CSE 373: Analysis of Algorithms

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Fall 2014

"Theory is when you know everything but nothing works.
Practice is when everything works but no one knows why.
In our lab, theory and practice are combined:
nothing works and no one knows why."

— A practical theoretician
(no one knows who)

Some Mostly Useless Information

- **Lecture Time:** TuTh 4:00 pm - 5:20 pm
- **Location:** Light Engineering Lab 102, West Campus

- **Instructor:** Rezaul A. Chowdhury
- **Office Hours:** TuTh 2:00 pm - 3:30 pm, 1421 Computer Science
- **Email:** rezaul@cs.stonybrook.edu

- **TA:** Ibrahim Hammoud
- **Office Hours:** MoWe 10:00 am - 11:30 am, 2110 Computer Science
- **Email:** firstname.lastname@stonybrook.edu

- **Class Webpage:**

  http://www.cs.sunysb.edu/~rezaul/CSE373-F14.html

Topics to be Covered

The following topics will be covered (hopefully)

- elementary data structures
- sorting and searching
- greedy algorithms
- divide-and-conquer algorithms
- dynamic programming
- graph algorithms
- randomized algorithms
- parallel algorithms and multithreaded computations
- NP-completeness and approximation algorithms

Grading Policy

- Problem solving (4 homework problem sets):
  40% (highest score 15%, lowest score 5%, and others 10% each)
- Problem design (4 themes, one per homework problem set):
  10% (each worth 2.5%)
- In-class midterm (Thursday, Oct 16, 4:00pm – 5:20pm):
  15%
- Final exam (Monday, Dec 15, 2:30pm – 5:00pm, location: TBD):
  35%

Each homework problem set and exam will include additional problems for graduate students taking the course as CSE 587.

Graduate and undergraduate students will be graded separately.

Groups and Supergroups

Groups for Problem Solving:

Each group will consist of a pair of students

- each group will submit only one copy of hand-written solutions for each homework problem set
- each group member must write down solutions for two problem sets

Supergroups for Problem Design:

Each supergroup will consist of a pair of groups (4 students)

- each supergroup will submit only one copy of hand-written problem for each theme
- each supergroup member must write down problem for one theme

Textbooks

Required

- Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein.
  Introduction to Algorithms (3rd Edition), MIT Press, 2009.

Recommended

- Sanjoy Dasgupta, Christos Papadimitriou, and Umesh Vazirani.
  Algorithms (1st Edition), McGraw-Hill, 2006.
- Jon Kleinberg and Éva Tardos.
  Algorithm Design (1st Edition), Addison Wesley, 2005.
- Steven Skiena.
  The Algorithm Design Manual (2nd Edition), Springer, 2008.

What is an Algorithm?

An algorithm is a ***well-defined computational procedure*** that solves a well-specified computational problem.

It accepts a value or set of values as ***input***, and produces a value or set of values as ***output***

What is an Algorithm?

An algorithm is a ***well-defined computational procedure*** that solves a well-specified computational problem.

It accepts a value or set of values as ***input***, and produces a value or set of values as ***output***

Example: ***mergesort*** solves the ***sorting problem*** specified as a relationship between the input and the output as follows.

Input: A sequence of n numbers $\langle a_1, a_2, \dots, a_n \rangle$.

Output: A permutation $\langle a'_1, a'_2, \dots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Desirable Properties of an Algorithm

√ Correctness

- Designing an incorrect algorithm is straight-forward

√ Efficiency

- Efficiency is easily achievable if we give up on correctness

Surprisingly, sometimes incorrect algorithms can also be useful!

- If you can control the error rate
- Tradeoff between correctness and efficiency:

  Randomized algorithms

  (Monte Carlo: always efficient but sometimes incorrect,

  Las Vegas: always correct but sometimes inefficient)

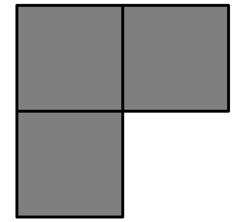
  Approximation algorithms

  (always incorrect!)

Algorithmic Puzzles

Tromino Cover

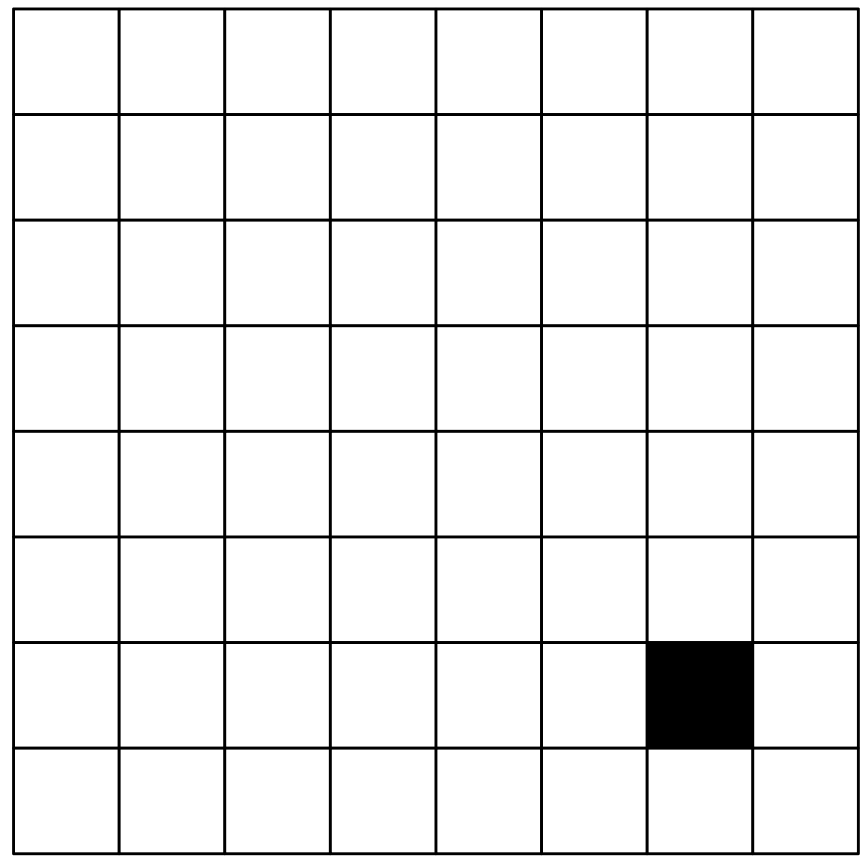
A *right tromino* is an L-shaped tile formed by three adjacent squares.



Puzzle: You are given a $2^n \times 2^n$ board with one missing square.

– you must cover all squares except the missing one exactly using right trominoes

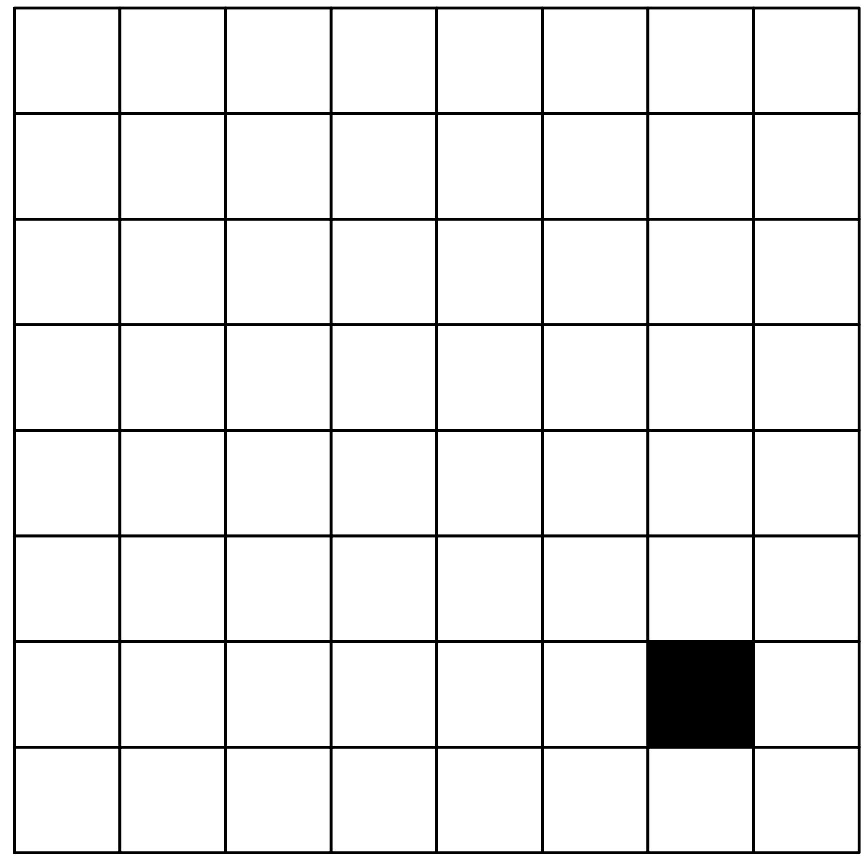
– the trominoes must not overlap



$2^3 \times 2^3$ board

Tromino Cover

Steps

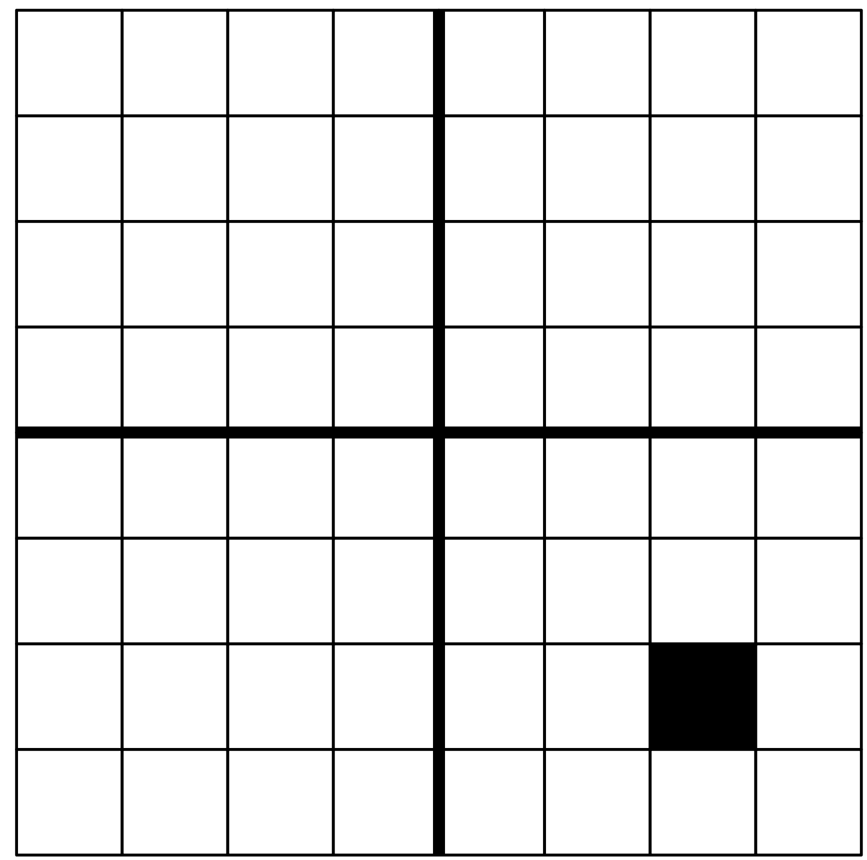


$2^3 \times 2^3$ board

Tromino Cover

Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.



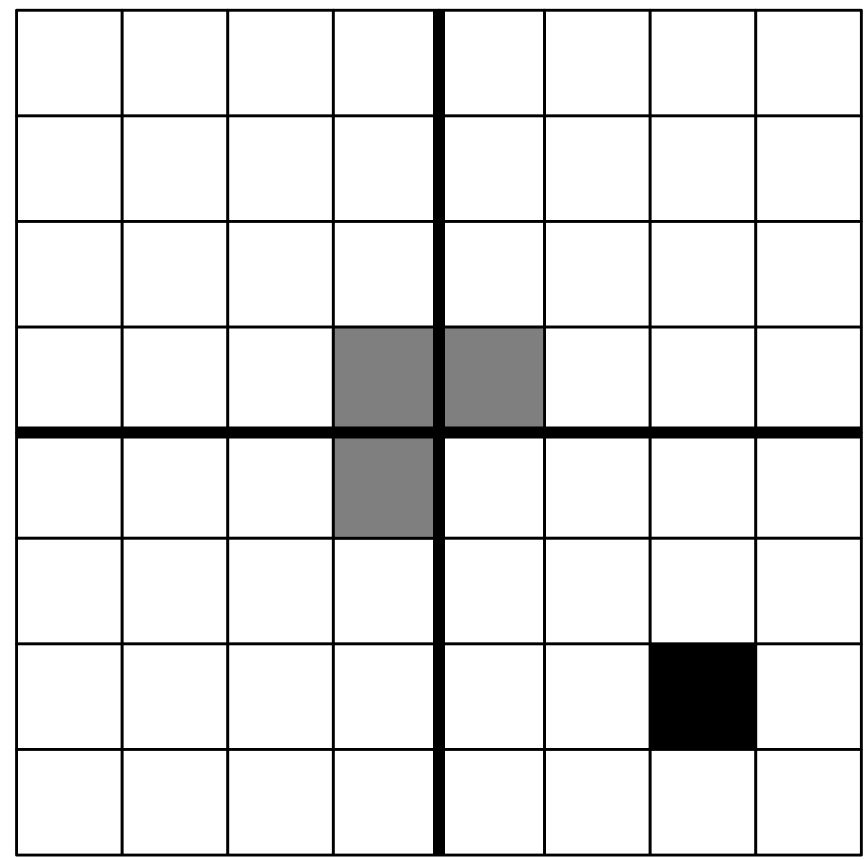
$2^3 \times 2^3$ board

Tromino Cover

Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.



$2^3 \times 2^3$ board

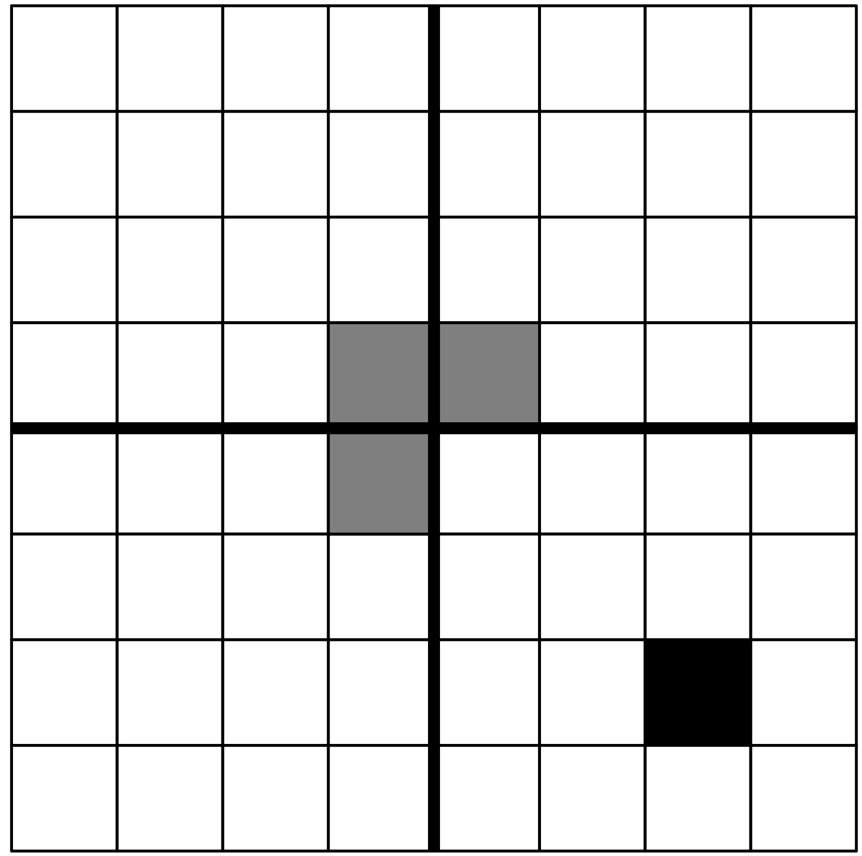
Tromino Cover

Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!



