Asim Shariff CSC 323

<u>Experiments Scheduling</u>

a) The optimal solution is the solution that complete all the steps in order with the fewest number of switches. So, if we schedule the students that can do the most consecutive steps in ascending order, then this would minimize the number of switches, since the student are doing as many steps in a row as they can. Suppose that we are trying to assign *k* number of steps to *j* number of students. Suppose the optimal solution has Student *i* completing steps {*m*, … *n*}, where *m* < *n* and *m* to *n* are consecutive numbers in increasing order. Now we have two subproblems: which student will do steps {1, … , (*m*-1)} and which student will do steps {(*n*+1), … ,*k*}. These two subproblems can be solved the same way as the overall problem, so this shows that the problem has an optimal substructure. [I used the meeting schedule greedy problem from class as a guide/inspiration].

b) The greedy algorithm that can be used to find the optimal solution would be to schedule the students that can do the most consecutive steps in a row. This means that we are scheduling the student can do as many steps in a row in ascending order. If a student can do a lot of consecutive steps in a row, then this would minimize the amount of switching it would take between students to complete all the steps.

c) Code is in the file PhysicsExperiment.java

d) The while loop completes once the number of remaining steps is equal to 0. To find the student that can finish the most steps in a row or largest number of consecutive 1's in the signUpTable would take O(n^2) in the worst case. Everything else in the while loop would take constant time. So, in the worst case it will take O(n^3) if we take in account the time it takes for the while loop times the time it takes to perform everything in the while loop. So, the greedy algorithm I described will take polynomial time in the worst case.

e) Greedy Algorithm: Always schedule the student that can do the most consecutive steps in ascending order, while there are still steps to be scheduled.

Proof: Let ALG represent our algorithm.

ALG : $<S_1, S_2 , \dots , S_K>$

Let S_i represent a student that can perform *x* number of consecutive steps in ascending order. This means that *k* indicates the total number of students we have, so it (*k*-1) will indicate the number of switches we have after we complete all the steps. Note, say Student A can do steps 1,3,4 and Student B can do step 2, for the proof we will consider Student A as two separate students, since the algorithm considers consecutive steps in ascending order. So, we would have Student A be considered as S_1 and S_3 and Student B be S_2. So, in total we will have 3 students after we complete all the steps, which is equal to 3-1 which is 2, and this is the total number of switches that we have after we complete all the steps. Assume there exist some optimal solution, OPT, which claims to have fewer number of switches than our algorithm.

OPT: $<S_1', S_2' , \dots , S_L'>$, where L<K.

So, the optimal solution would have less number of students, which also means that it would have less number of switches. Let *i* $(1 \leq i \leq L)$ be the smallest index where $S_i \neq S_i'$. So, all the students up to *i* is the same ($<S_1, S_2 , \dots , S_{(i-1)}> = : <S_1', S_2' , \dots , S_{(i-1)}'>$). So, at the *i*th index we have different students performing a different amount of steps. We can replace S_i' with S_i because the algorithm takes the student that can perform the longest consecutive number of steps in order and it wouldn't worsen the optimal solution, since if a student can finish the longest number of steps in order, then that shouldn't increase the total number of switches. Now we would have the same students performing the same steps in both solutions up to $S_{(i+1)}$ and $S_{(i+1)}'$. We can use this same cut and paste logic up to S_L and S_L'. The optimal solution claims that this is the last student needed to complete all the steps with the least amount of switches. Our algorithm only stops if all the steps are scheduled in order, so if the optimal solution stopped earlier, then there must have been some steps that were skipped or missed. This is a contradiction, since there is no optimal solution that can schedule all the steps with a fewer number of switches. Therefore, our algorithm yields an optimal solution.

<u>Public, Public Transit</u>

a) I would adapt Dijkstra's algorithm that we learnt in class to help solve this problem. Dijkstra's algorithm is used to help find the shortest path of a graph. So, I would modify the algorithm to keep track of the path that we took. So, it would keep track of all the nodes or stations that we stop at, on the way to the destination along the shortest path. Once I have this information, I would use the first matrix, frequency matrix, and starting time to find the next available train on the shortest path that we found from Dijkstra's algorithm. I will also add in the time it took to wait for the train at each stop on the shortest path and the time it took to get from one station to the next station that was on the shortest path.

b) Dijkstra's algorithm would take worst case O (V^2), if I implement Dijkstra's algorithm using a priority queue and a min heap or just use a Fibonacci heap, this worst case can be improved. Once I have the information of the shortest path from Dijkstra's algorithm, traversing through the shortest path and calculating the shortest time it takes to get from station u to station v would take O(n) in the worse case, where n is the length of the shortest path or number of stations along the shortest path. So the overall runtime would be O ($V^2 + n$), n would always be less than or equal to the number of vertices, since n is the path we are taking so we can have at most V stations along a path, so this runtime can be simplified to O (V^2).

c) The algorithm that this is implementing is Dijkstra's algorithm.

d) I would modify the algorithm to help keep track of the stations along the shortest path. To do this, I would have another array, which keeps track of the previous nodes on the graph. I could use this array to backtrack my way from the destination to the starting point and find the shortest path that we took.

e) The current runtime complexity of “shortestTime” is O (V^2), where V is the number of vertices. If we keep track of which vertices have edges between them in the graph in an adjacency list or adjacency matrix, and we use a binary heap, then we can make the algorithm faster. The runtime complexity of the optimal implementation is O (E log V), where E is the number of edges and V is the number of vertices. [I used the website below to figure out how to make Dijkstra’s algorithm faster: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/]

f) Code in FastestRoutePublicTransit.java

g) Code in FastestRoutePublicTransit.java