2022 SBU ICPC Selection Contest Editorial

December 7, 2022

A - War

Setter: *Shawn Mathew,* ***Validator:*** *Yimin Zhu*
Editorialist: *Tanzir Pial* ***Tag:*** *Implementation*

This is a direct implementation problem where you have to compare the given cards, find out the winner of the round and keep track of total number of wins by both players and the number of ties.

B - Happy Elimination

Problem Setter: *Yimin Zhu,* ***Validator:*** *Jiarui Zhang*
Editorialist: *Yimin Zhu, Tanzir Pial* ***Tag:*** *Dynamic Programming*

The naive solution tries all possible taps and takes $O(n!)$. Using dynamic programming to store the recomputations for the same subarray and reusing them later, the time complexity can drop to $O(n^3)$.

Let C denote the block array and let $dp[i_{left}][i_{right}][k]$ represent the best points to gain for $C[i_{left} : i_{right}]$(inclusive) followed by k blocks with the same value as $C[i_{right}]$. In this state, we consider all ways we can eliminate the i_{right}-th block (now or later) and pick the option that maximizes our total gain.

- We can tap the i_{right}-th block and the solution will be $dp[i_{left}][i_{right} - 1][0] + (k + 1) * (k + 1)$.
- We can select the i'_{right}-th block where $C[i_{right}] == C[i'_{right}]$ and $i_{left} \leq i'_{right} < i_{right}$ and eliminate the i_{right}-th box when we tap the i'_{right}-th box. To make this happen, we must eliminate all boxes from $i'_{right} + 1$ to $i_right - 1$ so that i'_{right} and i_{right} become contiguous. So we eliminate those first and then do this. The solution in this case is: $dp[i_{left}][i'_{right}][k + 1] + dp[i'_{right} + 1][i_{right} - 1][0]$

We pick the solution that maximizes our gain.

C - Purrfection

Problem Setter: *Michael Wolf-Sonkin,*
Validator: *Astra Kolomatski*
Editorialist: *Michael Wolf-Sonkin* ***Tag:*** *Math, Observation*

The first thing to notice is that trying to compute T by brute force will take $O(n^2)$ time, and since n can be as large as $2 * 10^5$ you will get TLE. Consider the two cases:

- $n > m$: There are only m possible remainders after modding by m, so there must exist two numbers in the array that have the same remainder. This means their difference is divisible by m, and thus the entire product is $0 \mod m$.
- $n \leq m$: We can brute force the solution since the max value of m is small.

D - Meow You Doin?

Problem Setter: *Michael Wolf-Sonkin,* ***Validator:*** *Tanzir Pial,*
Editorialist: *Michael Wolf-Sonkin* ***Tag:*** *Graph*

This question asks for the number of pairs of vertices (x, y) such that all paths of x to y pass through vertex a as well as vertex b. To solve this we can use two modified DFS. Perform DFS at a returning when it reaches b, marking all visited nodes. Perform DFS at b returning when it reaches a, marking visited vertices. Since the graph is connected, every node was either only marked by a, only marked by b, or marked by both a and b. Let x be some node only marked by a and let y be some node only marked by b. It is the case that any path from x to y passes through both a and b since b couldn't reach x without passing through a and a couldn't reach y without passing through b. The solution is the number of nodes only marked by a multiplied by the number of nodes only marked by b.

E - Dice Auction

Problem Setter: *Astra Kolomatski,* ***Validator:*** *Tanzir Pial*
Editorialist: *Tanzir Pial* ***Tag:*** *Game Theory*

There are a number of different solutions for this problem. The first observation is that the opponent has a 4 sided dice whereas you have a 6 sided dice, this tips the balance in your favor as you have more information. A simple strategy for winning is: always bet your roll + expected value of opponent's roll. Since opponent's expected value is 2.5, you can bet your roll + 3. This strategy is sufficient for winning the game.

F - Stop the overflow

Problem Setter: *Zafar Ahmad,* ***Validator:*** *Yimin Zhu,*
Editorialist: *Zafar Ahmad, Tanzir Pial* ***Tag:*** *Geometry*

It is a simple geometry problem where you need to find the volume of the water jar ($\pi r^2 h$) and divide it by the area of the water tap opening (πR^2). Finally, you can print the output rounding to the third decimal using printf-like directives. Make sure you use larger data types for floating point numbers that have at least 32 bits (e.g., *double* and *long double* in C++), otherwise you will run into precision issues. In competitive programming it is a good practice to never use data types that have less than 32 bits for floating point calculations.

G - String Problem

Problem Setter: *Jiarui Zhang,* ***Validator:*** *Tanzir Pial*
Editorialist: *Tanzir Pial* ***Tag:*** *Hashing, Z-Function*

This was the hardest problem in this problemset. This can be solved using either Hashing or Z-Function. We need to divide the given string S in this format: $(AB)^iC$. We iterate over all possible lengths L of AB and then iterate over i in a nested loop. Then we can check:

- If the prefix of the length $L \times i$ of S can be represented as a repeated pattern of length L. This part can be done in $O(1)$ using hashing or Z-function.
- We also need to check the number of possible A where $F(A) \leq F(C)$. This can be done in O(1) too by keeping two cumulative array (one starting from the beginning, one starting from the end) where we keep track of number of alphabet characters having odd frequency.

The overall complexity of the solution is $O(n \times H(n)) \leq O(nlogn)$ where $H(n)$ denotes the n-th harmonic number.

H - RothLand

Problem Setter: *Tanzir Pial,* ***Validator:*** *Shawn Matthew,*
Editorialist: *Tanzir Pial* ***Tag:*** *Math, Binary Search*

The number of ducks is D initially and rate of increase in the number of kittens is P% per year. Let's divide P by 100 and then we can use the formula of compound interest here to get that after X years, the number of ducks will be: $D \times (1 + P)^X$

Now we can do a binary search over X to find out which is the lowest value of X that satisfies the following equation:

$D \times (1 + P)^X \geq N$ (Where N is the number of people given in NCS)

So the time complexity is $O(\log^2 ans)$. Now if you take the highest end of your binary search range too large, then you may get TLE. You can do some calculation to find that answer can never be more than around 5000 for the given constraints.

Another way to solve the problem would be to use a direct formula. From the above formula, we can come to this: $X \geq \frac{\log(\frac{N}{D})}{\log(1+P)}$

We can get the minimum value of X then. But you have to take care of the cases where $P \leq N$ i.e. the answer is 0.

Since the number of test cases were huge (250k), there was a lot of file input and output operations. The number of test cases were kept higher to prevent linear solutions from passing, but it also unfortunately gave TLE verdicts to some of the correct solutions. This was fixed during the contest by increasing the time limit.

In competitive programming, it is a good practice to decrease the time needed for file I/O as much as possible especially when the number of test cases are huge. There are resources online for faster I/O for competitive programming.