ICT In-House Contest 1 Solutions

ICT Officers

November 2019

1 Introduction

This write-up is going to outline the basic thoughts and motivations behind solving the problems from our first 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.

2 Problem A: Multidimensional Travels

Derek the prestidigitator transports Spandan to an n-dimensional hyper-cubical world. He gives our hero, Spandan, his challenge: "After each second, you will walk along an edge of this hyper-cube to an adjacent vertex. You have t seconds. After the time is up, you must be back here, at the vertex where you started. Otherwise, you will never return to Earth!" How many paths can Spandan take to return to Earth? (Note: Spandan can return to his starting position at any time, as long as he ends up there by the end of the t seconds).

3 Solution to Problem A: Multidimensional Travels

This is an example of a problem that is a little difficult to understand but very simple to code (partly due to the generous bounds).

Let's start by looking at a n-dimensional unit cube. This cube will have coordinates $(0, 0, ..., 0), (0, 0, ..., 1), ...,$ $(1, 1, ..., 1)$. If we let vertex A have coordinates $(0, 0, ..., 0)$, then its neighbors will have vertices $(1, 0, 0, ..., 0)$, $(0, 1, ..., 0), ..., (0, 0, ..., 1)$. Notice that if two vertices are connected by an edge, then their coordinates differ by exactly 1 coordinate. Thus, vertices 1 edge away from A will have one 1 in their coordinates, vertices 2 edges away from A will have two 1's in their coordinates, and so on. Using coordinates allows us to get a better understanding of higher-dimensional cubes.

The key insight is to notice the recursive nature of the problem: if we are on vertex A and want to find the number of ways to return to vertex A after t seconds, we can simply find the sum of the number of ways to reach vertex A after $t - 1$ seconds from each of the neighbors of A. Expressed more mathematically:

$$f(A, t) = \sum_{a \in N(A)} f(a, t - 1),$$

where $f(A, t)$ represents the number of ways to reach vertex A starting on vertex A, and $N(A)$ represents the set of neighbors of A. Converting this to DP gives

$$DP[i][j] = (n + 1 - i) \cdot DP[i - 1][j - 1] + (i + 1) \cdot DP[i + 1][j - 1],$$

for the number of of ways to reach a vertex i edges away from A in j seconds (starting from vertex A). Therefore, our answer is $DP[0][t]$. (Notice that we only need to store 2 columns since we only use the previous timestep values in our calculations).

4 Problem B: Word Reactions

Spandan and Richard are bored in English class, so they decide to play a game called Word Reactions. In this game, one player creates a list of words and the other player guesses the number of words left after all word reactions occur. In a list of words, word reactions cause consecutive pairs of identical words to disappear. For example, given the list of words [“hello”, “hi”, “hi”, “hello”, “bye”], “hi” and “hi” react, and the resulting list becomes [“hello”, “hello”, “bye”]. Then, “hello” and “hello” react and the list becomes [“bye”], after which no word reactions occur. Given a list of words, print the number of words remaining after all word reactions.

5 Solution to Problem B: Word Reactions

This was meant to be the easiest problem of the contest. There are a lot of ways you could have solved this problem, but the easiest is to just use a stack. As you go through each word, if the value at the top of the stack matches it, then pop from the stack, otherwise, add the word to the stack. At the end, just print out the length of the stack.

6 Problem C: Whack-a-Mole

The ICT Officers find themselves trapped in a field full of man-eating moles, and the only way to escape the field is to whack all the moles! Luckily, the officers have a supply of mallets with which to whack the moles. There are p moles and q mallets. Each mole has a strength s and each mallet has a fortitude f and a mole with strength s can only be whacked by a mallet with fortitude $f \geq s$. Each mallet can only whack 1 mole.

7 Solution to Problem C: Whack-a-Mole

This problem is a very simple application of a stay-ahead greedy strategy. To solve the problem, all we need to do to whack a certain mole is find the mallet with least fortitude that is at least as great as the strength of the mole. To find such a mallet, we can simply use a binary search.

