2022 North American Qualifier
Solution Outlines

The Judges

Feb 5, 2023

Problem A - Beast Bullies

Problem

- Given N animals of various strengths, an *attack* is where the weakest animal is targeted by some other animals, and will be driven away unless enough animals come to its defense. Each animal wishes to not be driven away, but also wants as many other animals to be driven away as possible. If all animals act optimally, how many remain?

Initial Observations

- When $N = 2$, since all animal strengths are unique, the weaker animal will always be driven away.
- When $N = 3$, if the two weaker animals cannot match the stronger animal in strength, they will both be driven away. However, if they do match the stronger animal, then from the analysis when $N = 2$, we see that no animals are driven away in this case.
- This motivates looking at the animals from strongest to weakest.

Problem A - Beast Bullies

Solution

- Sort the animals in decreasing order of strength, maintaining the current stable set of animals, which starts with just the strongest animal.
- Looping over the other animals in decreasing order, maintain a candidate set of animals. The moment that the candidate set of animals has sum of strengths greater than or equal to the current stable set, merge that set into the stable set and reset the candidate set.
- The answer is the final size of the stable set.
- Note that the explicit sets do not need to be maintained, it suffices to maintain the number of animals and the sum of the strengths.

Problem A - Beast Bullies

Proof of Optimality

- We can prove the correctness of this algorithm using a proof by strong induction on the number of times the stable set is updated.
- Since we are consistently removing the weakest animal, we can instead consider the reverse process and figure out which subset of the strongest animals left remaining can be added to the set.
- By the induction hypothesis, since the existing set of animals is stable, if more animals are to be added to the stable set, the new animals that are added must sum in strength to be at least the sum in strength of the current stable set.
- This stable set must necessarily be minimal in size.

Problem B - Birthday Gift

Problem

- Count the number of integers equivalent to $b \pmod{225}$ that have exactly a digits and have no two adjacent digits being equal.

Initial Observations

- If we build the number digit by digit, we can update the residue modulo 225 by multiplying by 10 and then adding the digit.
- Therefore, there is an $\mathcal{O}(a)$ DP where the state is the residue of the current number modulo 225 and the last digit used in the number. This DP has 2250 states.
- We can optimize this DP using exponentiation by squaring, but this requires multiplying matrices that are 2250×2250 in size, which is too slow.

Problem B - Birthday Gift

Solution

- Note that $225 = 25 \times 9$. The residue of a number modulo 25 only depends on the last two digits.
- Therefore, if we manually loop over all possible candidates for the last two digits, we can reduce the number of states to 90 - the last digit of the number and the current residue modulo 9.
- We still need to optimize the DP with exponentiation by squaring. The complexity is $\mathcal{O}(90^3 \log a)$.

Problem C - Class Field Trip

Problem

- Given two strings of lowercase letters, print the sorted version of the two strings, concatenated with each other.

Solutions

- Because the strings are short, there are many different viable solutions.
- Solution 1: Concatenate the two strings into some list of characters, sort the list with a library sort, and print it out.
- Solution 2: Find the smallest character over both strings, print it, and remove it. Repeat until both strings are empty.
- Solution 3: Maintain a frequency count of each letter, then loop over the letters in order, printing each letter the number of times it appears,
- Solution 4: Do the 'merge' step in merge sort on the two strings to sort them.

Problem D - Ghost Leg

Problem

Read a "Ghost Leg" permutation and output the permuted order.

Solution

- The Ghost Leg input is really just a step-by-step description of the items that must be swapped in order to create the permutation.
- Read n and m from the first line of input.
- Create an array of integers, P, of length n and initialize it such that $P_i = i$.
- For each of the m remaining lines, read a and swap P_a with P_{a+1}.
- Output the elements of P.

Problem E - MazeMan

Problem

Given a maze with multiple "dots" and multiple "entrances" from which each player can enter through, calculate the minimum number of players necessary to eat all the **reachable** dots, and how many dots are not reachable because they are walled off.

Solution

- Set a global counter to be zero.
- As you iterate from entrance 'A' to entrance 'W', check how many "dots" you can eat (e.g, you may use BFS/DFS or DSU to do this).
- If the count is nonzero, increment the global counter by one and replace each visited "dot" with "space".
- Report the global counter together with a count of the remaining "dots".

