2022 Pacific Northwest Division 1 Solutions

The Judges

Feb 25, 2023

Three Dice

Problem

- You are given a list of three-letter words. Is it possible to construct three dice such that, for each word, it is possible to arrange the dice in such a way that the top faces can form the word? All 18 possible letters on the three dice must be distinct.

Initial Observations

- If a word has two or more identical letters, it is impossible.
- If 19 or more distinct letters appear over all words, it is impossible.
- If fewer than 18 distinct letters appear, we can pick arbitrary unique letters that do not appear to fill in the other faces.
- If letters α and β appear in the same word, they must appear on different dice.

Three Dice

Solution

- If the faces and dice are all distinguishable, there are 18! ways to arrange the letters.
- The faces of a die are indistinguishable before adding letters, so we can divide out a factor of 6!.
- The dice are also indistinguishable before adding letters, so we can divide out a factor of 3!.
- This leaves us with $\frac{18!}{(6!)^3 \cdot 3!} = 2858856$ combinations to try.
- We can use recursive backtracking to enumerate and try all of these, pruning when an assignment is clearly invalid.
- Though not necessary to solve the problem, we can recursively try assigning the letters that have the most constraints first to prune the search space.

Alchemy

Problem

- You are given a string of lowercase letters. In a single operation, you take two adjacent characters and mutate both of them. Compute the minimum number of operations needed to make the string a palindrome.

Initial Observations

- If the outermost characters match, neither should be changed.
- If the outermost characters do not match, it is not always optimal to make them match with one operation! The sample case `vetted` shows this - we need more than 2 operations if we make the outermost characters match, but we can do 2 operations by doing `vetted` to `gutted` to `guttug`.

Alchemy

Solution

- We can solve this with dynamic programming. We can reduce this problem to the following - you are given a binary string where in a single operation, you look at two adjacent indices - a 1 must be flipped to a 0, whereas a 0 can stay as either a 0 or be changed to a 1. Your goal is to make the string be all 1's.
- To solve this problem, you can maintain for a state of the form (length of prefix that is all 1's, whether the next bit has been forcibly flipped) the minimum number of operations needed to get to that state.
- To convert the original problem to this reduced one, construct the binary string from left to right by looping over pairs of characters in the original string from the outside going to the middle, adding a 1 if the characters match and a 0 if they don't.

Champernowne Count

Problem

- The nth Champernowne word is obtained by concatenating the first n positive integers in order. Compute how many of the first n ($1 \leq n \leq 10^5$) Champernowne words are divisible by k ($1 \leq k \leq 10^9$).

Solution

- n is large enough that it is not practical to store the integers using arbitrary precision integers.
- However, k is small, so we can maintain each Champernowne word modulo k.
- When transitioning from the nth Champernowne word to the $(n+1)$th, we can multiply by 10^s and add $(n+1)$, where s is the number of digits in $n+1$. This should be maintained modulo k.
- Be careful about integer overflow, 64-bit integers suffice.
- Challenge: Can you solve this for small k but very large n?

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.

Color Tubes

Problem

- You have $n + 1$ tubes each with the capacity to hold three balls. There are $3n$ balls distributed among the tubes, three balls each of n distinct colors. In a single move, you can take a ball from one tube and move it on top of all the other balls in a tube that has fewer than three balls in it. In $20n$ moves or fewer, get all tubes to be either completely empty or have all three balls of some color.

Solution

- There are many different approaches to get this to happen within $20n$ moves. We'll outline one approach that fills in the left n tubes. This solution will operate in multiple phases.

Initialization

- We start by emptying the rightmost tube, arbitrarily moving balls from there into tubes to the left that have space. This takes at most three moves.
- We proceed by making tube 1 be monochromatic, at which point future moves will not interact with it at all. We need to be able to perform this in fewer than 20 moves due to the overhead we incurred.

Making the Leftmost Tube Monochromatic

- Let the bottom ball in the leftmost tube have color c. We will move all balls with color c into this tube.
- If the tube is already monochromatic, we're done.
- If the topmost ball has color c and the middle one doesn't, we can reverse the two balls as follows:

Making the Leftmost Tube Monochromatic, continued

- Let the leftmost tube be l, the rightmost tube with balls be r, and the empty tube be e. Move a ball from r to e, the topmost ball with color c into e, the middle ball from l to r, the topmost ball with color c from e to l, and the last ball from e back to l. This takes five operations.
- Now, it remains to move balls from other tubes into the leftmost tube.
- If such a ball is not the bottom-most ball in its tube, we can remove the incorrect balls out of tube l into e, any balls above that ball into e, and then move that ball directly into l. Moving all balls back into e, this takes at most seven moves to fix one ball.
- If such a ball is the bottom-most ball in its tube, we can reverse the entire tube by moving all balls into tube e, at which point we can apply the above logic to move balls out of l until we can take the (now topmost ball) from e and move it into l. This takes at most eight moves.

