CS2040S: Data Structures and Algorithms

Discussion Group Problems for Week 12

For: April 3–April 7

Below is a proposed plan for tutorial for Week 12. In Week 12, we will continue talking about graph modelling, and continuing to look at shortest path problems. (This week we will mostly focus on problems that can be solved via Bellman-Ford and Dijkstra's algorithm.)

1 Review Questions

Problem 1. (Kahn's Algorithm)

topological sort: post order DFS or kahn's algo

Reference to Kahn's algorithm:
https://www.geeksforgeeks.org/topological-sorting-indegree-based-solution/

You have seen in lecture how Kahn's Algorithm work. The algorithm aims to find a topological ordering of a directed graph by removing nodes without any incoming edges, removing all edges originating from that node, and finally add that node to the ordering. This is repeated until there is no more nodes left without any incoming edges. What kind of data structure can we use to help us decide which node to remove next? What would the time complexity be?

Problem 2. (Shortest Path in a Tree)
Suppose you have an arbitrary undirected tree with weighted edges, you want to find the shortest distance between a pair of nodes. Clearly you can use dijkstra, is there a simpler (and faster) way to do it?

Optional: What if you are given Q pairs of nodes that you want to find the shortest distance between. Clearly there is an $O(QN)$ solution. Can you do better? (There are multiple $O(NlogN + Q)$ solutions.

2 Problems

Problem 3. Overcooked

Relevant Kattis Problems:

- https://open.kattis.com/problems/pickupsticks
- https://open.kattis.com/problems/reactivity

- https://open.kattis.com/problems/easyascab

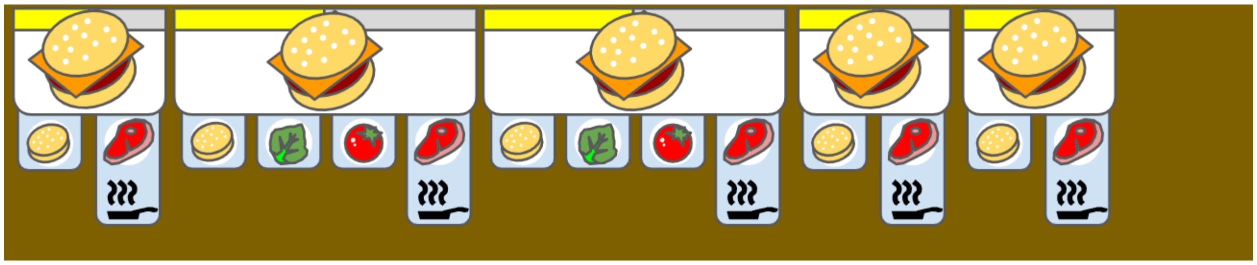

Figure 1: *Overcooked.* (Matthew Ng Zhen Rui)

After many years of playing Overcooked, you finally have fulfilled your dream of opening your own restaurant. On the opening day you are given a list of reviewers that will be coming. You want to ensure that they are served in a timely manner so that they leave good reviews for your restaurant.

You want to determine in what order you want to serve your dishes. There are a lot of dishes to serve out but you can't serve them in any order. Given a set of n dishes you know certain foods must be served after another (for example, you will serve appetizers like mushroom soup before the main course like filet mignon). Given a list of n dishes as well as k constraints on relative orderings of the dishes, output a valid sequence in which the dishes can be served. You may assume that such an ordering always exists.

In addition, you want to ensure that if there are multiple possible ways to order the dishes, you output the one which is *lexicographically* the smallest (So that it looks presentable on a menu).
As an example, let's say there were 4 dishes: Garlic Bread, Mushroom Soup, Filet Mignon Burger and Banana Split. Now, if we are given the constraints as follows:

- Garlic Bread must be served before Filet Mignon Burger
- Mushroom Soup must be served before Filet Mignon Burger
- Banana Split must be served after everything else

These give the following valid orders:

- Garlic Bread, Mushroom Soup, Filet Mignon Burger, Banana Split
- Mushroom Soup, Garlic Bread, Filet Mignon Burger, Banana Split

However, we will want to output the former, since it is lexicographically smaller than the latter (since Garlic Bread is alphabetically before Mushroom Soup).

Optional : In order to protect against potential modifications of the order of dishes being served, you also want to check whether any valid ordering of the dishes is unique. Note that if this is the case then the valid order will instantly be the smallest lexicographically as well. How would you do this?