Problem F - Metronome

Problem

- A metronome ticks 4 times for every 1 revolution of the winding key.
- For a given song of a given length, how many revolutions must the key be wound?

Solution

- Just divide the input by 4.
- Be sure to use reals, not integers.
- In Java:

```
Scanner sc = new Scanner(System.in);
PrintStream ps = System.out;
ps.println(sc.nextDouble()/4.0);
```

Problem G - Movie Night

Problem

- You want to go with the movies with some of your friends. However, a given person will only go to the movies if their best friend also goes. Compute the number of distinct sets of friends that can go to the movies with you.
- The set must be nonempty, and because the number of sets can be very large, the answer should be printed modulo $10^9 + 7$.

Initial Observation

- Treat the people as vertices and best friendship as a directed relationship.
- This graph is known as a *functional graph*, where every node has outdegree exactly one.

Problem G - Movie Night

Functional Graphs

- Functional graphs always contain at least one cycle. To see this, pick an arbitrary starting vertex and go to the vertex it is pointing to. By the pigeonhole principle, after repeating this $N + 1$ times, you will have seen some vertex twice. This vertex must be in the cycle.
- Functional graphs can contain multiple cycles. Different cycles are independent.

Solving The One-Cycle Case

- What happens if the graph has exactly one cycle?
- Every vertex in the cycle must be selected.
- What about vertices that are not in the cycle but point towards it?

Problem G - Movie Night

Solving The One-Cycle Case, Continued

- For now, we'll consider a vertex v that points directly to a vertex in the cycle, and only consider vertices that eventually point to v.
- Let $f(i)$ be the number of valid subsets of vertices that contain vertex i, considering only vertices that eventually point to i.
- We can show that $f(i) = 1 + \prod_{g \in P} f(g)$, where P is the set of parent vertices of i - vertices that directly point to i.
    - If person i does not go, then no one who considers person i their best friend can go and there is only one valid such subset in this case.
    - If person i does go, then everyone who considers person i their best friend can independently attend or not.
- Therefore, for a given cycle, the number of nonempty subsets that are valid is $\prod_{v \in S} f(v)$, where S is the set of all vertices not in the cycle that point directly at the cycle.

Problem G - Movie Night

Full Solution

- It remains to solve the problem when there are multiple cycles in the graph.
- Let $g(C)$ be the number of possibly empty valid sets of vertices only considering vertices in the weakly connected component C. This is equal to the quantity computed prior, incremented by one to handle the case where we don't select any vertices.
- The total number of valid sets is therefore the product of $g(C)$ over all weakly connected components in the original graph, minus one to exclude the empty set.

Problem H - Platform Placing

Problem

- Given a sorted sequence x of n integers, compute the maximum possible sum of a sequence y of n nonnegative real numbers that satisfy the following constraints:
  - $s \leq y_i \leq k$
  - $\frac{y_i + y_{i+1}}{2} \leq x_{i+1} - x_i$ for $1 \leq i < n$.
- Output -1 if no such sequence exists.

Initial Observations

- If $x_{i+1} - x_i < s$, then no such sequence exists.
- Otherwise, a sequence is guaranteed to exist by setting all $y_i = s$.
- All y_i values, except for y_1 and y_n, are present in two inequality constraints relating to some x_i.

Problem H - Platform Placing

Solution

- Set all $y_i = s$.
- In increasing order of i, increase y_i as much as possible while still respecting the inequality constraints.
- We can show that this is correct by an exchange argument with an optimal solution, as the prefix sum of y must always stay ahead of the prefix sum of an optimal sequence.

Problem J - Room Evacuation

Problem

- Given a room with some people and some exits, where people can only move one square at a time and no square can have more than one person in it at a time, what is the maximum number of people that can exit in T time units?

Motivating the Solution

- Imagine that we look at the room over the different time units - for every person that escapes, these people must be at pairwise disjoint locations at every point in time.
- Modeling this as a graph, we're looking for the maximum number of vertex-disjoint paths, which can be modeled as the maximum number of edge-disjoint paths in a similar graph.
- This problem can be solved by computing a maximum flow.