$2^3 \times 2^3$ board

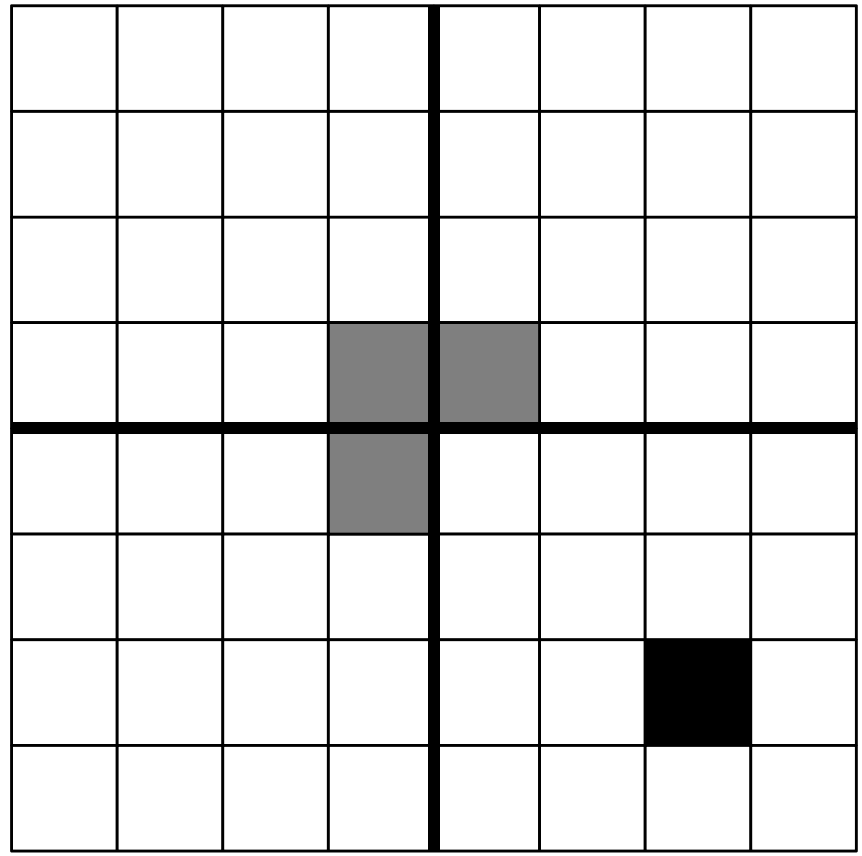
Tromino Cover

Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!



$2^3 \times 2^3$ board

– Solve each smaller subproblem recursively using the same technique.

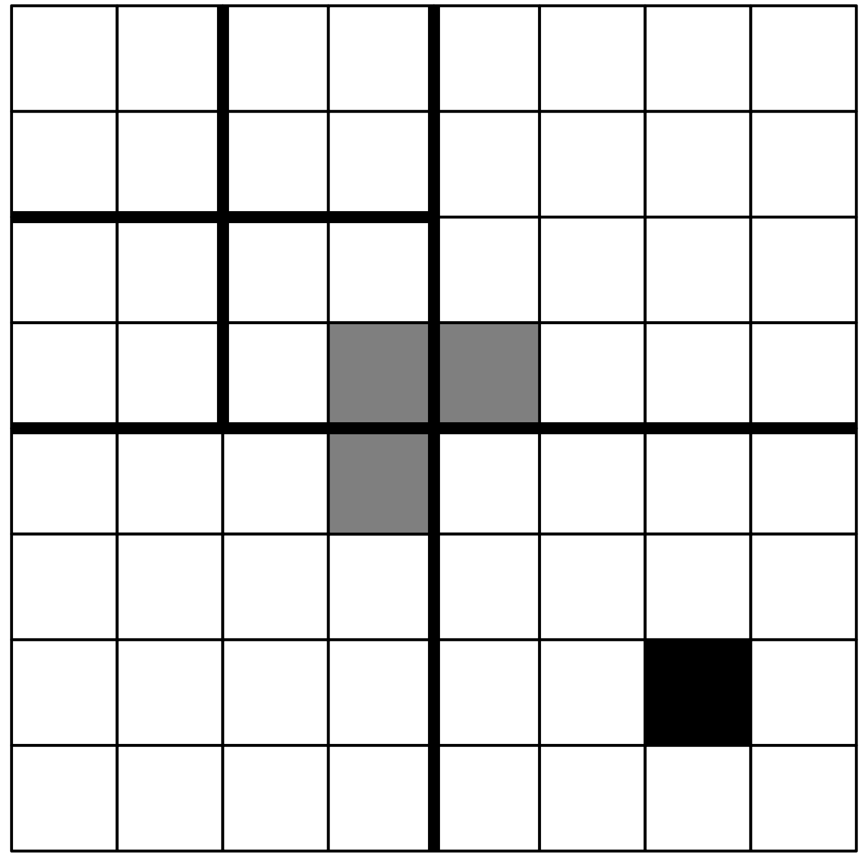
Tromino Cover

Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!



$2^3 \times 2^3$ board

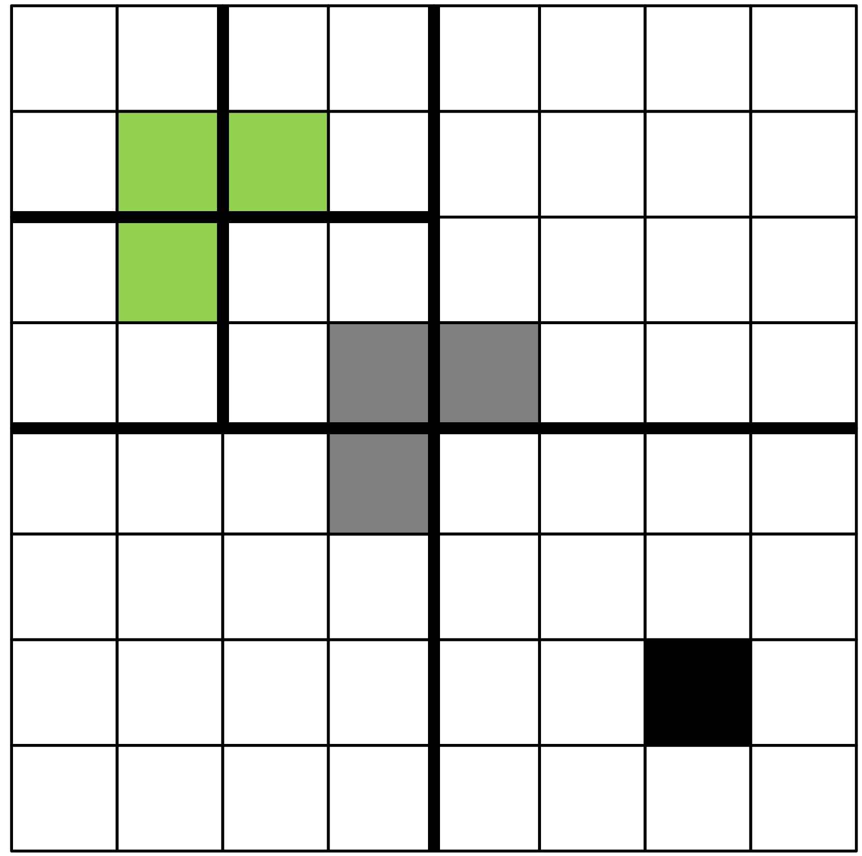
– Solve each smaller subproblem recursively using the same technique.

Tromino Cover

Steps

- Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.
- Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

  This reduces the original problem into 4 smaller instances of the same problem!
- Solve each smaller subproblem recursively using the same technique.



$2^3 \times 2^3$ board

Tromino Cover

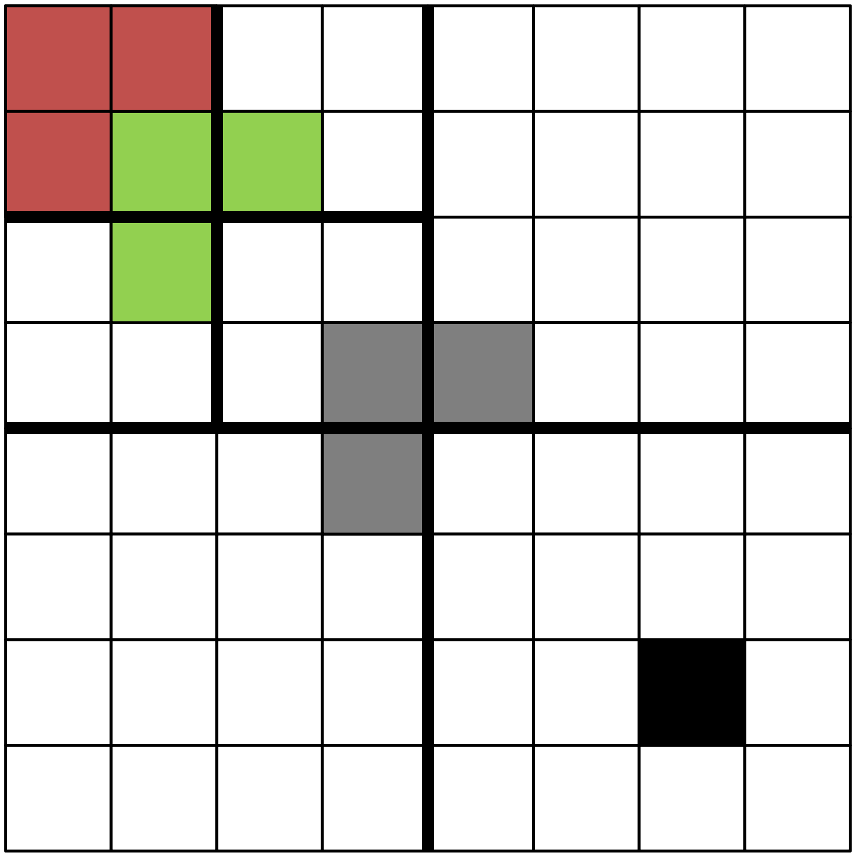
Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!

– Solve each smaller subproblem recursively using the same technique.



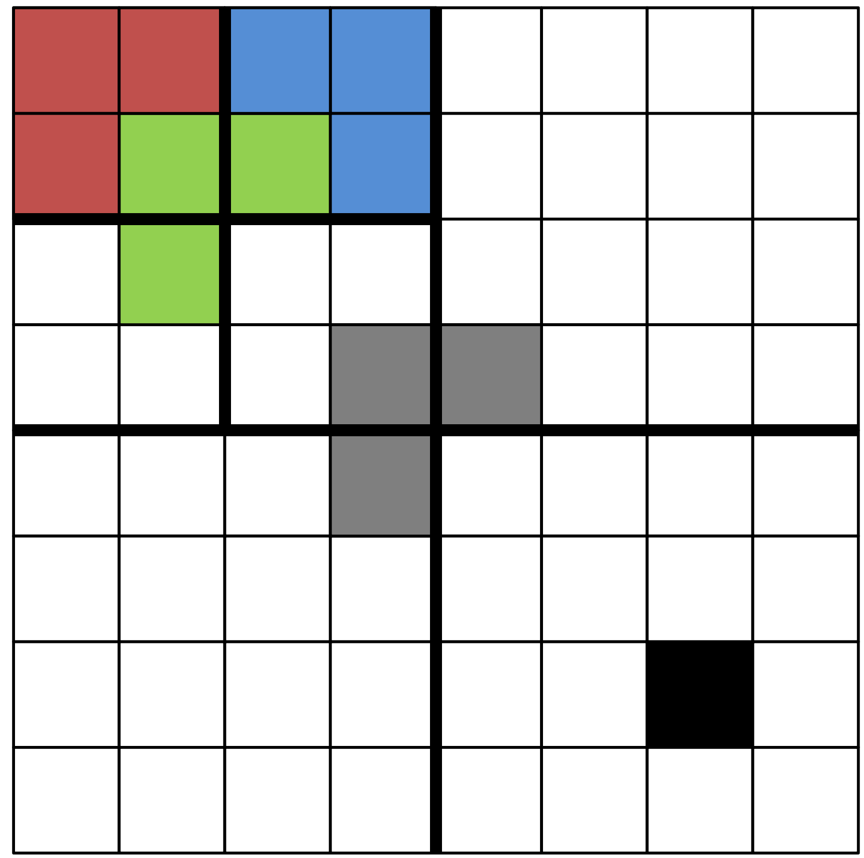
$2^3 \times 2^3$ board

Tromino Cover

Steps

- Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.
- Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

  This reduces the original problem into 4 smaller instances of the same problem!
- Solve each smaller subproblem recursively using the same technique.



$2^3 \times 2^3$ board

Tromino Cover

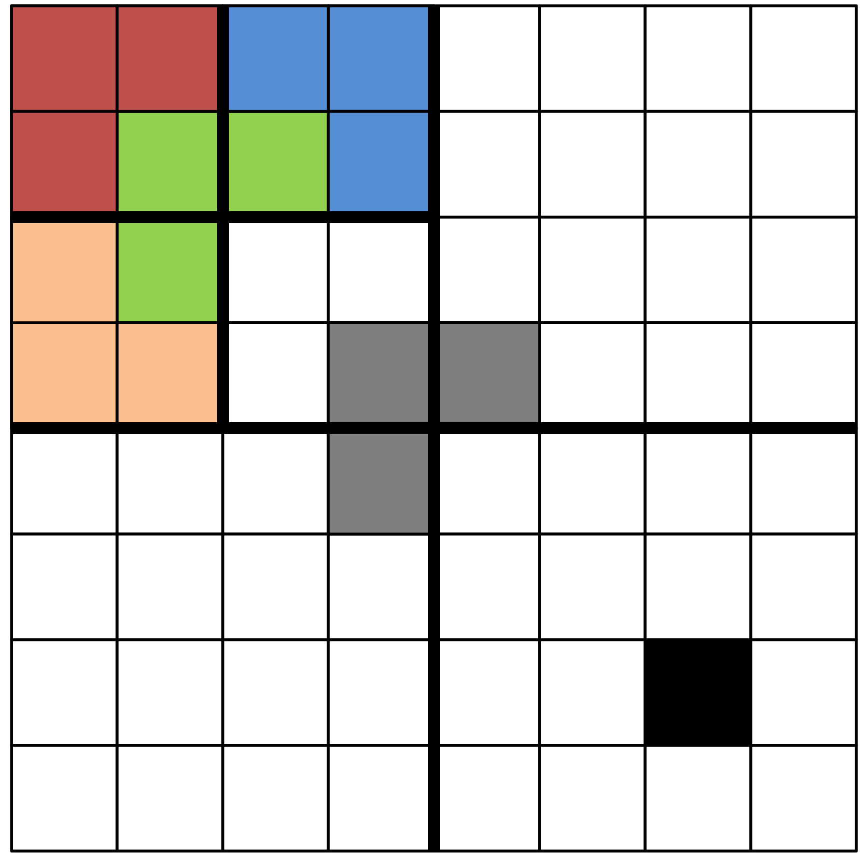
Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!

– Solve each smaller subproblem recursively using the same technique.



