2022 North Central Regional Solutions

The Judges

Feb 25, 2023

Repetitive Song

Problem

- Compute the length of the longest subsequence that appears more than once in a list of strings.

Initial Observations

- If every string is unique, then no subsequence appears more than once.
- Therefore, the only strings that matter are strings that appear more than once.
- If two equal strings are adjacent, the answer is $n - 1$.

Repetitive Song

Solution

- We are therefore looking for the closest pair of strings that are equal.
- Loop over the strings in order and maintain a map from string to the rightmost index where we have observed it.
- If we see a string more than once, update the minimum distance in indices we have seen.
- If the minimum distance between indices is d, the answer is $n - d$.

Exponent Exchange

Problem

- Alice has x dollars and Bob has $b^p - x$ dollars. In one operation, one person can give b^x dollars to the other. What is the minimum number of operations k such that, if both Alice and Bob and permitted to perform k operations, one person ends up with b^p dollars?

Initial Observations

- If the number of dollars Alice and Bob each have is divisible by b, then there is no reason for either person to give b^0 dollars to the other.
- In general, if the number of dollars both people is divisible by b^x, the only moves they should make should involve amounts greater than or equal to b^x.
- It also doesn't make sense for Alice and Bob to both give the other b^x dollars at any point.

Exponent Exchange

Solution

- We iterate on x from 0 to $p-1$, at that stage we assume that Alice and Bob will exactly operate using b^x dollars and both Alice and Bob have dollars divisible by b^x.
- We use dynamic programming. The state we maintain is whether Alice has more or less money relative to her starting amount, and the number of operations she has performed. We map this state to the mininum number of operations that Bob needs to perform.
- Naively, there are too many states to maintain. To prune the number of states, note that as the number of operations Alice performs increases, the number of operations Bob does must decrease. Pruning states that are in violation of this makes this run in time.

Chocolate Chip Fabrication

Problem

- You want to make a chocolate chip cookie. In a given turn, you add some cookie squares to your existing cookie. A given square can only be added if it is on the boundary of the cookie or if some adjacent square is not yet filled with a cookie. Compute the minimum number of turns needed to construct the chocolate chip cookie.

Initial Observations

- It seems difficult to know which squares we can fill in first.
- However, if we consider the last turn, we know which squares cannot be filled in on the last turn - any square which is surrounded by cookie on all four sides must be filled in prior to the last turn.
- Therefore, we can consider the reverse process of “eating" the cookie in the minimum number of turns, where a cookie square can be eaten if it is on the boundary or some adjacent square is empty.

Chocolate Chip Fabrication

Soluton

- We can solve this other problem using breadth-first search. All squares that are on the boundary or have some adjacent square empty are initialized to a turn counter of 1, and all other squares are set to a turn counter of infinity.
- We maintain a queue of squares we are processing, initialized with the squares that have a turn counter of 1.
- Remove a square from the queue, and if any adjacent squares have a turn counter of infinity, update the turn counter to 1 more than the current turn counter, and append that square to the queue.
- The answer is the maximum turn counter over all squares.

Advertising ICPC

Problem

- A grid of letters is *advertising ICPC* if a 2×2 subgrid spells out ICPC. Count the number of ways to fill in missing letters in the grid such that the grid is advertising ICPC.

Solution

- Even though the grid is small, there are too many ways to fill in blank grid squares for recursive backtracking to work.
- However, if we fill in the squares in row-major order, we note that the only letters which matter are the previous $c + 1$ letters, and whether a 2×2 subgrid spells out ICPC.
- There are 3^{c+1} ways for the previous $c + 1$ letters to be arranged, and we can use dynamic programming to maintain the transitions as we add letters.
- Challenge: Can you solve it in $\mathcal{O}\left(2^{\min(r,c)}\right)$?

Sun and Moon

Problem

- The sun and the moon align for an eclipse occasionally. It was d_s years ago when the sun was last in the right place, and d_m years ago when the moon was last in the right place. The sun is in the right place once every y_s years, and the moon is in the right place once every y_m years. When will the next eclipse happen?