Problem J - Room Evacuation

Solution

- We'll construct a graph with $T + 1$ layers. Each layer will effectively simulate the state of the room at that time.
- For each layer, we construct $2RC$ nodes, two nodes for each location in the room. We'll define an 'in' node and an 'out' node, and connect the 'in' node to the 'out' node with an edge of capacity one.
- Between each layer, we'll connect the 'out' node of the previous layer with the 'in' node of the next layer if the nodes correspond to the same or adjacent locations.
- Connect every 'out' node of every exit with the sink, and the source to the 'in' node of every location with a person in layer 0.
- The maximum flow on this graph is the answer.

Problem K - Smallest Calculated Value

Problem

- Given three integers and the four basic arithmetic operators with no precedence, what's the smallest nonnegative number you can compute?

Solution

- Compute a list of all the possibilities of the first two.
- For each, figure out the possibilities with the last.
- Find the smallest ≥ 0.

Problem K - Smallest Calculated Value

Solution in Java

```
private List<Integer> ops(int x, int y) {
  ArrayList<Integer> result = new ArrayList<Integer>(4);
  result.add(x+y);
  result.add(x-y);
  result.add(x*y);
  if(x%y==0) result.add(x/y);
  return result;
}
// some code omitted
ArrayList<Integer> values = new ArrayList<Integer>(16);
for(int x: ops(a, b)) values.addAll(ops(x, c));
int smallest = Integer.MAX_VALUE;
for(int x: values) if(x>=0 && x<smallest) smallest = x;
ps.println(smallest);
```

Problem L - Spidey Distance

Problem

- In the spidey distance metric, traveling from (x, y) to a horizontally or vertically adjacent point is 1 unit, but traveling to a diagonally adjacent point is 1.5 units.
- Consider all points at most s spidey distance units away from the origin, and count these S. Of these points, count the number of them T that are also in the taxi (Manhattan) distance t.
- Compute $\frac{T}{S}$ in reduced form. One way to do this is to find the greatest common factor and divide both by the greatest common factor. We can use the Euclidean algorithm for this.

Problem L - Spidey Distance

Observations

- To find either area of points, we notice that there is symmetry over both the x and y axis, so we can compute for one quadrant, making sure to properly add in the points on an axis, multiply by 4 and add the origin.
- The taxi distance in the first quadrant will be a triangle, specifically the points under the line $y = -x + t$. We can calculate this with $\frac{t(t-1)}{2}$. Note: this does not include either axis.
- For an individual point, it is inside the taxi distance if $x + y \leq t$.
- The spidey distance will always contain the points in the taxi distance of the same value, and will (with values greater than 3) have additional points further out from the origin. This is not obvious to calculate.
- For an individual point, it is inside the spidey distance if $\min(x, y) \cdot 1.5 + \max(x, y) - \min(x, y) \leq s$.

Problem L - Spidey Distance

Observations, continued

- If s is at least as large as t, then all taxi points are contained in the spidey area, so we can compute the two and reduce.
- If s is less than $\left\lfloor \frac{t}{3} \right\rfloor$ then all the spidey points are contained inside the taxi points, and the answer is 1.
- If the distance values fall between these, then things are a bit more challenging, as there are spidey points that are outside the taxi distance, and there are taxi points that are outside the spidey distance.
- Notice that on the diagonals, for sufficiently large spidey distances, starting with the axis, you move two columns further before the next diagonal out starts to have points on it. So taxi points outside the spidey distance can be calculated.
- Notice that the number of spidey points on a diagonal with y-intercept of y, outside the triangular area is $s - 3(y - s)$.

Problem L - Spidey Distance

Solution Approaches

- We can iterate over every point in the first quadrant where x or y is inside the spidey distance and test if the point is inside the spidey distance, and then inside the taxi distance. This will be too slow as it is $\mathcal{O}(s^2)$.
- We can iterate over every point in the first quadrant on the band of points outside the triangular portion of the spidey distance. This is still $\mathcal{O}(s^2)$.
- We can calculate the number of spidey points in a row/column by starting at s and tracing the border around, then comparing the values to the taxi distance on that row/column. This is $\mathcal{O}(s)$.