8 Problem D: Minimum Spanning Tree

Sarah would like to create a minimum spanning tree. There are N cities ($1 \leq N \leq 40,000$) and M routes ($1 \leq M \leq 100,000$). Each route has a cost and note that only 3 routes can share the same cost.

Please help Sarah compute the minimum total cost which connects all of the cities as a minimum spanning tree **and** indicate whether there is a single minimum spanning tree that can be created or if there are multiple minimum spanning trees that can be created.

9 Solution to Problem D: Minimum Spanning Tree

This problem involves finding the minimum total cost of the minimum spanning tree and whether there is a single or multiple minimum spanning tree(s) that can be created. In our previous Minimum Spanning Tree lecture, we discussed the two algorithms for finding a minimum spanning tree: Kruskal’s Algorithm and Prim’s Algorithm. Of these two, Kruskal’s Algorithm processes edges in increasing order of their weights since it greedily adds the edges. Note that a key point of Kruskal’s Algorithm is that it repeatedly adds the edges of minimum weight to the minimum spanning tree except when adding said edge forms a cycle (which violates the tree structure).

Just as a sidenote, remember that when the costs are distinct, only one minimum spanning tree can be created; otherwise, if the costs are not distinct there is a possibility that multiple spanning trees can be created. To find out if a single or multiple minimum spanning tree(s) can be created, you need to first identify the set of routes that share the same cost and do not share the same parent. Place these routes into a set and process these routes one by one to see if they can merge. If all the routes can be merged, a single minimum spanning tree can be created. Otherwise, if any one of the routes fails to be merged it means that a cycle was created, implying that multiple minimum spanning trees can be created.

10 Problem E: Everyone Loves Popeye's

Somu loves Popeyes. He loves Popeyes so much that he mapped out all the franchises in his city. With the release of the new chicken sandwich, Somu's hungrier than ever for some good chicken. But there's a problem. The sandwich is so popular that stores are selling out within hours. So Somu wants to buy as many chicken sandwiches as possible in the smallest amount of time.

Let S_i represent the Popeyes closest to Somu's house. Each Popeyes franchise has a limited number of sandwiches Somu can buy. Branches that are close to each other are connected by a road that takes a certain number of minutes to cross. Somu plans on starting from S_i and ending his Popeyes run at any other Popeyes, F_i, in his city. However, because of how little time he has, no matter which F_i he chooses, he has to take the shortest path from S_i to F_i. Help Somu find the F_i for which the shortest path from S_i will net him the most chicken sandwiches for the amount of time spent on the path i.e the highest chicken to time ratio.

11 Solution to Problem E: Everyone Loves Popeye's

The solution to this problem involves implementing Dijkstra's Algorithm with an extra layer of difficulty. When using Dijkstra's Algorithm to solve a programming problem, oftentimes we are only concerned with the total path costs of the shortest paths from a given start node. In this problem, however, the goal was to find the shortest path which yielded the best chicken-time ratio, which didn't necessarily mean the shortest of the shortest paths. While iterating through the graph provided using Dijkstra's, you had to keep track of the path costs as well as the accumulated chicken counts for all possible shortest paths from the given start node. To find the best destination, you had to iterate through all of the shortest paths, and determine the path with the maximum chicken-time ratio.

12 Problem F: Projects

Sarah is working on a sequence of N projects ($1 \leq N \leq 100$). Initially, Sarah agreed to work on project p for a hours ($0 \leq a \leq 10$). Later, due to a requirement change, Sarah's clients want her to work on project p for b hours ($0 \leq b \leq 10$). Sarah has the following three options: 1) She can add one hour to any project i and charge X dollars per hour ($0 \leq X \leq 1000$). 2) She can remove one hour from any project i and charge Y dollars per hour ($0 \leq Y \leq 1000$). 3) She can transfer one hour from any project i to project j and charge Z dollars per hour times $|i - j|$ ($0 \leq Z \leq 1000$). Please compute the minimum total cost for Sarah to complete her projects.

