smallest elements in the order of their ranks minimizing the number of calls to the $k$-sorter function.
Percentile sort algorithm (asymptotically optimal)

The PERCENTILESORT algorithm is a generalization of quicksort. This algorithm takes an \( n \)-sized unsorted array \( A[1 \ldots n] \) as input, uses a \( k \)-sorter repeatedly, where \( k \in [2, n] \), and returns the top \( m \in [1, n] \) elements of the sorted array. The time complexity of the algorithm is the number of times the algorithm calls \( k \)-SORTER.

We explain the algorithm in two parts:

[Part 1. Selection algorithm.] In this part, we want to find the element with rank \( rank \) using \( k \)-sorter in the sorted array of \( A \) as shown in SELECTION algorithm. If \( n \) is not greater than \( k \), we can simply sort the given array \( A \) and return \( A[rank] \).

If \( n \) is greater than \( k \), then we compute the answer as follows. Split the \( A \) array into \( \lceil \frac{n}{k} \rceil \) chunks of size at most \( k \). The last chunk might have less than \( k \) elements. Sort each of the chunks using a \( k \)-sorter. Then select the median element (i.e., the middle element) of each of the sorted chunks and store them in the medians array. Now select the median element of the medians array recursively and call it pivot. Once we have the pivot, partition the original \( A \) array except pivot into leftarray and rightarray such that all elements in leftarray are less than pivot and all elements in rightarray are greater than pivot. To achieve this, we split all elements of \( A \) except pivot into \( \lceil \frac{(n-1)}{k-1} \rceil \) chunks each of size at most \( k - 1 \). We add/consider the pivot element in each of these chunks and sort each of these at most \( k \)-sized chunks using a \( k \)-sorter. Based on these results, we can obtain leftarray and rightarray. The number of elements in leftarray plus one is stored in size. If \( rank < size \), then the answer is present in leftarray. So, we recursively find the element with rank \( rank \) in the sorted array of leftarray. Similarly, if \( rank > size \), then the answer is present in rightarray. So, we recursively find the element with rank \( rank - size \) in the sorted array of rightarray. If \( rank = size \), then pivot itself if the answer.

[Part 2. Percentile sort algorithm.] This part consists of three steps.

[Step 1. Find \( R \) points and \( S \) points.] Split the \( A \) array into \( \lceil \frac{n}{k} \rceil \) chunks of size at most \( k \). The last chunk might have less than \( k \) elements. Sort every chunk using a \( k \)-sorter.

We define \( t \)-items in array[1 \ldots \ell] as follows. Find the elements at ranks

\[
1, \quad \left\lfloor \frac{\ell}{t} \right\rfloor + 1, \quad 2 \left\lfloor \frac{\ell}{t} \right\rfloor + 1, \quad \ldots, \quad (t-2) \left\lfloor \frac{\ell}{t} \right\rfloor + 1, \quad (t-1) \left\lfloor \frac{\ell}{t} \right\rfloor + 1
\]

in the \( \ell \)-sized array. If array is sorted, then the \( t \)-items can be found considering array[\( i \times \lceil \ell/t \rceil + 1 \)] for all \( i \in [0, t-1] \). If array is unsorted, then the \( t \)-items can be found by using the SELECTION algorithm in array for rank \( i \times \lceil \ell/t \rceil + 1 \) for all \( i \in [0, t-1] \).

Set target number of points \( t = \max \left( 2, \left\lceil \sqrt{\frac{\sqrt{n}}{\log k}} \right\rceil \right) \). Create a dynamic array \( R \). Find the \( t \)-items in the first chunk and add them to the \( R \) array. Similarly, find the \( t \)-items in the second chunk and add them to the \( R \) array. Repeat this process for all chunks of \( A \). At the end we will have all such points in the \( R[1 \ldots r] \) array such that

\[
r = \begin{cases} 
\frac{n}{k} \times t & \text{if } (n \mod k) = 0, \\
\frac{n}{k} \times t + 1 & \text{if } (n \mod k) \in [1, t-1], \\
\frac{n}{k} \times t & \text{if } (n \mod k) \in [t, k-1]. 
\end{cases}
\]

(55)
Figure 139: PercentileSort algorithm to find the top $m$ elements of an $n$-sized unsorted array using a $k$-sorter.
**Selection**($A[1...n], rank, k$)

**Input:** An unsorted array $A[1...n]$, $rank \in [1,n]$ and $k \in [2,n]$

**Output:** Element at rank $rank$ in $A[1...n]$ using a $k$-sorter

1. if $n \leq k$ then \{ K-SORTER($A[1...n]$); return $A[rank]$ \}

2. // Find pivot - the median of medians of $A$ ...............................  

3. Split array $A$ into $[n/k]$ chunks each of size at most $k$

4. Each of the first $[n/k]$ chunks will have size $k$

5. If $n \mod k \neq 0$, then the $[n/k]$th chunk will have size less than $k$

6. Sort each of the $[n/k]$ chunks using a K-SORTER and find its median

7. Store all the $[n/k]$ medians in the median array

8. $pivot \leftarrow \text{SELECTION}\left(\text{median}\left[1...\left\lceil\frac{n}{k}\right\rceil\right], \left\lceil\frac{n}{k}\right\rceil, k\right)$

9. // Partition $A$ into leftarray and rightarray based on pivot .........

10. Create dynamic arrays leftarray $\leftarrow [\ ]$ and rightarray $\leftarrow [\ ]$

11. Split array $A$ into $[(n-1)/(k-1)]$ chunks each of size at most $k-1$

12. Consider pivot in each of these $[(n-1)/(k-1)]$ chunks and run a K-SORTER

13. Based on these results, partition all elements of $A$ except pivot into

leftsubarray and rightsubarray such that elements in leftsubarray are less
than pivot and elements in rightsubarray are greater than pivot

14. // Recursively call selection algorithm based on rank and size ....

15. size $\leftarrow |\text{leftarray}| + 1$

16. if $rank < size$ then return $\text{SELECTION}(\text{leftarray}, rank, k)$

17. if $rank > size$ then return $\text{SELECTION}(\text{rightarray}, (rank - size), k)$

18. if $rank = size$ then return pivot

**MakeBuckets**($A[1...n], S[1...s]$)

1. Initialize buckets with partitioning elements ............................

2. for $i \leftarrow 1$ to $s$ do bucket$i$ $\leftarrow [S[i]]$

3. for $i \leftarrow 0$ to $\left\lceil\frac{n}{k-s}\right\rceil - 1$ do

4. // Take $s$ elements from $S$ and $k-s$ from $A$. Compare the $k-s$
elements elements of $A$ with the partitioning elements .........

5. chunk$\_i$ $\leftarrow A[i \times (k-s) + 1 \ldots \min((i+1) \times (k-s), n)]$

6. temp $\leftarrow [\ ];$ temp.Add($S[1...s]$); temp.Add(chunk$\_i$)

7. temp $\leftarrow$ K-SORTER(temp)

8. // Distribute chunk$\_i$ elements to its respective buckets .........

9. index $\leftarrow s$

10. for $j \leftarrow |\text{chunk}_{i+1}|$ to $1$ do

11. | if temp$[j] = S[index]$ then index $\leftarrow index - 1$

12. else if temp$[j] > S[index]$ then bucket[index].Add(temp$[j]$)

13. return bucket$[1...s]$

Figure 140: Auxiliary functions in the PercentileSort algorithm.
These points are called \textit{R points}.

Similarly, find the \(t\)-items in the \(R[1 \ldots r]\) array using the \texttt{SELECTION} algorithm and add them to the \(S[1 \ldots s]\) array, where \(s = t\). These points stored in the \(S\) array are called \textit{S points}.

[\textbf{Step 2. Make buckets.}] In this step, we make as many buckets as there are \(S\) points. We create buckets \(\text{bucket}[1 \ldots s]\) such that \(\text{bucket}[i]\), for \(i \in [1, s - 1]\), stores elements of \(A\) in the range \([S[i], S[i + 1])\), i.e., greater than or equal to \(S[i]\) and less than \(S[i + 1]\), and \(\text{bucket}[s]\) stores elements of \(A\) in the range \([S[s], \infty)\). We will not have any elements less than \(S[1]\) because \(S[1]\) will be the minimum element of \(A\). We make buckets as follows. First, initialize \(\text{bucket}[i]\) with \(S[i]\) for all \(i\). The buckets will be populated as follows.

\[
\begin{array}{ccccccc}
& \text{bucket}[1] & & \text{bucket}[2] & & \text{bucket}[s] \\
\end{array}
\]

\textbf{Figure 141:} Bucketing using \(S[1 \ldots s]\) array.

Split the \(A\) array into \(\left\lceil \frac{n}{k - s} \right\rceil\) chunks such that each chunk has size at most \(k - s\). In iteration \(i\), copy the elements of \(\text{chunk}_i\) and \(S[1 \ldots s]\) into an empty dynamic array \(\text{temp}\). Now sort \(\text{temp}\), which has size at most \(k\), using a \(k\)-sorter. The contents of \(\text{temp}\) is then populated in the \(\text{bucket}\) array as shown in \textbf{Figure 141}. All elements to the right of (the last instance of) \(S[i]\) and to the left of (the first instance of) \(S[i + 1]\) will be larger than \(S[i]\) and smaller than \(S[i + 1]\) and hence will be added to \(\text{bucket}[i]\). The elements to the right of the last instance of \(S[s]\) will be larger than \(S[s]\) and hence will be added to \(\text{bucket}[s]\). In this way, we execute all iterations and populate all buckets.

[\textbf{Step 3. Recursively sort each bucket.}] We create a dynamic array called \(\text{sorter}\). We recursively sort the first bucket \(\text{bucket}[1]\) using \texttt{PERCENTILESORT} and store the result in \(\text{sorter}\). We add this \(\text{sorter}\) array to \(\text{sorter}\). We check if \(\text{sorter}\) has fewer than \(m\) elements. If \(\text{sorter}\) has fewer than \(m\) elements we recursively sort the second bucket \(\text{bucket}[2]\) using \texttt{PERCENTILESORT} and repeat the process above. We repeat the process until \(\text{sorter}\) has greater than or equal to \(m\) elements. Finally, we return the first \(m\) elements of \(\text{sorter}\).

