COP 4516 Spring 2023 Week 15 Team Final Contest Solution Sketches

Problem A: Connect the Dots

This problem doesn't have too much problem obfuscation. The problem statement insinuates that the solution involves trying each permutation of visiting the other dots, and then checking to see if any of the segments intersect, not counting the vertices of the polygon. We can do this by skipping over checking a segment's intersection with the previous and next segments in the list of segments. Most of the difficulty in this question is in stitching together the implementation of both brute force and line segment intersection together correctly to solve the problem.

Problem B: USACO Grade

This problem is meant to be the banger in the set. Read in the information and set up an appropriate if statement and output. Arrays and a loop can be used to clean up the solution, but there are so few cases, it doesn't matter.

Problem C: Grow Tree Grow

The problem is asking for cycle detection, which can be achieved with a disjoint set. Specifically, create a disjoint set. As each edge gets added, run the union function. The first time the union function returns false (meaning that both nodes were already connected), is the first time a cycle occurs. The bounds are small enough that alternative methods to solve the problem work as well. (You can "search from scratch" for each subgraph to see if it has a cycle via counting number of connected components, for example.)

Problem D: Counting Oranges in Orange County

The operations in the problem strongly insinuate the use of a binary index tree, since we can either change the number of oranges on a tree, or query a contiguous range of trees. This takes care of half of the problem, leaving the primality query. To solve this, a second BIT can be used where we put one in an index if that tree has a prime number of oranges or 0 if it does NOT have a prime number of oranges. The trick is maintaining this BIT quickly. Since the description states that no tree will ever have more than 10^7 oranges, we can pre-compute all the primes up to 10^7 before processing the data via a prime sieve. Then, whenever we change the number of oranges on a tree, we must look at the old number of oranges and the new number of oranges to determine if we must make any changes to the bit storing primality. There are two situations where we must change this BIT: when the old number of oranges was composite and the new number is prime, OR if the old number was prime and the new number is composite. In the first situation we add one to the appropriate index. In the second situation, we subtract one. This is easiest to do by just maintaining the actual array itself, but not using it for queries. (The judge solution has two BITs, one for sum queries, one for primality queries and one array, maintaining the values.)

Problem E: Order Statistics

Use an ordered set to store all unique values. Then, extract the values and map them to indexes 1 to u, where u is the number of unique values. Create an array of u lists, where the i^{th} list will store each index, in sorted order, of the i^{th} smallest value. To populate this list, go through the values in the original list a second time, this time using the map to figure out which smallest value each item is, and then add its index to the back of the appropriate list of indexes. For example, if our unique values are 2, 3, 5, 6 and 8, and we come across a 3 in index 6, then we want to add 6 to list 2, since 3 is the second smallest value. To handle queries, just go to list k, looking for the value (if it exists) stored in index m.

Problem F: Scientist

We want to match scientists to virus outbreaks, and want the maximum matching. Thus, we can set up a network flow graph to solve the problem. We create a vertex for each scientist and each virus outbreak. We add an edge from the source to each scientist with capacity 1. We add an edge from each virus outbreak to the sink with capacity 1. Then, we add an edge between each scientist and virus outbreak, if that scientist can extinguish that virus outbreak. In order for a scientist to be able to extinguish a virus outbreak, the scientist must be able to reach it, AND that virus must be on its list of viruses that scientist can extinguish. We can check the former via BFS or DFS (the most we have to run is 26 from the scientists, which will run pretty fast), and the latter is just a lookup, probably most easily implemented via a Boolean array or bitmask.

More than likely, just like Connect the Dots, the difficulty here isn't problem obfuscation, but implementation. There are quite a few moving pieces to all get correct.

Problem G: Senior Design

This is meant to be the second easiest question in the set. Follow the directions given. Sort the array, and then look at the array from index k to index n-k-1, inclusive. Add these values. This is the numerator of the answer (not reduced). The denominator is simply n – 2k. To reduce the fraction, find the GCD of these two values. Then, divide both by this GCD and display the answer.

Problem H: Subway Surfers

This looks to be a DP similar to the Tri question from team contest #5. Unfortunately, one complication is that you can go back and forth on a row. (In the Tri question, you could always come from "previous" spots, so to speak.) One way to view this DP is by stating that dp(i, j) equals the maximum number of coins you can get starting at row i, column j. In this case, what you want to try to do is get as many coins as you can on your row (this involves a bunch of casework looking for '#' and 'c'), and then move forward to the next row. In moving forward, you'll have 0, 1, 2 or 3 options, depending on the board configuration, so you must simply try each possibility, and take the one that leads to the maximum answer. Much like a couple other problems in this set, the implementation of this is main point of difficulty.