Jonathan Arenson May 10, 2019

Greedy Algorithms & Graphs Assignment

1. Experiments Scheduling

a. Describe the optimal substructure of this problem.

If student signed up for a step, schedule that student for the current step. Move to the next step and decrement total # of steps. Also, if the student doesn't sign up for the current step you have to find the student that has the most ones and make that student the current student. Continue doing so until schedule is filled and there are no more steps.

b. Describe the greedy algorithm that could find an optimal way to schedule the students.

Choose the student who is available to do the steps that were signed up for. The students should do the max steps that are left. Continue on until all steps are finished using the fewest amount of students.

c. The actual code

d. What is the runtime complexity of your greedy algorithm?

The runtime complexity is O(mn^2). This is because n amount of steps and m amount of students.

Total number of steps * cost of each step = n * n * m

In time complexity is = O(n) * O(n*m) = O(mn^2)

e. Give a full proof that your greedy algorithm returns an optimal solution.

Alg: S1, S2, …, Sn

Opt: O1, O2, …, On

Let's say the alg has j students and opt has n students.

Assume j > n.

There is an *i* where the alg and opt don't have same amount of switches. The alg will schedule the student to the current step if that student had signed up for that. If the case is that the alg doesn't switch students it proceeds to Sn where it'll eventually switch out. As a result, it would become S1, S2,..Sz, Oz+1,Oz+2, ..On and then the alg would then have the same amount of switches in the end.

2. Public, Public Transit

a. Describe an algorithm solution to this problem.

This can be solved using Dijkstra's algorithm with a couple of changes. The most important change is to consider the wait time of the public transit from u to v. Next, calculate the wait time from u to v. As arrival time at u is given, the wait time can then be calculated from this and the frequency. After the wait time is figured out then add that to the cost of the path to v, which goes through u.

b. What is the complexity of your proposed solution in (a)?

The complexity is O(V^2) as it is similar to Dijkstra's algorithm.

c. See the file FastestRoutePublicTransit.java, the method "shortestTime". Note you can run the file and it'll output the solution from that method. Which algorithm is this implementing?

This algorithm is implementing Dijkstra's single source shortest path.

d. In the file FastestRoutePublicTransit.java, how would you use the existing code to help you implement your algorithm? The existing code only handles one piece of data per edge, so describe some modifications.

Since the existing code only handles one piece of data per edge to get the time between u and v. The algorithm has to account for the wait time depending on if the train is there or not.

e. What's the current complexity of "shortestTime" given V vertices and E edges? How would you make the "shortestTime" implementation faster? Describe any algorithm changes or data structure changes. What's the complexity of the optimal implementation?

The current complexity of “shortestTime” is O($V^2 * \lg v$). If the algorithm were changed to either Fibonacci heap or adjacency list the complexity would then improve to O(E + V lg V).

f. The actual code

g. Extra credit (15 points): I haven't set up the test cases for "myShortestTravelTime", which takes in 3 matrices. Set up those three matrices (first, freq, length) to make a test case for your myShortestTravelTime method. Make a call to your method from main passing in the test case you set up.