[\textbf{Complexity.}] We observe that \(r = O\left(\frac{n}{\sqrt{k \log k}}\right)\) and \(s = t = O\left(\frac{n}{\log k}\right)\). The number of \(k\)-sorter operations required for different steps are as follows. \texttt{SELECTION}: \(O\left(\frac{n}{k}\right)\), \texttt{FINDRPOINTS}: \(O\left(\frac{n}{k}\right)\), \texttt{FINDSPPOINTS}: \(O\left(s \cdot \frac{n}{k}\right) = O\left(\frac{n}{k \log^2 k}\right)\), and \texttt{MAKEBUCKETS}: \(O\left(\frac{n}{k - s}\right) = O\left(\frac{n}{k}\right)\). There would be \(s\) buckets and the maximum size of each bucket is \(O\left(\frac{n}{k}\right)\). Let \(T(n)\) represent the worst-case number of \(k\)-sorter operations for \texttt{PERCENTILESORT}.

Then, using all the complexities given above we have

\[
T(n) = \begin{cases} 
1 & \text{if } n \leq k, \\
S \cdot T\left(\frac{n}{s}\right) + O\left(\frac{n}{k}\right) & \text{if } n > k.
\end{cases}
\]

Solving the equation and substituting for \(s\) and \(r\), we get
Number of races (or \(k\)-sorter operations) = \(O\left(1 + \frac{n \log n}{k \log k}\right)\).
Soldiers Firing

Problem

There is a long line of soldiers with the army general at one end and all of them facing unarmed terrorists. The army general needs to communicate to the group to shoot simultaneously. The problem constraints are as follows: the soldiers can meet and discuss the strategy on the previous day of execution; the actions of the soldiers are in lock-step (or synchronized); each soldier has constant memory and can only communicate with the immediate neighbors, i.e., at each time step each soldier takes an action based on what they themselves and their immediate neighbors were already doing in the previous time step. What strategy must the soldiers follow to synchronize firing?

Solution

This puzzle, often called the firing squad synchronization problem, is one of the finest puzzles from cellular automata and automata theory. The puzzle is more computational than algorithmic and more algorithmic than mathematical. We present several strategies for solving the puzzle.

[Fundamentals: Context-sensitive grammar.] Some most popular models of computation with increasing computational powers are as follows:

1. Regular grammar, deterministic finite automata (DFA), or regular expressions
2. Context-free grammar (CFG) or push-down automata (PDA)
3. Context-sensitive grammar (CSG) or linear-bounded automata (LBA)
4. Unrestricted grammar (UG) or Turing machines (TM)

that have varying computational powers and hence solve different classes of computational problems.

The simplest computational model that can be used to solve this puzzle is context-sensitive grammar. A context-sensitive grammar is a grammar \( G = (N, \Sigma, P, S) \), where \( N \) is a set of nonterminal symbols, \( \Sigma \) is a set of terminal symbols, \( P \) is a set of transition/production rules, \( S \) is the start symbol, and all rules in \( P \) are of the form:

\[
rAs \to rts
\]

where \( A \) is a nonterminal i.e., \( A \in N \); \( r \) and \( s \) are strings of nonterminals and terminals, i.e., \( r, s \in (N \cup \Sigma)^* \) and \( t \) is a nonempty string of nonterminals and terminals, i.e., \( t \in (N \cup \Sigma)^+ \).

Solutions

Every solution for solving this puzzle is a special case of a context-sensitive grammar and has several characteristics such as:
Our aim is to design optimal solutions, which minimize the time first, minimize #states second, minimize #rules third, and minimize state change complexity fourth. Solutions that minimize the time only are called time-optimal solutions.

Some concepts that we use for all the solutions for this puzzle are as follows:

- **Start state.** Starting/quiescent state of each of the soldiers.
- **Final state.** Firing/synchronization state which all soldiers end up with together.
- **#Rules.** Total number of state transition rules.
- **#States.** Total number of nonterminals used in the solution.
- **Time.** Number of timesteps required to reach the final state from start state.
- **State change complexity.** Number of times the states of the soldiers change.

Table 55 summarizes all solutions.

<table>
<thead>
<tr>
<th>Solution</th>
<th>Time</th>
<th>#States</th>
<th>#Rules</th>
<th>State changes</th>
<th>Idea</th>
</tr>
</thead>
<tbody>
<tr>
<td>McCarthy-Minsky (1957)</td>
<td>≤ 3n</td>
<td>13</td>
<td>138</td>
<td>O(n log n)</td>
<td>—</td>
</tr>
<tr>
<td><strong>Time-optimal solutions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goto (1962)</td>
<td>2n − 2</td>
<td>thousands</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Goto reconstructed (2018)</td>
<td>2n − 2</td>
<td>165</td>
<td>4378</td>
<td>O(n log n)</td>
<td>—</td>
</tr>
<tr>
<td>Waksman (1966)</td>
<td>2n − 2</td>
<td>16</td>
<td>202</td>
<td>O(n²)</td>
<td>D&amp;C</td>
</tr>
<tr>
<td>Balzer (1967)</td>
<td>2n − 2</td>
<td>8</td>
<td>165</td>
<td>O(n²)</td>
<td>D&amp;C</td>
</tr>
<tr>
<td>Gerken (1987)</td>
<td>2n − 2</td>
<td>7</td>
<td>105</td>
<td>O(n²)</td>
<td>D&amp;C</td>
</tr>
<tr>
<td>Mazoyer (1987)</td>
<td>2n − 2</td>
<td>6</td>
<td>119</td>
<td>O(n²)</td>
<td>D&amp;C</td>
</tr>
<tr>
<td>Clergue-Verel-Formenti (2018)</td>
<td>2n − 2</td>
<td>6</td>
<td>80</td>
<td>O(n²)</td>
<td>Local search</td>
</tr>
</tbody>
</table>

Table 55: Solutions for the soldiers firing puzzle.
Each solution can be described in one of two ways:

1. **[State transition rules.]** This style uses a list of state transition rules. Each state transition rule will be of the form $UVW \rightarrow Y$, where, if a soldier is in state $V$, the soldier on the left is in state $U$, the soldier on the right is in state $W$, then the soldier's new state is $Y$.

2. **[State transition tables.]** This style uses a set of transition tables. If $V$th table's $U$th row $W$th column being set to $Y$ means that if a soldier is in state $V$, the soldier on the left is in state $U$, the soldier on the right is in state $W$, then the soldier's new state is $Y$.
state is $Y$.

[McCarthy-Minsky's solution.] Figure 142 gives the state transitions. This solution which is non-optimal with respect to time has 13 states and 138 rules and takes at most $3n$ time. In the transitions, $\|$ and $Q$ denote the leader state and the initial inactive state of all soldiers, respectively. The algorithm makes use of parallel divide-and-conquer and divides the original problem of size $n$ into subproblems of size $\frac{n}{2}$ to

Figure 143: Waksman's solution.

MATHEMATICAL AND ALGORITHMIC PUZZLES
solve the problem.

**[Waksman’s solution.]** Figure 143 gives the state transitions. This solution has 16 states and 202 rules. In the transitions, \( P_0 \) and \( Q \) denote the leader state and the
initial inactive state of all soldiers, respectively. The algorithm makes use of parallel divide-and-conquer and divides the original problem of size $n$ into a subproblems of size $\frac{n}{2}$ to solve the problem.

[Balzer’s solution.] Figure 144 gives the state transition tables. This solution has 8 states and 165 rules. In the transition tables, $M$ and $L$ denote the leader state and the initial inactive state of all soldiers, respectively. The algorithm makes use of parallel divide-and-conquer and divides the original problem of size $n$ into a subproblems of size $\frac{n}{2}$ to solve the problem.

[Gerken’s solution.] Figure 145 gives the state transition tables. This solution has 7 states and 105 rules. In the transition tables, $>$ and $/$ denote the leader state and the initial inactive state of all soldiers, respectively. The algorithm makes use of parallel divide-and-conquer and divides the original problem of size $n$ into a subproblems of
size $\frac{n}{2}$ to solve the problem.

[Mazoyer’s solution.] Figure [146] gives the state transition tables. This solution has 6 states and 119 rules. In the transition tables, $G$ and $L$ denote the leader state and the initial inactive state of all soldiers, respectively. The algorithm makes use of parallel divide-and-conquer and divides the original problem of size $n$ into a subproblems of size $\frac{2n}{3}$ to solve the problem.

[Clergue-Verel-Formenti’s solution.] Figure [147] gives the state transition tables. This solution has 6 states and 80 rules. In the transition tables, $A$ and $G$ denote the leader state and the initial inactive state of all soldiers, respectively.

Problems

1. [Optimal solution.] Design an optimal solution to solve the puzzle.

References

The puzzle was proposed by John Myhill (1957). The first solution was given by John McCarthy and Marvin Minsky and published by Moore, but it was a non-optimal solution. The first time-optimal solution was proposed by Goto (1962) which required thousands of states but the first implementation of its reconstructed version appeared in [Umeo et al., 2018] with 165 states, 4378 rules and $O(n \log n)$ state change complexity. Other time-optimal solutions are presented in [Waksman, 1966], [Balzer, 1967], [Noguchi, 2004], [Gerken, 1987], and [Mazoyer, 1987] as summarized in Table 55. [Clergue et al., 2018], [Verel, 2022] used the iterated local search algorithm to compute 706 of 6-state time-optimal solutions (if we consider only unique solutions considering renaming of states as shown in [Nguyen and Maignan, 2020]). Two of these solutions required 80 rules. [Nguyen and Maignan, 2020] used reductions and local search algorithm to compute millions of 6-state time-optimal solutions.

An excellent survey on the puzzle, its solutions, and its variants is [Umeo, 2009]. The puzzle has been generalized to higher dimensional grids, graphs, and arbitrary starting point for the leader. It is unknown if there exists any solution (optimal or not) that requires only 5 states.
Figure 148: McCarthy's space-time diagram.

Figure 149: Waksman's space-time diagram.

Figure 150: Balzer's space-time diagram.
Figure 151: Gerken's space-time diagram.

Figure 152: Mazoyer's space-time diagram.
Majority Vote

Problem

An election was held in a democratic nation to elect their next leader. The citizens of the nation voted for their favorite candidates. It is now time to find whether someone won the election. Winning the election means getting a majority of votes. Given a set of elements, an element is a majority in that set if that element occurs greater than 50% of the number of elements in that set. If there is no majority in an election, there will be a re-election and the process repeats until there is a majority.

So, how do you find whether someone won an election?

