ICT In-House Contest 2 Solutions

ICT Officers

March 2020

Introduction

This write-up is going to outline the basic thoughts and motivations behind solving the problems from our second in-house contest for this year. 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, or else you'll get stuck in an inescapable cycle of giving in easily. As usual, feel free to message one of us if any part of this write-up is unclear.

Problem A: Corporate Crisis (Part 2)

Spandan's company has experienced yet another unfortunate series of events and all of their vending machines have malfunctioned and he needs to replace all of them. However, Spandan, being a resourceful CEO, had planned for such an event. According to his company's rules, the company is allowed a period of D days to deal with all this chaos and reorganize. Sarah is a vending machine magnate and her machines are of the utmost quality. However, the funky thing about her machines is that they are only sold on a specific day, after which they are off the market. Each machine has a cost, produces a certain profit P_i everyday it is used, and has a return price R_i, and is only available on day D_i. If Spandan buys machine M_i on day D_i, then it can be used from day $D_i + 1$. If Spandan decides to return a machine, he cannot use that machine on the day he returns it, but he can use the money he gains from the return to buy a new machine. Help Spandan maximize his profit during this period.

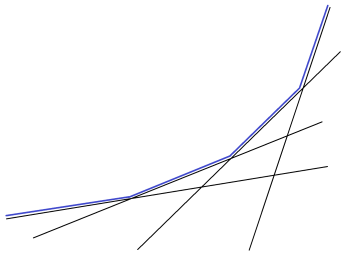
Solution to Problem A: Corporate Crisis (Part 2)

The naive solution to this problem would be:

- Maintain a solutions set and sort the machines by their day of availability
- For each machine:
  - Find the maximum amount of money it is possible to have before purchasing this machine and if there is enough money, add the solution to the set
- Find the maximum money in the solutions set

However, this $O(N^2)$ solution is not going to work on all the test cases. The most inefficient part of the last approach was iterating through the machines and solutions every time. We need to keep our solutions set as small as possible by removing partial solutions which are useless as soon as we can.

What we notice is that all of our partial solutions are lines and eventually when a particular solution gets overtaken by another line with greater slope, it becomes redundant and does not matter to us any more. Visually, our solution would look something like the blue curve shown below.



Since we are going to be dealing with lines, it would serve us to keep our partial solutions in order of increasing slope.

1. Every time we are adding a new solution, keep removing redundant first solutions.
2. If there are parallel lines, one must be redundant.
3. The new solution you are evaluating could be redundant; if the next solution is better than the new solution, do not add the new solution.
4. If the next solution is worse than the new solution currently and till the end of the period, then remove it. Keep doing this till the conditions are not satisfied any more.
5. If the previous solution in the solutions bag is worse than the new solution by this day, then the previous solution is redundant. Keep removing this till the conditions are not satisfied any more.

If we use a data structure like a TreeSet (in Java) or a std::set (in C++), we can find the next, previous, and first solutions in $O(\log n)$ time. When we have three slopes s_i, s_j, and s_k, where $s_i < s_j < s_k$ is satisfied, the day on which machine j overtakes i must come before the day on which machine k overtakes machine i. After doing all of this, we can finally add the new solution to the solutions bag (assuming it didn't get disqualified in any of our previous operations). Once we have finished looping through all the machines, we simply pick the solution from the solutions set that produces the most money by the end of the period.

Problem B: Corporate Crisis

Our beloved captain Spandan is the CEO of a company that is in the midst of a major crisis currently! And he needs your help! Spandan's company consists of N structures and $N - 1$ communication sockets established between them. It also has a headquarters building that has communication sockets established with each of the N structures. Due to varying distances between each structure, the time it takes to send a message between them varies. However, due to superior cables connecting the headquarters to each of the structures, it takes no time to send messages to any structure from the headquarters (and vice versa). Output the minimum time it will take for a message to be transmitted from the headquarters to all the specified structures and back to the headquarters.

Solution to Problem B: Corporate Crisis

An important thing to note in this problem is that since sending a message from the headquarters to a structure (and vice versa) doesn't cost us any time, the structure that the message is first sent to and the last structure the message is sent to do **NOT** have to be the same. The general idea is that we queue all the structures of degree 1 (that is to say, leaves) that don't need to receive the message and one by one delete them until all of the degree-1 structures are those that need to receive the message. Before we arrive at the solution for this problem, let's analyze a slightly different problem where the message needs to start and end at the same place (ignoring the headquarters). In this case, the shortest path we can take is twice the length of the Euler tour of the tree. However, since the problem doesn't constrain us to starting and ending at the same structure (excluding headquarters), the shortest path we can take involves passing through every edge twice except for the diameter of the graph. To find this, we can use two BFS's. First, conduct a BFS from a

node a to the farthest node b and then conduct a BFS from node b to the farthest node c. The diameter of the tree is the length of the path from b to c. So, the solution is $\mathbf{2 \cdot}$ **Euler tour - diameter**. This solution is O(N) time complexity.