$2^3 \times 2^3$ board

Tromino Cover

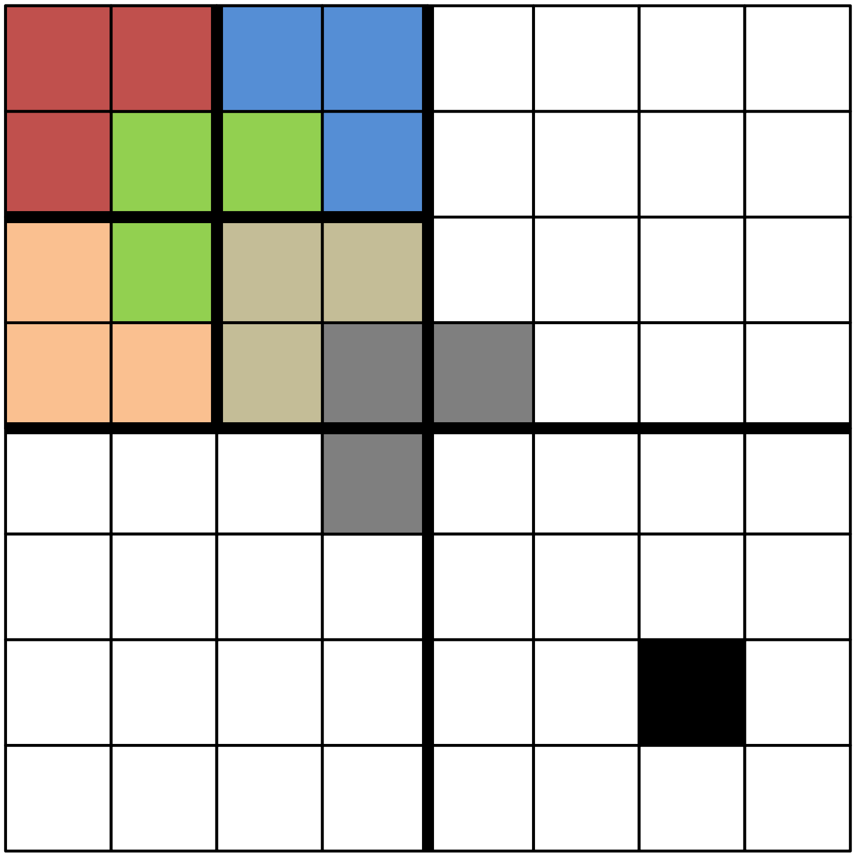
Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!

– Solve each smaller subproblem recursively using the same technique.



$2^3 \times 2^3$ board

Tromino Cover

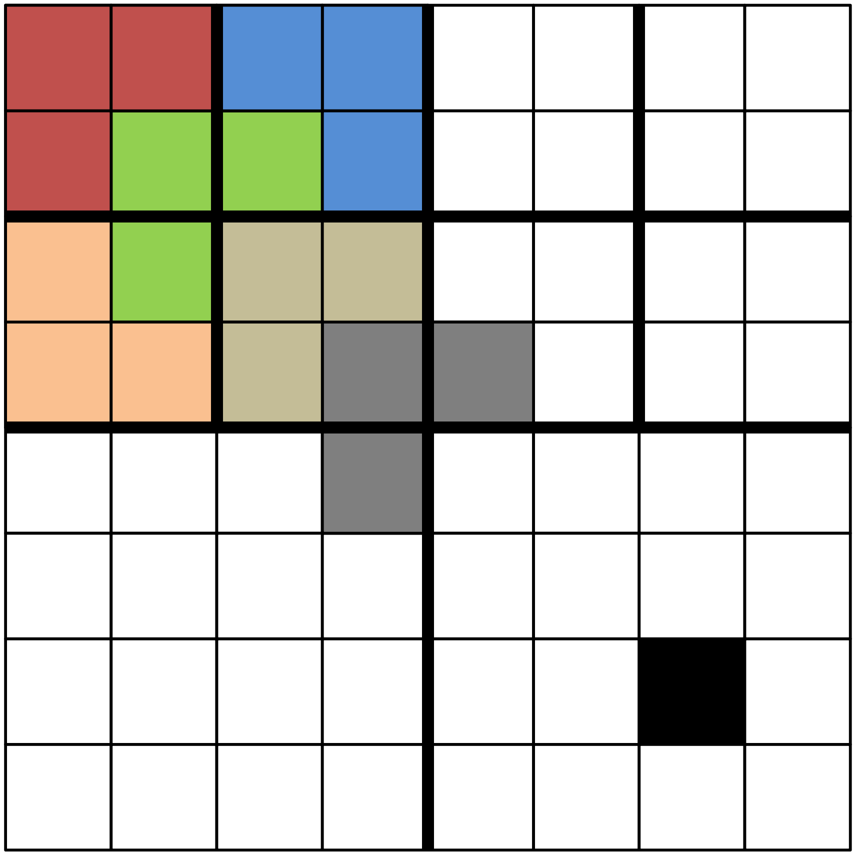
Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!

– Solve each smaller subproblem recursively using the same technique.



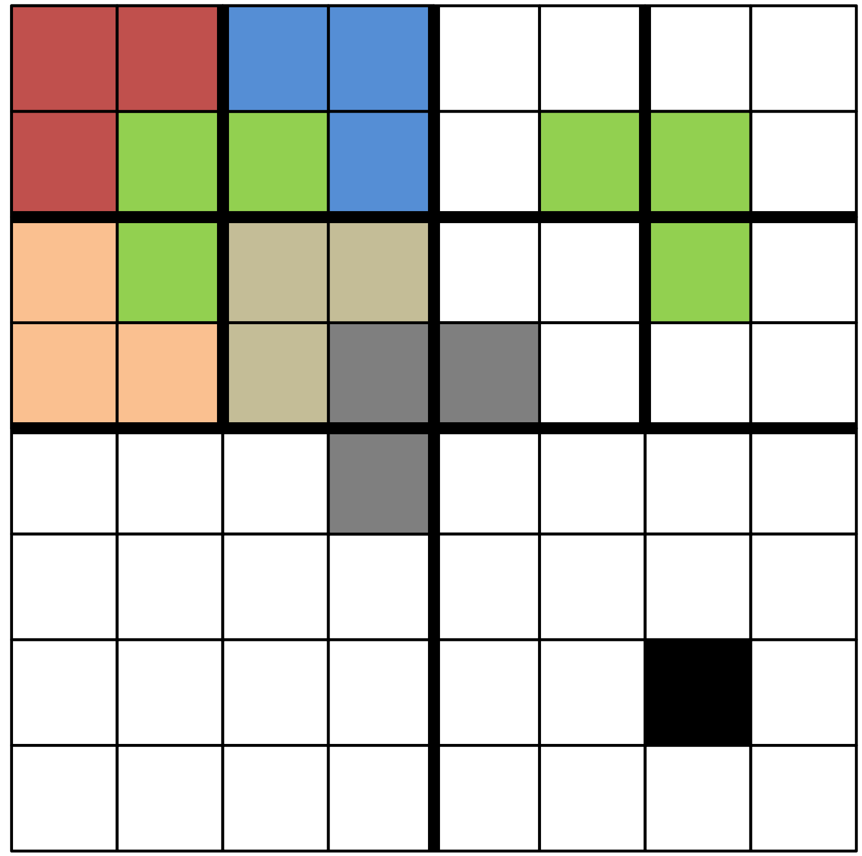
$2^3 \times 2^3$ board

Tromino Cover

Steps

- Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.
- Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

  This reduces the original problem into 4 smaller instances of the same problem!



$2^3 \times 2^3$ board

- Solve each smaller subproblem recursively using the same technique.

Tromino Cover

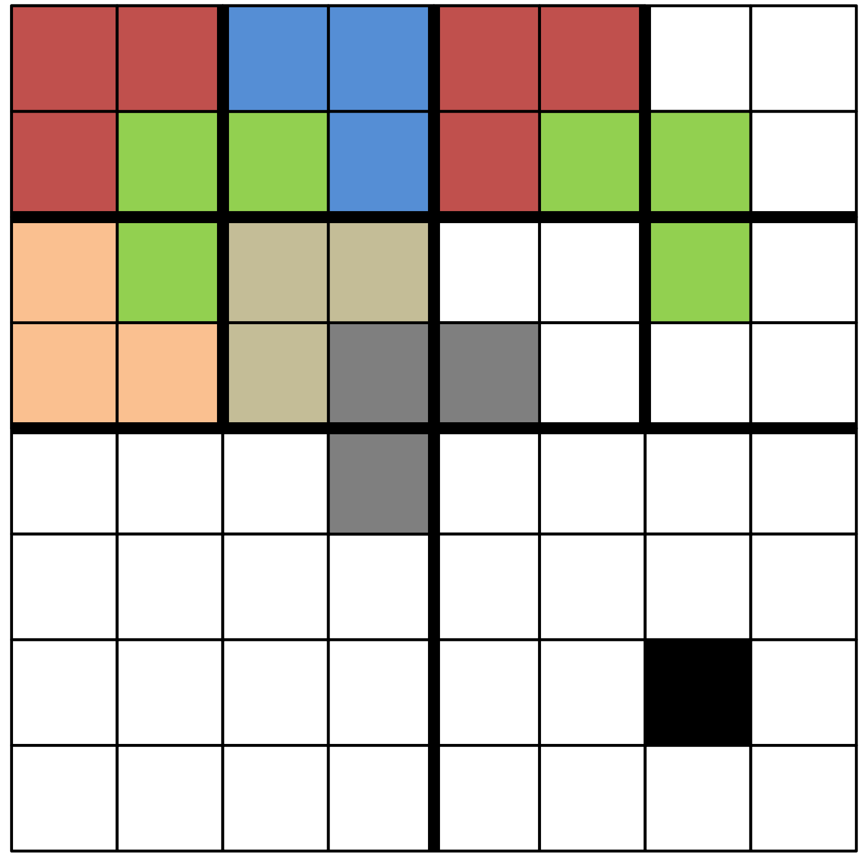
Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!

– Solve each smaller subproblem recursively using the same technique.



$2^3 \times 2^3$ board

Tromino Cover

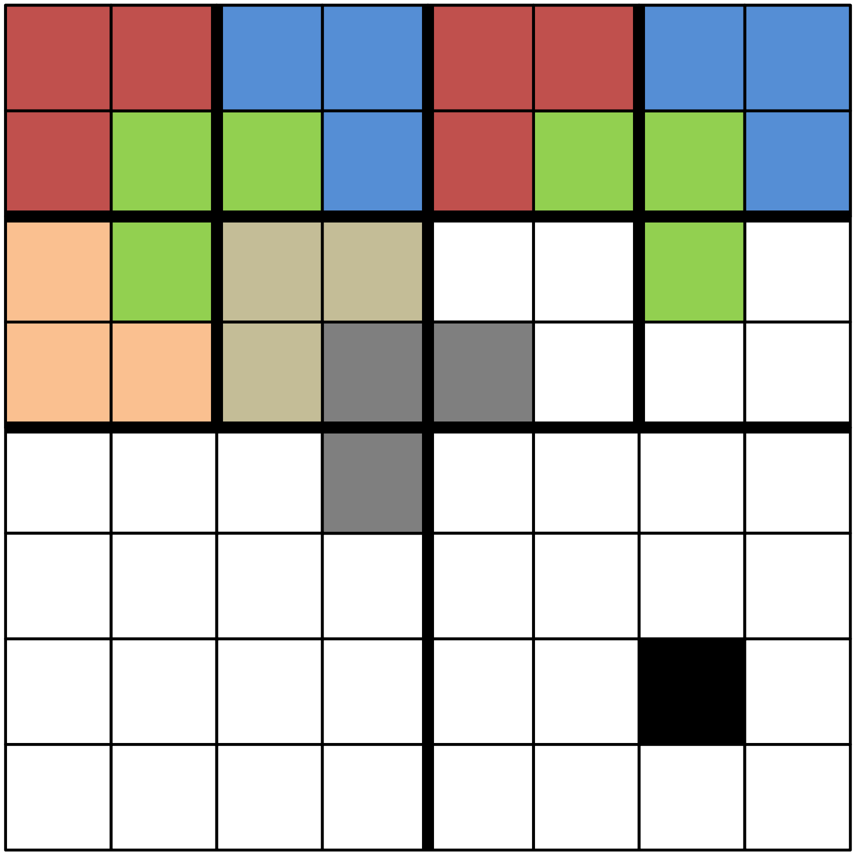
Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!

– Solve each smaller subproblem recursively using the same technique.



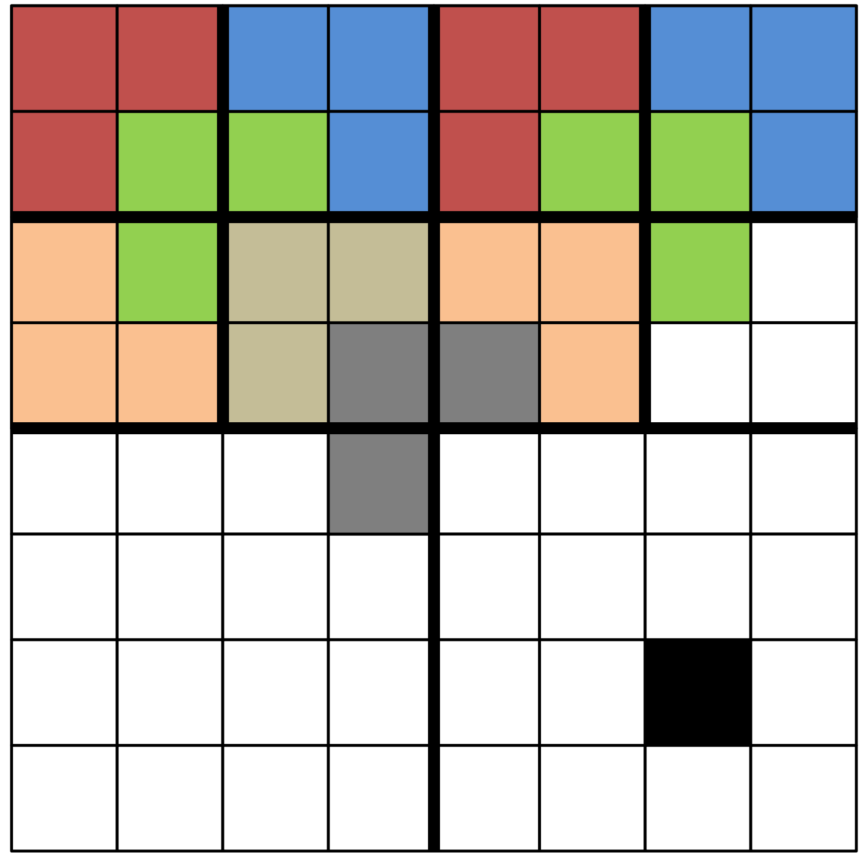
$2^3 \times 2^3$ board

Tromino Cover

Steps

- Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.
- Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

  This reduces the original problem into 4 smaller instances of the same problem!
- Solve each smaller subproblem recursively using the same technique.



$2^3 \times 2^3$ board

Tromino Cover

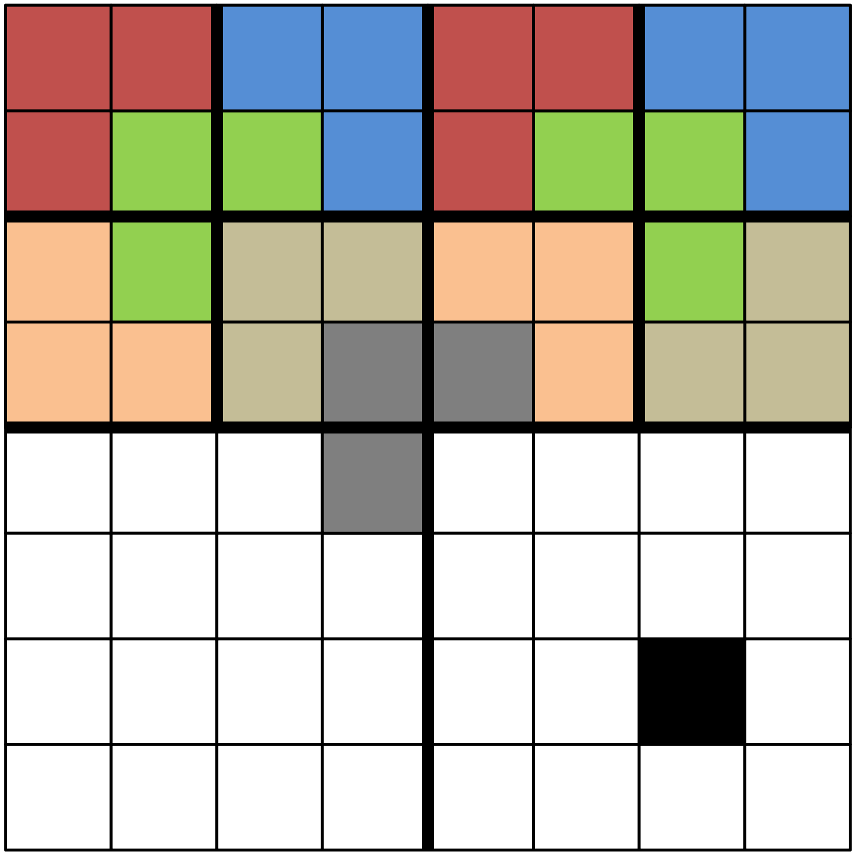
Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!