Solution

This problem is frequently asked in technical interviews. Let’s first formalize the problem.

[Problem statement.] Given a sequence of natural numbers $A_1, A_2, \ldots, A_n$, find the majority in the sequence or determine that none exists.

We will assume the general case that the number of possible candidates is unlimited and that they are assigned arbitrary but distinct natural numbers. We are now ready to see several solutions for solving the puzzle, as summarized in Table 56.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Extra Space</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(1)$</td>
<td>–</td>
</tr>
<tr>
<td>Sorting-based</td>
<td>$\Theta(n \log n)$</td>
<td>$\Theta(n)$</td>
<td>Can’t solve our puzzle.</td>
</tr>
<tr>
<td>Divide-and-conquer</td>
<td>$\Theta(n \log n)$</td>
<td>$\Theta(\log n)$</td>
<td>–</td>
</tr>
<tr>
<td>Probabilistic</td>
<td>$\Theta(n \log \frac{1}{\epsilon})$</td>
<td>$\Theta(1)$</td>
<td>Can’t solve our puzzle. Success $&gt; 1 - \epsilon$.</td>
</tr>
<tr>
<td>Hashing-based</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
<td>Can’t solve our puzzle.</td>
</tr>
<tr>
<td>Median-based</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
<td>Can’t solve our puzzle.</td>
</tr>
<tr>
<td>Boyer-Moore multipass</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
<td>–</td>
</tr>
<tr>
<td>Boyer-Moore singlepass</td>
<td>$\leq 2n - 2$</td>
<td>$\Theta(1)$</td>
<td>–</td>
</tr>
<tr>
<td>Fischer-Salzberg</td>
<td>$\leq \lfloor 3n/2 \rfloor - 2$</td>
<td>$\Theta(n)$</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 56: Comparison of algorithms to solve the majority vote puzzle

[Brute force algorithm.] A brute force algorithm is to count the number of occurrences of every element in the array. If the count is greater than $\lfloor n/2 \rfloor$ for some element, then that element is the majority element. The algorithm is given in MAJORITY-BRUTEFORCE.
**MAJORITY-BRUTEFORCE**($A[1...n]$)

**Input:** Array $A[1...n]$ of natural numbers.

**Output:** Majority element if it exists and $-1$ otherwise.

1. for $i$ ← 1 to $[n/2]$ do
2. count ← COUNTOCCURRENCES($A[i...n], A[i]$)
3. if count > $[n/2]$ then return $A[i]$
4. return $-1$

**COUNTOCCURRENCES**($A[l...h], k$)

1. count ← 0
2. for $i$ ← $l$ to $h$ do
3. if $A[i] = k$ then count ← count + 1
4. return count

**MAJORITY-SORT**($A[1...n]$)

1. $A[1...n]$ ← SORT($A[1...n]$)
2. $i$ ← 1
3. for $j$ ← 2 to $n$ do
5. if $(j - i) > [n/2]$ then return $A[i]$
6. $i$ ← $j$
7. if $(n - i + 1) > [n/2]$ then return $A[i]$
8. return $-1$

**MAJORITY-D&C**($A[1...n]$)

1. return D&C($A[1...n]$)

D&C($A[low...high]$)

1. if $low = high$ then return $A[low]$
2. $m$ ← $(high - low + 1); mid$ ← $((low + high)/2)$
3. $l_{majority}$ ← D&C($A[low...mid]$)
4. $r_{majority}$ ← D&C($A[(mid + 1)...high]$)
5. $l_{count}$ ← COUNTOCCURRENCES($A[low...high], l_{majority}$)
6. $r_{count}$ ← COUNTOCCURRENCES($A[low...high], r_{majority}$)
7. if $l_{count} > [m/2]$ then return $l_{majority}$
8. if $r_{count} > [m/2]$ then return $r_{majority}$
9. return $-1$

**[Sorting-based algorithm.]** Sort the sequence of votes using a fast $Θ(n \log n)$ sorting algorithm (such as merge sort, quicksort with a linear-time median finding algorithm, heapsort, etc). Count the number of occurrences of each element in the array and then
find the majority element. The algorithm is given in MAJORITY-SORT.

\[ \langle \text{Time, Space} \rangle \text{ complexity to determine majority vote is } \langle \Theta(n \log n), \Theta(n) \rangle. \]

This sorting-based solution has a limitation, i.e., it assumes that we can compare two elements (e.g., candidates) in the given sequence with comparison operators (<, =, and >). We can compare two elements with comparison operators if they are integers, real numbers, or any objects with a total order. However, we cannot compare two elements using comparison operators if they are logos, images, songs, colored balls, coins of different weights, and chip-based identity cards without personal information.

The other solutions to this puzzle assume that there is an equality operator (=). The assumption is that given two elements \(a\) and \(b\), we can check whether \(a = b\). Hence, other solutions can be used to determine a majority element, assuming an equality operator, even in a sequence of logos, images, songs, colored balls, coins of different weights, chip-based identity cards (without personal information), etc.

**[Divide-and-conquer algorithm.]** There is a beautiful algorithm that can determine the majority element using divide-and-conquer. First, we split the input array into two or almost two equal halves: left and right. We recursively compute the majority element in both the left and the right halves. Suppose the majority element in the left and right halves be \(l_{\text{majority}}\) and \(r_{\text{majority}}\), respectively. Let \(m\) be the size of the input array. If \(l_{\text{majority}}\) occurs greater than \([m/2]\) times in the input array, then \(l_{\text{majority}}\) is the majority element in the input array. On the other hand, if \(r_{\text{majority}}\) occurs greater than \([m/2]\) times in the input array, then \(r_{\text{majority}}\) is the majority element in the input array. Finally, if neither \(l_{\text{majority}}\) nor \(r_{\text{majority}}\) occurs more than \([m/2]\) times in the input array, then there is no majority element in the input array. The algorithm is given in MAJORITY-D&C.

The time taken by the algorithm can be easily computed using the recurrence 
\[ T(n) = 2T([n/2]) + \Theta(n). \]

\[ \langle \text{Time, Space} \rangle \text{ complexity to determine majority vote is } \langle \Theta(n \log n), \Theta(\log n) \rangle. \]

**[Hashing-based algorithm.]** Hash map is a good data structure to use in applications that require computing #occurrences (also called frequencies or multiplicity). We create a hash map to store #occurrences of each distinct element in the sequence. The hash map will be a collection of \((\text{key}, \text{value})\) pairs, where \(\text{key}\) is the distinct element and \(\text{value}\) is #occurrences of that element in the sequence. We traverse the input array and check if element \(A[i]\) is present in the hash table. If \(A[i]\) is present, we increment its frequency by 1. If \(A[i]\) is not present, we insert \(A[i]\) into the hash map with its frequency set to 1. If frequency of \(A[i]\) is greater than \([n/2]\), then \(A[i]\) is the majority element. If even after traversing the array completely we are not able to find the majority element, then there is no majority. The algorithm is given in MAJORITY-HASHING.

\[ \langle \text{Time, Space} \rangle \text{ complexity to determine majority vote is } \langle \Theta(n), \Theta(n) \rangle. \]
### Mathematical and Algorithmic Puzzles

**MAJORITY-HASHING(A[1...n])**

1. Create hash map \( H \) to insert (element, frequency) pairs
2. for \( i \leftarrow 1 \) to \( n \) do
   3. if \( H.\text{ContainsKey}(A[i]) \) then \( H.\text{Add}((A[i], H.\text{GetValue}(A[i]) + 1)) \)
   4. else \( H.\text{Add}(A[i], 1) \)
5. if \( H.\text{GetValue}(A[i]) > \lceil n/2 \rceil \) then return \( A[i] \)
6. return \(-1\)

**MAJORITY-PROBABILISTIC(A[1...n])**

1. for \( i \leftarrow 1 \) to \( \lfloor \log_2 n \rfloor \) do
   2. random \( \leftarrow \text{RANDOM}(A[1...n]) \)
   3. count \( \leftarrow \text{COUNTOCCURRENCES}(A[1...n], D&C) \)
   4. if \( \text{count} > \lceil n/2 \rceil \) then return random
5. return \(-1\)

**MAJORITY-MEDIAN(A[1...n])**

1. median \( \leftarrow \text{SELECTION}(A[1...n], \lfloor n/2 \rfloor) \)
2. count \( \leftarrow \text{COUNTOCCURRENCES}(A[1...n], \text{median}) \)
3. if \( \text{count} > \lfloor n/2 \rfloor \) then return median
4. return \(-1\)

**MAJORITY-BOYERMOORE-MULTIPASS(A[1...n])**

1. return MAJORITY-MULTIPASS(A[1...n], \(-1\))

**MAJORITY-MULTIPASS(A[1...n], tiebreaker)**

1. Create a dynamic array \( B \leftarrow [ ] \)
2. for \( i \leftarrow 1 \) to \( n - 1 \) increment 2 do
   3. if \( A[i] = A[i+1] \) then \( B.\text{Add}(A[i]) \)
4. if \( n \mod 2 = 1 \) then tiebreaker \( \leftarrow A[n] \)
5. if \( B \) is empty then return tiebreaker
6. \( c \leftarrow \text{MAJORITY-MULTIPASS}(B, \text{tiebreaker}) \)
7. if \( c = -1 \) then return \(-1\)
8. count \( \leftarrow \text{COUNTOCCURRENCES}(A[1...n], c) \)
9. if \( \text{count} > \lfloor n/2 \rfloor \) or \( \text{count} = \lfloor n/2 \rfloor \) and \( c = \text{tiebreaker} \) then
   10. return \( c \)
11. return \(-1\)

This hashing-based solution has a limitation, i.e., it does not work with our puzzle because we need to assume that we cannot take the hash value of physical logos, images, songs, colored balls, coins of different weights, chip-based identity cards (without personal information), etc.
[Median-based algorithm.] A nice idea is as follows. *If there exists a majority element, then it must be the median (i.e., $\lfloor n/2 \rfloor$th smallest element).* So, we find the median of the sequence using a median-finding algorithm. We then check if the median is a majority element or not. If the median is not a majority element, then there is no majority element. The algorithm is given in MAJORITY-MEDIAN.

This median-based solution has a limitation, i.e., it does not work with our puzzle because the linear-time selection algorithm used to compute the median makes use of the partition logic and this partition function uses comparison operations similar to that in the sorting-based solution.

