Algolympics 2024 Finals
Solution Discussion

April 6, 2024

- Input parsing problem.
- Just implement the solution described in the problem statement.

H – Rainbow Energy

- Just do it!
- Try all subsets of size m and directly compute the luminosity of each one
- $\binom{n}{m} < 2^n$ but $n = 15$ only
- $O(n \cdot 2^n)$

F – Meat Sudoku

- How many 4×4 sudokus do you think there are?
- Not too hard to show the number is, say, < 1000 by pen and paper (and not too hard to improve this bound)
- Or, just enumerate all of them using complete search to get an exact value (use backtracking... or 16 nested `for` loops, idk)

There are only 288 different 4×4 sudokus

F – Meat Sudoku

- Feasible to, for each test case, just iterate over all 288 different 4×4 sudokus.
- For each one, count the number of places a “wrong clue” was placed in the given puzzle.
- Use a counter to tally the number of sudokus for which the number of wrong clues is $\leq k$.
- $O(288 \cdot 16) = O(1)$ per test case.

E – Minimum Cost Spanning Subgraph

- Don't let the problem statement goad you into overcomplicating things :))
- Imagine *rearranging the vertices in a line* $c_1, c_2, ..., c_n$ (where the c_i are the technology ratings in sorted order). We just want a large enough a so that adjacent vertices have an edge.
- The condition we want to satisfy, as written in the statement is: "there is an edge between c_i and c_{i+1} for all i".
- Answer: $\max_{1 \le i < n} |c_i - c_{i+1}|$, can be computed with a single $O(n)$ loop.

(Yes, this is in fact **not** a graph problem.)

J – Reservoir Doggos

- We want to order n jobs, where job i takes t_i time to do, and a cost of v_i is incurred per second until job i is finished.
- Intuitively, it would be better to prioritize jobs which are short to finish (low t_i) which mitigate the most cost (high v_i).

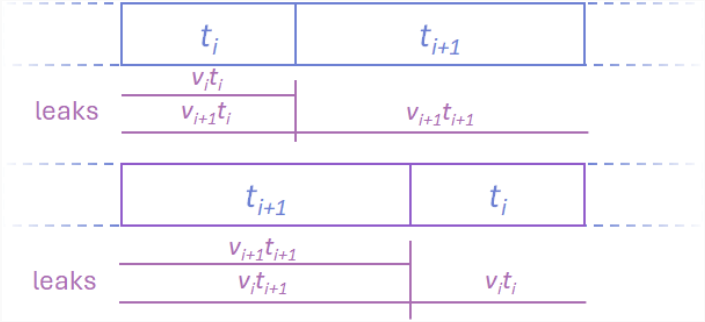
Solution

Greedy: Sort jobs by decreasing $\frac{v_i}{t_i}$.

- Sorting the jobs can be done in $O(n \lg n)$
- Computing the total leaks can be done in $O(n)$ by considering the jobs in reverse-time order.

J – Reservoir Doggos

- Can derive sorting order
- Suppose we have two consecutive jobs i and $i+1$, and swapping the two jobs would make the total leaks *worse*.



- The total leak in this time interval before the swap is $v_i t_i + v_{i+1} t_i + v_{i+1} t_{i+1}$
- The total leak in this time interval after the swap is $v_{i+1} t_{i+1} + v_i t_{i+1} + v_i t_i$

J – Reservoir Doggos

- If the jobs are worse after the swap:

$$v_i t_i + v_{i+1} t_i + v_{i+1} t_{i+1} \leq v_{i+1} t_{i+1} + v_i t_{i+1} + v_i t_i.$$

- Simplifying,

$$v_{i+1} t_i \leq v_i t_{i+1}$$

- Equivalently,

$$\frac{v_i}{t_i} \geq \frac{v_{i+1}}{t_{i+1}},$$

  which means the jobs are sorted in decreasing $\frac{v_i}{t_i}$.