Food Processor

Problem

- You have n different blades. Blade i can cut pieces of size at most m_i, cutting them in half in h_i seconds. Blades reduce the size at an exponential rate. Compute the minimum number of seconds needed to convert food that is originally size t to size s.

Solution

- For a given piece size, we want to use the blade with the minimal h_i rate. We can ignore blades where $m_i \leq s$ or $m_i > t$.
- We need to be able to solve the equation $t \cdot 0.5^{\frac{x}{h_i}} = s$ for x. Taking logarithms, we can show that $x = \frac{h_i \cdot \log\left(\frac{t}{s}\right)}{\log 2}$.
- We need to reevaluate the best blade for all m_i values in $[s, t]$. We can do this by maintaining the blades sorted by their m_i values. It is too slow to enumerate all eligible blades for each check.

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.

Branch Manager

Problem

- In a rooted tree, people navigate through the tree by always traveling to the descendant with the lowest ID. n people start at the root and wish to get to specific destinations, traveling through the tree in order. Before each person starts traveling, you can permanently delete some edges from the tree. Compute the index of the first person who cannot make it home.

Initial Observations

- Use the Euler tour technique to represent the tree. Specifically, DFS through the tree in sorted order of children. Let s_v be the time when we first see vertex v in the DFS, and let e_v be the time when we return from vertex v in the DFS.
- We are therefore looking for the first vertex v where there exists a vertex u appearing before v in the destination order list where $e_v < s_u$.

Branch Manager

Solution

- If we compute the Euler tour of the tree, we can simply loop over the destination vertices in order, track the maximum s_v we have seen, and see when some e_v is less than the maximum e_v seen prior.
- Note that it is not strictly necessary to compute the Euler tour beforehand and then loop over the destination vertices in order. We can perform a preorder traversal of the tree. Prior to returning from the recursive call from a vertex v, we can visit any vertex that is in the call stack of the DFS, so we can loop over destination vertices until we see one we cannot visit.

Counting Satellites

Problem

- Find a string of length at most 5×10^3 that contains k ($1 \le k \le 10^{18}$) subsequences of the word `SATELLITE`.

Solution

- We show one solution that generates strings using around 4×10^3 characters at most. It is possible to do better.
- For convenience, we will actually construct k subsequences of the reverse, `ETILLETAS`. We can reverse the string to get our desired answer. We do this because both 'S' and 'A' appear exactly once in the string. After reading through the full solution, note that we can do better by not giving 'S' special treatment.

Counting Satellites

Solution, continued

- Create eight blocks - block i will consist of 2^{i-1} copies of each letter in ‘ETILLET‘ in order. The eight blocks will be separated by various numbers of `A`’s. The last character in the string will be `S`.
- After block 8, each instance of the letter `A` will contribute $8711794301899425 \approx 8 \times 10^{15}$ subsequences of the form `ETILLETA`. In general, we can show that each block will be separated by at most 300 `A`’s.
- The other letters alone comprise roughly 1800 characters, and $1800 + 8 \times 300$ easily fits in the given bound.

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.

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 $\mathcal{O}\left(2^{\min(r,c)}\right)$?

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.

Lone Knight

Problem

- Given an infinite chessboard with n ($1 \leq n \leq 10^3$) rooks, quickly answer queries of the form - can a knight hop from square (x_1, y_1) to (x_2, y_2)?

Initial Observations

- If a chessboard has dimensions at least 4×4, a knight can get from any square to any other square.
- If a chessboard has at least two rows, then within a given row, if a knight can visit some square, it can visit the square four squares to its left or right.
- n rooks divide the chessboard into $\mathcal{O}(n^2)$ sub-chessboards.

Lone Knight

Solution

- Within a sub-chessboard with at least two rows and two columns, squares are indistinguishable when their rows and columns are equivalent modulo 4.
- We can use a disjoint set data structure to maintain the connected components after reducing by parity.
- To handle the special case where a sub-chessboard has exactly one row or column, two squares are in the same component only if that component has size strictly greater than 1.