TJCT Long 1 Solutions

ICT Officers

July 2020

1 Introduction

This write-up will outline the solutions and motivations behind the problems in our first weekend contest (Amrita themed). These problems were meant to introduce you to harder problems in the USACO Silver and Gold levels. Remember that we will cover all of these topics throughout the year, so don't feel discouraged if you don't understand the concept yet. The solutions are given in decreasing order of number of people who solved the problem. If you haven't attempted the problems yet, we highly recommend that you at least give the problems an honest effort before looking at our solutions. Feel free to message us if any part of this write-up is unclear.

2 Problem A: Amrita's Ice Cream Business

Problem Credits: Anirudh Bansal

A simple brute force approach works for this problem. We can try all the possible intervals of heights from (0, 17) to (83, 100). To calculate the cost of an interval, we must consider three cases for each stack. If an elevation is in the interval, the cost is 0. If an elevation x is less than the interval's lower bound a, the cost is $(x-a)^2$. If an elevation x is greater than the interval's upper bound b, the cost is $(x-b)^2$. The total cost of an interval is the sum of the individual costs. We must iterate through each possible interval and find the minimum cost. The total time is $O(N)$.

3 Problem B: Amrita's Autocorrect

Problem Credits: Joshua Zhang

Thir problem required knowledge of Tries. The Trie had to be modified so that each node in the Trie keeps track of how many valid words are in it's subtree. Queries of type one are the same as adding to a normal Trie, but we increment the number of words in the subtree of each node as we go down. Queries of type two can be answered by getting the number of valid solutions in the subtree of the node corresponding to our parameter. Overall time complexity is $O(N \cdot M)$, since we need to query N times and the depth of our Trie is dictated by M.

4 Problem F: Amrita's Bacteria

Problem Credits: Joshua Zhang

This problem was mostly implementation. To begin, we'll store the bacteria as an array of M arrays. Each array in our big array represents a single type of bacteria, and has K strings of length K. Next, we can create a matrix of $N \cdot K$ rows and N columns, this is our answer matrix. We'll iterate over the N by N matrix of bacteria now. For a bacteria on row i and column j, we'll set answer$[K \cdot i ... K \cdot i + K - 1][j]$ equal to the corresponding array of strings for that bacteria type. Then we'll print our answer array, going row by row. Overall time complexity is $O(MK + NK^2)$. The bounds of the problem are such that a well optimized $O(N^2K^2)$ solution would work in time. Note: String concatenation in some languages is very, very slow. Avoid it at all costs!

5 Problem C: Amrita's Fear of Spiders

Problem Credits: Amrita Sahu
This problem was a somewhat straightforward implementation of floodfill and connected components. First, you can floodfill using DFS to find the sizes of each spider/non-spider component. Then, you can iterate through each of the components adjacent to the starting block to find which one will the have largest size when combined with the non-spider blocks it connects to. Remember, the problem is asking you to maximize the totally number of reachable spots, not just the number of spiders blocks that are cleared!

6 Problem D: Amrita's Pyschic Abilities

Problem Credits: Aarav Bajaj
This problem can be solved using offline queries, where we sort the queries in an order that is convenient for us to process, and then return the answers to each query in the correct order. First, we sort the queries in order of increasing r-index. Then we will loop through the array of values and answer each query that has the same r-index as the index we are currently on. We will keep track of the index of the last time each value that appears has been visited. By maintaining a BIT for range sums, we can keep an array that has an element of 1 at an index if that index is the last time we have seen that value, and 0 otherwise. When we see a duplicate value, then will simply switch the previous occurrence of the value to 0, and the new occurrence to 1. The answer to each query is then just the range sum from l to r, since each value in that range will be represented exactly once by the BIT. BITs support range queries and updates in $log(N)$ time, so this gives a time complexity of $O(NlogN)$.

7 Problem I: Amrita's Brain

Problem Credits: Pranav Mathur
We are asked to answer several queries on paths on a tree. If we can modify the problem such that these queries resemble segment tree range queries, we can use Heavy Light Decomposition to answer each query in $O(log(N))$ time.

We will change the problem to apply HLD by processing the queries offline. Sort the queries in decreasing order of k, then process the queries as follows.

1. Begin by labeling all nodes with value 0
2. Before we process a process with craziness value k, update all nodes that have value greater than or equal to k to 1.
3. Calculate the sum of all values from a to b using our decomposition.

Note that this procedure gives the correct answer to each query because we are processing the queries in decreasing order of k, so the only values that will by summed in the tree sum query are the values greater than or equal to k. Total time complexity is $O(N + QlogN)$ (initial decomposition + queries).

Note that since the queries on the tree are sum queries, this problem can also be solved using a DFS and preorder traversal. Both solutions are included in Github.

8 Problem H: Amrita the Boss

Problem Credits: Aarav Bajaj
We will try to maximize the difference between the original and new power of the array. The operation can be thought of as either a right or left cyclic shift. First we'll see an $O(N^2)$ solution and then an optimization to make it $O(NlogN)$. Let l and r be the endpoints of the shift.

