Adam vanWestrienen
CSCI 323 – Prof. Yao
Assignment #2 – Greedy Algorithms & Graphs Assignment

1. Experiments Scheduling

a) Describe the optimal substructure of this problem.

Because we want to minimize the number of switches overall, our optimal subproblem is going to include the current step plus the student(s) who have signed up for the greatest number of steps. These students will also be in the optimal solution. The formula this optimal subproblem is for k steps, when $1 \leq k \leq n$, an optimal solution for k steps will be part of the solution for k + 1 steps.

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

The greedy algorithm to find an optimal way to schedule students is somewhat like the Activity Selection Problem, where the greedy choice is to always pick the next activity whose finish time is least among the remaining activities and where the start time is more than or equal to the finish time of the previous activity. However, in scheduling students we want to look at what is the next sequential step and then which student has signed up for the most steps.

Because we are trying to finish all the steps in order with the least amount of switching between students, our goal is then to minimize the number of different students involved. Therefore, the greedy choice is to always select and schedule the students who can do the greatest number of steps in relation to the next step. If we do continually do that, it will lead us to the optimal solution of least number of switches.

c) Code

See code in PhysicsExperiment.java

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

With the steps for each student already sorted the runtime is O(m*n) where m is the number of steps and n is the number of students.

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

Lets say we have an algorithm that returns $u_1, u_2, \ldots, u_n$ with L number of switches as a solution. Furthermore, lets assume there exists an optimal algorithm, $v_1, v_2, \ldots, v_n$ with K number of switches which is the optimal solution, and K < L.

Claim: $i(1 \leq i \leq n)$ is the smallest index where $u_i \neq v_i$

If both algorithms are the same and make same switches of steps then ($u_1, u_2, . . ., u_i, v_{i+1}, . . ., v_n$) is also a solution. However, if my algorithm continues on consecutive steps and the optimal algorithm switches, then we can use my algorithm throughout, continually modifying the optimal algorithm using the cute and paste method until my algorithm replaces the optimal algorithm resulting in ($u_1, u_2, . . ., u_i, v_{i+1}, . . ., v_n$). But by design of our algorithm if there exists a student who can do the current step plus more consecutive steps resulting in fewer switches, the algorithm will select that student first. This contradicts that there is an algorithm that can make less switches than our algorithm. Thus our greedy algorithm will give us the optimal solution.

2. Public, Public Transit

a) Describe an algorithm solution to this problem.

One solution to this problem (as we see with the “shortestTime” method) is to use a variation of Dijkstra’s shortest path algorithm. The algorithm works like a minimum spanning tree solution where you keep track of the shortest distance or weight at each step of the way. You have the distances traveled in one set and tracking which vertices have not been included in another set. Therefore, if you come to a route with a shorter path you can update the distance and know which have already been accounted for. Here, however, instead of just keeping track of the distance or length, we need to account for a few other data points for each route to our destination. From u to v we need to account for the first train of the day, how frequently the train passes through, and the start time x.

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

The complexity of the proposed solution would be $O(E + V^2)$ where E is number of Edges and V is the number of vertices in the graph. This reduces to $O(V^2)$.

c) Which algorithm is the “shortestTime” method implementing?

The method “shortestTime” is implementing the Dijkstra’s shortest path algorithm which is a greedy algorithm.

d) How would you use the existing code to help you implement your algorithm?

To implement “myShortestTravelTime” algorithm, we can use Dijkstra’s shortest path algorithm used in the “shortestTime” method and expand it to include the additional parameters we need to account for. To do this we include graphs for first trains and how frequently trains pass through so we take those measurements into account when we update our time array.

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

The current complexity of "shortestTime" is $O(E + V^2)$. To implement "shortestTime" to be faster you would need to represent the input graph as an adjacency list instead of the matrix. Additionally, you would use a heap to track each vertex in the graph whereby you are removing which vertices are the minimum rather than updating an array. The complexity of this updated solution would be O(E log V).

f) Code

See "myShortestTravelTime" method in the file FastestRoutPublicTransit.java.
Please Note I also used a few helper methods:
- "calculateTime" to calculate travel time from one station to the next.
- "toAlpha" to convert station numbers to letters to make the graph easier to see.
- "runTest" to run "myshortestTravelTime"

g) Extra Credit

See the code in the main method of the file FastestRoutePublicTransit.java

Resources for this assignment include:

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

https://www.geeksforgeeks.org/dijkstras-algorithm-for-adjacency-list-representation-greedy-algo-8/

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/