[Probabilistic algorithm.] We can compute the majority element with success probability greater than $1 - \epsilon$ for any desired real number $\epsilon \in (0, 1)$. Fix $\epsilon \in (0, 1)$. Select a random element of the array and check if it is indeed the majority element. If yes, then return the majority. If no, select a random element for the second time and check if it is the majority. Repeat this process $\lceil \log_2(1/\epsilon) \rceil$ times. The majority element will be found with greater than $1 - \epsilon$ probability. The algorithm is given in MAJORITY-PROBABILISTIC.

\[
\text{Prob. that the majority is not selected after 1 iteration} < 1/2 \\
\text{Prob. that the majority is not selected after 2 iterations} < (1/2)^2 \\
\text{Prob. that the majority is not selected after } \lceil \log_2(1/\epsilon) \rceil \text{ iterations} < (1/2)^{\lceil \log_2(1/\epsilon) \rceil} < \epsilon
\]

Hence, the probability that the majority is selected after $\lceil \log_2(1/\epsilon) \rceil$ iterations is greater than $1 - \epsilon$.

This solution has a limitation. It does not guarantee the selection of the majority as the algorithm is probabilistic.

[Boyer-Moore’s multipass algorithm.] Suppose there are $n$ elements in a multiset $A$, where $n$ is even for simplicity. Create another multiset $B$. Step 1. Remove a pair of elements from $A$. Step 2. If the two elements are equal, then add one element to $B$. In contrast, if the two elements are distinct, then eliminate both the elements. Step 3. Repeat Step 1 until there are no elements left in $A$. An important invariant to observe here is that the majority element in $A$ is same as the majority element in $B$. If there is no majority element in $A$, then there is no majority element in $B$. We can apply the three steps to set $B$ and reduce the sample space for the majority candidate. If we repeatedly apply this idea, we will find the majority element.

The algorithm is given in MAJORITY-BOYERMOORE-MULTIPASS that works for arbitrary natural number $n$. The initial value of tiebreaker for the function will be $-1$ (meaning no majority element). We compare $A[1]$ and $A[2]$, $A[3]$ and $A[4]$, so on. If a pair of elements of $A$ are identical then add that element to $B$. If the number of elements in $A$ is odd then the last element $A[n]$ will be the tiebreaker. We recursively
compute the majority candidate of \( B \) in \( c \) and store the number of occurrences of \( c \) in the array \( A \) as \textit{count}. If \( \text{count} > \lfloor n/2 \rfloor \) then \( c \) is the majority element. If \( \text{count} = \lfloor n/2 \rfloor \) and \( c = \text{tiebreaker} \), it means that the number of times \( c \) appears in the array \( A \) is slightly more than \( \lfloor n/2 \rfloor \) due to \textit{tiebreaker}, which automatically implies \( c \) is the majority element. In all other cases, there is no majority element and hence \(-1\) is returned.

\[
\langle \text{Time, Space} \rangle \text{ complexity to determine majority vote is } \langle \Theta(n), \Theta(n) \rangle.
\]

\[\text{Boyer-Moore's singlepass algorithm.}\] \textsc{Majority-BoyerMoore-Singlepass} gives the algorithm. There are two variables in the algorithm: \( c \), the majority candidate and \( m \), the number of unpaired occurrences of the majority candidate \( c \). The intuition behind \( m \) will be made clear later. When \( m > 0 \), then \( c \) represents the majority element. When \( m = 0 \), the element stored in \( c \) does not mean anything. It is not possible for \( m \) to be negative.

We now describe the algorithm. In the first iteration, we start by setting \( c \leftarrow A[1] \) and \( m \leftarrow 1 \), i.e., the number of unpaired occurrences of the majority candidate \( A[1] \) is 1. In the \( i \)th iteration for \( i \in [2, n] \), we do the following. If \( m = 0 \), then the \( i \)th element \( A[i] \) is the majority candidate and hence we set \( c \leftarrow A[i] \) and \( m \leftarrow 1 \). If \( m > 0 \), then we check if \( A[i] \) equals \( c \). If \( A[i] = c \), then we increment \( m \) by 1. If \( A[i] \neq c \), then we decrement \( m \) by 1. The reason is simple. If \( A[i] = c \), the number of unpaired occurrences of \( c \) increases by 1. If \( A[i] \neq c \), then we can eliminate \( A[i] \) (which is not equal to \( c \)) and one copy of \( c \) and hence we decrement \( m \) by 1. Observe that in this process \( m \) never becomes negative because when \( m \) hits 0, it again becomes positive. At the end, we count the number of occurrences of the majority candidate \( c \). If that count is greater than \( \lfloor n/2 \rfloor \), then \( c \) is the majority. If not, then there is no majority.

\[
\langle \text{Time, Space} \rangle \text{ complexity to determine majority vote is } \langle 2n - 2, \Theta(1) \rangle.
\]

\begin{center}
\textbf{References}
\end{center}

Boyer-Moore’s single pass algorithm is presented in [Boyer and Moore, 1991]. Fischer-Salzberg’s algorithm is presented in [Fischer and Salzberg, 1982] and it minimizes the number of element-to-element equality comparisons. The pseudocodes of Boyer-Moore’s single pass algorithm and Fischer-Salzberg’s algorithm are given in [Kleerekoper, 2016]. A parallel algorithm to compute the majority is given in [Cafaro and Tempes, 2011]. The generalization of Boyer-Moore’s single pass algorithm to compute element(s) that occurs greater than \( \lfloor n/k \rfloor \) number of times are discussed in [Misra and Gries, 1982].
MAJORITY-BOYERMOORE($A[1...n]$)

1. $c \leftarrow A[1]; m \leftarrow 1$
2. // Stage 1. Eliminate all except one candidate $c$ .................
3. for $i \leftarrow 2$ to $n$ do
4.     if $m = 0$ then // $c \leftarrow A[i]; m \leftarrow 1$
5.     else
6.         if $c = A[i]$ then $m \leftarrow m + 1$
7.         else $m \leftarrow m - 1$
8. // Stage 2. Check whether $c$ is a majority ........................
9. if $m \neq 0$ then
10.    count $\leftarrow \text{COUNTOCCURRENCES}(A[1...n], c)$
11.    if $count > \lfloor n/2 \rfloor$ then return $c$
12. return $-1$

MAJORITY-FISCHERSALZBERG($A[1...n]$)

1. // Stage 1. Find the majority candidate $c$ .........................
2. Create two stacks $L$ and $B$
3. for $i \leftarrow 1$ to $n$ do
4.     if $L$ is empty or $L$.Top() $\neq A[i]$ then
6.     else if $B$ is not empty then $L$.Push($B$.Pop())
8. $c \leftarrow L$.Top()
9. // Stage 2. Confirm if the candidate is the majority ..............
10. while $L$ is not empty do
11.    item $\leftarrow L$.Pop()
12.    if item $= c$ then
13.        if $L$ is empty then $B$.Push($c$)
14.        else if $L$ is not empty then $L$.Pop()
15.    else if item $\neq c$ then
16.        if $B$ is empty then return $-1$
17.        else if $B$ is not empty then $B$.Pop()
18.    if $B$ is empty then return $-1$
19. return $c$
Mouse in a Hole

Problem

There are five holes on the ground in a straight line from left to right. One of them is occupied by a mouse. Every night, the mouse moves to a neighboring hole, either to the left or to the right. Every morning, we get a chance to inspect a hole of our choice. What strategy would ensure that the mouse is eventually caught?

Solution

The strategy may not be easy to find out directly. We will solve the puzzle for small instances. We will then see if there is a pattern among those solutions.

We denote a solution strategy as

\[ \text{Strategy} = [h_1, h_2, \ldots, h_d] \]

where \( h_i \) is the hole inspected on day \( i \in [1, d] \).

When there is only 1 hole, we can catch the mouse in only one attempt by checking the hole, i.e., [1].

When there are 2 holes, we can catch the mouse in two attempts. There are two optimal strategies to do this: (i) [1, 1] and (ii) [2, 2]. Consider the first strategy [1, 1]. We search for the mouse in the first hole twice. If the mouse was in the first hole initially, it gets caught in the first attempt. If the mouse was in the second hole initially, after one night it moves to the first hole. Then, it gets caught in the second attempt. The second strategy can be explained similarly.

When there are 3 holes, we can catch the mouse in two attempts and there is only one optimal strategy for this: [2, 2].

When there are 4 holes, we can catch the mouse in four attempts and there are two optimal strategies for this: (a) [2, 3, 3, 2] and (b) [3, 2, 2, 3]. Consider the first strategy [2, 3, 3, 2]. We search for the mouse in the second hole on the first day, third hole on the second day, third hole on the third day and second hole on the fourth day. If the mouse starts from an even numbered hole i.e 2 or 4, then it is caught in the first half of the attempts: [2, 3]. If the mouse starts from an odd numbered hole i.e 1 or 3, then it is caught in the second half of the attempts: [3, 2]. The second strategy can be explained in a similar way.

When there are 5 holes, we can catch the mouse in 6 attempts. There are 4 optimal strategies: [2, 3, 4, 4, 3, 2], [4, 3, 2, 2, 3, 4], [2, 3, 4, 2, 3, 4] and [4, 3, 2, 4, 3, 2].

Any optimal strategy works as follows. If the mouse starts from an even numbered hole, then it is caught in the first half of the attempts. If the mouse starts from an odd numbered hole, then it is caught in the second half of the attempts. Figure 154 helps
Figure 154: Optimal strategy \([2, 3, 4, 5, 6, 5, 4, 3, 2]\) for \(n = 7\). Color blue represents possible mouse positions and color red represents holes that we inspect. The figure depicts the scenarios when the mouse starting position is an even hole (left) or an odd hole (right). Similar figures can be drawn for other optimal strategies as well.

You to visualize an optimal strategy when there are 7 holes. An important thing to realize about the problem is that if the mouse is in an even-numbered hole on day \(i\), then it will move to an odd-numbered hole on day \(i + 1\), and vice versa. So, the mouse is restricted to always staying on the blue cells as shown in the figure.