For a right cyclic shift, the difference in power will be

$$\Delta_{l,r} = (a_l \cdot (l+1) + a_{l+1} \cdot (l+2) + \ldots + a_r \cdot l) - (a_l \cdot (l) + a_{l+1} \cdot (l+1) \ldots + a_r \cdot r) = a_r \cdot (l-r) + \sum_{i=l}^{r-1} a_i$$

For a left cyclic shift, the difference in power will be

$$\Delta_{l,r} = (a_l \cdot (r) + a_{l+1} \cdot (l) + \ldots + a_r \cdot (r-1)) - (a_l \cdot (l) + a_{l+1} \cdot (l+1) \ldots + a_r \cdot r) = a_l \cdot (r-l) - \sum_{i=l+1}^{r} a_i$$

We can find these values for each possible choice of l and r in $O(1)$ time with prefix sums, giving a time complexity of $O(N^2)$. To optimize, first we rewrite the above equations.

For a right cyclic shift: $\Delta_{l,r} = (a_l \cdot r - sum_r) + (sum_l - a_l \cdot l)$
For a left cyclic shift: $\Delta_{l,r} = (a_r \cdot l - sum_{l-1}) + (sum_{r-1} - a_r \cdot r)$

Notice that the second term of each equation depends only on l and r respectively. Therefore, for each possible value of l in the first equation, the problem simplifies to have n lines (slope r, y-intercept $-sum_r$) and find the maximum value among all lines evaluated at a_l. The same idea applies to the left cyclic shift. This can be done in $O(NlogN)$ with the convex hull trick. For more details, see `http://wcipeg.com/wiki/Convex_hull_trick`.

9 Problem E: Amrita's Bets

Problem Credits: Amrita Sahu
This problem had a couple methods you could use - dynamic programming or finding the closed form expression. This solution will focus on the second method. Furthermore, to find the mathematical expression you can either use Catalan numbers or approach it as a standard permutation problem. Finding the expression for Q in the problem is relatively straightforward, it's the number of distinct ways to arrange W wins and L losses:

$$Q = \binom{W+L}{W}$$

The final number of 10 dollar bills Amrita has must be $M + W - L$ (if this is less than 0, then we already know the probability is 0). To calculate the numerator, we must find the number of arrangements of W wins and L losses such that at some point in the sequence, the number of preceding wins plus M is less than the number of preceding losses. We can do this by imagining a graph of a given sequence, where each "bet" corresponds to either an increase of 1 (W) or a decrease of 1 (L). The graph will begin at $(0, M)$ and end at $(N, M + W - L)$. For a particular sequence to be counted (Amrita cannot pay her debts), it must cross y = -1. If we imagine the rightmost intersection with y = -1, and reflect it over y = -1, we will have a new graph starting at $(0, M)$ and ending at $(N, 2 - M - W + L)$. Furthermore, for each distinct graph that follows those two requirements, there is a corresponding graph in which Amrita cannot par her debts. Thus, we need to find the number of such graphs by finding $\binom{W+L}{A}$ where A is the number of new "wins". You can solve for by setting up a system of equations. Finally, you should end up with an equation that looks like this:

$$P/Q = \frac{\binom{W+L}{L-1-M}}{\binom{W+L}{W}} = \frac{L!W!}{(L-1-M)!(W+1+M)!} = \frac{L(L-1)(L-2)...(L-M)}{(W+M+1)(W+M)...(W+1)}$$

The formula simplifies quite nicely into a form where complexity is O(M). The last step requires finding the modular inverse of Q (which can be done via the extended Euclidean algorithm or a variety of other ways). Remember to keep track of the modular arithmetic throughout the solution!

10 Problem G: Amrita's Town

Problem Credits: Pranav Mathur
This problem is very similar to the 2019 USACO Gold problem "Shortcut". In that problem, there is a single

pun-center at node 1 and Amrita builds one road in attempt to reduce total travel time. To solve this simplified version, we run Dijkstra's algorithm from node 1 to get the shortest path from each node k to node 1. We also keep track of the path taken to node 1, keeping in mind that if two paths are tied, we must pick the lexicographically smallest path (see solution code for implementation details). Now, here's the trick. Using our shortest paths, we determine the number of people p_i that pass through node i on their journey to the pun-center. Note that any person passing through node i has a travel time of at least $dist(i)$ (where $dist(i)$ is the length of the shortest path from node i to node 1). Thus, if Amrita builds a road from node i to node 1, she will reduce the total travel time of all people by $p_i * (dist[i] - T)$. We calculate the maximum of this value over all i to get our answer.

Solving the problem with multiple pun-centers requires a simple modification: instead of running Djikstra's from node 1, we run multi-source Djikstra's from each of the K pun centers. Note that with this change, each house will automatically be assigned to the nearest pun-center and the rest of the algorithm works exactly the same as it does in the single pun-center case. Overall time complexity is $O(MlogN + N)$ (Djikstra + post-Djikstra processing).