CS323
Assignment 2
Dennis Chang

Physics Experiment

a. Describe the optimal substructure of this problem.

$For\ a\ value\ k, where\ 1 \leq k < n,$
$an\ optimal\ solution\ for\ k\ steps\ is\ part\ of\ a\ solution\ for\ k + 1\ steps$

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

Starting from step 1, select the student with the longest consecutive steps and schedule that student. Start from the step afterwards and select the longest consecutive steps and schedule that student. If there are more than one student that share the longest consecutive steps, choose the first one.

d. What is the runtime complexity of your greedy algorithm? Again, you don't need to factor in the setup of the lookup table, just scheduling algorithm.

$O(m * n)$
The worst-case scenario would be if each student could be present for each step; my greedy algorithm would check the longest consecutive steps each of m students could take, which would be n iterations. This would result in m * n checks.

e. In your PDF, based on your answer to part b, give a full proof that your greedy algorithm returns an optimal solution.

$The\ ALG\ solution\ returns\ a_1, a_2, a_3, \dots, a_n\ with\ x\ switches$
$Assume\ OPT\ solution\ returns\ o_1, o_2, o_3, \dots, o_n\ with\ y\ switches$
$where\ x > y$

$At\ some\ value\ i, where\ 1 \leq i \leq n, a_i\ and\ o_i\ differ$
$Since\ ALG\ takes\ the\ longest\ consecutive\ steps, a_i\ and\ o_i\ can\ only$
$differ\ at\ value\ i\ both\ ALG\ and\ OPT\ solutions\ are:$
$1. making\ a\ switch$
$2. or\ OPT\ is\ making\ a\ switch\ while\ ALG\ doesn't$

$If\ both\ algorithms\ are\ making\ a\ switch, then\ a_1, a_2, a_3, \dots, a_i\ and$
$o_1, o_2, o_3, \dots, o_i\ are\ the\ same\ and\ the\ former\ can\ replace\ the\ latter$
$resulting\ in: a_1, a_2, a_3, \dots, a_i, o_{i+1}, \dots, o_n$

$If\ ALG\ is\ continuing\ while\ OPT\ is\ switching, we\ can\ continue\ with$
$ALG\ until\ a_s\ where\ ALG\ switches\ and\ replace\ o_1, o_2, o_3, \dots, o_s\ with$
$a_1, a_2, a_3, \dots, a_s\ (since\ ALG\ has\ less\ switches)\ which\ will\ result\ in$
$a_1, a_2, a_3, \dots, a_s, o_{s+1}, \dots, o_n$

$We\ can\ use\ the\ cut - and - paste\ method\ until\ ALG\ replaces\ OPT$
$This\ would\ result\ in\ ALG\ with\ the\ same\ or\ less\ number\ of\ switches$

Public, Public Transit

a. Describe an algorithm solution to this problem. Feel free to talk about how you would adapt an algorithm we covered in class.

The Dijkstra algorithm would be suitable for this problem since it's a single source problem. When computing the weight of the path, we would also need to account for any wait time that might occur when a transferring train is not currently available. To figure out the wait time, we have two scenarios:

1. the first train has not arrived yet or
2. the first train has already passed (and a subsequent train is not here)

If the first train has not arrived yet, we will have to wait for the first train. We need to add a wait time equal to:

$$waitTime = firstArrivalTime - currentTime$$

If the first train has already passed and a subsequent train is not here yet, we need to figure out how long until the next train:

$$waitTime = frequency - ((currentTime - firstArrivalTime) \% frequency)$$

$(currentTime - firstArrivalTime)$

Would tell us how long it has been since the first train has passed

$((currentTime - firstArrivalTime) \% frequency)$

Would tell us how long it has been since the last train has passed since trains arrive at regular frequencies

$frequency - ((currentTime - firstArrivalTime) \% frequency)$

Would tell us how long it will be until the next train

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

The worst-case scenario for a Dijkstra algorithm would be a dense graph resulting in an $O(\ (|V|^2)\ \log|V|)$. The addition of finding out wait time per transfer should be negligible; it only requires retrieving data from matrices and some simple arithmetic computations. The issue would be that a computation is required for every transfer. A dense graph might have $|V|^2$ transfers so an additional $O(|V|^2)$ might be necessary to add to run time resulting in an $O(\ (|V|^2)\ \log|V| + (|V|^2))$.

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?

The method is implementing a Dijkstra algorithm. It creates an array holding the shortest travel time to any given node, which values are initialized to a max value. Starting at the source node, we travel to adjacent nodes and test to replace the current travel time value with a lesser travel time value. This process is repeated until all nodes are processed.

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.

The only modifications would be additions to the current "shortestTime" method. Instead of just adding "times[v] = times[u] + graph[u][v]" which is summing node u's lowest time with the length of edge(u,v) to get the time for node v, the algorithm would add an additional wait time depending on whether or not a transferrable train is present. In my response to question(a), I outlined an additional wait time that would be added to the current computation of times[v]. The additional wait time will be computed using the additional first & frequency matrices.

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 $O(\ (|V|^2)\ \log|V|)$.
According to Wikipedia, storing the graphs in the form of an adjacency list and using a self-balancing binary search tree, binary heap, pairing heap, or Fibonacci heap as a priority queue would result in better efficiency.
The use of a Fibonacci heap results in complexity of $O(\ (|E|) + |V| \log|V|)$.