2023 North America Championship Solutions

The Judges

May 27, 2024

Arrested Development

Problem

- You are given n tasks and each task must be assigned to one of two interns.
- Based on the task and intern, each task may take a different amount of time. Assign the tasks to minimize T, which is the maximum of the sum of task times assigned to each intern.
- Bounds: $1 \leq n \leq 50$, $1 \leq a, b \leq 10^5$.

Arrested Development

Solution

- We can solve this problem with dynamic programming. Specifically, define $f(i, k)$ to be the minimum sum of times that the second intern can be assigned given that the first i tasks are assigned and the first intern's sum of task times is exactly k. In the event it is not possible for the first intern to have k minutes worth of tasks, we define $f(i, k) = \infty$.
- We can then iterate over $f(n, k)$ and compute $\min_k \max(k, f(n, k))$.
- If we let k be the maximum time of one task, we can compute this DP in $\mathcal{O}(n^2 k)$.

Champernowne Substring

Problem

- Consider the infinite string obtained by writing the positive integers in increasing order.
- You are given a regex of length at most 25, find the first match of the regex within the infinite string.

Champernowne Substring

Solution

- Due to the small bounds of the problem, we can solve the problem with brute force on the length of the first number that touches the regex and how many digits of the first number do not appear within the regex.
- There are a lot of tricky cases to handle, so some things that may aid in the implementation include special handling of the following cases:
  - Cases where the first number is less than 100.
  - Cases where a carry is observed to a new power of 10.

Solution, continued

- After handling the above cases, we can reduce the given problem to: Given two regexes of x and $x+1$ where both numbers have k digits, and x is not $9 \pmod{10}$, compute the minimum value of x that is compatible with both regexes.
- It may help to brute force the last two digits of the first number.
- In this reduced problem, all digits are effectively uniquely determined so there is no casework.

Comparator

Problem

- You are given a series of statements in a programming language called IFFY, used for defining comparators on bitstrings.
- Bounds: $0 \leq n \leq 2 \cdot 10^5$, $1 \leq k \leq 10$.

Comparator

Solution

- First, parse each expression and convert it to a truth table on x and y.
- There are only 2^4 such tables an expression can evaluate to, so any IFFY program only has at most $16k^2$ useful statements.
- Then evaluate the program on every pair of bitstrings after filtering out the useless statements. In expectation, each invocation will only use 4 statements, so this process takes $4 \cdot 4^k$ time.
- Use bitsets to compute the transitive property quickly.
- Overall runtime is $O(n + |s| + 4 \cdot 4^k + 8^k)$, but with a low constant factor thanks to the bitsets.

House Deconstruction

Problem

- You are given n people and m houses on a circle.
- Let $f(S)$ be the minimum cost of a matching between the n people and a set S of n houses, where the cost between the person and a house is the distance along the circle.
- Find both the minimum value of $f(S)$ and the number of sets S that achieve that minimum value.
- Note we are only counting the number of sets of houses, not the number of ways to assign people to houses.

Solution

- Draw an arc from each person to their matched house. You can prove the following statements with exchange arguments:
  1. no arcs will intersect,
  2. there are no unmatched houses with arcs going over them, and
  3. if one arc is nested within another, then both travel in the same direction.
- With these criteria, there are only two potential houses that each person could be matched with.
- We can find these houses using a stack, and build a graph using only these edges.
- Because there are strictly more houses than people, each component in this graph is a line graph.
- All possible costs on a line graph can be computed in linear time using prefix / suffix sums.

Magic Bean Cube

Problem

- You are given a puzzle comprising 30 beads, 10 of three different colors, and the ability to rotate beans along one of four cycles.
- Given a puzzle layout, figure out how to get all ten beads of a given color into their assigned cycles with at most 240 moves.

Magic Bean Cube

Solution

- We can define a primitive on one of the colored cycles by rotating it once clockwise once, rotating the shared cycle clockwise once, rotating the colored cycle counterclockwise once, and rotating the shared cycle counterclockwise once.
- This has the effect of swapping two beads in specific locations and reordering the other beads within the two cycles that are touched.
- Therefore, with six moves, we can move one bead from an incorrect cycle to a correct cycle, and in at most 180 moves we can use this primitive to solve the puzzle.

Manhattan Walk

Problem

- You are in the top-left corner of an $r \times c$ grid, where each square in the grid has an arrow pointing either down or right along with a timer uniformly at random from $[0, p]$. The arrow changes directions when the timer runs down to zero and resets to p.
- You can only move in the direction specified by the arrow in the square you are in.
- You can only see the arrow in the square you are currently in.
- Compute the expected time you wait for arrows while traveling to the bottom-right square with an optimal strategy.