13 Solution to Problem F: Projects

This problem uses dynamic programming with an additional sequence transformation to compute the minimum total cost needed to complete the projects. First, transform each hour Sarah will be working on a project before the requirement change and after the requirement change into two separate arrays of length of 1000. This is because both the arrays containing the hours Sarah will be working on a project before the requirement change and after the requirement change have at most 100 projects each of which she needs to work on for at most 10 hours, giving an array of length 1000. For example, given the sample input and output in the problem, let array A indicate the hours Sarah will be working on a project before the requirement change and let array B indicate the hours Sarah will be working on a project after the requirement change.

Array A: 1 2 3 4

0	1	2	3	4	5	6	7	8	9	10
0	1	2	2	3	3	3	4	4	4	4

Array B: 4 3 2 0

0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	2	2	2	3	3

Our goal is to transform one project hour sequence into another at a minimum cost given three possible operations: insertion of a hour (at cost X), deletion of a hour (at cost Y), or transfer of a hour (at cost Z multiplied by the magnitude of the change). This can be accomplished in $O(N^2)$ time (where N is 1000) using dynamic programming.

Each subproblem $DP[i][j]$ that we solve along the way represents the minimum cost of transforming the first i hours of the source sequence into the first j hours of the target sequence. Starting with the base case, $DP[0][j]$ is equal to the product of j and X and $DP[i][0]$ is equal to the product of i and Y. The matrix is shown below to give you an idea of these base cases:

	0	1	2	3	4	5	6	7	8	9
0	0	100	200	300	400	500	600	700	800	900
1	50									
2	100									
3	150									
4	200									
5	250									
6	300									
7	350									
8	400									
9	450									
10	500									

Next, we can proceed to calculate the rest of the values in the matrix. By double looping from index 1 to $n1$ and 1 to $n2$, we can first set $DP[i][j]$ equal to infinity. Then, based on the three possible operations, we can say:

- $DP[i][j]$ is equal to $Math.min(DP[i][j], DP[i][j-1] + X)$
- $DP[i][j]$ is equal to $Math.min(DP[i][j], DP[i-1][j] + Y)$
- $DP[i][j]$ is equal to $Math.min(DP[i][j], DP[i-1][j-1] + Z * |A[i] - B[j]|)$

By repeating the above steps, we can fill the matrix and solve for the minimum total cost for Sarah to complete her projects.

14 Problem G: Panic Attack

First quarter is over, and everyone who has Dr. Zacharias Computer Vision 1 ended with a 93.7 percent! Each person is panicking a certain amount, and the level of their panic can be given a score, S. Some aspiring young scientists, like Spandan, are panicking very hard because they expected to get a 110 percent in the class and have a high panic score. On the other hand, bad students, like Richard, expected to get an F but ended with an A, so they aren't panicking at all and have a low panic score.

One day, the students were sitting around in a circle to discuss the fate of their Computer Vision grades. However, people want to sit next to other people with roughly the same panic score as themselves in order to relate better with the people next to them. The awkwardness of the circle can be calculated by finding the maximum absolute difference in panic score between any two people.

Help the Computer Vision 1 students arrange themselves so that the awkwardness of the circle is minimized.

15 Solution to Problem G: Panic Attack

This problem may look intimidating at first, but it's fairly easy. First observe that we have to sort the scores because naturally, if the points were sorted, they would have the smallest possible absolute difference between them. However, since it's a circle, the first and last two points would have a large difference. Hence, we want to separate the points as such:

$$S_1, S_3, S_5 ... S_{LargestOddIndex}, S_{LargestEvenIndex}, ... S_2.$$

This ensures that each score is next to a score with index two greater than it.