– Solve each smaller subproblem recursively using the same technique.



$2^3 \times 2^3$ board

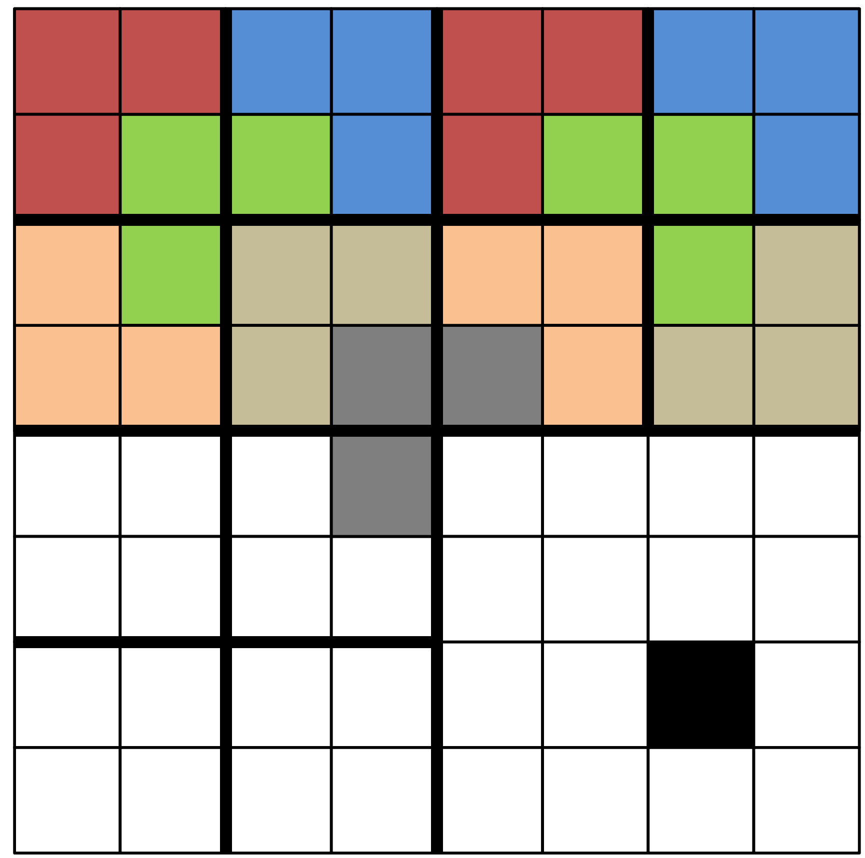
Tromino Cover

Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!



$2^3 \times 2^3$ board

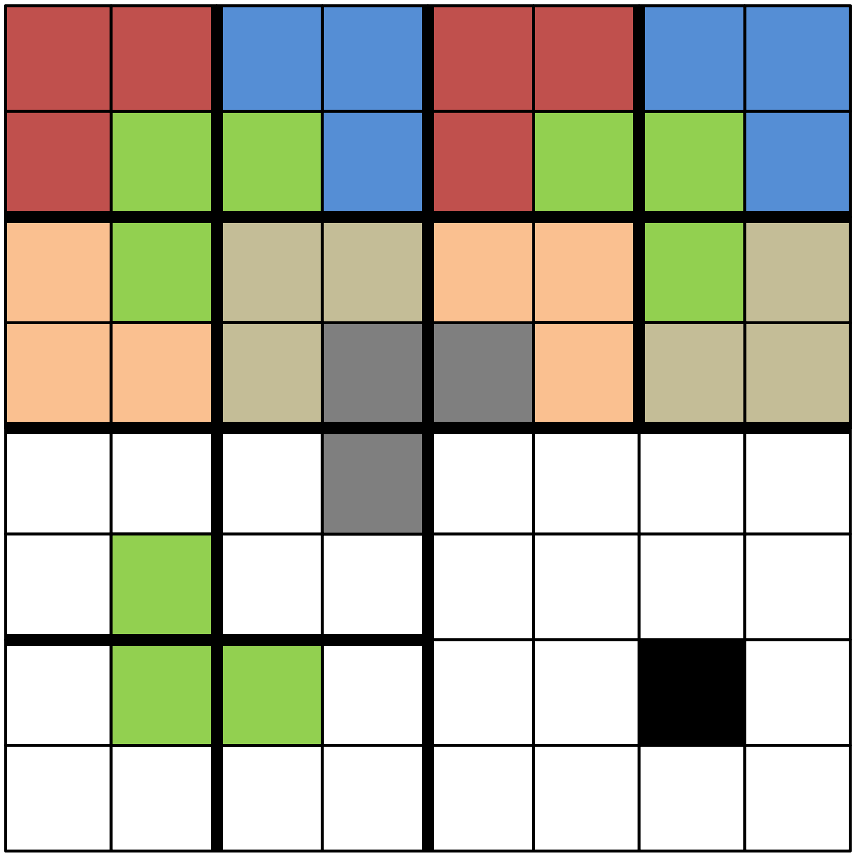
– Solve each smaller subproblem recursively using the same technique.

Tromino Cover

Steps

- Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.
- Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

  This reduces the original problem into 4 smaller instances of the same problem!
- Solve each smaller subproblem recursively using the same technique.



$2^3 \times 2^3$ board

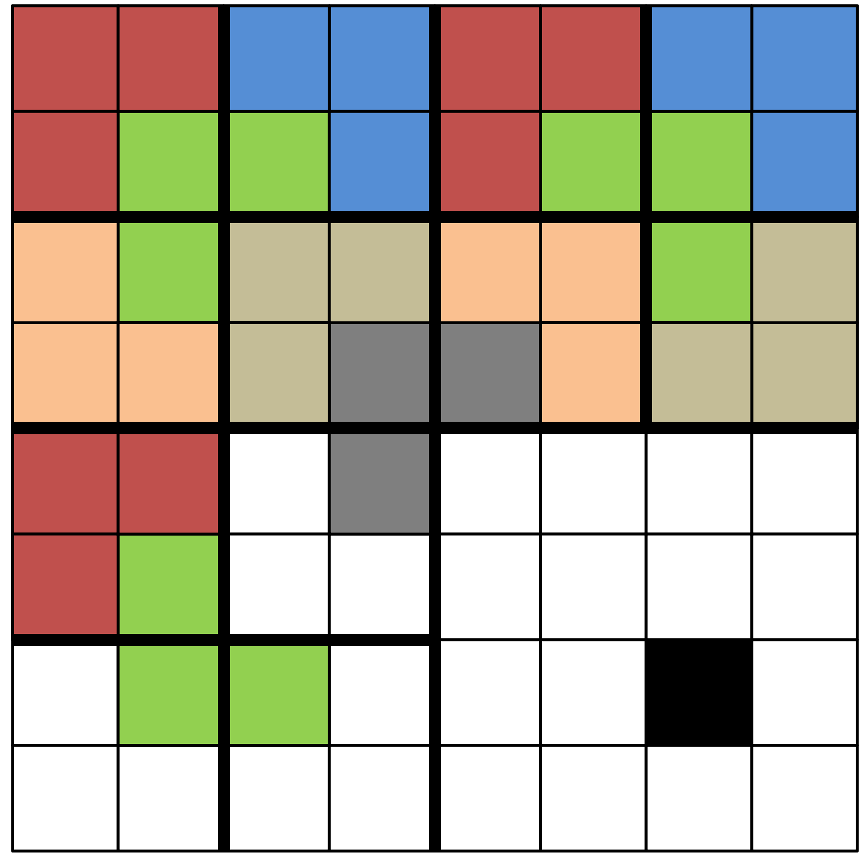
Tromino Cover

Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!



$2^3 \times 2^3$ board

– Solve each smaller subproblem recursively using the same technique.

Tromino Cover

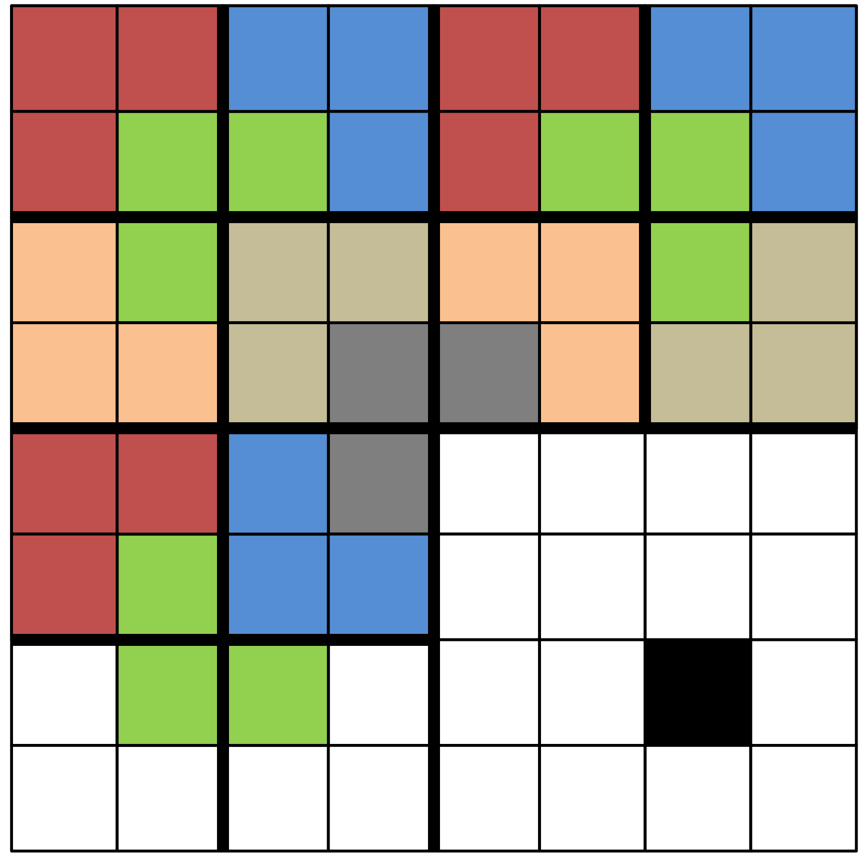
Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!

– Solve each smaller subproblem recursively using the same technique.



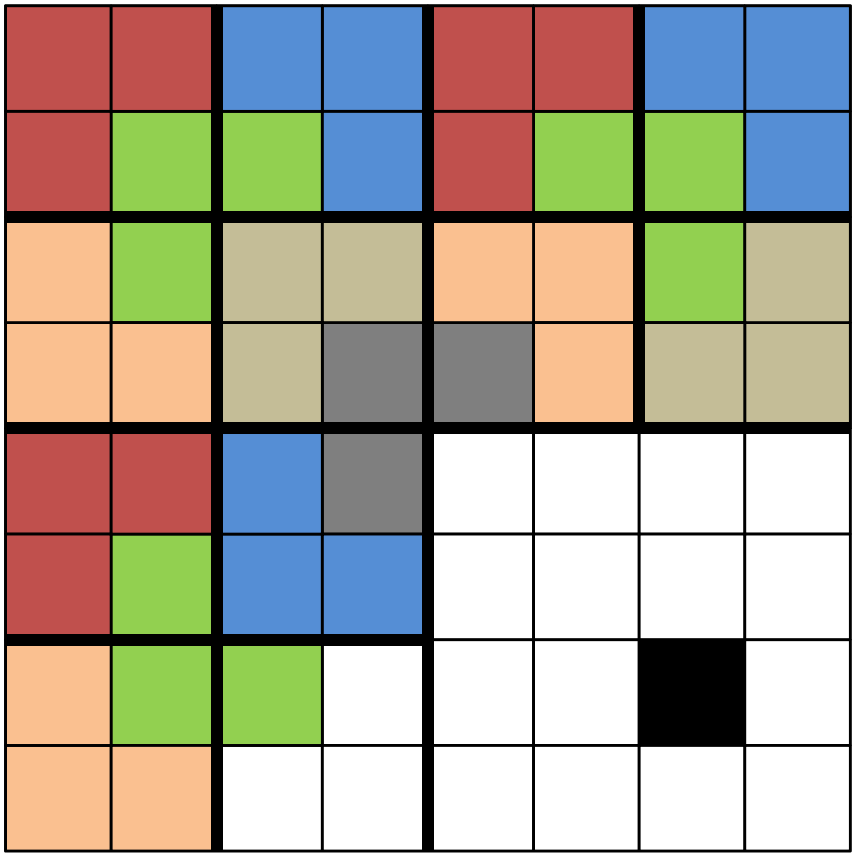
$2^3 \times 2^3$ board

Tromino Cover

Steps

- Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.
- Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

  This reduces the original problem into 4 smaller instances of the same problem!
- Solve each smaller subproblem recursively using the same technique.



$2^3 \times 2^3$ board

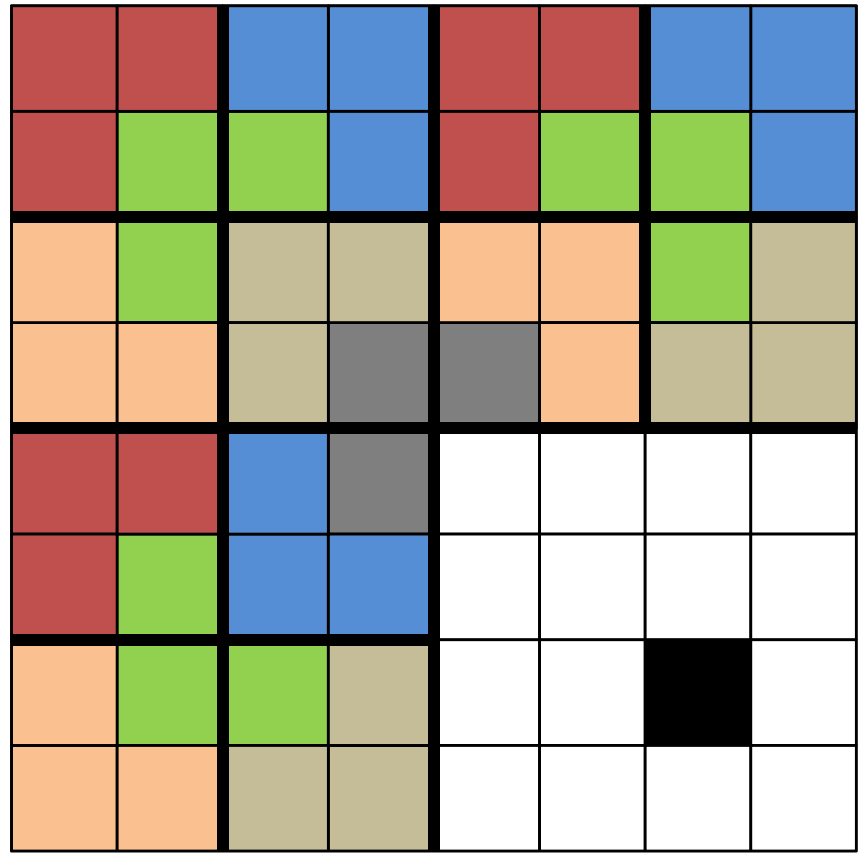
Tromino Cover

Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!



$2^3 \times 2^3$ board

– Solve each smaller subproblem recursively using the same technique.