**[Generalization.]** We generalize the solution to \(n \geq 4\) holes. The optimal strategies for the problem are given in the MOUSEINAHOLE algorithm. There are four optimal strategies when \(n\) is odd and there are two optimal strategies when \(n\) is even.

\[
\text{MOUSEINAHOLE}(n)
\]

**Input:** Number of holes \(n \geq 4\) in a straight line.

**Output:** Optimal strategies to catch the mouse.

1. number of attempts \(= 2(n - 2)\)
2. \textbf{if} \(n\) is odd \textbf{then}
3. number of optimal strategies \(= 4\)
4. first optimal strategy \(= [2, \ldots, (n - 1), (n - 1), \ldots, 2]\)
5. second optimal strategy \(= [(n - 1), \ldots, 2, 2, \ldots, (n - 1)]\)
6. third optimal strategy \(= [2, \ldots, (n - 1), 2, \ldots, (n - 1)]\)
7. fourth optimal strategy \(= [(n - 1), \ldots, 2, (n - 1), \ldots, 2]\)
8. \textbf{else if} \(n\) is even \textbf{then}
9. number of optimal strategies \(= 2\)
10. first optimal strategy \(= [2, \ldots, (n - 1), (n - 1), \ldots, 2]\)
11. second optimal strategy \(= [(n - 1), \ldots, 2, 2, \ldots, (n - 1)]\)

**Problems**

1. Prove the correctness of the \textsc{MouseInaHole} algorithm.
2. What is the strategy if the holes are connected as a circle or a tree or a graph? If there is no solution, then solve the problem assuming multiple people to catch the
mouse. Is there a strategy to catch the mouse?

3. A variant of the puzzle is the *cops and robbers* puzzle. Suppose there is a connected graph. First a cop and then a robber occupy a vertex in the graph. After that they move one after the other along the edges of the graph. The cop wins if the cop ends up at the same vertex as that of the robber. The robber wins if the robber can escape from the cop forever. The problem can be generalized to multiple cops. Try to analyze the problem.

**References**

Refer to [Bonato and Nowakowski, 2011](#) for the cops and robbers puzzle and its other variants.
Truth Teller

Problem

A strange island has \( n \) people. It is known that strictly more than half of the people are truth tellers and they always tell the truth. The remaining people are unreliable who may speak the truth or lie arbitrarily. Each person of the island knows who is who.

Any visitor who visits the island must find out at least one truth teller by asking queries. An \((i, j)\) query is a question to person \( i \) asking if person \( j \) is a truth-teller or a liar. If a visitor succeeds in finding at least one truth teller correctly in the minimum number of queries, then the visitor survives. On the other hand, if the visitor fails to find at least one truth teller in the minimum number of queries, then the visitor will be killed by the islanders.

Suppose you visit the island. How would you find the identity of at least one truth teller in the minimum number of queries?

Solution

The puzzle belongs to logic. There are several logical puzzles like the current puzzle involving truth tellers, liars, and unreliable people (who may speak the truth or lie). This puzzle is a variant where there are truth tellers and unreliable people only.

We solve the following two subproblems:

- [One truth-teller.] Find one truth-teller in minimum number of queries.
- [All identities.] Find the identities of all people in minimum number of queries.

The problem has applications in distributed computing, where a few computers are faulty. Working computers always answer correctly and faulty computers can be arbitrarily correct. We need to identify the faulty computers.

[Notations.] The notations used in the subsequent sections are summarized in Figure 155. We know that \( t > u \), or in other words \( t \in [u + 1, n] \).

We represent the system of truth-tellers, unreliable people, and queries as a graph. A person \( P_i \) is denoted by a node labeled \( P_i \). A query \( q(P_i, P_j) \) is denoted by a directed edge from node \( P_i \) to node \( P_j \) with the result of the query as the edge label. The identities of \( P_1, \ldots, P_n \) are unknown. The possible types of query results are shown in Figure 156. The problem asks us to \((i)\) find one truth-teller node, and \((ii)\) reveal the identities of all people, with a minimum number of directed edges.

Majority vote solution (non-optimal)

The core idea of the solution is the following observation. If people \( P_1, P_2, P_3, \ldots, P_m \) are asked about the identity of one person, say, \( P_{m+1} \), and if the majority of \( P_1, P_2, P_3, \ldots, P_m \)
**Notation** | **Meaning**  
--- | ---  
\( n \) | Total number of people.  
\( t \) | Number of truth-tellers.  
\( u \) | Number of unreliable people.  
\( P_i \) | Person \( i \).  
\( T \) | A truth-teller.  
\( U \) | An unreliable person.  
\( q(P_i, P_j) \) | Query asking \( P_i \) to reveal the identity of \( P_j \).  
\( Q_T \) | Minimum #queries required to identify a truth-teller.  
\( Q_{all} \) | Minimum #queries required to reveal all identities.

---

Figure 155: Notations and meanings.

are truth-tellers, then the majority answer will be the correct identity of \( P_{m+1} \).

The solution is as follows. Every person \( P_i \) for all \( i \in [1, n] \) queries about the identity of a specific person \( P_j \). As more than half of the people are truth-tellers, the majority answer reveals the identity of person \( P_j \). Using this method for all \( P_j \), we need \( n \cdot n = n^2 \) queries in total to reveal all identities.

\[ Q_T = Q_{all} = n^2. \]

We can reduce the number of queries using several optimizations. Two such optimizations are shown here: (i) Self-queries i.e., people querying about themselves, of the form \( q(P_i, P_i) \), are not necessary and can be removed. (ii) When all people give their opinions about the identity of a particular person, those who hold minority views are always the unreliable people.

Let’s improve the solution. In the first iteration, choose a random person among \( P_1, \ldots, P_n \). Ask everyone to reveal the identity of this person. If the person is a truth-teller, then the person can reveal others’ identities. On the other hand, if the person is not a truth-teller, then in the second iteration choose another random person again among \( P_1, \ldots, P_n \), excluding the person chosen in the previous iteration, and repeat the process.

**Pairing solution (non-optimal)**

The core idea of this solution is the following observation. From a set of people with a strict majority of truth-tellers, if a pair consisting of at least one unreliable person is removed, then the strict majority property is maintained in the remaining set.

In the first iteration, we group people in pairs. Each person is asked to reveal the identity of the other person in the pair. Each pair result is represented in the form \( [q(P_i, P_j), q(P_j, P_i)] \). From Figure 156, the only four possible results in a pair can be \([TT], [TU], [UT], [UU]\). From the pairing results, the pair identities in the form \( \langle P_i, P_j \rangle \) can be derived as shown in Table 57.

If the pair result is \([TU]\) or \([UT]\), then at least one person in the pair is an unreliable person and that pair can be removed from the set to maintain the majority of truth-tellers in the remaining set. In other words, all pairs whose results are either \([TU]\) or \([UT]\) can be removed retaining the majority (of truth-tellers) property.
<table>
<thead>
<tr>
<th>Pair result</th>
<th>Pair identity</th>
</tr>
</thead>
<tbody>
<tr>
<td>([TT])</td>
<td>(\langle T, T \rangle) or (\langle U, U \rangle)</td>
</tr>
<tr>
<td>([TU])</td>
<td>(\langle U, U \rangle) or (\langle U, T \rangle)</td>
</tr>
<tr>
<td>([UT])</td>
<td>(\langle U, U \rangle) or (\langle T, U \rangle)</td>
</tr>
<tr>
<td>([UU])</td>
<td>(\langle U, U \rangle) or (\langle U, T \rangle) or (\langle T, U \rangle)</td>
</tr>
</tbody>
</table>

Table 57: Pair identities that can be derived from the pair results.

If the number of people is odd, then there will be one person who is unpaired. In that case, we check if the last person is a truth-teller or not (asking \(n\) queries about that person and taking the majority opinion). If the last person is a truth-teller, we can easily reveal the identities of everyone else. If the last person is an unreliable person, then we neglect the last person and consider the remaining even number of people.

We can eliminate a minimum of \(\left\lceil \frac{n}{2} \right\rceil\) people after the first iteration.

In the second iteration, the remaining people are regrouped to form pairs and the process is repeated. In the worst case, the truth-teller is found at the last level after \(\log n\) iterations. Once the truth-teller is found, others’ identities can be revealed easily.

In the worst case, the number of queries required to identify a truth-teller i.e., \(Q_T(n)\) is given by the recurrence

\[
Q_T(n) \leq \begin{cases} 
Q_T \left( \frac{n}{2} \right) + 2n & \text{if } n > 1, \\
1 & \text{if } n = 1.
\end{cases}
\]

The term \(2n\) comes from two components: (i) \(n\) queries for checking if the unpaired person is a truth-teller, and (ii) \(2\left\lfloor n/2 \right\rfloor\) queries from all pairs. Solving the recurrence, we get

\[
Q_T = 4n \text{ and } Q_{\text{all}} = 5n.
\]

**Probabilistic solution (non-optimal)**

We can reduce the number of queries using the probabilistic approach, but it has a side-effect of producing errors. Larger the number of queries, lesser the probability of error.

The approach is as follows. In the first iteration, choose a random person among \(P_1, \ldots, P_n\). Ask everyone to reveal the identity of this person. If the person is a truth-teller, then that person can reveal others’ identities. On the other hand, if the person is not a truth-teller then in the second iteration choose a random person among \(P_1, \ldots, P_n\), excluding the already chosen person, and repeat the process. The probability of error (not choosing a truth-teller) after several iterations is computed below. Let \(c > 1\) be a constant.

Probability of error after first iteration

\[
\frac{n - t}{n} < \frac{1}{2}
\]

Probability of error after \(k\) iterations

\[
\left(\frac{n - t}{n}\right)^k < \left(\frac{1}{2}\right)^k
\]

316
Probability of error after $k = c \log n$ iterations $\leq \left(\frac{n-t}{n}\right)^{c \log n} < \left(\frac{1}{2}\right)^{c \log n} = \frac{1}{n^c}$

$Q_T = c \log n$, where $c > 1$, with error probability less than $\frac{1}{n^c}$.

**Gorlin’s chaining solution (non-optimal)**

The core idea in this method is to form a chain of people giving $T$ answers successively. In such a chain, if truth tellers exist then they all should be together at the end of the chain and the last person must be a truth teller. We do not have this nice property when we construct a chain of $U$ answers successively.

Initially, we number the people $P_1, P_2, \ldots, P_n$. We ask $P_1$ about $P_2$. If the answer is $T$, we keep them in the chain and ask $P_2$ about $P_3$. The process is continued until the answer is $U$. Suppose $P_i$ answers $U$ when asked about $P_j$. This means that at least one of $P_i$ and $P_j$ is unreliable (see Figure 156). Then we remove both $P_i$ and $P_j$ from the chain as they do not affect the majority of the truth tellers in the remaining people. We continue the process by querying the predecessor of $P_i$ about the successor of $P_j$. We stop when all persons have been considered. Then we get a chain of $T$ answers as shown below.

