COP 4516 Spring 2025 Week 15 Team Final Contest Solution Sketches

Problem A: Dan the Delivery Man

The key observation is that the graph presented is a rooted tree and that the amount of time to travel from a parent node to a child node is 1 unit of time, but that one can instantly travel to any ancestor node. This means that effectively, we incur a cost of 1 unit of time for each edge that has to be traveled. There are several ways to accomplish this. Perhaps the easiest is to trace the path from each delivery spot to the root, simply marking each edge touched. (No recursion is necessary here because the parent array stores where to go.) After marking all edges for all paths from delivery point to the root, just count the total number of edges marked.

Problem B: Fizz or Buzz, Cuzz

This is one of the two bangers in the set. Just use mod and follow the directions (order of the checks given.)

Problem C: Lighting Fuses

As of the time of the writing of these sketches, it remains to be seen if this should have been a 20 point question or not. It seems clear that the problem is close to bipartite matching as we need to match fuses with fireworks. The problem is that there might be more fuses and those extra fuses also have a cost; we can't ignore them. It some sense, all unused fuses are in the same category – we can set them up to match with a single dummy firework. Now the problem becomes what capacities do we assign in our graph? Typically with matching problems we use 1 for our capacities. In particular, if we are able to match two items, then we connect them with an edge with capacity 1, otherwise, we don't connect them.

The trouble here is that we don't immediately know what should be "matchable" (connected) and what shouldn't.

Thus, what we can do is very similar to the Museum Guards problem covered in the lecture notes. We can binary search the time needed (the output value). It must be one of the times in the input value, so instead of running the binary search from 1 to 1,000,000,000 which would take 30 iterations, we can just run the binary search on the values in the input (there are at most 10,100 of these, resulting in at most 14 iterations of the binary search.) The time limit should be generous enough (at least in C++ and Java) to allow either approach. Once we make a guess for a particular time, we create the network flow graph as follows (other designs are possible):

1. Connect the source to each fuse node with capacity 1. All fuses must be matched in the end. (This is n edges.)
2. Connect each fuse to each firework if and only if their pairing time is less than or equal to our target time. These edges should all have capacity 1. (There could be quite a few edges here.)
3. Connect each fuse with an extra dummy firework if the burn time for the fuse on its own is less than or equal to the target time. These edges should have capacity 1.

4. Connect each regular firework node (there are m of these) with the sink with capacity 1. Connect the dummy firework node (absorbing all the unused fuses) to the sink with capacity n – m. **We do this because we must have exactly n – m fuses that are unmatched.**

Run the max-flow algorithm on this graph. If the maximum flow equals n, then that means the real answer is less than or equal to the target value attempted. If the maximum flow is less than n, then the real answer must be strictly greater than the target value tried.

Problem D: Geode Slices

Of all the shapes for the outer most geode, the greedy selection is the convex hull of the given points. Any other polygon will be inside the convex hull and can't contain "more" than the convex hull. This reasoning works recursively so the algorithm is as follows on an initial set S of points.

Set a counter to 0.
While more than 2 points are left in S:

1. Compute the convex hull of S, call this set of points T.
2. Add 1 to a counter for this being a layer.
3. Remove all points in T from S. (So compute the set S – T and reset S to this new set.)

The answer is the value of the counter.

Note: if there are 333 nested triangles, an $O(n^2)$ convex hull algorithm might TLE. But, we were generous with the time limit, so it's possible one of these could pass. We decided that the implementation details here were tricky enough to potentially allow the slower convex hull algorithm.

Since coordinates go up to 10^6, long long in C++ or long in Java must be used. (The code for convex hull looks at the product of the difference between pairs of coordinates, so this product is on the order of 10^{12}.)

Problem E: Missing Contests

This is the other banger in the set. While it's probably not necessary, with questions like this it's always best to avoid division. Mathematically, we want $\frac{m}{n} \geq \frac{p}{100}$. Since all three variables are positive, the former is true if and only if $100m \geq np$, a check which requires no division.

Problem F: Osmosis

The first move of the game is special because no matter which move is made, the multiplying factor will be b. After that, the score of a move is dependent on three things: left endpoint, right endpoint, and which token was moved previously.

From a recursive viewpoint, define `f(left, right, previous)` to be equal to the best score you can achieve if your tokens start on index left and index right, respectively, with previous storing which token was moved previously. (One possible convention is to set previous = 0 if the left token was previously moved and set previous = 1 if the right token was previously moved.)

Taking this together with the beginning observation, the final answer to the question is the maximum of `f(0, n-2, 1) + v[0]*v[n-2]` and `f(1, n-1, 0) + v[1]*v[n-1]`. The first of these terms represents the best score when we move the right token first (from index n-1 to index n-2) and the second term represents the best score when we move the left token first (from index 0 to index 1.)

Of course, if written with traditional recursion, this solution will earn a TLE verdict. Just use memoization to make sure the solution runs fast enough. In addition, the answer can be pretty large, so long long in C++ or long in Java must be used.

Problem G: Returning Papers

This is a standard probability problem from COT 3100. The sample space is the number of subsets of students for which Arup has papers to return, or $\binom{n}{p}$. To count the number of items in this sample space that are "successes", we must count the number of ways we can choose **s** students out of the **a** students who attended AND choose **p – s** students out of the **n – a** students who skipped class. Both sets of choices are independent of one another, so we want their Cartesian Product, thus we'll multiply. It follows that the total number of successes is $\binom{a}{s}\binom{n-a}{p-s}$. Generate Pascal's Triangle under mod for the first 301 rows and then either use modular inverse or modular exponentiation (knowing that $10^9 + 7$ is prime so x^{-1} mod $10^9 + 7$ is really $x^{1000000005}$ mod (10^9+7).) to compute the final answer. Of course, use long long in C++ or long in Java.

Problem H: Rhino Count

While the problem seems like it'll be a difficult implementation problem, all that really needs to be stored is an array where we store the last day number we saw each rhinoceros. Thus, if rhino[7] = 12, this means the last time we saw the rhino with ID number 7 was on day 12. If it's day 17, since 17 – 12 > 4, we'll issue an extra ranger for that day. If it's day 23, since 23 – 12 > 10, we'll do a full search for that rhino. The order of output requested is fairly straight-forward since we can just loop through the rhino array described above.