Problem 4. (Tourism)

Relevant Kattis Problem: https://open.kattis.com/problems/maximizingwinnings

You are off to travel the world in your trusty old beat-up Chevrolet. You have a map, a full tank of gas, and you are off. Just as soon as you figure out where you want to go. Looking at the map, you notice that some roads look very appealing: they will make you happy with their winding curves and beautiful scenery. Other roads look boring and depressing: they will make you unhappy with their long straight unwavering vistas.

Being methodical, you assign each road a value (some positive, some negative) as to how much happiness driving that road will bring. You may assume that the happiness is spread out uniformly over the entire road, for every road on your map. You may also assume that every road spans a distance of 1 kilometer. Pondering a moment, you realize that you can only travel a fixed number k kilometers. Starting from here on the map, find the destination that is at most k kilometers away that will bring you the most net happiness. (Write out your algorithm carefully. There is a subtle issue here!)

Problem 5. (Strange Car)

At this point, you are probably well acquianted with the shortest path problem. Now, let's shake things up a little.

You have finished work and want to go home. Unfortunately, some guy decided to trash your car and it no longer works. But fret not as your friend however decides to lend you his car. So your problems are solved right?

Well, not quite. As you start the engines, there is a strange note written on the door. It states that the car can only be parked after moving every 5 kilometers! Indeed the odemeter is currently a multiple of 5! If you have not travelled a distance exactly a multiple of 5, the doors will not unlock and you will be stuck until the odemeter is a multiple of 5!

Nonetheless there might still be hope. The town you are in is strange. Buildings are connected by unidirectional roads exactly 1 kilometer long. How long then will you take to reach your home AND be able to stop? (You can visit your home multiple time before being able to stop).

Optional Extension 1: Now, what if the care can only be parked after moving arbitrary k kilometer, here k can be large.

Optional Extension 2: Now, the roads are bidirectional BUT are of arbitrary length (but are of integer kilometer length). However, you are sure that the total length of the all roads in kilometer is small. What if k is still large? (Adapted from: https://open.kattis.com/problems/kwalks)

Problem 6. (Roads with discount)

Adapted from: NOI 2022 Finals Task 1 (Voting cities)
https://codebreaker.xyz/problem/votingcities

You have a country with multiple cities connected by unidirectional roads, and travelling along each road will cost a corresponding amount of money in bus fare. Some of these cities are good cities. You want to get from a city A to any good city in the cheapest way possible. At the start of your journey, you are given the choice to buy 5 different vouchers. They can be used on one road per voucher, reducing its costs by $10\%, 20\%, 30\%, 40\%, 50\%$ respectively. You cannot use multiple vouchers on a single road to stack effects. You can choose to buy all 5 vouchers, or none at all.

Alas, you do not know your starting city A, nor the prices of the vouchers. As such, you thought of Q different scenarios. For each scenario, you start at a different city and are given different voucher prices, find the cheapest possible way to get to a good city. (Note that the cost of bus fares doesn't change in each of the scenarios, e.g. it will always take 3 dollars to travel from city A to city B regardless of which city you start from)

Problem 6.a. For this part, let us consider the subproblem where there are no vouchers. How would you then answer the Q scenarios effectively?

Problem 6.b. Now, we will consider the full problem with vouchers involved. How would you answer this question? Hint: Transform the graph, perhaps we can keep track of which vouchers we have used already somehow?

Problem 7. (How many arrays, hard)

Adapted from: https://open.kattis.com/problems/permutationarrays

Problem 7.a. You are given an unknown array A of length n and a few facts about it. You know that all the elements are a number from 0 to $2^N - 1$. You also are given m constraints of the following tuple: (l, r, k). This means that $A_l \oplus A_{l+1} \ldots \oplus A_r = k$ ($\oplus$ is the XOR operator, e.g. $5 \oplus 4 = 1$. If you are unsure how this operator works, try converting the numbers to binary forms first and XOR bit by bit). You want to find out how many different arrays (or none) satisfies your constraints.

First Hint: Instead of considering the number of arrays, would it be easier to consider the number of prefix XOR arrays? How would you convert the constraint from the original array to this new array and what does the number of prefix xor arrays tell you about the number of original arrays?

Second Hint: Try to model this as a graph where the goal is to find how many ways you can assign values to nodes. What would be the nodes? How about the edges?

Note that we define the prefix XOR array B to contains $n+1$ entries, where entry $B_0 = 0$ and

$B_i = A_1 \oplus A_2 ... \oplus A_i$.

Problem 7.b. (Optional)

What if now, you are given the constraints one by one? At the start, you have no constraints. Everytime you receive a constraint, you want to know at present moment, how many arrays satisfies all previous constraints you have seen. Of course, we could naively run the above algorithm from scratch everytime we receive a constraint, but can we do better?