![Diagram of Gorlin's chaining solution]

We remove a pair of people if one of the two persons involved in a query is unreliable. Hence, the majority property is maintained in the chain and at least one of the people in the chain is a truth teller. It is easy to verify that if there is a truth teller in the chain then the identities in the chain must be of the form: $U^*T^*$ i.e., a set of 0 or more unreliable people followed by one or more truth tellers. So, the last person in the chain must be a truth teller. We can identify a truth teller using CHAINING ALGORITHM. Except the first person in the chain, every other person’s identity would be queried once. Hence, the total number of queries asked to identify a truth teller will be $n - 1$.

Once we know a truth teller, we can ask $n - 1$ questions to him to reveal the identities of everyone else.

$Q_T = n - 1$ and $Q_{all} = 2n - 2$.

**Aigner’s bucket solution (optimal $Q_T$)**

The core idea of this method is similar to the previous one i.e., linking all people who give a series of $T$ answers consecutively but we do it a bit more efficiently.

We construct a data structure consisting of buckets $B_1, B_2, \ldots, B_k$ and a dump $D$. Each bucket $B_i$ consists of $2^a$ number of people in hypercube-like tree (instead of chains) with all $T$ answers and a single sink $S_i$ as shown in Figure 157. The number of queries used to construct a bucket $B_i$ is $2^a - 1$ because a tree data structure with $t$ nodes has $t - 1$ edges.
### CHAINING ALGORITHM($P_1, \ldots, P_n$)

**Input:** $n$ people $P_1, \ldots, P_n$.

**Output:** Find a truth teller.

1. $i \leftarrow 1; \ j \leftarrow 2; \ 1$.prev $\leftarrow -1$
2. **while** $j \leq n$ **do**
3. 
4. 
5. **if** $qresult = T$ **then**
6. 
7. **else**
8. 
9. **if** $i$.prev $= -1$ **then**
10. 
11. **else**
12. 
13. **return** $P_i$

Initially each bucket $B_i$ contains person $P_i$ for $i \in [1, n]$ and dump $D$ is empty. We pick two buckets $B_i$ and $B_j$ such that $2^{a_i} = 2^{a_j}$ and ask a query to $S_i$ about the identity of $S_j$.

### BUCKET ALGORITHM($P_1, \ldots, P_n$)

**Input:** $n$ people $P_1, \ldots, P_n$.

**Output:** Identify a truth teller.

1. $B_i \leftarrow P_i$ for $i \in [1, n]$
2. **while** buckets $B_i$ and $B_j$ have the same size **do**
3. 
4. **if** $qresult = T$ **then**
5. 
6. **else**
7. 
8. **if** $i$.prev $= -1$ **then**
9. 
10. **else**
11. 
12. **return** $S_\ell$

If the query result is $T$ then we add a directed edge from $S_i$ to $S_j$, the entire new $(2^{a_i} + 1)$-sized tree is moved to $B_j$, and $B_i$ is deleted. The sink $S_j$ will be the sink of $B_j$. On the other hand, if the query result is $U$, then at least one of $S_i$ or $S_j$ is an unreliable person. If $S_i$ (resp. $S_j$) is $U$, then all people in $B_i$ (resp. $B_j$) are $U$'s. Therefore, we throw both $B_i$ and $B_j$ into dump $D$. We continue the process until no two buckets are of the
same size.

At this point, the bucket sizes will be distinct $2^{b_1} < 2^{b_2} < \cdots < 2^{b_\ell}$ i.e., $2^{b_1} + \cdots + 2^{b_{\ell-1}}$. Also, $S_\ell$ (sink of $B_\ell$) must be a truth teller. If the person is not a truth teller then the number of unreliable people will exceed the number of truth tellers. The strategy is given in BUCKETALGORITHM. Once we know a truth teller, others’ identities can be easily found.

We need to compute $Q_T$. When we throw two buckets of equal size $2^{c_i-1}$ into dump $D$, we would have asked $2(2^{c_i-1} - 1) + 1 = 2^{c_i} - 1$ questions. Let $b(n)$ be the number of 1-bits in the binary representation of number $n$. Then, the total number of questions would be

$$Q_T \leq \sum_{i=1}^{\ell} (2^{b_i} - 1) + \sum_{i=1}^{m} (2^{c_i} - 1)$$

$$= \left( \sum_{i=1}^{\ell} 2^{b_i} + \sum_{i=1}^{m} 2^{c_i} \right) - (\ell + m)$$

$$= n - (\ell + m) \quad \left( \because n = \sum_{i=1}^{\ell} 2^{b_i} + \sum_{i=1}^{m} 2^{c_i} \right)$$

$$\leq n - b(n) \quad (\because \ell + m \geq b(n))$$

$Q_T = n - b(n)$ and $Q_{all} = 2n - b(n) - 1$, where $b(n)$ is the number of 1-bits in the binary representation of $n$.

Blecher-Shlosman’s reduction solution (optimal $Q_{all}$)

We explain the solution when $n = 2k + 1$ i.e., odd. It is straightforward to extend the solution when $n$ is even. We use mathematical induction to prove that the minimum number of questions asked in this strategy when $n = 2k + 1$ is $3k$.

We ask queries to $P_2, P_3, \ldots, P_n$ one-by-one about the identity of $P_1$. We stop the process if any of the following two events occur:

[Truth teller found.] Suppose $k$ persons answer $T$. Then the person $P_1$ must be a truth teller. This is because if $P_1$ is an unreliable person, then all the $k$ persons will be unreliable which cannot happen. It is easy to see that the people who said
are unreliable. Once we know that \( P_1 \) is a truth teller, we ask \( P_1 \) to reveal the identities of all the other persons. The minimum number of questions required to reveal everyone’s identities is clearly \( 3k \).

**Problem reduction.** Suppose the number of persons who answer \( U \) is more than the number of persons who answer \( T \). Let the number of people who answer \( U \) be \( m \), where \( m \geq 1 \). Then the number of people who answer \( T \) will be \( m - 1 \).

If \( P_1 \) is unreliable, then the \( m - 1 \) persons who said \( P_1 \) is \( T \) are unreliable. On the other hand, if \( P_1 \) is a truth teller, then the \( m \) persons who said \( P_1 \) is \( U \) are unreliable. Therefore, at least half of this \( 2m \) group of persons are unreliable. The number of questions asked in this group is \( m + m - 1 = 2m - 1 \). Let’s call this group Group \( A \).

We will call the remaining group of \( (2k + 1) - (2m) = 2(k - m) + 1 \) persons Group \( B \). It is easy to see that Group \( B \) must contain a majority of truth tellers. We ask \( 3(k - m) \) queries to reveal everyone’s identities in Group \( B \) using induction. We then select a truth teller from this group to reveal the identities of \( P_1 \) and all persons of Group \( A \) who revealed the true identity of \( P_1 \).

We need to show that if this event occurs, then the minimum number of questions required to reveal everyone’s identities is \( 3k \).

\[
\text{Max #questions} = \#\text{Questions to Group } B + \#\text{Questions to Group } A \\
= (3(k - m)) + ((2m - 1) + (1 + m)) = 3k
\]

**Related problems**

**Heaven or hell.** A person dies and finds himself standing in front of two doors: one that leads to heaven and another that leads to hell. Two guards stand in front of the gates. One is a truth teller and the other is a liar. They answer either yes or no. The person does not know which door leads to what and also who among the two guards is a truth teller. He must ask one and only one question to any one of the guards to determine which door leads to heaven. The person after getting the reply for his brilliant question finds his way to heaven. What was the question of the person?

**Solution.** If two questions are allowed, then we can find the way to heaven easily. The first question can be used to identify the identities of the two guards, e.g.: the question “Is \( 1 + 1 = 2 \)” can be asked to any guard. Once we find the identities from the first question, we can know the door to be taken for heaven from the second question, e.g.: pointing to a door the person can ask “Does this door lead to heaven?”

If only one question is allowed, then what is the question? The fundamental properties every answer statement, which in fact is a question itself, must satisfy are: (i) The question must be a compound statement made up of possibly two simple statements. (ii) The truth teller and the liar must provide the same answer to the question.
The question can be any of the following:
- If I were to ask you if this door leads to heaven, what would you say?
- If I were to ask the other guard if this door leads to heaven, what would she/he say?
- Of the two statements: you are a liar, and this door leads to the heaven, is one and only one of them true?

The analysis of the first statement is as follows. If the dead person points to the door that leads to heaven and asks the question “If I were to ask you if this door leads to heaven, what would you say?”, then regardless of whether the guard is a truth teller or not, the guard is forced to say yes. On the other hand, if the person points to the door that leads to hell, then regardless of whether the guard is a truth teller or not, the guard is forced to say no. If the guard’s answer is yes, then the person can go through the door being sure that it leads to heaven. If the guard’s answer is no, then he can go through the other door being sure that it leads to heaven. The analysis is summarized in Table 58. A similar argument can be made to the other two statements as well.

<table>
<thead>
<tr>
<th>Pointed door</th>
<th>Truth teller</th>
<th>Liar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leads to heaven</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Leads to hell</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 58: Answers of the truth teller and the liar for the question “If I were to ask you if this door leads to heaven, what would you say?”

Problems
1. [Generalization.] Assume we have a set of $n$ people of $t$ truth tellers and $n - t$ unreliable persons. How can we: (i) find a truth teller, and (ii) reveal the identities of all $n$ persons asking minimum number of questions if (a) $t$ is known, and (b) $t$ is unknown?
2. [Heaven or hell variants.] (i) The person knows that ha and na are the words for yes and no in the native language of guards, but the person doesn’t know which word means what. How can the person find the way to heaven? (ii) Suppose that there are three guards instead of two: one is a truth-teller, another a liar, and yet another tells truth or lies arbitrarily. What is the minimum number of questions the person should ask to find the way to heaven?

References
The chaining solution was proposed by Andrey Gorlin (2015). The bucket solution for optimal $Q_T$ is in [Aigner, 2004]. The reduction solution was developed by S. Shlosman (published in [Blecher, 1983]) and [Wildon, 2010], independently. An excellent literature review of the problem and its variants is given in [Cheng et al., 2013].
How To Write A Pseudocode?