- What if the only command was `COUNT A <i> <j>`?
- Standard segment tree / Fenwick tree problem.
- So... just make 26 segment trees! (or Fenwick trees)

I – Ready Player Juan

- Let $\texttt{opt}(i)$ be the minimum challenge if we are only fighting bosses i and onwards.
- We have a decision here—do we fight the ith boss head-on, or in an ambush?
- After this decision, we must minimize the challenge from fighting bosses $i+1$ and onwards, *which is of the same form as our original problem*.
  - Even if all values are scaled by $\times 2$, the optimal decisions would still be exactly the same.
  - So, the minimum challenge would still be the same, but scaled up by $\times 2$.
- This gives us a recurrence which is $O(n)$ with memoization.

$$\texttt{opt}(i) = \min \begin{cases} t_i + \texttt{opt}(i+1) & \text{(attack head-on)} \\ a_i + 2 \cdot \texttt{opt}(i+1) & \text{(ambush)} \end{cases}$$

K – Space Colonizers

- Never bad to explore a new cell whenever possible.
- Let p be your current power rating. You can enter a cell if and only if...
  - $p > x_{i,j}$
  - $p + x_{i,j} \geq \max_{(a,b) \in \text{ neighbors}} x_{a,b}$
- Combined into one condition,

$$p \geq \max(x_{i,j} + 1, \max(x_{a,b}) - x_{i,j})$$

K – Space Colonizers

- Let $w_{i,j} = \max(x_{i,j} + 1, \max(x_{a,b}) - x_{i,j})$
- Among all cells in the "frontier" (adjacent to a cell you've visited), consider the cell with the minimum $w_{i,j}$.
  - If you can visit it—why not!
  - If you can't visit, then you can't visit *any* of the other cells on the frontier, either, and you're stuck. Exit now.
- Can be solved with basically just a BFS, but using a priority queue instead of a queue to find the next cell to visit.

M – Yet Another Arbitrary Polynomial Problem

Prerequisites:

- Algebraic geometry
  - Hasse's theorem on elliptic curves
  - Mordell-Weil theorem
  - Mazur's torsion theorem
- Gröbner basis
- Quintic Diophantine equations

...just kidding 😜

M – Yet Another Arbitrary Polynomial Problem

- I have no idea what to do.
- When all else fails, brute force for small cases.
- Here are all solutions with $1 \leq a, b, c \leq 500$ (just use three nested for loops).

```
13 42 14
17 69 7
104 84 56
136 138 28
351 126 126
459 207 63
```

M – Yet Another Arbitrary Polynomial Problem

- Use your Intelligent Human Mind 🧠 to notice the following pattern:

```
13 42 14
                                    17 69 7

104 84 56
                                    136 138 28

351 126 126
                                    459 207 63
```

- $(104, 84, 56) = (8 \times 13, 2 \times 42, 4 \times 14)$
- $(136, 138, 28) = (8 \times 17, 2 \times 69, 4 \times 7)$
- $(351, 126, 126) = (27 \times 13, 3 \times 42, 9 \times 14)$
- $(459, 207, 63) = (27 \times 17, 3 \times 69, 9 \times 7)$

M – Yet Another Arbitrary Polynomial Problem

The polynomial

$$2023ab^3 + 1858abc + 1084670664a^2 = 2024b^2c^2 + 1966b^4c + 36051705c^3$$

- **Recurring idea in math:** If I have a solution, can I use it to generate more solutions?
- **Observation:** If (a, b, c) is a solution, then so is $(k^3a,\ kb,\ k^2c)$ for any positive integer k.

Proof: Plugging in $(k^3a,\ kb,\ k^2c)$ gives

$$2023k^6ab^3 + 1858k^6abc + 1084670664k^6a^2 = 2024k^6b^2c^2 + 1966k^6b^4c + 36051705k^6c^3$$

$$k^6(2023ab^3 + 1858abc + 1084670664a^2) = k^6(2024b^2c^2 + 1966b^4c + 36051705c^3)$$

If (a, b, c) satisfies the original equation, then the equation will still be satisfied after we multiply k^6 to both sides.

