CIS 477 (Fall 2022) Disclosure Sheet

Name: Kevin Lopez

HW # 6

Yes [✓ No] Did you consult with anyone on parts of this assignment, including other students, TAs, or the instructor?

Yes [✓ No] Did you consult an outside source (such as an Internet forum or a book other than the course textbook) on parts of this assignment?

If you answered Yes to one or more questions, please give the details here:

By submitting this sheet through my Blackboard account, I assert that the information on this sheet is true.

This disclosure sheet was based on one originally designed by Profs. Royer and Older.

Kevin Lopez
CIS477
11/24/22
HW6

1) **Problem 1:** Suppose you are given a weighted, undirected graph G with vertex set V, a node $s \in V$, and a set of nodes $U \subseteq V$. Assume that s is not in U. Design an efficient algorithm to find the lightest weight spanning tree T of G such that every node in U is directly connected to s in that tree by a single edge. If it is not possible to do so, your algorithm should say so.

a) To do this algorithm, you start at node s and do a breadth first search to all adjacent nodes. By that notion, S would be connected to every possible node of U by a single edge. Yet this is incomplete because not every node in vertex set V is available in U, U is a part of V but not completely connected through V as it has less nodes.

Problem 2: Recall the stepping stones problem from the last homework, in which you are trying to cross a river by stepping on a sequence of stones. Suppose that you have the same problem, except now each stone has a *slip score* s_i. s_i ranges from 0 to 1, and a slip score of p indicates that if you step on that stone, there is p probability of slipping and falling into the river. As before, you begin at distance 0 and wish to cross to distance R, and can only step from one stone to another if they are within M units of distance of one another. Like the previous problem, you can only move forward, not backward.

Design an efficient algorithm to find the route across the river that has the lowest probability of falling into the river. Your algorithm should report which stones to step on. It may be helpful to remember the rules of probability: If one event has probability x and another has probability y, the probability of both occurring is xy (assuming they are independent, which in this case, they are).

2)

a) You can solve this problem with a dag pathfinding algorithm. You assign the nodes with the probabilities of slipping on them. The edges are only connected if they are M units of distance of each other and only face forward. You then multiply each of the paths that are connected and surpass R distance and find the lowest probability of slipping.

Problem 3: Suppose that you are traveling along a highway by bus. There are n bus stations along the highway. At each station, you can either get off your current bus, get on a new bus, or stay on your current bus. The

1

cost for taking a bus from station i to station j is given by $c_{i,j}$. There is no relationship between the different c values. You cannot go backwards (i.e., you can only go from station i to station j if $j > i$). You need to go from the first bus stop to the last bus stop.

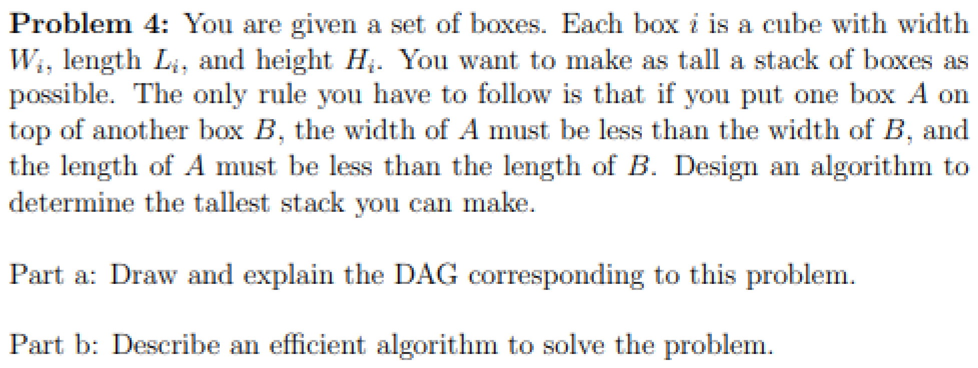
(a) Design a dynamic programming algorithm to determine the minimum cost of such a trip. Hint: Let $C[i]$ represent the cheapest way to get to rental shop i.

(b) Now suppose that there are also bus routes that go backwards, and they have their own costs (if $c_{i,j}$ is the cost of going from i to j, then $c_{j,i}$ is the cost of going from j to i). Assume all costs are positive. Design an efficient algorithm to determine the minimum cost of a trip from the first bus stop to the last bus stop.

3)

a) C[i] is the cheapest way to get to shop i. You can do this by creating the subproblem of finding the next cheapest bus. At each station starting at station 0. If the cost of getting to station j and taking another bus is less than staying on the current bus, you take the next bus. You keep doing this for all n bus stations to the last station i.

b) You can solve this with a DAG pathfinding algorithm. In this case the nodes are represented as the n bus stations and the edges are weighted with the cost C_{ij} of getting to the next bus station. Starting at bus station 0, you find all the possible paths to get to the final bus station. You then choose the cheapest solution and find your answer.



Problem 4: You are given a set of boxes. Each box i is a cube with width W_i, length L_i, and height H_i. You want to make as tall a stack of boxes as possible. The only rule you have to follow is that if you put one box A on top of another box B, the width of A must be less than the width of B, and the length of A must be less than the length of B. Design an algorithm to determine the tallest stack you can make.

Part a: Draw and explain the DAG corresponding to this problem.

Part b: Describe an efficient algorithm to solve the problem.

4)

a) The dag will be a longest pathfinding search algorithm however it will have a caveat. Instead of one value per node, it will have two. In this case, it will be the width and length. The edges will only point to the next node if the width and length are less than the node previous to it. Then you will find the longest series of edges. Another way of doing this is by making the edges be equal to the height and finding the largest sum of edges in the DAG.

b) To do this algorithm, you find the maximum amount of ways to stack the boxes and choose the one with the largest height. You iterate and organize the boxes based on length and width in increasing order, but starting with the first element as the bottom box. Then you add up the heights based on the list and they fit the conditions. You continue iterating and finding the combinations based on each box. Then choose the largest height

Extra Credit: Suppose you are attempting to determine the chemical composition of a mountain. You have a list of possible substances, and to determine whether a substance is present in the mountain, you can see how it reacts to a laser probe at a particular frequency. Each substance s_i has a range $[l_i, h_i]$, where $l_i < h_i$, of frequencies that it will react to; if you shoot a laser at a frequency within this range, then if the substance is in the mountain, you will see a reaction unique to that substance. For example, suppose there are three substances you are testing for, with frequency ranges [1, 3], [4, 6], [1, 5] (in MHz). If you fire a laser at frequency 2 MHz, that laser lies within the frequency ranges for substances 1 and 3, so it will test for those substances, and if you fire a second laser at frequency 5 MHz, that will test for substance 2.

Because laser probes are expensive, your goal is to test for each possible substance with the smallest number of laser probes. Design an efficient algorithm to determine the minimum number of laser probes needed to test for all substances. (Just return the number, not the frequencies.)

5)

a) This is similar to the scheduling program, except you are looking for the most overlaps instead of scheduling them in separate groups. You can order the frequencies by most numbers overlapped. This also ensures the fewest number of laser probes because it combines the frequencies that can be tested together.