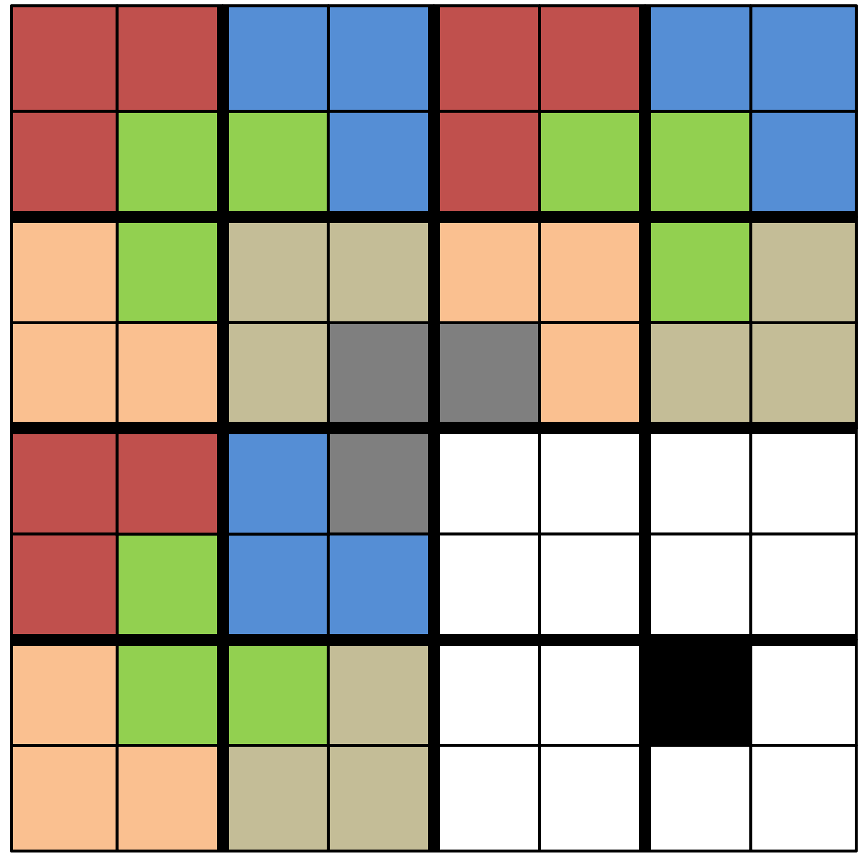
Tromino Cover

Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!



$2^3 \times 2^3$ board

– Solve each smaller subproblem recursively using the same technique.

Tromino Cover

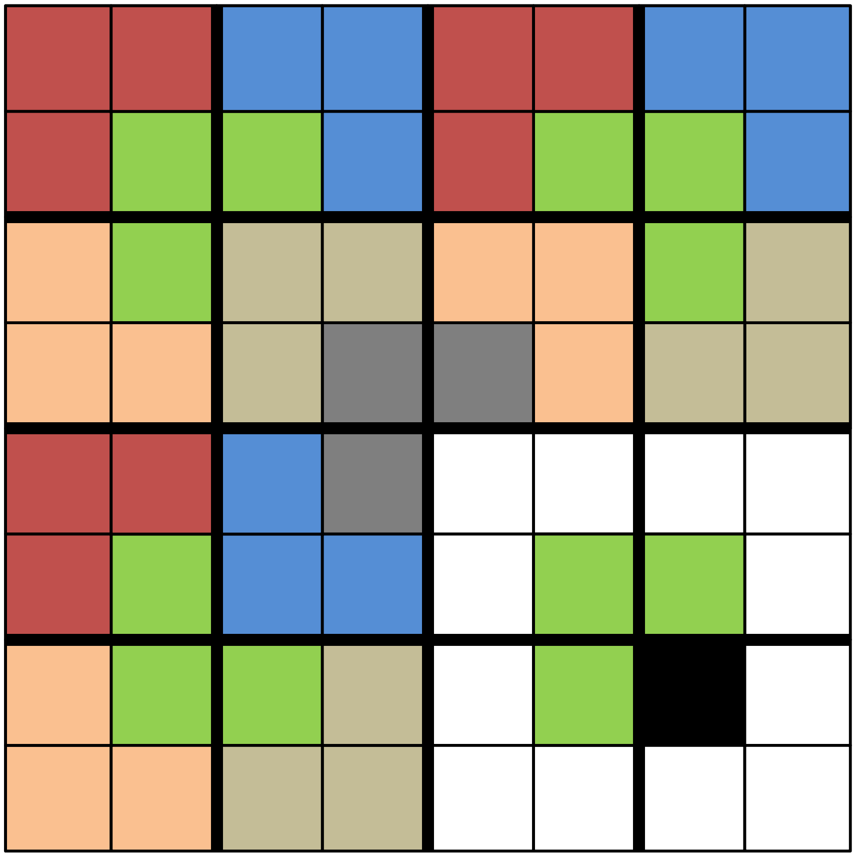
Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!

– Solve each smaller subproblem recursively using the same technique.



$2^3 \times 2^3$ board

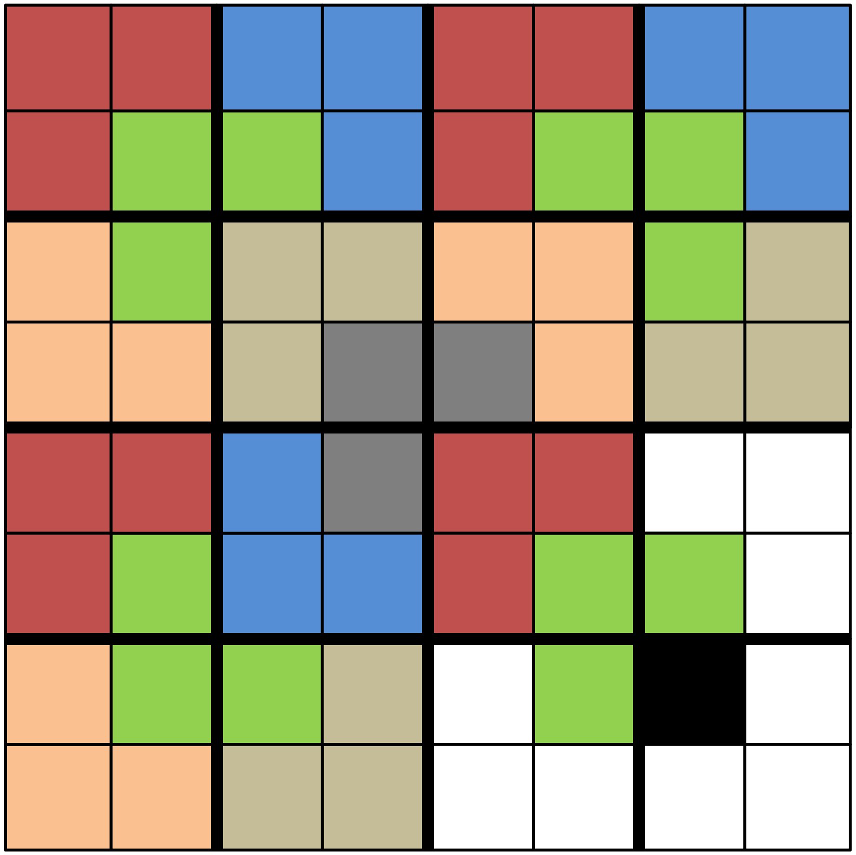
Tromino Cover

Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

  This reduces the original problem into 4 smaller instances of the same problem!



$2^3 \times 2^3$ board

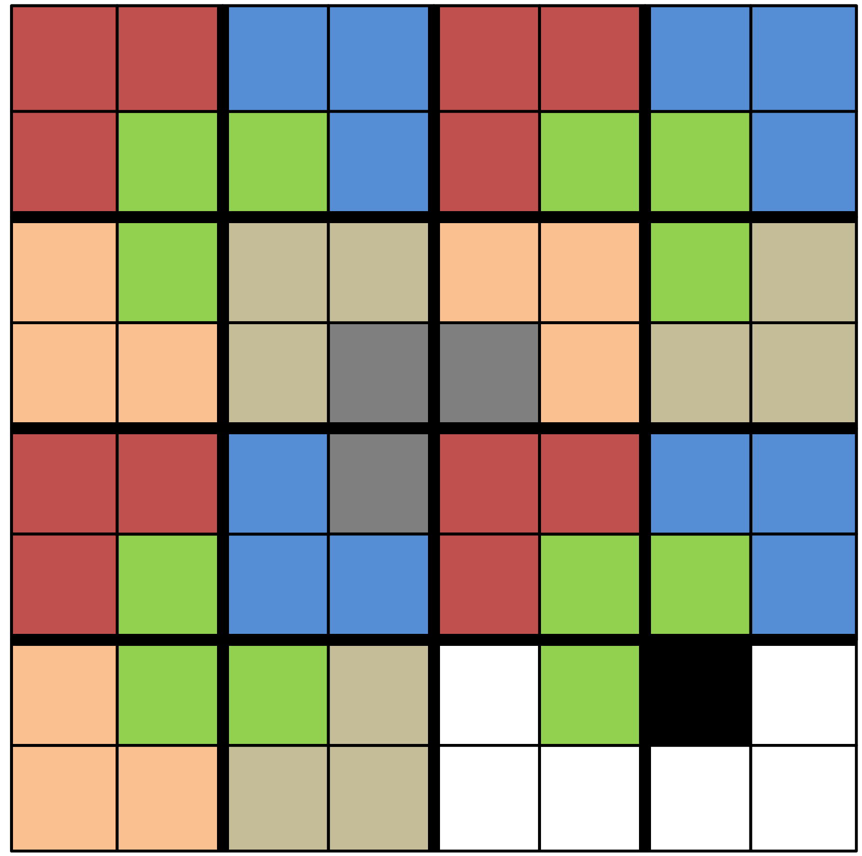
– Solve each smaller subproblem recursively using the same technique.

Tromino Cover

Steps

- Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.
- Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

  This reduces the original problem into 4 smaller instances of the same problem!
- Solve each smaller subproblem recursively using the same technique.



$2^3 \times 2^3$ board

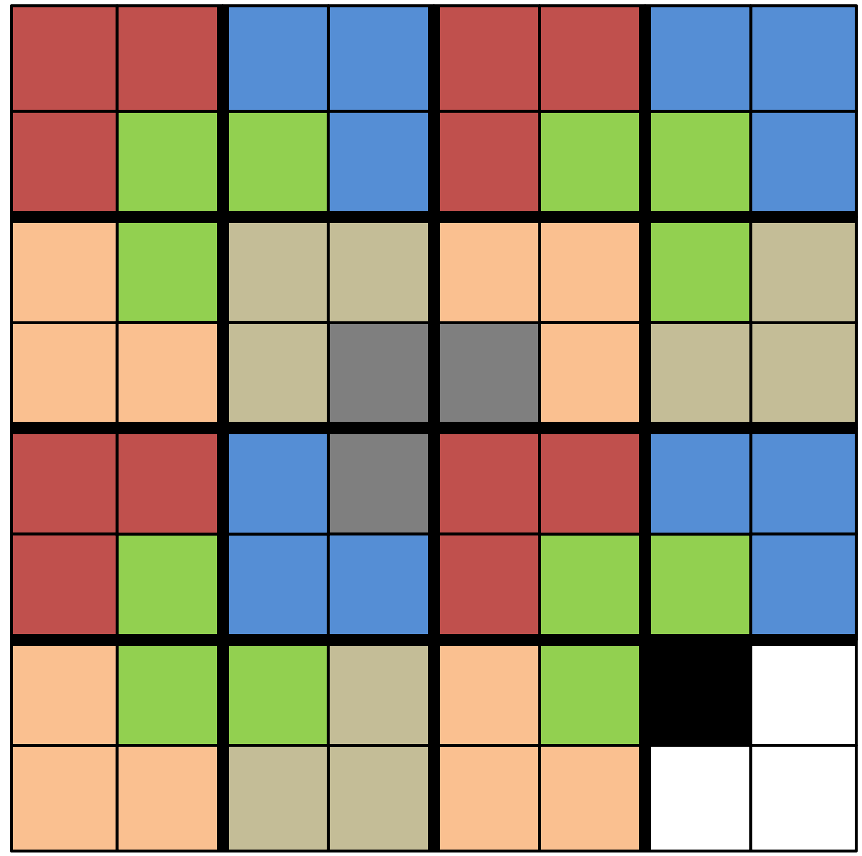
Tromino Cover

Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!



$2^3 \times 2^3$ board

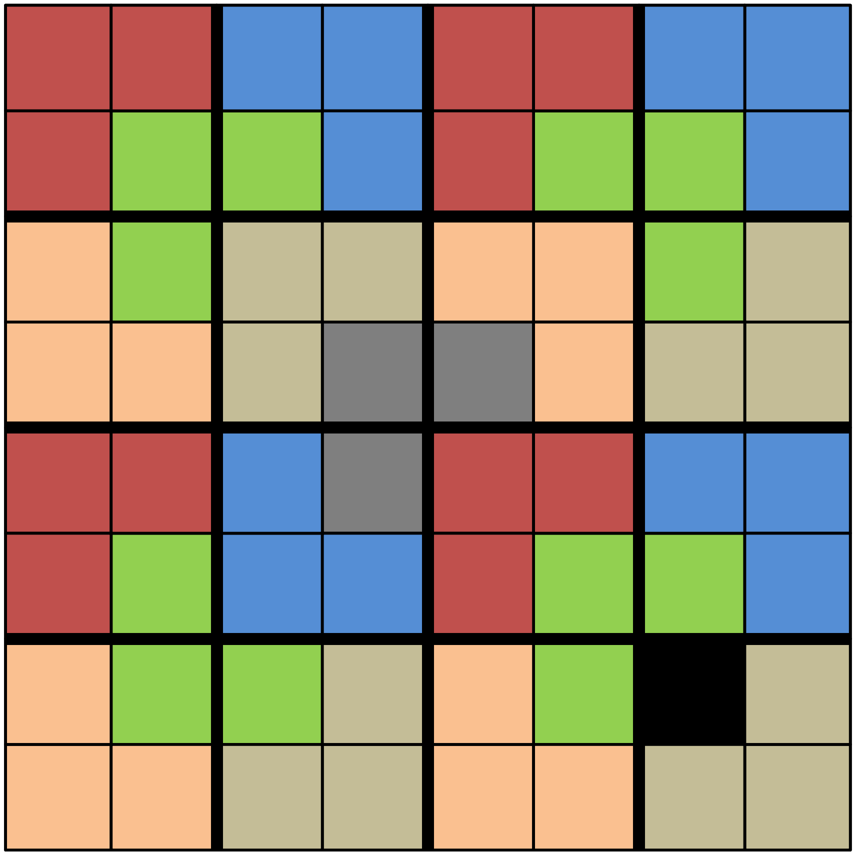
– Solve each smaller subproblem recursively using the same technique.

Tromino Cover

Steps

- Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.
- Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

  This reduces the original problem into 4 smaller instances of the same problem!
- Solve each smaller subproblem recursively using the same technique.



$2^3 \times 2^3$ board

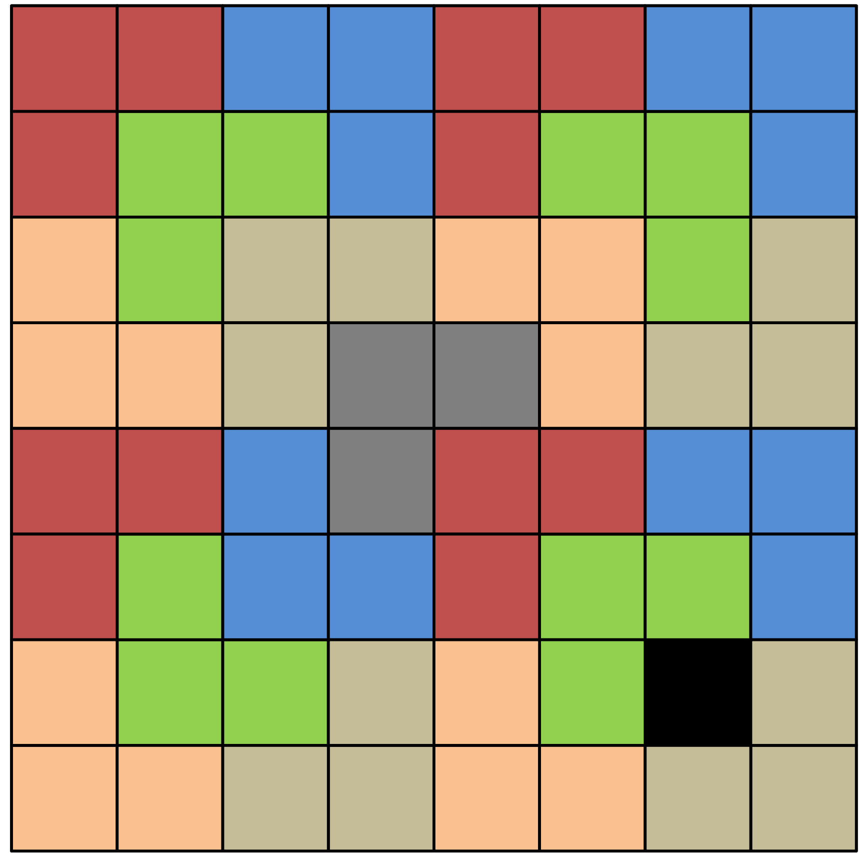
Tromino Cover

Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!