Problem C: Prerequisites

Spandan and Derek want to take the same classes at school. However, they have different interests, and they realize that they will eventually have to take different classes. Their school offers N classes ($1 \leq N \leq 10^5$), conveniently labelled $1, 2, \ldots, N$. Each class has exactly one prerequisite (except class 1, which has no prerequisites), and multiple classes may have the same prerequisites (note that this makes $N - 1$ total prerequisites). The classes and their prerequisites form a tree with class 1 as the root. Given Q queries ($1 \leq Q \leq 10^5$) of the form a b, where Spandan wants to take class a and Derek wants to take class b, determine the last class that Spandan and Derek will take together, given that their first class is class 1.

Solution to Problem C: Prerequisites

This problem is a direct application of Lowest Common Ancestors. The problem states that Spandan and Derek chronologically take classes together and they eventually end up taking different classes, and it asks to find the last common class they take. We can think of this backwards: given the two classes that Spandan and Derek eventually take, we wish to find the first class that is a prerequisite for both. My implementation used 2^n jump pointers so it was able to run in $O(Q + N \log N)$, which was enough to get all test cases.

Problem D: Mountain Climbing Trip

For spring break, Sarah is going on a mountain climbing trip with Harsha, Richard, Spandan, and Somu. Since Sarah is an avid mountain climber, she is very excited and wants to plan which mountains the group should climb beforehand. She is provided with a map detailing the locations of mountains and rivers. The map has M ($1 \leq$M≤ 12) rows and N ($1 \leq$N≤ 12) columns. Each cell of the map is marked with either a 1 or 0 to represent a mountain or river, respectively. Please help Sarah compute the number of options the group has to climb the mountains given the following requirements:

1) Sarah can choose to visit any number of mountains from none to all.
2) Sarah cannot choose to climb mountains that share a side.

Solution to Problem D: Mountain Climbing Trip

Since each row contains binary numbers (either 0 or 1), the number of possible variations of states is equivalent to 2^N ($1 \leq$N≤ 12). Next, we can pre-process the valid states for each row. This can be done by looping through each possible state for a specific row and eliminating the invalid ones. Because we are only interested in the valid states and the total number of valid states, we can use a State class to maintain the above information. Each State object will have an integer to keep track of the valid states for a particular row. Invalid states can be ruled out by using bit-wise operations. After determining the valid states, we can store them into a two-dimensional array $f[i][j]$. For the first row, each valid state will be 1. We can solve row 2 to M by using traditional dynamic programming. A state j in row i will need to be tested with each valid state in row $i - 1$ to guarantee that the mountains do not share a side. Lastly, the number of options the group has to climb the mountains given the requirements will be the sum of the states on the last row because we utilized top-down dynamic programming from left to right.

Problem E: Coronavirus

In the bleak near future, Novel Coronavirus COVID-19 has broken out in America. Assume America can be modeled by an r my c grid, where each cell in the grid represents a different city. Each city has an interaction score from 1 to s. Cities with the same score tend to interact with each other a lot, which is less than ideal for containing Coronavirus. Coronavirus spreads very rapidly. In one day, the virus can spread from the current city to any of the cities side-adjacent to it or it can spread to a city with the same interaction score.The Center for Disease Control wants you to calculate the minimum number of days it will take the Coronavirus to spread from city (r1,c1) to city (r2,c2) for p different city pairs.

Solution to Problem E: Coronavirus

Because the number of queries can range up to 10^5, we need to process the queries in less than $O(logn)$. Thus, this problem requires a clever preprocessing. In order to do the preprocessing, we do a multi-sourced BFS for each interaction score from 1 to s. This will let us find the shortest distance between each city in a grid to another city with a specific score. For each query, loop through all the colors (maximum 40 colors). The answer is: ans = min (Manhattan Distance, ans, 1 + dis[r1][c1][color] + dis[r2][c2][color]). The solution is $O(n * m + p)$.

Problem F: More Smarter

Somu is one of the groundkeepers at the local park. After a grueling morning of work, Somu hears the promising bell of a passing food truck. The Burger Truck is giving out free burgers to everyone, but only on one condition. To dig into a delicious, free burger, you had to have graduated high school and secured a diploma. Somu can only watch as his workmates all eat their scrumptious burgers. The truth is, Somu dropped out of TJ all those years ago, thinking he was "too cool for school" and with dreams to become a multi-billionaire. Clearly, that didn't happen, and now Somu is stuck with an empty stomach. So he does the most sensible thing possible: he goes back to high school to get his diploma.