Algorithms are expressed in three major ways: (i) natural language explanation, (ii) visual representations, and (iii) pseudocodes. Natural language explanation is so high-level that it might be extremely difficult to transform the description into a computer program and analyze the algorithm’s complexity. In contrast, a computer program is so low-level that it is extremely difficult to understand the high-level idea of the algorithm and the logical parts of the algorithm. Pseudocode is a description (its visual counterpart is called a flowchart) that consists of step-by-step instructions in a language that is a hybrid of natural language phrases and program structures that are programming language neutral. The aim of a pseudocode is to make it easy to understand the high-level idea (or logical parts) of the algorithm via comments and also make it easy to transform the logic to any general-purpose programming language (e.g. C, C++, Java, Python, etc).

HOWTOWRITEAPSEUDOCODE gives ideas on how to write a pseudocode. If we want to make use of data structures in algorithms, we can use them as shown in HOWTOUSEDATASTRUCTURES-1 and HOWTOUSEDATASTRUCTURES-2.
HowToWriteAPseudocode(variable, array[1...n], matrix[1...m, 1...n])

**Input:** Description of input parameters.

**Output:** Description of output.

1. // Assignment: Statements are separated by a semicolon ............
2. \[a \leftarrow 1; a \leftarrow \left(\frac{\lceil d \rceil + \lfloor e \rfloor}{\sqrt{n} + \log_3 k} + a^b \mod c\right)\]
3. // Assignment: Simple variable names must be all small letters ....
4. currenpathcost \leftarrow 0; optimalsolutionslist \leftarrow [ ]
5. // Assignment: \[A[i] \leftarrow 0, B[i] \leftarrow i, and C[i] \leftarrow B[i] \text{ for all } i \in [1,n] \] ....
6. \[A[1...n] \leftarrow [0...0]; B[1...n] \leftarrow [1...n]; C \leftarrow B\]
7. // Array operations (includes And, Or, BitwiseAnd, BitwiseOr etc) ..
8. \[a \leftarrow \text{Sum}(A[1...n]) + \text{Max}(A[1...n]) + \text{Min}(A[1...n])\]
9. // Conditional: If-else ladder ........................................
10. if \[a > 0\] then
11. \[a\text{ is positive}\]
12. else if \[a = 0\] then
13. \[a\text{ is neither positive nor negative}\]
14. \[b \leftarrow 1\]
15. else if \[a < 0\] then
16. \[a\text{ is negative}\]
17. // Conditional: Compact if-else ladder ................................
18. if \[a > 0\] then \[a\text{ is positive}\]
19. else if \[a = 0\] then \{ \[a\text{ is neither positive nor negative}; b \leftarrow 1\}\}
20. else if \[a < 0\] then \[a\text{ is negative}\]
21. // Loop: \[i\text{ takes values } a,a+k,a+2k, \text{ so on such that } i \leq n \] ....
22. for \(i \leftarrow a\) to \(n\) increment \(k\) do
23. \[\text{print } A[i]\]
24. // Loop: \[i\text{ takes values } a,a+1,a+2, \text{ so on such that } i \leq n \] ....
25. for \(i \leftarrow a\) to \(n\) do
26. \[\text{print } A[i]\]
27. // Loop: \[i\text{ takes values } n,n-k,n-2k, \text{ so on such that } i \geq a \] ....
28. for \(i \leftarrow n\) downto \(a\) decrement \(k\) do \text{ print } A[i]
29. // Loop: \[i\text{ takes values } n,n-1,n-2, \text{ so on such that } i \geq a \] ....
30. for \(i \leftarrow n\) downto \(a\) do \text{ print } A[i]
31. // Loop: \[i\text{ takes values } n,n-k,n-2k, \text{ so on such that } i \geq a \] ....
32. \(i \leftarrow n\) while \(i \geq a\) do \[i \leftarrow i - k\]
33. // Function invocation ...................................................
34. \(output \leftarrow \text{ALGORITHM2}(a,A[1...n])\)
35. \text{return } output

\[\text{ALGORITHM2}(a,A[1...n])\]

1. \text{return } A[a]
HowToUseData Structures 1()

1. // Static array

2. Create an array A[1...n]

3. A[i] ← a // set ith value of array. time: O(1)

4. a ← A[i] // get ith value of array. time: O(1)

5. // Dynamic array (or array list or vector)

6. Create a dynamic array A ← []

7. A.Add(a) // add element at last. time: Θ(1)

8. a ← A.First() // return the first element. time: Θ(1)

9. a ← A.Last() // return the last element. time: Θ(1)

10. // 2-D matrix

11. Create a 2-D matrix M[1...m, 1...n]

12. M[i, j] ← a // set (i, j)th value of matrix. time: Θ(1)

13. a ← M[i, j] // get (i, j)th value of matrix. time: Θ(1)

14. // Singly linked list (SLL)

15. Create a SinglyLinkedList L

16. a ← L.First() // return the first element. time: Θ(1)

17. a ← L.Last() // return the last element. time: Θ(1)

18. L.AddFirst(a) // add element at first. time: Θ(1)

19. L.AddLast(a) // add element at last. time: Θ(1)

20. a ← L.RemoveFirst() // remove element at first. time: Θ(1)

21. // Circularly singly linked list (CSLL)

22. Create a CircularlySinglyLinkedList L

23. a ← L.First() // return the first element. time: Θ(1)

24. a ← L.Last() // return the last element. time: Θ(1)

25. L.AddFirst(a) // add element at first. time: Θ(1)

26. L.AddLast(a) // add element at last. time: Θ(1)

27. a ← L.RemoveFirst() // remove element at first. time: Θ(1)

28. L.Rotate() // move first element to last. time: Θ(1)

29. // Doubly linked list (DLL)

30. Create a DoublyLinkedList L

31. a ← L.First() // return the first element. time: O(1)

32. a ← L.Last() // return the last element. time: O(1)

33. L.AddFirst(a) // add element at first. time: O(1)

34. L.AddLast(a) // add element at last. time: O(1)

35. a ← L.RemoveFirst() // remove element at first. time: Θ(1)

36. a ← L.RemoveLast() // remove element at last. time: Θ(1)

37. // Stack (implemented using dynamic array or SLL)

38. Create a stack S

39. S.Push(a) // add element at top. time: Θ(1)

40. a ← S.Pop() // remove element at top. time: Θ(1)

41. a ← S.Top() // return element at top. time: Θ(1)
# How To Use Data Structures-2()

42. // Queue (implemented using a circular dynamic array or SLL) .......
43. Create a queue Q
44. Q.Enqueue(a)    // add element at last. time:Θ(1)
45. a ← Q.Dequeue() // remove element at first. time:Θ(1)
46. a ← Q.Top()    // return element at top. time:Θ(1)
47. // Double-ended queue (deque) ........................................
48. Create a deque D
49. D.AddFirst(a)    // add element at first. time:Θ(1)
50. D.AddLast(a)     // add element at last. time:Θ(1)
51. a ← D.RemoveFirst() // remove element at first. time:Θ(1)
52. a ← D.RemoveLast() // remove element at last. time:Θ(1)
53. // Balanced search tree with size n ..............................
54. Create a BalancedSearchTree T
55. T.Add(a)    // add element. time:Θ(log n)
56. T.Remove(a) // remove element. time:Θ(log n)
57. a ← T.Search(a) // check if element exists. time:Θ(log n)
58. T.InOrder(a) // inorder traversal. time:Θ(n)
59. T.LevelOrder(a) // levelorder traversal. time:Θ(n)
60. // Hash map (assuming perfect hash function) ......................
61. Create a hash map H to store (key, value) pairs
62. H.GetValue(k)  // return the value associated with key k. time:Θ(1)*
63. H.Add((k, v))  // add a pair. time:Θ(1)*
64. (k, v) ← H.Remove(k) // return the pair with key k. time:Θ(1)*
65. a ← H.Search(k) // search for pair with key k. time:Θ(1)*
66. // Minimum-heap with size n ........................................
67. Create a min-heap H to store (key, value) pairs
68. H.Add((k, v))  // add a pair. time:Θ(log n)
69. (k, v) ← H.Min() // return pair with min key. time:Θ(1)
70. (k, v) ← H.RemoveMin() // remove pair with min key. time:Θ(log n)
71. // Graph (implemented using adjacency-map or adjacency-matrix) ......
72. Create a graph G
73. G.GetEdge(u, v) // get an edge.
74. a ← G.OutDegree(v) // return the outdegree of a vertex.
75. a ← G.InDegree(v) // return the indegree of a vertex.
76. L ← G.OutgoingEdges(v) // return all outgoing edges from a vertex.
77. L ← G.IncomingEdges(v) // return all incoming edges to a vertex.
78. G.AddVertex(x) // add a vertex.
79. G.RemoveVertex(v) // remove a vertex.
80. G.AddEdge(u, v, x) // add an edge.
81. G.RemoveEdge(e) // remove an edge.
Linear Diophantine Equation

Consider the following equation

\[ a_1x_1 + a_2x_2 + \cdots + a_nx_n = c \]  \hspace{1cm} (56)

where \(a_1, \ldots, a_n\) and \(c\) are integers. This equation is called a *linear Diophantine equation* in \(n\) unknowns. We are interested in finding all integer solutions to this equation. In this section, we present an algorithm to generate all solutions for solving such a linear Diophantine equation.

To simplify exposition, we will use shorthand notations. For two arrays \(U = [u_1, u_2, \ldots, u_n]\) and \(V = [v_1, v_2, \ldots, v_n]\) and a constant \(w\), we can define operations \(U + V\), \(U - V\), \(U \circ V\), \(U \cdot V\), and \(wU\) as follows:

- **Addition.** \(U + V = [u_1 + v_1, u_2 + v_2, \ldots, u_n + v_n]\)
- **Subtraction.** \(U - V = [u_1 - v_1, u_2 - v_2, \ldots, u_n - v_n]\).
- **Element-wise product.** \(U \circ V = [u_1v_1, u_2v_2, \ldots, u_nv_n]\)
- **Dot product.** \(U \cdot V = [u_1v_1 + u_2v_2 + \cdots + u_nv_n]\)
- **Scalar product.** \(wU = Uw = [wu_1, wu_2, \ldots, wu_n]\).

Suppose \(A = [a_1, a_2, \ldots, a_n]\) and \(X = [x_1, x_2, \ldots, x_n]\), then we want to find integer solutions of \(A \cdot X = c\).