$2^3 \times 2^3$ board

– Solve each smaller subproblem recursively using the same technique.

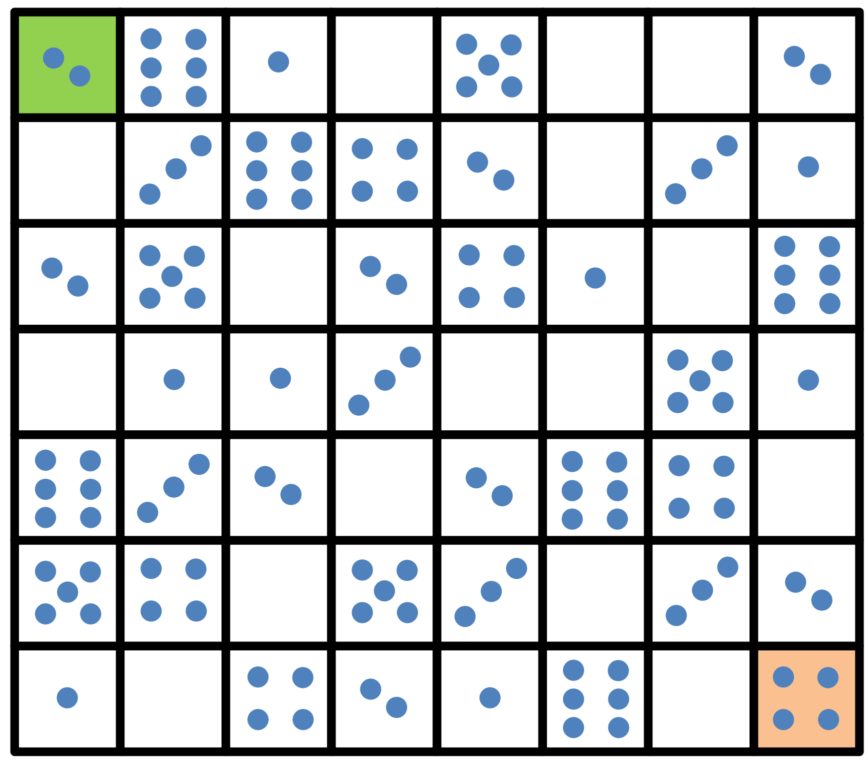
– This algorithm design technique is called *recursive divide & conquer.*

Collecting Coins

Puzzle: A robot moves from the top-left corner to the bottom-right corner of an $m \times n$ grid. At each step it either moves one cell to the right or one cell down from its current location.

Each cell of the grid contains zero or more coins. The robot collects the coins from every cell it visits.

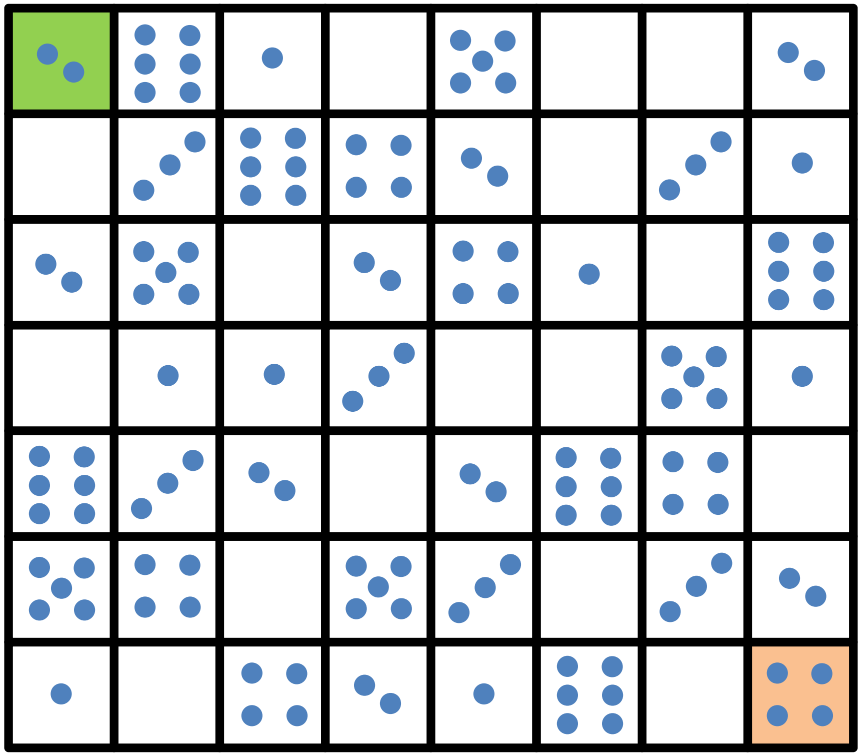
What path the robot should take to collect the maximum number of coins?



8×7 grid

Collecting Coins

Let us first count the number of coins in each cell.



C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & \text{if } i < 0 \text{ or } j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & \text{otherwise} \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & \text{if } i < 0 \text{ or } j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & \text{otherwise} \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & \text{if } i < 0 \text{ or } j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & \text{otherwise} \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & \text{if } i < 0 \text{ or } j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & \text{otherwise} \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8 →	1	0	5	0	0	2
2	2	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & if\ i < 0\ or\ j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & otherwise \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	0	5	0	0	2
2	2	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & \text{if } i < 0 \text{ or } j < 0 \\ \max\{C[i, j-1], C[i-1, j]\} + c_{i,j}, & \text{otherwise} \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	0	5	0	0	2
2	2	11	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i,j]$

Let $C[i,j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i,j]$

$$C[i,j] = \begin{cases} 0, & \text{if } i < 0 \text{ or } j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & \text{otherwise} \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	0	5	0	0	2
2	2	11	6	4	2	0	3	1
3	4	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & \textit{if } i < 0 \textit{ or } j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & \textit{otherwise} \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	5	0	0	2
2	2	11	6	4	2	0	3	1
3	4	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & if\ i < 0\ or\ j < 0 \\ \max\{C[i, j-1], C[i-1, j]\} + c_{i,j}\,, & otherwise \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	5	0	0	2
2	2	11	17	4	2	0	3	1
3	4	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & \text{if } i < 0 \text{ or } j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & \text{otherwise} \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	5	0	0	2
2	2	11	17	4	2	0	3	1
3	4	16	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & \text{if } i < 0 \text{ or } j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & \text{otherwise} \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	5	0	0	2
2	2	11	17	4	2	0	3	1
3	4	16	0	2	4	1	0	6
4	4	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & \text{if } i < 0 \text{ or } j < 0 \\ \max\{C[i, j-1], C[i-1, j]\} + c_{i,j}, & \text{otherwise} \end{cases}$$

c	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	14	14	14	16
2	2	11	17	21	23	23	26	27
3	4	16	17	23	27	28	28	34
4	4	17	18	26	27	28	33	35
5	10	20	22	26	29	35	39	39
6	15	24	24	31	34	35	42	44
7	16	24	28	33	35	41	42	48

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i,j]$

Let $C[i,j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i,j]$

$$C[i,j] = \begin{cases} 0, & if\ i < 0\ or\ j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & otherwise \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	14	14	14	16
2	2	11	17	21	23	23	26	27
3	4	16	17	23	27	28	28	34
4	4	17	18	26	27	28	33	35
5	10	20	22	26	29	35	39	39
6	15	24	24	31	34	35	42	44
7	16	24	28	33	35	41	42	48

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & \mathit{if}\ i < 0\ \mathit{or}\ j < 0 \\ \max\{C[i, j-1], C[i-1, j]\} + c_{i,j}, & \mathit{otherwise} \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	14	14	14	16
2	2	11	17	21	23	23	26	27
3	4	16	17	23	27	28	28	34
4	4	17	18	26	27	28	33	35
5	10	20	22	26	29	35	39	39
6	15	24	24	31	34	35	42	44
7	16	24	28	33	35	41	42	48

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & \mathit{if}\ i < 0\ \mathit{or}\ j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & \mathit{otherwise} \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	14	14	14	16
2	2	11	17	21	23	23	26	27
3	4	16	17	23	27	28	28	34
4	4	17	18	26	27	28	33	35
5	10	20	22	26	29	35	39	39
6	15	24	24	31	34	35	42	44
7	16	24	28	33	35	41	42	48

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & \text{if } i < 0 \text{ or } j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & \text{otherwise} \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	14	14	14	16
2	2	11	17	21	23	23	26	27
3	4	16	17	23	27	28	28	34
4	4	17	18	26	27	28	33	35
5	10	20	22	26	29	35	39	39
6	15	24	24	31	34	35	42	44
7	16	24	28	33	35	41	42	48

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i,j]$

Let $C[i,j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i,j]$

$$C[i,j] = \begin{cases} 0, & \text{if } i < 0 \text{ or } j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & \text{otherwise} \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	14	14	14	16
2	2	11	17	21	23	23	26	27
3	4	16	17	23	27	28	28	34
4	4	17	18	26	27	28	33	35
5	10	20	22	26	29	35	39	39
6	15	24	24	31	34	35	42	44
7	16	24	28	33	35	41	42	48

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & \textit{if } i < 0 \textit{ or } j < 0 \\ \max\{C[i, j-1], C[i-1, j]\} + c_{i,j}, & \textit{otherwise} \end{cases}$$

c	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	14	14	14	16
2	2	11	17	21	23	23	26	27
3	4	16	17	23	27	28	28	34
4	4	17	18	26	27	28	33	35
5	10	20	22	26	29	35	39	39
6	15	24	24	31	34	35	42	44
7	16	24	28	33	35	41	42	48

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i,j]$

Let $C[i,j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i,j]$

$$C[i,j] = \begin{cases} 0, & \textit{if } i < 0 \textit{ or } j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & \textit{otherwise} \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	14	14	14	16
2	2	11	17	21	23	23	26	27
3	4	16	17	23	27	28	28	34
4	4	17	18	26	27	28	33	35
5	10	20	22	26	29	35	39	39
6	15	24	24	31	34	35	42	44
7	16	24	28	33	35	41	42	48

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & \textit{if } i < 0 \textit{ or } j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & \textit{otherwise} \end{cases}$$

C	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	14	14	14	16
2	2	11	17	21	23	23	26	27
3	4	16	17	23	27	28	28	34
4	4	17	18	26	27	28	33	35
5	10	20	22	26	29	35	39	39
6	15	24	24	31	34	35	42	44
7	16	24	28	33	35	41	42	48

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

$$C[i,j] = \begin{cases} 0, & \text{if } i < 0 \text{ or } j < 0 \\ \max\{C[i, j-1], C[i-1, j]\} + c_{i,j}, & \text{otherwise} \end{cases}$$

c	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	14	14	14	16
2	2	11	17	21	23	23	26	27
3	4	16	17	23	27	28	28	34
4	4	17	18	26	27	28	33	35
5	10	20	22	26	29	35	39	39
6	15	24	24	31	34	35	42	44
7	16	24	28	33	35	41	42	48

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i,j]$

Let $C[i,j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i,j]$

$$C[i,j] = \begin{cases} 0, & \text{if } i < 0 \text{ or } j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & \text{otherwise} \end{cases}$$

c	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	14	14	14	16
2	2	11	17	21	23	23	26	27
3	4	16	17	23	27	28	28	34
4	4	17	18	26	27	28	33	35
5	10	20	22	26	29	35	39	39
6	15	24	24	31	34	35	42	44
7	16	24	28	33	35	41	42	48

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i,j]$

Let $C[i,j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i,j]$

$$C[i,j] = \begin{cases} 0, & \text{if } i < 0 \text{ or } j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}, & \text{otherwise} \end{cases}$$

c	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	14	14	14	16
2	2	11	17	21	23	23	26	27
3	4	16	17	23	27	28	28	34
4	4	17	18	26	27	28	33	35
5	10	20	22	26	29	35	39	39
6	15	24	24	31	34	35	42	44
7	16	24	28	33	35	41	42	48

Collecting Coins

Let $c_{i,j}$ = number of coins in cell $[i, j]$

Let $C[i, j]$ = max number of coins the robot can collect if it goes from cell $[1,1]$ to cell $[i, j]$

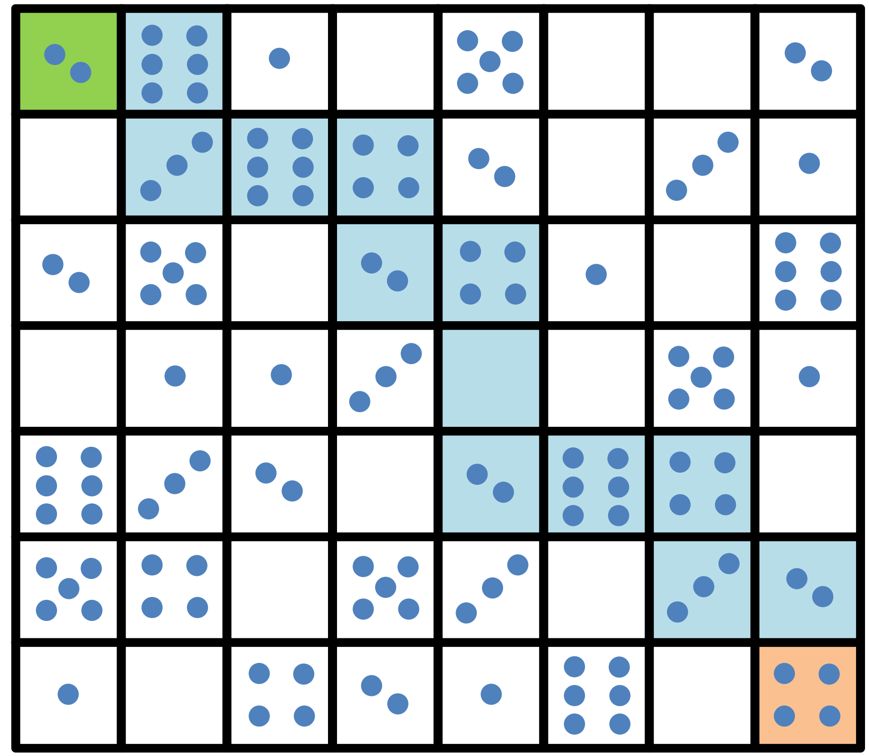
$$C[i,j] = \begin{cases} 0, & if\ i < 0\ or\ j < 0 \\ \max\{C[i,j-1], C[i-1,j]\} + c_{i,j}\,, & otherwise \end{cases}$$

c	1	2	3	4	5	6	7	8
1	2	6	1	0	5	0	0	2
2	0	3	6	4	2	0	3	1
3	2	5	0	2	4	1	0	6
4	0	1	1	3	0	0	5	1
5	6	3	2	0	2	6	4	0
6	5	4	0	5	3	0	3	2
7	1	0	4	2	1	6	0	4

C	1	2	3	4	5	6	7	8
1	2	8	9	9	14	14	14	16
2	2	11	17	21	23	23	26	27
3	4	16	17	23	27	28	28	34
4	4	17	18	26	27	28	33	35
5	10	20	22	26	29	35	39	39
6	15	24	24	31	34	35	42	44
7	16	24	28	33	35	41	42	48

Collecting Coins

C	1	2	3	4	5	6	7	8
1	2	8	9	9	14	14	14	16
2	2	11	17	21	23	23	26	27
3	4	16	17	23	27	28	28	34
4	4	17	18	26	27	28	33	35
5	10	20	22	26	29	35	39	39
6	15	24	24	31	34	35	42	44
7	16	24	28	33	35	41	42	48



This algorithm design technique is called *dynamic programming*.