M – Yet Another Arbitrary Polynomial Problem

- Do we have primitive solutions to “seed” our algorithm?
- Yes! We found them earlier!
- $(13, 42, 14)$ and $(17, 69, 7)$
- Output these solutions, scaled up by various values of k, and you will have solved the problem 🎉.

C – Deal Breaker

- You have f flaws, and n subsets $A_1, A_2, ..., A_n$ of the f flaws.
- Each set A_i has an associated desirability rating a_i.
- Each query is a set of dealbreakers B. Find i such that
  - A_i and B are *compatible* ($A_i \cap B = \varnothing$)
  - a_i is maximized.
- **Observation:** Note that $A_i \cap B = \varnothing \iff A_i \subseteq \overline{B}$. We need to find a *subset* A_i of B which maximizes a_i.
- **Observation:** $f \leq 20$. The time complexity of the solution can be exponential in f.

C – Deal Breaker

Intuition

Use DP. Subproblem relation: $A_i \subseteq X$ iff $A_i = X$ or $A_i \subseteq X \setminus \{e_j\}$ for some flaw e_j.

- Suppose the f flaws are $e_1, e_2, ..., e_f$.
- Let $\texttt{opt}(X)$ be the maximum a_i such that $A_i \leq X$, and $-\infty$ if no such i exists. Then $\texttt{opt}(X)$ satisfies:

$$\texttt{opt}(X) = \begin{cases} -\infty & \text{if } X = \varnothing \\ \max\left(a_i, \texttt{opt}(X \setminus \{e_1\}), ..., \texttt{opt}(X \setminus \{e_f\})\right) & \text{if } X = A_i \text{ for some } i \\ \max\left(\texttt{opt}(X \setminus \{e_1\}), ..., \texttt{opt}(X \setminus \{e_f\})\right) & \text{otherwise} \end{cases}$$

- Given a query B, output $\texttt{opt}(\overline{B})$.
- **Implementation:** Can represent sets X with bitmasks.
- **Complexity:** $O(f2^f)$ time, $O(2^f)$ space.

G – Pandemic

- **Observation:** When someone participates in a handshake, their status flips from Healthy to Infected (or vice-versa)
- Since each person always participates in $n - 1$ handshakes, their final status is fixed, **regardless of order**.
- The game is rigged!

G – Pandemic

- **Observation:** The original status of everyone can be recovered by “undoing” the handshakes in the input
- The final statuses will then **always** be either the same as or the opposite as the original statuses (depending on the parity of $n - 1$)

G – Pandemic

- If the desired statuses are not equal to the predetermined statuses, the answer is 0
- If they are, then *any* permutation of the remaining $\frac{n(n-1)}{2} - m$ handshakes will be valid
- So, in that case, the answer is

$$\left(\frac{n(n-1)}{2} - m\right)!$$

G – Pandemic

How to compute $x!$ if x is big, though?

- If $x \geq \texttt{MOD}$, then $x! \equiv 0 \pmod{\texttt{MOD}}$
- So we only care about the case where $x < \texttt{MOD}$
- Here, we must use... **the forbidden technique**

G – Pandemic

- Let $y \leq x$. If you know $y!$, then $x!$ can be computed in $x - y$ steps:

$$x! = x \cdot (x-1) \cdot (x-2) \cdot \cdots \cdot (y+1) \cdot y!$$

- So, **precompute offline and then hard-code** the answer to $y!$ at various breakpoints
- A sensible choice is evenly-spaced with intervals of size $\approx \sqrt{\texttt{MOD}}$
- The file size limit is 256 kB (DOMjudge will tell you this if you try to upload something too large), and this is plenty generous

L – SwapSwap++++

- Given c lines: `SWAP <i> <j>`, find two permutations of the lines π_1 and π_2 so that the resulting programs produce the same output.
- **Observation**: Suppose we can rearrange d of the c lines so that those d lines result in the same output in either arrangement. Then we can move those d lines to the start of the program, then not touch the remaining lines.