To get his diploma, Somu needs to complete one assignment. This assignment consists of finding "streaks" of words in a large text. A "streak" is a sequence of words for which each word is present in a predetermined dictionary of words. A streak begins with a word in the dictionary and ends with a word in the dictionary that's preceding a non-dictionary word, or the end of the text. Somu wants to avoid the smaller size streaks, so he imposes a certain threshold the streak size needs to meet to be considered. Note that all streaks will have unique start points and endpoints. For example, if a streak began at 2 and ended at 4 in the list [1,2,3,4,5,6,...], then the next possible streak would start at 5. A word at an indexed position cannot be in two separate streaks at the same time.

Not learning from his prior life experiences, Somu doesn't want to do the assignment himself, so he enlists your help from a listing he found on Craigslist for work. Help Somu find the biggest streaks for the texts given, and in turn, achieve his goal of finally eating the burger which has curved him since the dawn of time, 6 hours ago.

Solution to Problem F: More Smarter

This problem is meant to be the most straightforward of the problems in this contest. There are no complicated algorithms or data structures that need to be implemented. It only requires concepts that are prevalent in a typical APCS class. The general solution involves iterating through each word of the text, accumulating streak counts and relevant information, then sorting the relevant streaks for final output. The worst case time for this program would be O(N + S log S), where N is the number of words in the text, and S is the number of streaks found. One point to note is that it is only necessary to store the streaks whose length is greater than or equal to the threshold. In this way, we can avoid having to sort through large lists of streaks which are comprised of mainly irrelevant streaks. This will reduce S in the run time expression, and in turn reduce the average run time of your program. It is also important to choose a data structure which supports

O(1) contains method calls for the dictionary of words, as without it, we would have to loop through the dictionary for each word in the text, which would increase the worst case to O($ND + S \log S$), where D is the number of words in the dictionary. This would be too slow to fulfill the constraints of the test cases provided.

Problem G: Universal Basic Income

Every year, the citizens of the country Teejayland receive a Universal Basic Income (UBI) from the government. The UBI is distributed such that citizens of city i receive a_i units of Teejayland currency. Unfortunately, you are the only data scientist working in the Teejayland government, so the economic wellbeing of Teejayland depends on your abilities! The government wants you do be able to accomplish two tasks. 1. Keep track of UBI reforms. There comes a time where for every city i∈[l,r], the government reforms the amount of UBI given to city i to the number of positive divisors of a_i. 2. Sum UBI spending. For every city i∈[l,r], sum all the a_i in this range.Your job is to help the government compute their UBI spending across certain cities.

Solution to Problem G: Universal Basic Income

Compute the number of factors for money values beforehand using a double for-loop. The crucial observation is that this problem converges very quickly. In fact, for values up to 10^6, we need at most 6 updates in order to get the number of factors to either 1 or 2. Thus, this problem can be solved by implementing two Segment Trees, one range min tree and one range sum tree. Whenever we update a range, update the values for both Seg Trees. If in a segment we realize that the maximum number is 2, then we skip the update. The rest of the solution is standard seg tree stuff: implement build, sum/max, and update functions for seg trees normally. Off of the crucial observation, the problem complexity changes from $O(m * n * logn)$ to $O(m * logn)$.

Problem H: Richard's Cookies

It's almost time for ICT and the officers don't have cookies! On Friday morning, Richard reveals his secret stash of cookies. His stash is made of N ($4 \leq N \leq 10^5$) jars of cookies arranged in a line, with the ith jar containing c_i cookies ($1 \leq c_i \leq 10^5$). The other officers rejoice, but Richard tells them that they can't take any of the cookies because he's hungry. Not wanting to make the rest of the officers sad, Richard agrees to let the officers pick four of the jars. The officers will get to bring the second and fourth jars to the next ICT meeting, and Richard will get to eat the cookies from the first and third jars. Find the **net number** of cookies that the officers will receive, where the net number of cookies is defined as the difference between the number of cookies that the officers get and the number of cookies that Richard eats. In other words, if the officers pick jars w, x, y, z ($w < x < y < z$), the net number of cookies they will receive is $c_z - c_y + c_x - c_w$ cookies (Note: This number MAY be negative!).