We computed solutions to larger and larger instances of the given problem using saved solutions to smaller overlapping instances of the same problem until we found a solution to the given instance.

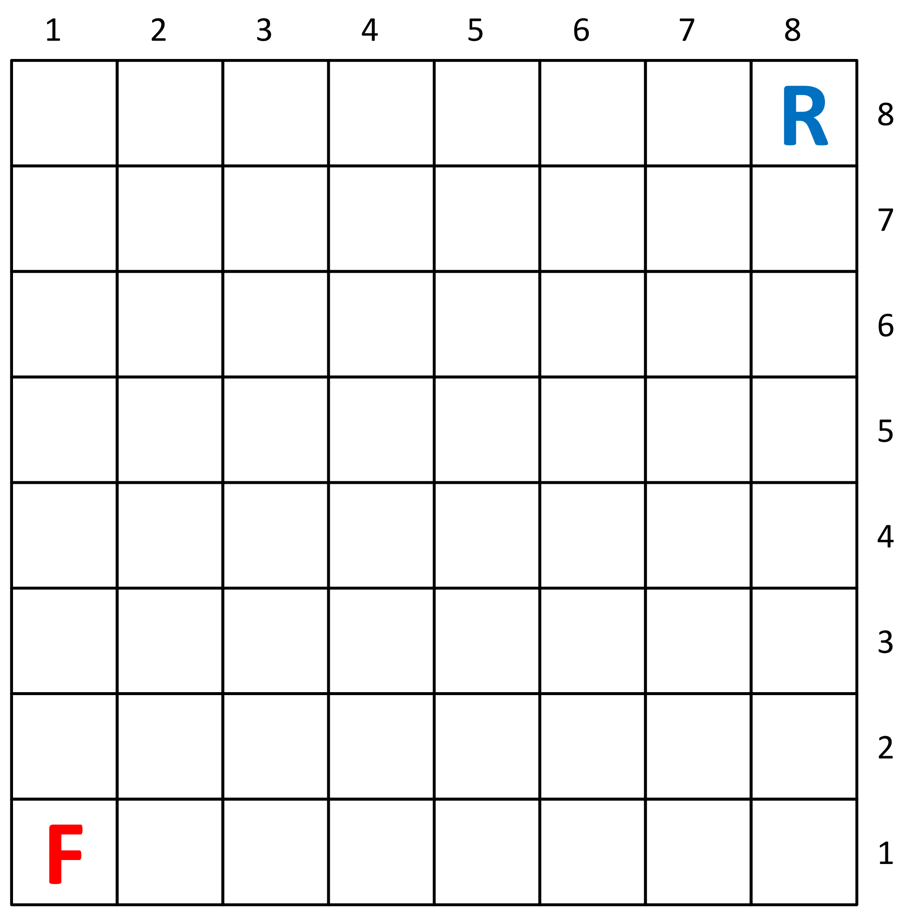
Rooster Chase

Puzzle: A framer (**F**) is trying to catch a rooster (**R**) on a $2n \times 2n$ grid.

- **F**'s initial position: bottom-left corner of the grid
- **R**'s initial position: top-right corner of the grid
- **F** and **R** move alternately until **R** is captured
- each move is to a neighboring cell horizontally or vertically
- **R** is captured when **F** moves to a cell occupied by **R**

What algorithm should **F** use to catch **R** when

- **F** moves first?
- **R** moves first?



8×8 board

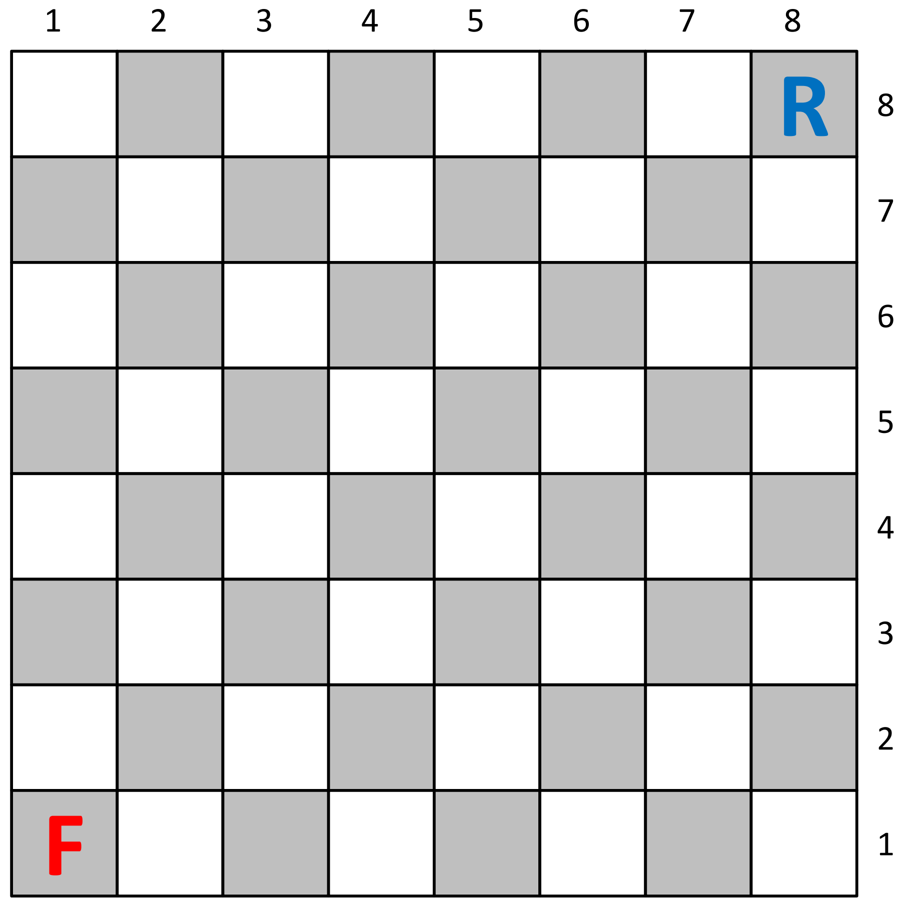
Rooster Chase

Initially, both **F** and **R** are on cells of the same color.

If **F** is to catch **R** in the next move, **R** must be in a cell horizontally or vertically adjacent to **F**'s current cell.

So, right before **F** catches **R**, they must be in cells of opposite color.

But that will never happen if **F** moves first!



8×8 board

So, if **F** moves first, **F** will never be able to catch **R**!

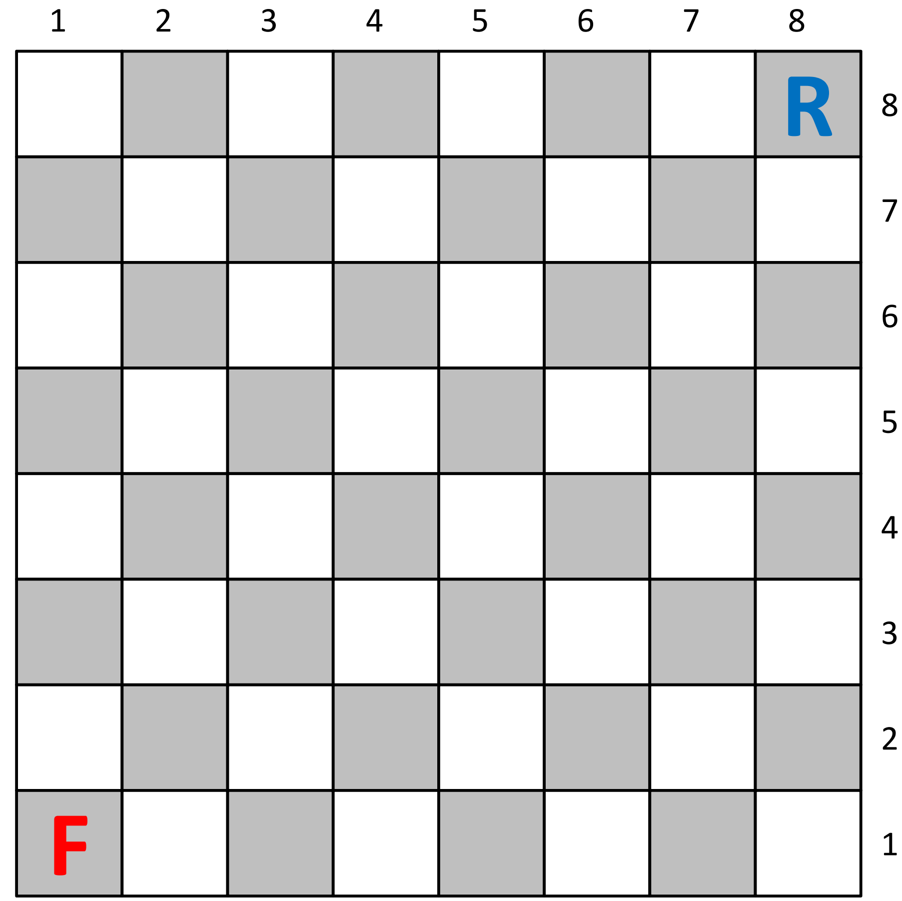
Rooster Chase

If **R** moves first, then **F** will be able to catch **R** in at most $2n - 1$ moves of its own (or $2(2n\ - 1)$ total moves)!

Let $(i_F, j_F) =$ **F**'s current location, and $(i_R, j_R) =$ **R**'s current location.

F's algorithm is as follows:

$\boldsymbol{i_R - i_F > j_R - j_F}$: **F** moves up
$\boldsymbol{i_R - i_F < j_R - j_F}$: **F** moves right
$\boldsymbol{i_R - i_F = j_R - j_F}$: cannot happen



8×8 board

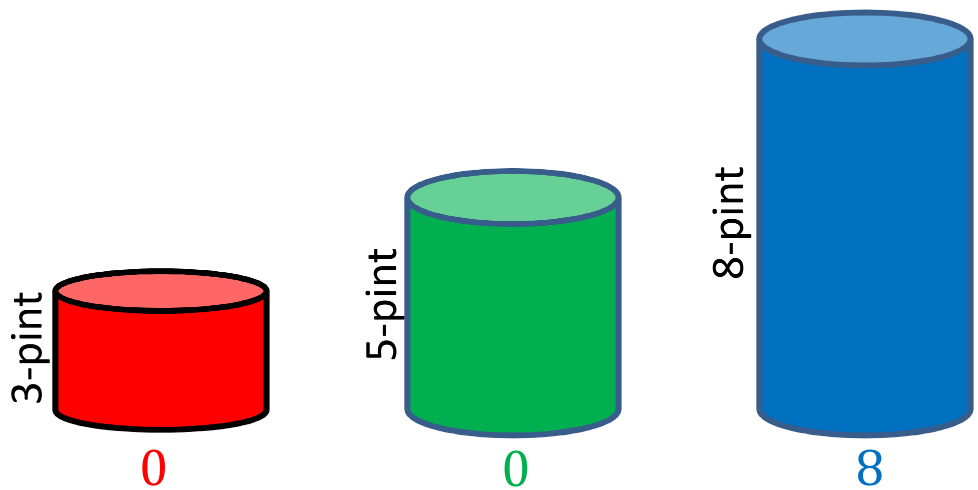
This is a *greedy algorithm* as **F** always chooses the option that looks the best (i.e., reduces the Manhattan distance to **R**) at the time of choice.

Three Jugs

Puzzle: You are given three jugs

- one 8-pint jug full of water
- one empty 5-pint jug
- one empty 3-pint jug

Your task is to get exactly 4 pints of water in one of the jugs by completely filling up or emptying jugs into others. Minimize the number of times you fill up / empty jugs.