We now explain an efficient algorithm to solve \(A \cdot X = c\) (i.e., Equation 56) in three stages.

**Stage 1. Check solvability**

Checking if the given Diophantine equation is solvable or not helps in saving time before actually solving the equation. Equation 56 is solvable if and only if \(c\) is a multiple of the greatest common divisor of \(a_1, \ldots, a_n\). If the equation is solvable, there are an infinite number of integer solutions. In other words,

Equation 56 is solvable if and only if \(c \mod d = 0\), where \(d = \text{GCD}(a_1, a_2, \ldots, a_n)\).

**Stage 2. Find a solution**

We find a solution \(X^* = [x_1^*, x_2^*, \ldots, x_n^*]\) to Equation 56.

**[Step 1. Initialize]** Set \(m = n\). Initialize \(B = [b_1, b_2, \ldots, b_n]\) array with the \(A\) array. Let \(X^{(b)}\) for \(i \in [1, n]\) be an \(n\)-sized array with the \(i\)th value set to 1 and the remaining values set to 0. Here, \(X^{(c)}\) denotes a solution of Equation 56 with \(c = z\), in other words, \(B \cdot X^{(c)} = z\).

**[Step 2. Find quotients and the last remainder]** Find the quotient array \(Q = [q_1, q_2, \ldots, q_{m-1}]\) and the last remainder \(b_{\text{rem}}\) that satisfies the equation:

\[ b_1 = [q_1, q_2, \ldots, q_{m-1}] \cdot [b_2, b_3, \ldots, b_m] + b_{\text{rem}} \]
such that
\[ b_1 = q_1 b_2 + r_1, \text{ where } r_1 \in [0, b_2) \]
\[ r_1 = q_2 b_3 + r_2, \text{ where } r_2 \in [0, b_3) \]
\[ \vdots \]
\[ r_{m-2} = q_{m-1} b_m + r_{m-1}, \text{ where } r_{m-1} \in (0, b_m) \]

In other words,
\[ q_i = \begin{cases} 
\lfloor b_i \div b_{i+1} \rfloor & \text{if } i = 1, \\
\lfloor r_{i-1} \div b_{i+1} \rfloor & \text{if } i \in [2, m-1] 
\end{cases} \]
\[ r_i = \begin{cases} 
q_{i+1} \mod b_i & \text{if } i = 1, \\
r_{i-1} \mod b_i & \text{if } i \in [2, m-1] 
\end{cases} \]

where,
\[ b_{\text{rem}} = \begin{cases} 
q_{m-1} & \text{if } m \geq 2, \\
q_1 & \text{if } m = 1. 
\end{cases} \]
is called the last remainder.

**[Step 3. Update solution \( X \) that corresponds to the last remainder]** Find a solution \( X^{(b_{\text{rem}})} \) for solving the equation \( A \cdot X = b_{\text{rem}} \) as follows:
\[ X^{(b_{\text{rem}})} = X^{(b_1)} - (q_1 X^{(b_2)} + q_2 X^{(b_3)} + \cdots + q_{m-1} X^{(b_{m-1})}) \]

Eliminate \( b_1 \) from the \( b \) array.
If \( b_{\text{rem}} \neq 0 \), then append \( b_{\text{rem}} \) at the end of the \( b \) array and re-index the remaining values of the \( b \) array as \([b_1, b_2, \ldots, b_m]\). Repeat Step 2.
If \( b_{\text{rem}} = 0 \), then \( X^{(b_{\text{rem}})} \) is one of the \( n - 1 \) generating vectors. We set \( m = m - 1 \) and then re-index the remaining values of the \( b \) array as \([b_1, b_2, \ldots, b_m]\).
If \( m > 1 \), go to Step 2.

**[Step 4. Find a solution]** At this point we will have \( n - 1 \) generating vectors \( X_{G_1}, X_{G_2}, \ldots, X_{G_{n-1}} \).

A solution to Equation 56 can be computed as 
\[ X^* = [x_1^*, x_2^*, \ldots, x_n^*] = \left( \frac{c}{b_{\text{rem}}} \right) X^{(b_{\text{rem}})} \]

It is important to note that \( b_{\text{rem}} \) is equal to \( \text{GCD}(a_1, a_2, \ldots, a_n) \). Hence, this algorithm can also be used as an algorithm to compute the greatest common divisor of \( n \) natural numbers.

**Stage 3. Find a generic solution**

If Equation 56 is solvable (from Stage 0), then its infinite solutions can be found using a solution \( X^* \) and the \( n - 1 \) generating vectors (from Stage 1) as shown below
\[ X = X^* + k_1 X_{G_1} + k_2 X_{G_2} + \cdots + k_{n-1} X_{G_{n-1}} \]

where \( k_1, k_2, \ldots, k_{n-1} \) are any integers.

**LINEARDIOPHANTINEEQUATIONSolver** gives the pseudocode of the algorithm. For efficient computation of the method given above, we use double ended queue
**LINEARDIOPHANTINEEQUATIONSolver**(*a[1...n], c, k[1...(n−1)]*)

**Input:** All integers *a[1],...,a[n]*, *c*, and *k[1],...,k[n−1]*.

**Output:** A solution in integers to the equation *a[1]x₁ + a[2]x₂ + ... + a[n]xₙ = c* that uses values of the *k* array

1. /* Stage 2. Find a solution ........................................................ */
2. /* Step 1. Initialize ................................................................. */
3. \( m \leftarrow n \)
4. Create a double ended queue \( b[1...n] \) and initialize it with \( a[1...n] \)
5. Create a double ended queue \( x[1...n] \), where each element is an *n*-sized array
6. \( x[1] \leftarrow [1, 0, ..., 0] \), \( x[2] \leftarrow [0, 1, ..., 0] \), ..., \( x[n] \leftarrow [0, 0, ..., 1] \)
7. /* Steps 2 and 3. Find quotients, the last remainder, and update solution \( X \) that corresponds to the last remainder .................. */
8. Create a quotient array \( q[1...(m−1)] \) and the last remainder \( r \)
9. \( j \leftarrow 1; \) Create the generating vectors \( g[1...(n−1)] \)
10. **while** \( m > 1 \) **do**
11. \( i \leftarrow 1 \)
12. \( q[i] \leftarrow \left\lfloor \frac{b[i]}{b[i+1]} \right\rfloor; \) \( r \leftarrow b[i] \mod b[i+1] \)
13. **for** \( i \leftarrow 2 \) **to** \( m−1 \) **do**
14. \( q[i] \leftarrow \left\lfloor \frac{r}{b[i+1]} \right\rfloor; \) \( r \leftarrow r \mod b[i+1] \)
15. Create an array \( y[1...n] \) and initialize \( y \) with \( x[1] \)
16. **for** \( i \leftarrow 2 \) **to** \( m \) **do**
17. \( y \leftarrow y − (q[i−1] \cdot x[i]) \) // \( X^{(b_{\text{rem}})} = X^{(b)} − \sum_{\ell=2}^{m} q_{\ell−1}X^{(b_{\ell})} \)
18. Delete the first element of the \( b \) array
19. Delete the first row of \( x \)
20. **if** \( r \neq 0 \) **then**
21. \( b[m] \leftarrow r \) // last remainder will be appended to the \( b \) array
22. Insert \( y \) into \( x \) as the last row
23. **else if** \( r = 0 \) **then**
24. \( g[j] \leftarrow y \) // \( X^{(b_{\text{rem}})} \) is one of the \( n−1 \) generating vectors
25. \( j \leftarrow j + 1; \) \( m \leftarrow m − 1 \)
26. /* Step 4. Find a solution ........................................................ */
27. \( \text{solution} \leftarrow \frac{c}{b[1]} \cdot x[1] \) // \( X^* \)
28. /* Stage 3. Find a solution that uses the \( k \) array values ....... */
29. **for** \( i \leftarrow 1 \) **to** \( n−1 \) **do**
30. \( \text{solution} \leftarrow \text{solution} + (k[i] \cdot g[i]) \) // \( X = X^* + \sum_{\ell=1}^{n−1} k_{\ell}X_{G_{\ell}} \)
31. **return** \( \text{solution}[1...n] \)

(deque) for both structures \( B \) and \( X \). This is because if we use a deque that is implemented as a circular dynamic array, insertion and deletion of elements at both first and last positions can be performed in \( O(1) \) amortized time. Figure [158] shows the execution of the algorithm on an example problem.
〈Time, Space〉 complexity to compute \( X^* \) is \( \Omega(n^2), \Theta(n^2) \).

References
Finding integer solutions to a single linear Diophantine equation has been covered in books [Uspensky and Heaslet, 2002] and [Schrijver, 1998] and studied by ancient Indian mathematicians [Dutta, 2002], [Lehmer, 1919], [Weinstock, 1960], [Bond, 1967], and [Kertzner, 1981]. The algorithm described above is given in [Morito and Salkin, 1979, Morito and Salkin, 1980], which heavily depends on previous work.
One solution: \( \frac{8913}{5677} = 1(5677) + 0(4378) + 3236 \)

\( \frac{33}{10} = 2(5) + 0(3) + 0 \)

\( 10 = 2(5) + 0(3) + 0 \)

\( 3 = 1(2) + 1 \)

\( 2 = 2(1) + 0 \)

\( 1142 = 1(638) + 3(157) + 33 \)

\( 157 = 4(33) + 2(10) + 5 \)

\( 2 = 2(1) + 0 \)

\( 1299 = 1(1142) + 0(638) + 157 \)

\( 157 = 4(33) + 2(10) + 5 \)

\( b \)

\( 2 \)

\( r \)

\( q \)

\( q_1 = \left\lfloor \frac{r_1}{b_3} \right\rfloor \)

\( q_2 = \left\lfloor \frac{r_2}{b_3} \right\rfloor \)

\( \left( (\varepsilon q)_X \cdot \overline{r_2} + (\varepsilon q)_X \cdot \overline{r_1} \right) - (\varepsilon q)_X \overline{b} = (\varepsilon(\varepsilon q))_X \overline{b} \)

\( (\varepsilon(\varepsilon q))_X \frac{\overline{b}}{c} = \overline{X} \)

\( \text{Computing quotients and remainders at every iteration:} \)

\( \varepsilon q \text{ mod } \overline{r_1} = \overline{r_1} \)

\( \frac{\varepsilon q}{\overline{r_1}} = \overline{r_1} \text{ mod } \overline{r_1} \)
References


331