Problem L - Spidey Distance

Solution Approaches, continued

- We can calculate the number of spidey points in a row/column by using the formula and applying as many diagonal steps as possible, then comparing this data to the taxi area on that row/column. This is $\mathcal{O}(s)$. (Solution 1)
- We can use the observations above about the number of points on a diagonal that are or are not part of the spidey distance, and the knowledge of the y-intercept to know if these points are part of the taxi distance. This is $\mathcal{O}(s)$. (Solution 2)
- We can look for a closed form for the number of spidey points, and then find a way to count the number of taxi points outside the spidey area. The closed form is hard to find, but it can be computed in $\mathcal{O}(1)$.

Problem L - Spidey Distance

Solution 1

- For each x in $[0, s]$, we'll find the maximum possible y that we can reach in spidey distance.
- If we fix the number of diagonal steps we do and perform those first, note that the region we can reach if we only go up and to the right is a right triangle.
- As the number of diagonal steps increases, the length of the hypotenuse decreases but its distance from the origin increases.
- With this observation, we can find all such y values in $\mathcal{O}(s)$ time by looping over the number of diagonal steps in decreasing order and only setting the maximum y value for x-coordinates we have not yet solved for.

Problem L - Spidey Distance

Solution 1, continued

- By symmetry, we can compute the minimum possible y values attainable. We can reflect across the y-axis to compute this information for negative values of x.
- We can now loop over x from $-t$ to t, counting the number of values inside the bounding box.
- We can compute the exact probability by taking the GCD of these two values to get the probability as a fraction in simplest form.

Problem L - Spidey Distance

Solution 2

- To calculate the number of spidey points in the first quadrant, we will count the number of points in the taxi distance of s and add to that the total: `for(i=1; s-3*i > 0; i++) total += s-3*i`.
- To calculate the number of taxi points we will count the number of points in the taxi distance of s and add to that the total: `for(i=1; s+i <= t; i++) total += s-3*i;`
- Remember the number of points in the taxi distance of s is $\frac{s(s-1)}{2}$.
- To get the total number of points, we take either of the above, add s (for a single axis), multiply that value by 4 (for the four quadrants), and then add 1 for the origin.
- Once we have these two values, we reduce to get our answer.

Problem M - Toll Roads

Problem

- Given an undirected weighted graph and multiple queries of pairs of vertices, for each pair of vertices report two values: w, the minimum weight such that it is possible to travel between the two vertices using edges with weight at most w, and k, the number of vertices it is possible to reach starting at either vertex and using edges with weight at most w.

Solving the $Q = 1$ case

- Sort the edges in the graph, and use a disjoint set data structure maintaining the size of each component.

Problem M - Toll Roads

Solution 1 - Augmenting the $Q = 1$ solution

- It is too slow to naively check each query after sorting the edges and adding them incrementally.
- To do this more quickly, for each vertex, we annotate which queries use a given vertex as an endpoint. When we merge two components together, we check if both components share a common query, and merge these annotations otherwise.
- When checking and merging the annotations, operate on the smaller set.

Problem M - Toll Roads

Solution 2 - Minimum Spanning Tree + Disjoint Set

- Note that edges which are not in the minimum spanning tree are irrelevant to any query.
- We can therefore compute the minimum spanning tree first and then for every query, compute the maximum weight edge on the path between those two vertices. We can do this with an augmented sparse table in the same way that we can compute LCA of two nodes quickly.
- After this, we can run the $Q = 1$ solution again but check the size of the connected component after merging in all edges with weight at most w.

Problem M - Toll Roads

Solution 3 - Parallel Binary Search

- An alternate solution to the $Q = 1$ case is to binary search for the minimum possible value of w using a disjoint set to check whether two vertices are connected.
- We can extend this solution when Q is large by doing *parallel binary search*, where we simultaneously binary search for the answer to all queries at the same time.
- To do this, in a single iteration of binary search, establish candidate w values for each query that we are checking for. Add the edges in sorted order by weight and when all edges with a specific weight have been added, check all queries that have candidate w value equal to that weight and adjust the binary search boundaries accordingly.