Solution to Problem H: Richard's Cookies

This problem is a DP problem. My solution used 4 different lookup tables. The first stored the maximum value of c_z, going from right to left. The second stored the maximum value of $c_z - c_y$, using the values from the previous lookup table. The third lookup table stored the max of $c_z - c_y + c_x$ and the last one stored the max of $c_z - c_y + c_x - c_w$. The final answer was the first element of the last lookup table (first element since we went from left to right). Here was the DP I used ($LT1$, $LT2$, $LT3$, $LT4$ are the lookup tables with each element initialized to $-\infty$; C is the array containing the number of cookies):

$$LT1[i] = max(LT1[i+1], C[i])$$

$$LT2[i] = max(LT2[i+1], LT1[i+1] - C[i])$$

$$LT3[i] = max(LT3[i+1], LT2[i+1] + C[i])$$

$$LT4[i] = max(LT4[i+1], LT3[i+1] - C[i])$$
$$Ans \rightarrow LT4[0]$$

Problem I: Sarah's Rose Garden

In Sarah's backyard, she has a rose garden with N ($1 \leq$N≤ 5000) roses numbered 1 through N. The roses are arranged on a number line on the x-axis ranging from 1 to M ($1 \leq$M$\leq 100,000$). Note that no two roses share the same x-coordinate.

In order to maintain her rose garden, Sarah needs to water the roses everyday. Since she is very busy, she wants to build an automatic sprinkler to water the roses. Sarah is planning to buy automatic sprinklers from HSSSRCo. which charges sprinklers by length. The length is calculated based on the distance of the two roses plus one. For instance, if you have rose i at x-coordinate 5 and rose j at x-coordinate 8, the length would be $8 - 5 + 1$ which is equivalent to 4. Sarah can purchase automatic sprinklers of different lengths and costs from HSSSRCo. Please help Sarah determine the minimum cost for purchasing a set of sprinklers that will water her roses.

Solution to Problem I: Sarah's Rose Garden

Begin by instantiating an array of roses based on their positions. This array will have a length of N, which is the number of roses Sarah has in her garden. Next, sort the array based on x-coordinate. Since the automatic sprinklers are of different lengths and costs, we know that larger length sprinklers do not necessarily cost more than smaller length sprinklers. We are also allowed to overlap sprinklers, so we can pre-compute an array of minimum cost that covers any particular length in $O(M)$ time complexity. At this point, the problem simply becomes a traditional dynamic programming problem. After sorting the N roses, we can divide them into contiguous groups, each of which are covered with a sprinkler. This is now a dynamic programming problem with $N + 1$ states where the array $dp[n]$ is the minimum cost to cover roses 1 through N. In order to compute each value, we can look at all possible last intervals. The minimum cost for purchasing a set of sprinklers that will water Sarah's roses will be equivalent to the value stored in $dp[N]$.

Problem J: Time Crunch

After messing with the space-time continuum, Somu is now stuck in an infinite realm, and he can't find a way to get out. With seemingly no other option, Somu slumps to the ground, dejected and concerned about the rest of his life. However, as if his prayers were answered instantly, a portal gun pops up right in front of him.

After conducting various experiments with the portal gun, Somu came up with certain conditions which guard the gun against infinite use. First, the number of seconds it takes to move through a portal is seemingly random and never fixed. Second, due to the time-bending nature of the portal gun, it is possible to actually gain time, or go back in time when he enters a certain portal.

With these ideas in mind, given the location of the infinite realm, help Somu find the destination which will yield him the most efficient trip, i.e. the one where he gains the most amount of time back, or the one which takes the least amount of time to get to his destination.

Solution to Problem J: Time Crunch

Because this problem essentially lays out a graph with negative weights, our first logical step would be to take a look at the Bellman-Ford Algorithm. This algorithm is quite similar to Dijkstra's, except that it is compatible with negative weights for the edges of a graph. Bellman-Ford is based on relaxation, the idea that approximations to the real minimum distance are replaced by better approximations until they reach the true solution. The algorithm updates the minimum distance from the source to every other vertex, by iterating through each edge, comparing the calculated distance with the current established distance,

updating if necessary, and repeating this process V-1 times, where V is the number of vertices in the graph. A final check for negative cycles would yield an expected run-time of O(VE), where E is the number of edges in the graph. This is simply too slow for the constraints given by the test cases. One area of improvement would be avoiding having to repeat the process of updating for EACH vertex in the graph. This is what the Shortest Path Faster Algorithm (SPFA) achieves. An improvement of the Bellman-Ford algorithm, SPFA maintains a queue of vertices which have been relaxed, or updated with a smaller distance. This avoids having to iterate through every vertex and edge, as minimum distances are only affected by vertices whose distance has already been updated. The worst case time is still O(VE) for SPFA, but on average, it runs much faster as it avoids iterating through each vertex. With this time improvement, the solution is able to finally pass the constraints given by the test cases.