CS2040S: Data Structures and Algorithms

Discussion Group Problems for Week 11

For: March 29–April 2

Below is a proposed plan for tutorial for Week 11. In Week 11, 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, while next week we will talk about Dijkstra's algorithm.)

1 Review Questions

Problem 1. (Kahn's Algorithm)
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)
What if you want to find shortest path in an undirected rooted tree? What is the simplest way to find a good order to relax the edges?

2 Problems

Problem 3. (Exploring all Paths)

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

Sometimes, it seems like a good solution to a graph problem is to explore all possible paths in the graph. Starting from some vertex s, we explore all possible paths to some destination t, and examine the resulting cost. For example, imagine you have a highway road map, with tolls specified along various roads. You want to find the route with the smallest cumulative toll.

Perhaps we can explore all possible routes using a cleverly modified DFS or BFS? Why is this a bad idea? Draw a graph in which exploring all possible paths from s to t will not work well.

Problem 4. Overcooked

Relevant Kattis Problems:

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

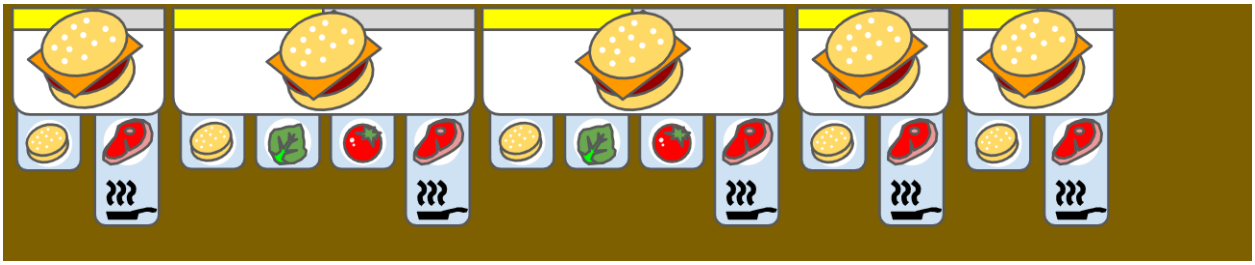

Figure 1: *Overcooked.* (Matthew Ng Zhen Rui)

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 5. (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 6. (Spaceship Troubles)

The wonderful, fantastic product made by the Whoozit Company is a newfangled spaceship with a fancy warp drive. Now, warp drives are not quite as useful as you might think: they only let you travel between a set of fixed locations, along fixed (directed) paths. One of the possible jobs at Whoozit Company is *Guinea Pig*, and you have just been promoted. They hand you a map of the universe (complete with locations and directed paths), put you in the new spaceship, and off you go.

At first, everything works fine. You start off at the little dot marked "Earth", chose one of the neighboring dots ("Alpha Centauri", at only 4.24 light years away), and hit the big green *GO* button. Whoosh. You made it in one piece! Wow, each hop takes exactly 1 minute, regardless of how far you go!

But then a little red light starts blinking, and the screen reads out the following error message: `Error: your warp drive is now broken. Abort, Retry, Fail?` After some fussing with the computer, you discover that the warp drive still works, but it has the following limitation: it can only jump five hops at a time. You cannot just jump to a neighbor of Alpha Centauri; you have to jump directly to some location that is exactly five hops away.

Describe an algorithm that will get you from Alpha Centauri back to Earth in minimum number

of hops, explain why it works, and give its efficiency.

Problem 7. (Roads with discount)

Adapted from: NOI 2022 Finals Task 1 (Voting cities)
https://github.com/noisg/noi_2022_finals/blob/main/statements.pdf

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

Problem 7.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 8. (How many arrays, hard)

Adapted from: https://nus.kattis.com/sessions/qs926n/problems/permutationarrays

Problem 8.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 $B_i = A_1 \oplus A_2 ... \oplus A_i$. Some facts of XOR you may find useful: $a \oplus a = 0$, $a \oplus b = k \iff a \oplus k = b$.

Problem 8.b. (Optional)

Note to tutors: This is an optional extension to the problem above and may be rather challenging conceptually. Tutors can feel free to skip it if they are not confident that the class can understand it or save it for future tutorials. However, I believe it features a clever use of small to large merging and binary search.

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?

Problem 9. (Internet Routing) **(Optional)**

https://open.kattis.com/problems/shortestpath3

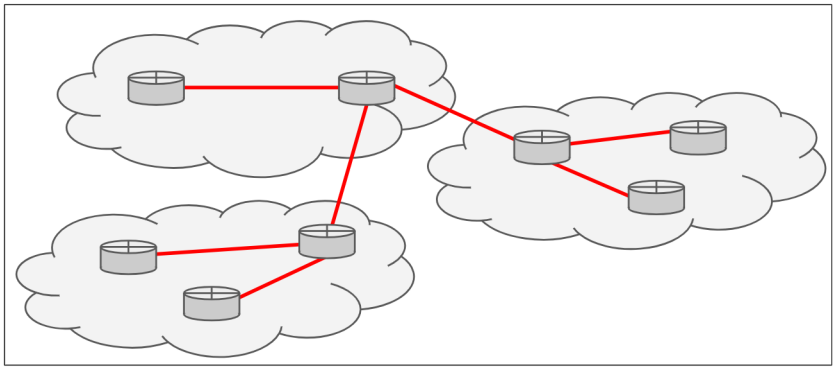

Figure 2: A Network Diagram

Imagine a hypothetical world where there are a set of computers connected by wires called *the internet*. Each of these computers has a unique address that is used for routing. (We might call this an IP address.) Each computer needs to maintain a routing table, i.e., a table that maps destination addresses to the next hop of the route. For example, if computer X has links to computers A, B, and C, and if it receives a message for destination D, the routing table might specify that that message should be forwarded to B. The goal is to forward the message to the destination using the minimum number of hops possible.

Problem 9.a. Collectively, the routing tables specify shortest path trees. How might you use **Bellman-Ford** to construct the routing tables?

Problem 9.b. Now imagine we want to construct the routing tables in parallel. That is, imagine that all the computers can send information to their neighbors at the same time. How might you implement a relax step so that all the computers can run it at once? Will that work? (Why or why not?) How long will the entire Bellman-Ford execution take?

Problem 9.c. Imagine that the computers run Bellman-Ford relax rounds forever. E.g., every one minute they send their vector of estimates to their neighbors and update their routing tables as necessary. Assume that a new edge is added to the network, e.g., A and B were not neighbors and now they are. Will the ongoing algorithm fix the estimates? If so, how long will it take? If not, what needs to be done to make it work?

Problem 9.d. Imagine that the computers run Bellman-Ford relax rounds forever. E.g., every one minute they send their vector of estimates to their neighbors and update their routing tables as necessary. Assume that one edge fails in the network, e.g., A and B were neighbors and now they are not anymore. Will the ongoing algorithm fix the estimates? If so, how long will it take? If not, what needs to be done to make it work?