# 2022 Mid-Central Regional Solutions

The Judges

Feb 25, 2023

# Blueberry Waffle

## Problem

- A waffle maker rotates 180 degrees every $r$ seconds. A blueberry waffle is inserted with the blueberries pointing up. After $f$ seconds, the waffle maker stops and rotates strictly fewer than 90 degrees back to horizontal. Are the blueberries pointing up or down?

## Solution

- The waffle maker makes a full rotation every $2r$ seconds. Therefore, we can take $f$ modulo $2r$. If $f$ is less than $\frac{r}{2}$ or greater than $\frac{3r}{2}$, then the blueberries are pointing up. When $f$ is equal to $\frac{r}{2}$ or $\frac{3r}{2}$, which is not allowed in the problem, the waffle maker is exactly vertical. When $f$ is greater than $\frac{r}{2}$ and less than $\frac{3r}{2}$, the blueberries are pointing down.

# 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 $\theta_i$ 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 $\alpha_i$ 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 $\theta$ smaller than the current point.
- We can maintain a segment tree keyed on index in the $\theta_i$ sort order. When we see point $j$, report the sum of all points seen so far with smaller $\theta$, 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.

# Everything is a Nail

## Problem

- You are given a ternary array. You are to construct a ternary array where all 0's are contiguous, all 1's are contiguous, and all 2's are contiguous. Maximize the number of indices where your constructed array matches the given array.

## Solution

- There are $\mathcal{O}(n^2)$ different ternary arrays you can construct, so checking all of them is too slow.
- However, if we construct our ternary array from left to right, the only information that matters is what integers have been used so far in our constructed ternary array and what the last added element is.
- Therefore, with dynamic programming, we can maintain the maximum number of integers we can match conditioned on having assigned the first $i$ integers, the set of ternary integers we have used so far, and what the $i$th integer in our ternary array is.

# Champernowne Count

## Problem

- The $n$th 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 $n$th 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 $n$?

# 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.

# 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.

# 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.

# Advertising ICPC

## Problem

- A grid of letters is *advertising ICPC* if a $2 \times 2$ 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 $2 \times 2$ 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 $\mathcal{O}\left(2^{\min(r,c)}\right)$?

# 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.

# 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.

# 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.