2022 Pacific Northwest Division 2 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?

Hunt the Wumpus

Problem

- Generate locations for four wumpuses on a grid, then simulate playing a game where you try to find them in the grid.

Solution

- This problem requires carefully following the rules stipuated in the problem. There are several things to be careful for.
- One tricky part is making sure that the four locations of the wumpuses are distinct. There are many ways to implement this - one can use a set or maintain a boolean array of size 100 to see which locations have been filled in.
- After that, carefully simulate the process to see if a location contains a wumpus. If a location is hit, make sure to remove the wumpus from that location.
- Be sure to print out all the messages exactly as written.

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.

Fading Wind

Problem

- Simulate a paper airplane flying with a fading wind.

Solution

- This problem requires carefully following the rules stipulated in the problem. It should suffice to translate the rules directly into code.
- The easiest way to compute $\left\lfloor \frac{x}{10} \right\rfloor$ is to use integer division. In languages like C++ and Java, `x / 10` when x is a positive integer will automatically give the result rounded down. In Python 3, `x // 10` will do the same.

Creative Accounting

Problem

- You are given an array of n integers. Your goal is to partition the array into subarrays of size k (except for possibly the first and last subarray) such that as many subarrays as possible have positive sum. Though $n \leq 3 \cdot 10^4$, k can only take on 10^3 distinct values.

Initial Observations

- Because k can only take on a small number of values relative to n, this hints at brute-forcing all possible valid values of k.
- If we precompute prefix sums - specifically $f(i)$ is the sum of the first i integers in the array for $0 \leq i \leq n$, we can compute the sum of all elements in an arbitrary subarray in $\mathcal{O}(1)$ time. Specifically, the sum of the subarray starting at index a and ending at index b is exactly equal to $f(b+1) - f(a)$.

Creative Accounting

Solution

- For a fixed starting point and a subarray size k, we can compute the number of subarrays with positive sum in $\mathcal{O}\left(\frac{n}{k}\right)$ time.
- Checking all k possible starting points for a subarray of size k therefore takes $\mathcal{O}(n)$ time.
- Checking all possible values of k, this algorithm therefore runs in $\mathcal{O}(nk)$ time.

I Could Have Won

Problem

- Alice and Bob are playing rock-paper-scissors - they each earn points with the first to earning k points winning a game, and points resetting to zero after. For what values of k does Alice win more games than Bob?

Solution

- Because the number of total points won by both Alice and Bob is at most $2 \cdot 10^3$, we can brute force all values of k up to the total number of points earned.
- We can directly simulate the result for a fixed value of k by maintaining the current count of points earned by both individuals as well as the number of games won by both individuals.

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.

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.

Distinct Parity Excess

Problem

- Define the *prime parity* of an integer k to be the number of distinct primes that divide k. For multiple queries $[a, b]$ compute the difference between the number of integers in $[a, b]$ with even prime parity and the number of integers in $[a, b]$ with odd prime parity.

Solution

- Start by using the sieve of Eratosthenes to compute all primes.
- For each prime p, loop over all its multiples and update the prime parity for all such multiplies with respect to p.
- Let $f(i)$ be the desired answer for the range $[0, i]$ - $f(i+1)$ is one larger than $f(i)$ if $i+1$ has even prime parity, and one smaller otherwise.
- The answer to a query $[a, b]$ is therefore $f(b) - f(a-1)$.

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.