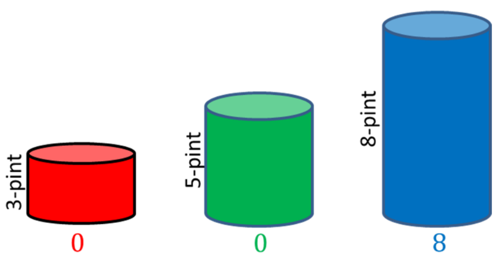


initial state

Three Jugs

$\underline{level\ 0}$

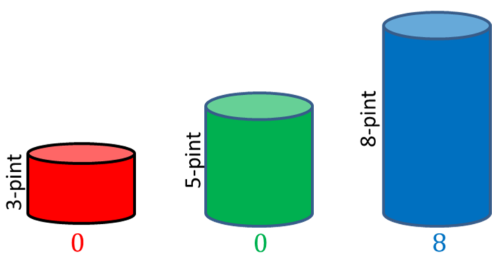
008



Three Jugs

$\underline{level\ 0}$ $\underline{level\ 1}$

$$008 \longrightarrow \left\{ \begin{array}{l} 305 \\ 053 \end{array} \right.$$



Three Jugs

level 0 | *level* 1 | *level* 2

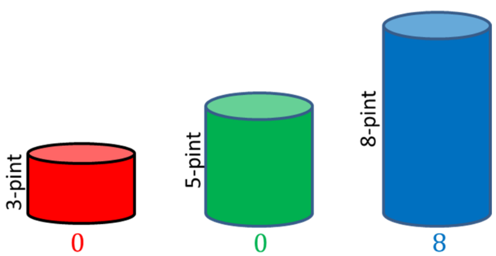
035
008
350

305

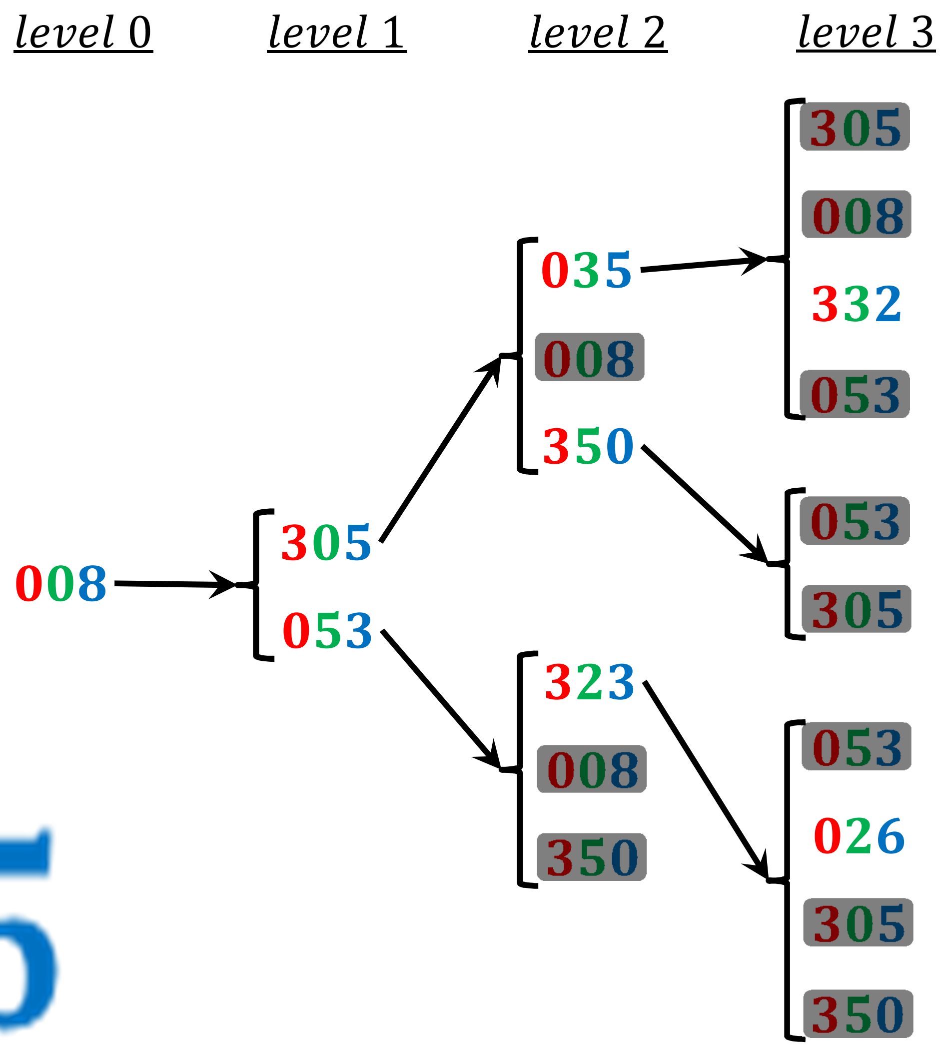
008

053

323
008
350

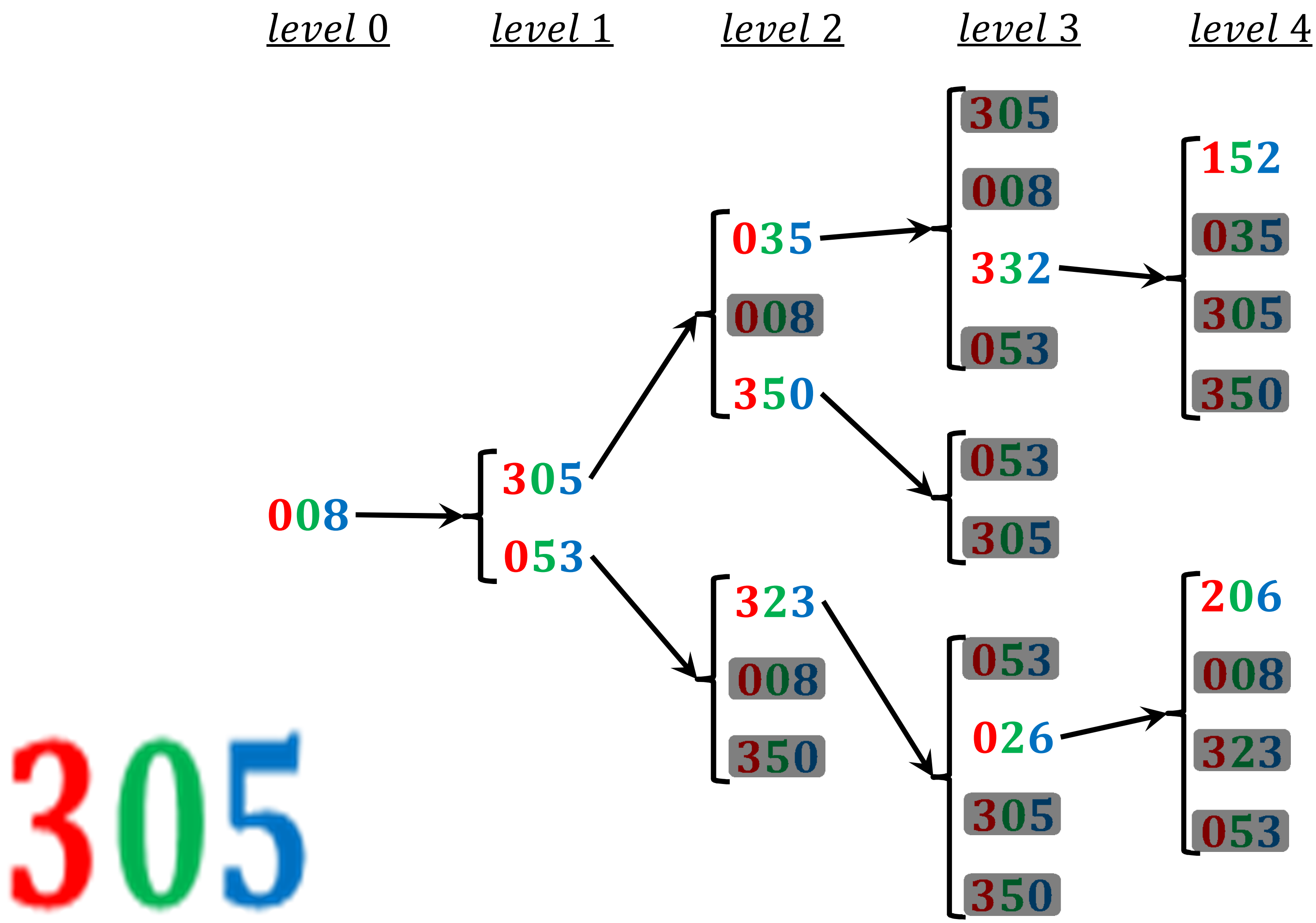


Three Jugs

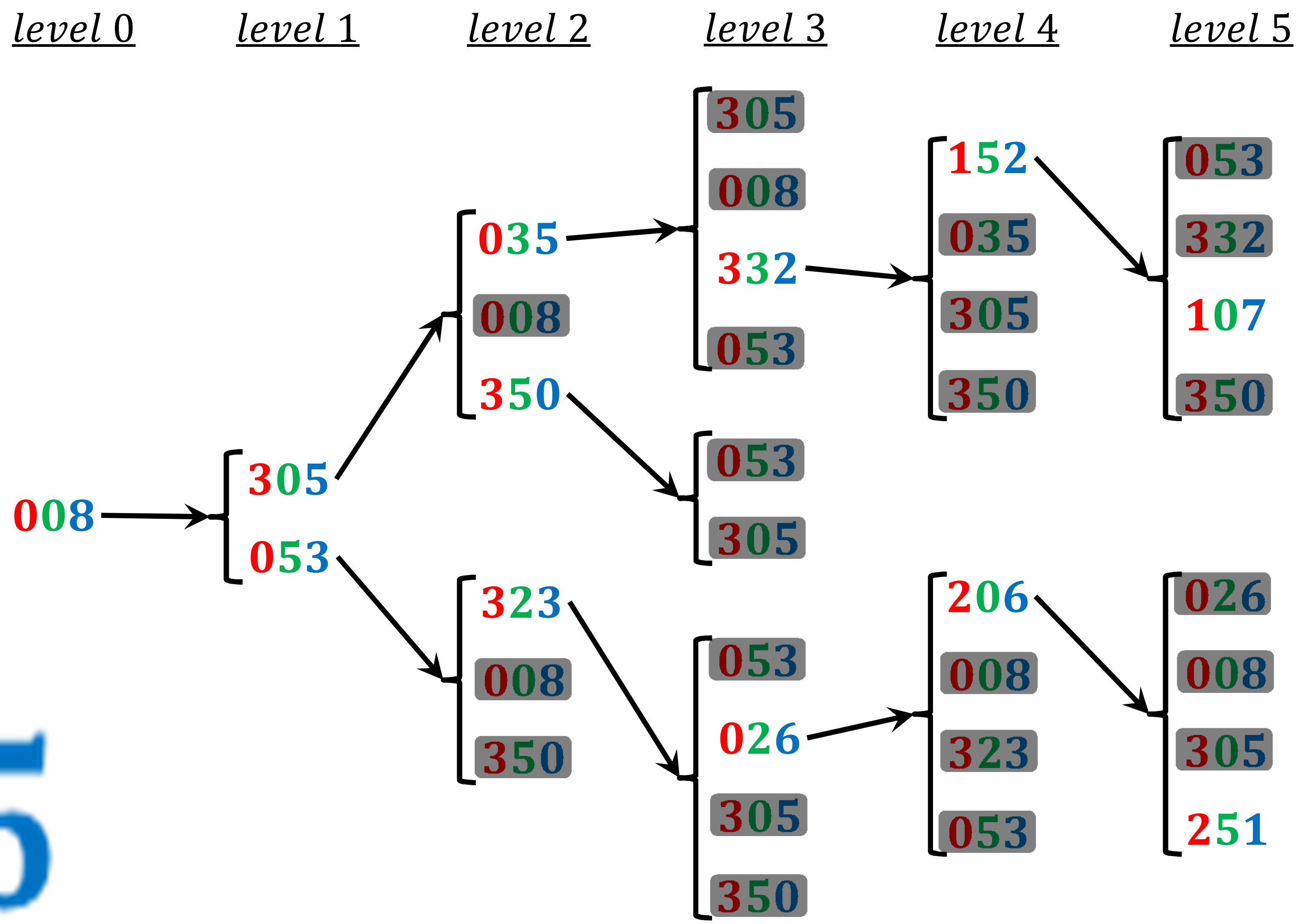


305

Three Jugs

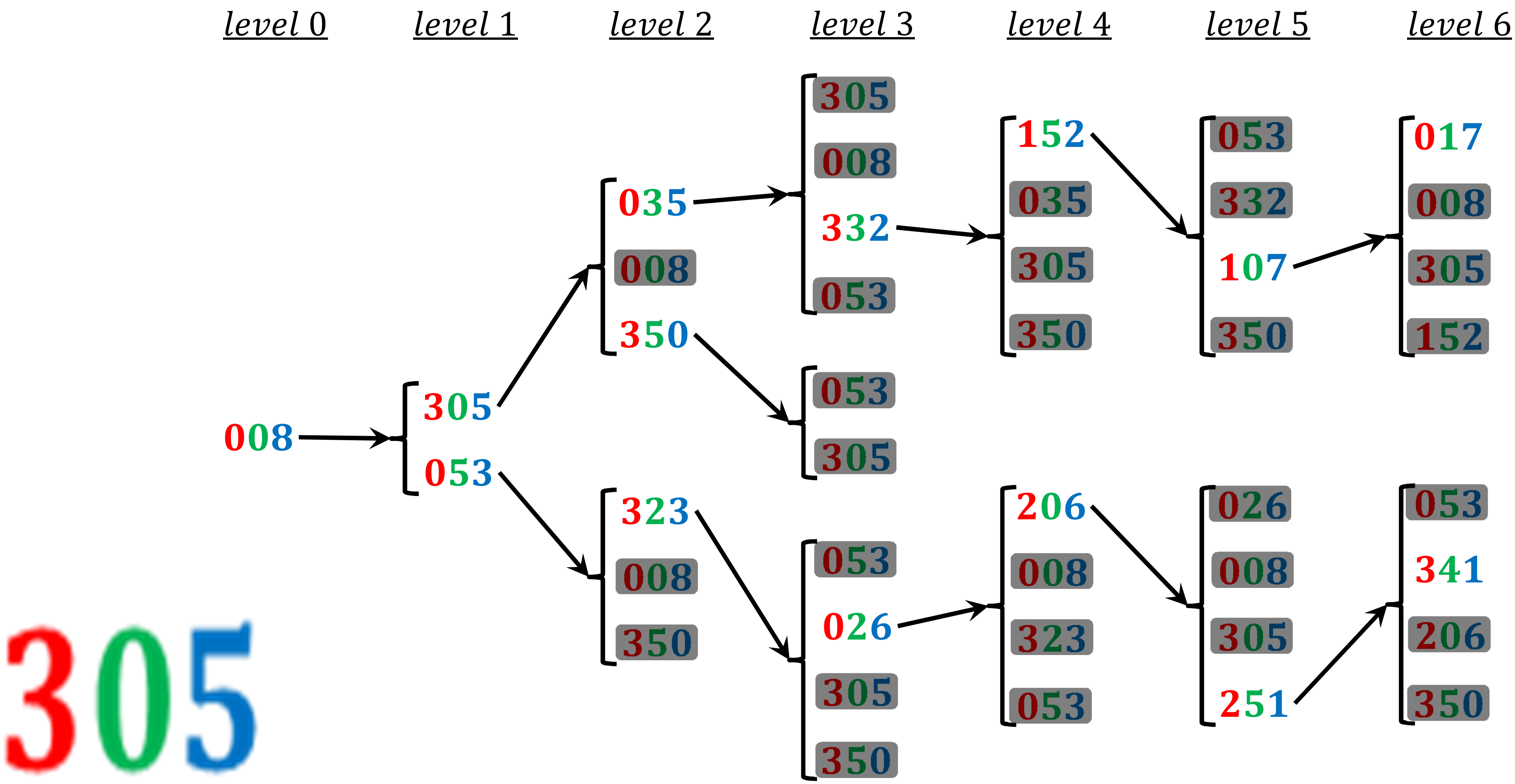

level 0
level 1
level 2
level 3
level 4
008
305
053
035
008
350
323
008
350
305
008
332
053
053
305
053
026
305
350
152
035
305
350
206
008
323
053
305

Three Jugs

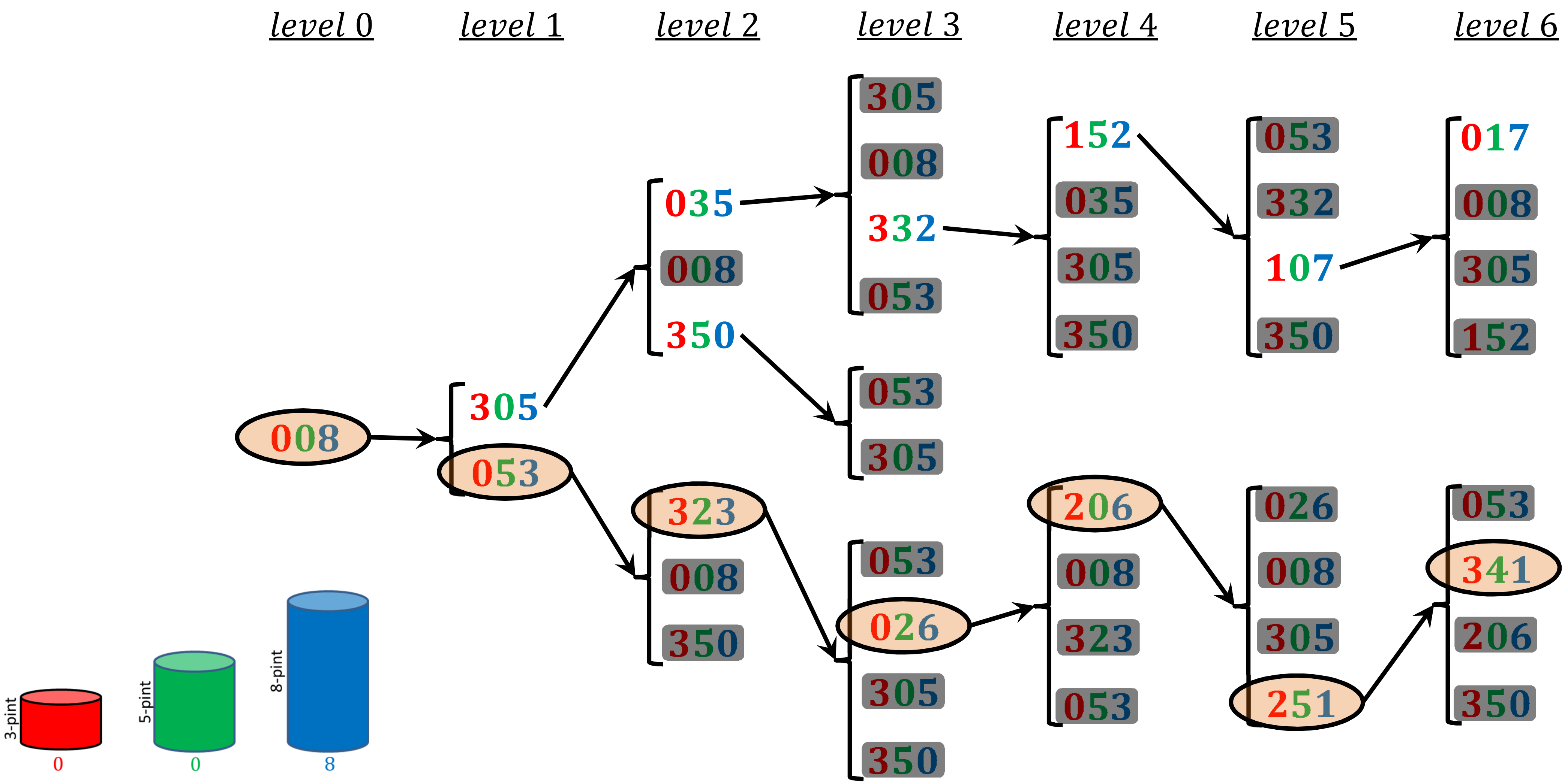


305

Three Jugs


level 0
level 1
level 2
level 3
level 4
level 5
level 6
008
305
053
035
008
350
323
008
350
305
008
332
053
053
305
053
026
305
350
152
035
305
350
206
008
323
053
053
332
107
350
026
008
305
251
017
008
305
152
053
341
206
350
305

Three Jugs



We have just used a *state-space search algorithm* called the *breadth-first search*.