Example. Note that the commands `SWAP 1 3` and `SWAP 3 1` are interchangeable. So if these two lines occur in the program, we can use them to generate two permutations of lines which produce the same output.

	Original Swap++		Permutation π_1		Permutation π_2
1	`SWAP 1 2`	5	`SWAP 1 3`	2	`SWAP 3 1`
2	`SWAP 3 1`	2	`SWAP 3 1`	5	`SWAP 1 3`
3	`SWAP 3 2`	1	`SWAP 1 2`	1	`SWAP 1 2`
4	`SWAP 4 5`	3	`SWAP 3 2`	3	`SWAP 3 2`
5	`SWAP 1 3`	4	`SWAP 4 5`	4	`SWAP 4 5`
6	`SWAP 5 1`	6	`SWAP 5 1`	6	`SWAP 5 1`

Observation

If there are two lines which perform identical swaps, then these can be permuted.

Example. Note that the commands `SWAP 1 2` and `SWAP 4 5` are interchangeable. So if these two lines occur in the program, we can use them to generate two permutations of lines which produce the same output.

	Original Swap++		Permutation π_1		Permutation π_2
1	`SWAP 1 2`	1	`SWAP 1 2`	4	`SWAP 4 5`
2	`SWAP 3 1`	4	`SWAP 4 5`	1	`SWAP 1 2`
3	`SWAP 3 2`	2	`SWAP 3 1`	2	`SWAP 3 1`
4	`SWAP 4 5`	3	`SWAP 3 2`	3	`SWAP 3 2`
5	`SWAP 1 3`	5	`SWAP 1 3`	5	`SWAP 1 3`
6	`SWAP 5 1`	6	`SWAP 5 1`	6	`SWAP 5 1`

Observation

If there are two lines which perform swaps on two disjoint pairs of elements, then these can be permuted.

L – SwapSwap+++++

View each line `SWAP <i> <j>` as an edge (i, j) in a graph with n vertices.

We already found constructions for:

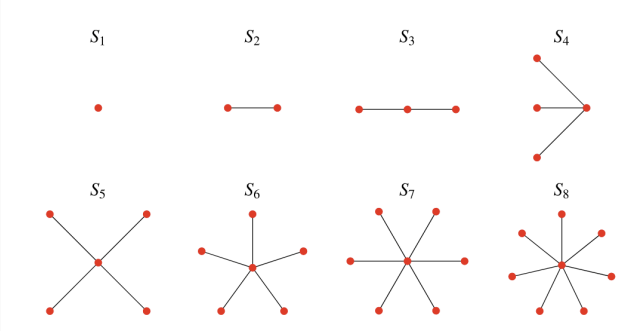
1. if there are two lines which perform swaps on two disjoint pairs of elements (**if the graph is not simple**)
2. if there are two lines which perform swaps on two disjoint pairs of elements (**if the graph has two disjoint edges**)

There are not many different kinds of graphs which don't satisfy either condition.

Observation (left as exercise)

If the graph contains a triangle (e.g. `SWAP 1 2`, `SWAP 2 3`, `SWAP 3 1`), then these can be permuted.

If we have a simple graph with no disjoint edges and no triangles, then the graph is a *star graph*.



This is where the problem *actually* gets tricky.

If the graph is a star graph, then the given program looks something like this...

```
SWAP 1 2
SWAP 1 3
SWAP 1 5
SWAP 1 4
SWAP 1 7
SWAP 1 8
```

You have a central index i that's constantly being swapped with other distinct indices j.

From here, we have to consider the elements $a_1, a_2, ..., a_n$.

The following can be proven and are left as an exercise. Assume the underlying graph is a star graph.

- (all distinct) If all $a_1, a_2, ..., a_n$ are distinct, then it is **impossible** to form permutations π_1 and π_2.
- (three of a kind) If there are three indices x, y, z where $a_x = a_y = a_z$, then it is **possible** to form permutations π_1 and π_2.
- (two pair) If there are four indices w, x, y, z where $a_w = a_x$ and $a_y = a_z$, then it is **possible** to form permutations π_1 and π_2.
- (one pair) If there are only exactly two indices x, y where $a_x = a_y$, then it is **impossible** to form permutations π_1 and π_2.