Solution

- We are guaranteed that an eclipse will happen in the next 5000 years.
- Therefore, we can check the years starting from one year in the future and check if the sun and moon will be in the right place - y is a valid year for an eclipse if $(y + d_m)$ is divisible by y_m and $(y + d_s)$ is divisible by y_s.
- There is a faster solution using the Chinese Remainder Theorem, but this was not required to solve the problem.

Streets Ahead

Problem

- Given a list of strings and multiple pairs of strings, compute for each pair of strings how many strings are between the paired strings.

Solution

- It is too slow to do a linear scan for each string for each query pair.
- Instead, we maintain a map from string to the index it is at in the list.
- We then output $|a - b| - 1$, where a and b are the indices of the two strings in the pair.

Restaurant Opening

Problem

- Given a grid of integers g where the *cost* of location (i, j) is $\sum_{x=1}^{n}\sum_{y=1}^{m} g_{xy}(|i-x|+|j-y|)$, compute the minimum possible cost over all locations in the grid.

Solution

- The grid is small, so we can brute force all locations.
- To compute the cost for a given location, we can have two nested loops to loop over all locations in the grid to accumulate the costs that the various grid locations contribute.

Triangle Containment

Problem

- You are given a bunch of weighted points (x, y) in the plane. For each point, its value is defined as the sum of the weights of the other weighted points strictly inside the triangle defined by it, $(0, 0)$, and $(b, 0)$. Compute the value of every point.

Initial Observations

- n is too large to directly check, for each point, which points are strictly inside the induced triangle - it is possible to construct $\mathcal{O}(n^2)$ pairs where one point is inside the induced triangle by another point.
- If we sort the points by their directed angle θ_i around the origin, note that in order for point i to have point j inside its triangle, $\theta_j < \theta_i$.
- By similar logic, if we sort the points by their directed angle α_i around $(b, 0)$, we get a similar relation.

Triangle Containment

Solution

- Sort the point in reverse order by angle around $(b, 0)$.
- Looping over all points in this given order, we see that the points inside the current triangle must precede the current point. However, those points must also have θ smaller than the current point.
- We can maintain a segment tree keyed on index in the θ_i sort order. When we see point j, report the sum of all points seen so far with smaller θ, and then activate that point in the segment tree.
- Due to the large numbers, exact integer arithmetic must be used when sorting points by angle. This can be done by using cross products.

Digits of Unity

Problem

- Count the number of sets of n positive integers each less than or equal to m where the bitwise AND of all the integers in the set has at least k bits turned on.

Solution (High-Level)

- The number of subsets is far too large to enumerate, even with backtracking.
- m is small though, so we could enumerate all possible bitwise ANDs that can result.
- We need to use the principle of inclusion-exclusion to handle overcounting.

Digits of Unity

Solution (Details)

- We need to precompute factorials and inverse factorials modulo 998244353. We can do the factorials in linear time directly. We can compute one inverse factorial by leveraging Fermat's Little Theorem, then compute the rest by observing $\frac{1}{i!} = \frac{i+1}{(i+1)!}$.
- We can also precompute, for an integer x, the number of ways to select a subset of y elements from a set of x elements where $k \leq y < x$.
- We can then enumerate all possible bitwise ANDs, counting the number of integers less than or equal to m that have all those bits turned on.

Bog of Eternal Stench

Problem

- You are given a weighted, directed graph. You start at vertex 1 and travel some edges to get to vertex n. When you traverse an edge with weight t, you gain t units of stench. Your stench can never be negative. What is the minimum stench you can end up with?

Bog of Eternal Stench

Solution

- Note that the answer is binary searchable. To verify if we can get to vertex v with at most k units of stench, we construct a new graph where an edge going from a to b with weight w in the old graph corresponds to an edge from b to a in the new graph with weight $-w$. This corresponds to reversing the process described in the problem, so traveling an edge in the new graph is akin to going back in time.
- We can now run Bellman-Ford in this graph, where we start at vertex n with k units of stench. We can traverse an edge in the new graph if and only if our stench would have been nonnegative. We maintain the maximum stench we can have in the graph conditioned on ending at vertex n with k units of stench.
- If we see a positive cycle in the graph, we can set all maximum values in the cycle to positive infinity. It is possible to end at vertex n with k units of stench if and only if it is possible to get to vertex 1 at all.