Federico Toscano
Collaborator: Karina Ibragimova
Greedy Algorithms

Experiments Scheduling

Problem A)

This problem exhibits optimal substructure because the optimal solution is achieved by scheduling the student with the most uncompleted steps. Then the sub problem of scheduling the student with the most uncompleted which do not include the steps completed by the first student. Then the sub problem of scheduling the student with the most uncompleted which do not include the steps completed by the first student and second student, etc.

Problem B)

The greedy algorithm that can solve this problem is achieved by greedily choosing the student with the most steps that are able to completed out of the ones that are left. This process will then be repeated until all steps are marked off as completed, resulting in the least amount of students needed.

Problem D)

The runtime of my greedy algorithm is O(n*m) where n is the number of students and m is the numbr of steps.

Problem E)

Hypothesis:
The algorithm return $S_1, S_2,\ldots, S_n$ with z different students
The optimal solution return $A_1, A_2,\ldots, A_n$ with v different students
With z > y
There exists i such that S_i != A_i which can occur in only one scenarios being that the optional solution makes a switch but my algorithm does not.
If the algorithm doesn't switch students, it will continue, until S_b where the algorithm finally switches. This would make the algorithm result in: $S_1, S_2,\ldots, S_b, A_{b+1}, A_{b+2},\ldots, A_n$. Eventually this would result in the algorithm have at most the same number of switches.

Federico Toscano
Collaborator: Karina Ibragimova
Greedy Algorithms

Public Transit Problem

Problem A)

This problem is simply a modified version of Djikstra's algorithm. Dijkstra's algorithm finds the shortest path between the source node and all other nodes in the graph. However the difference between this problem and Djikstra's is that we will need to account for the wait time that may occur if a train is not available at the current time because it is not at the station. If the first train hasn't arrived the wait time will be the first arrival time subtracted by the current time. If the train has arrived at least once but isn't available, then the wait time is the frequency of that train subtracted by the currentTime – firstArrivalTime % frequency.

Problem B)

The worst case scenario occurs similarly to Dijktras algorithm's worst case which is when the graph provided is a dense graph. The time complexit of this is $O(|V|^2 * \log|V|)$. However in the solution provided above, we also need to factor in the computing the wait time for every transfer which could at worst occur V^2 times. This result in a worst case time complexit of $O(|V|^2 * \log|V| + |V|^2)$.

Problem C)

It's implementing Dijktra's algorithm.

Problem D)

In order to use the existing code to implement my algorithm, the shortestTime method would also need to take into account any additional wait time that may be necessary depending on whether the train is present or not at the current time.

Problem E)

The time complexity of the shortestTime algorithm give V vertices and E edges is $O(|V|^2 * \log|V|)$, as stated in Problem B. If the graph used a Fibonacci heap instead of an adjacency matrix, the time complexity can be brought down to $O(|E| * \log|V||V|)$ which is significantly better.

Problem F)

Method implementation found within FastestRoutePublicTransit.java

Problem G)

Test case is found after line 172 where the professors comment is located.