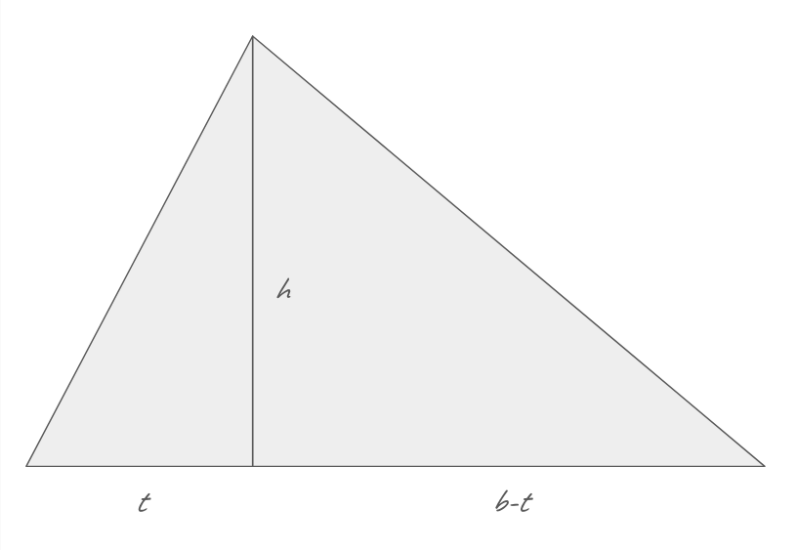
D – Look, Up in the Sky!

- Problem is reducible to **counting** the number of points (C, D) that yield the desired height.
- Then, divide by n^2.
- Remember to simplify to lowest terms.

D – Look, Up in the Sky!

- Use similar triangles to find a closed form expression of E's altitude in terms of (x_c, y_c) and (x_d, y_d).

D – Look, Up in the Sky!



D – Look, Up in the Sky!

- Similar triangles tell us the following:

$$\frac{y_c}{x_c} = \frac{h}{t}$$
$$\frac{y_d}{b - x_d} = \frac{h}{b - t}$$

  which lets you solve for t and h.

- From here, you can arrive at the following relationship.

$$\frac{x_c}{y_c} + \frac{b - x_d}{y_d} = \frac{b}{h}$$

A tangent:

- You have two arrays of integers c and d (each of length n). Answer the following query: Given s, count the pairs of indices (i, j) such that

$$c_i + d_j = s.$$

- Doesn't this sum somewhat resemble polynomial multiplication?
- Let $M = \max(c, d)$. This problem can be solved in $O(1)$ per query after an $O(M \lg M)$ pre-processing using generating functions.

D – Look, Up in the Sky!

- Let $C(x)$ be a polynomial such that the coefficient of x^t is the number of times that t appears in c.
  - Or, equivalently, $C(x) = \sum x^{c_i}$
- Define $D(x)$ similarly using d
- Then, the answer to the query is the coefficient of x^S in $C(x)D(x)$.
- Use FFT to quickly multiply the two polynomials in $O(M \lg M)$
- This looks like our original problem! We just need to transform the terms into integers.

D – Look, Up in the Sky!

- Let our arrays c and d be

$$c_i = x_i \frac{60}{y_i}$$
$$d_j = (b - x_j)\frac{60}{y_j}$$

  and let the s of each query be $60b/h$ (if this is an integer).
- Here, we multiply both sides by $60 = \mathrm{lcm}(1, 2, 3, 4, 5, 6)$ so the terms in the LHS are always integers.
- If $60b/h$ is not an integer, the count is 0.
- Let $Y = \mathrm{lcm}(1, 2, 3, 4, 5, 6)$. Then, this solution runs in $O((bY)\lg(bY))$