ICT June In-House Solutions

ICT Officers

July 2020

1 Introduction

This write-up will outline the solutions and motivations behind the problems in our first Intro In-House Contest (Aarav themed). These problems were meant to introduce you to the concepts commonly tested in the USACO Bronze and Silver Divisions. 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 B: Anirudh's Names

Problem Credits: Pranav

This problem asks to find the number of occurences of a string (in this case, "aarav") in an array. Since the number of elements in the array $N < 10^6$, we can simply loop through the array and count the number of occurences in $O(N)$ time.

3 Problem C: Aarav's Array

Problem Credits: Aarav

This problem asks for the minimum number of integers that must be inserted into the given array such that the median is M. There are several ways to approach this, but the easiest (not the fastest) is while M is not the median, add M to the array. This involves sorting the array each time M is added and then keeping track of how many times M is added until the median is M. This leads to an $O(nlog(n))$ run at most $n/2$ times, so the complexity is $O(n^2log(n))$.

A faster solution is to keep track of how many elements of the array are below M, above M, and equal to M. This allows us to determine immediately whether the median is above M or below M. If the median is above M, add something smaller than M to the array. If the median is below M, add something bigger than M to the array. Note that in this solution, we must also check if M is present or not in the original array, because if not, then we must add it. Since we are only adding elements to the array, the time complexity of this is $O(n)$, which easily passes in time.

4 Problem D: Corona

Problem Credits: Amrita

Each parent has a priority level based on their arrival time and their group. To solve this problem, we can use a priority queue, such that two parents are compared first by their group number and then by their arrival time. This will allow us to "help" or remove the parents from the top of the priority queue, as long as we make sure the total weight at one time does not exceed W. To find the answer, we just need to keep a count of the number of trips made, or the number of times the weight exceeds W (plus the last trip, if there are any remaining parents).

5 Problem E: NBA Fans

Problem Credits: Anirudh

The naive approach to this problem involves independently counting the number of 1's, 2's, and 3's for each interval. This solution's complexity is O($N * Q$), which, given the bounds of this problem, is insufficient.

A different approach uses a technique known as *prefix sums.* This method precomputes the number of fans of each team by subtracting the number of occurrences from the first element to the starting point of an interval from the number of occurrences from the first element to the ending point of an interval ($f(A, B) = f(1, B) - f(1, A)$, where f represents a function that calculates the number of occurrences of a key, and A and B are the starting and ending points of an interval, respectively).

Using prefix sums, the function f can be precomputed in O(N) and each query can be processed in O(1) time.

6 Problem F: Aarav's Fish

Problem Credits: Amrita

First, we can calculate x with a little arithmetic ($x = \lfloor \frac{N}{C} \rfloor$). Let y be the number of colors that should have $x + 1$ fish (such that $C - y$ is the number of colors that should have x fish). We can calculate y easily as well ($y = N–x * C$).

We can create an array of length C to keep track of how many fish there are of each color. Then we can traverse that array, and for the first y colors with $> x$ fish, increase our count by the difference between the number of fish and $x + 1$. For the rest of the colors with $> x$ fish, we will increase our count by the difference between the number of fish and x.

7 Problem G: Aarav's Lotion

Problem Credits: Anirudh

The answer to this problem is one half of the maximum distance between any two adjacent guards. Note that the positions must be sorted.

8 Problem H: Aarav's Safes

Problem Credits: Aarav

This problem asks for the minimum number of safes that must be covered with boards in order for all the money safes to be covered. We can use a greedy approach by noticing that placing B boards forms $B - 1$ gaps of empty safes in between the first and last board. Note that it is never optimal to start or end a board on a safe that contains no money because that board's endpoint could then be shortened to the closest money safe giving a better solution.

We can sort all the safes with money, and then store the lengths of all the gaps of empty safes in between each consecutive money safe. We can sort the gaps by length and choose the $B - 1$ gaps with the largest length, since they represent NOT choosing a largest number of empty safes. Summing these $B - 1$ values gives the total length of safes we are not choosing. Our answer is given by the total number of safes, S, minus this quantity.

We are sorting the money safes initially, and then looping through to find the gaps, sort them, and take the $B - 1$ largest ones. Sorting overshadows everything else, so the time complexity is $O(MlogM)$, which easily passes.

9 Problem I: Popular Aarav

Problem Credits: Pranav

This problem asks for $2b + g$, where b and g are the number of boys and girls who hear bad things about Aarav respectively. Note that the graph of friends has $N - 1$ edges and each person has at least one friend. Therefore the graph is a tree, so we can solve this problem by running a DFS (or BFS) starting at Aarav's node (node 1).

At each node, we keep track of what the node's corresponding person heard from the person before them. If the person is a girl, then we pass on the same sentiment to all of her friends and recursively DFS on those nodes. If the person is a boy, then we pass on a bad sentiment (regardless of what the boy heard) too all his friends and similarly DFS on those nodes. Each recursive call on a node returns the insecurity score of the subtree rooted at that node. Applying this recursion pattern to Aarav's root node will give his insecurity over all N students.

10 Problem J: Haunted House

Problem Credits: Joshua

This problem asks for the shortest path out of a haunted house. Without taking the scares into account, we can simply treat the map of the haunted house as a graph(a set of nodes connected to each other by edges) and run a breadth first search(BFS) on the graph.

In a BFS, we visit a node(in this case, a tile in the map), and add all unvisited nodes that it connects to into a queue, keeping track of how many steps it took to reach each node. We then repeat this process until we reach the "exit" node, or our queue has no more nodes. In the former case, we return the number of steps it took to reach the exit, and in the latter we return -1 because the empty queue indicates no way to reach the exit. If we visualize our nodes as a tree, then the source node would be at the first layer of the tree, then the nodes one step away, then two and so on. We can see that this method guarantees that the first path we find to a tile will also be the fastest.

However, we need to account for the effect of the scares, which force us to go a certain way. To do so, we modify our node to keep track of the direction Aarav is forced to go in, as well as which row and column he is at. Each time we process a node, we check if Aarav is at the exit. If he is, then we print the amount of steps it took to get there. Otherwise, we check if Aarav is scared. If he is, we either add a new node where Aarav moved in the direction he was forced to, or we set him back to normal if he couldn't go that way. If Aarav was normal or got reset to normal, we process this node by making the nodes where he has moved in all possible directions and add them to the queue. In order to keep the runtime low, we make sure to never revisit nodes that we have already visited(being at the same time but forced to go in a different direction is not the same node), since there is no point for Aarav to take more steps than neccessary to reach a certain point. We do this using an array of booleans and by setting a node's value to true in that array when we reach it for the first time.

Since there are five different directions and $N, M \leq 1000$, we need to visit at most $5 \cdot 10^6$ nodes. BFS is $O(N)$, where N is the number of nodes, so our solution should easily run in time.