Solution

- Define $f(x, y)$ to be the answer for a grid with x and y columns. Note that $f(x, y) = f(y, x)$.
- We can see that $f(1, n) = \frac{n-1}{4}$ since with probability $\frac{1}{2}$ we can travel directly to the right, and otherwise we wait $\frac{p}{2}$ seconds in expectation.
- $f(i, j)$ is clearly dependent only on $f(i-1, j)$ and $f(i, j-1)$ otherwise, let the values of those be a and b with $a < b$.
- Clearly, if we are able to transition to a directly, we should. Otherwise, we should wait only if waiting would take less time than going straight to b.
- The rest of the math is left as an exercise to the reader.

MountainCraft

Problem

- A mountain is defined by a peak at point (x, y), and by line segments that are parallel to the lines $y = x$ and $y = -x$ that go down to the y-axis, possibly clipped on the left by $x = 0$ and the right by $x = w$.
- Given a collection of mountains, we wish to compute the sum of the lengths of segments of mountains that are visible. If two mountains overlap in area, the affected segments are not visible.
- Operations consist of adding and removing mountains. After each operation, output the sum of lengths of visible segments.

MountainCraft

Solution

- If we project all segments down to the x-axis, we see that the answer is equal to $\sqrt{2}$ times the length of segments that are covered from above by some mountain.
- To solve this problem, we can maintain a segment tree. On a given range, we maintain the minimum number of segments observed at some point within the segment, along with the sum of lengths of segments covered within.
- Though it may seem like lazy propagation is necessary, it suffices to take a given segment and just break it up into a maximal collection of disjoint segments on the segment tree, and to track how many segments fully cover a given node.
- This is the same data structure used when computing the union of area of axis-aligned rectangles that may overlap.

Not Another Constructive!

Problem

- You are given a partial string. Change each question mark to some letter such that the resulting string has exactly k instances of `NAC` as a subsequence.

Not Another Constructive!

Verifying if a given string has k instances of NAC

- It is possible to naively count the number of subsequences in $\mathcal{O}(n^3)$ by just using three for loops.
- However, we can do this in $\mathcal{O}(n)$ by counting the number of subsequences exactly equal to `N`, `NA`, and `NAC`. When we see an `N`, we increment the first count by one. When we see an `A`, we increment the number of `NA`'s by the number of `N`'s. When we see a `C`, we increment the number of `NAC`'s by the number of `NA`'s.

Not Another Constructive!

Solution

- Naively, we can recursively backtrack on the string, keeping track of the prefix that has been fully determined, the number of `N`, `NA`, and number of `NAC` subsequences.
- This is too slow, but with memoization there are at most $\mathcal{O}(n^7)$ states, since we have to track the length of the prefix we have seen so far for a factor of $\mathcal{O}(n)$, the number of `N` subsequences for a factor of $\mathcal{O}(n)$, the number of `NA` subsequences for a factor of $\mathcal{O}(n^2)$, and the number of `NAC` subsequences for a factor of $\mathcal{O}(n^3)$,
- Most of these states are actually not reachable. If k is low, we can prune the search if the number of subsequences gets too high. If k is high, we can prune the search if it is impossible to get up to k subsequences theoretically even if all future characters are unconstrained.

Passport Stamps

Problem

- You have a passport with n pages. You will take a trip where on each trip, t_i contiguous pages are stamped and unusable in the future. Compute the first trip where you are at risk of not having enough contiguous pages.

Passport Stamps

Solution

- To check if the k-th trip is at risk, we see that $n - \sum_{i=1}^{k-1} t_i$ pages will be available, and there are at most k sections that they can be in. If $\left\lfloor \frac{n - \sum_{i=1}^{k-1} t_i}{k} \right\rfloor \geq t_k$, then it is guaranteed to be fine.
- We can therefore check k in increasing order.
- Note that the condition is not binary searchable!

Problem

- Ashley is solving n problems.
- Her skill can be modeled with two integers (s, t).
- A problem can only be solved if $l_i \leq s \leq r_i$ and $l_t \leq t \leq r_t$.
- After solving a problem, she may increase one of her skill points by 1.
- What is the maximum number of problems she can solve?

Solution

- Keep a set of (s, t) values that Ashley can achieve before solving a problem.
- For each such value, if she can solve the problem, then add $(s + 1, t)$ and $(s, t + 1)$ to the set.
- Implemented naively, this runs in $O(n^3)$, which is too slow for the given bounds.
- This complexity can be improved by removing elements from the set after processing them.
- Since the bounds are large, it is recommended to use a bitset to store the states and find elements in the set quickly.
- With proper implementation to only iterate over set bits, this reduces the runtime to $O(n^2)$.