Greedy Algorithms & Graphs Writeup

Experiments Scheduling:

a) Optimal Substructure:

When you know student i doing the next x possible steps they are available for is in the solution, you can append the optimal solution of the remaining n-x steps to find the overall optimal solution.

b) Greedy Algorithm:

A greedy algorithm to solve this problem would always choose the student who is available to do the highest number of successive next steps, and have them do all those successive possible steps. The pseudocode for the greedy algorithm is as follows:

```
Set current student to none
Iterate through each step:
    If current student can do current step, let them
    Else:
        Set max to 0
        Iterate through students:
            Set count to 0
            Iterate through remaining steps:
                If student can do step, increase count
                Else, break
            If count > max, set current to student and max to count
        Current student does current step
```

d) Runtime Complexity:

$O(mn^2)$ because it's doing constant work inside a loop through steps, inside a loop through students, inside a loop through steps.

e) Proof of Optimal Solution:

Public, Public Transit:

a) Algorithm Solution:

This problem can be solved with a simple modification to Dijkstra's algorithm for finding a single source shortest path. Dijkstra's algorithm already solves for the shortest path between any two nodes given the edge weights, therefore, the only necessary modification is to account for wait time. With Dijkstra's algorithm, this is simple, as you can solve and account for the wait times from u to v during the process of relaxing edge u, v. Calculating the wait time from u to v is simple, as you already have the arrival time at u and can figure out wait time from that and first & freq. Once you have the wait time, you simply add it to the cost of the potential path to v that goes through u during the relaxation process.

b) Complexity of Proposed Solution:

Since the modifications made to Dijkstra' algorithm won't increase the order of the run time, the run time is on the same order as the plain Dijkstra' algorithm. Therefore, a simple version would be O(|V|^2), but with more advanced data structures this can be improved upon further.

c) shortestTime Algorithm:

shortestTime is implementing a simple version of Dijkstra's algorithm for finding a single source shortest path.

d) Code Modifications:

As mentioned earlier, there is only one main modification needed to adapt Dijkstra's algorithm (which the code implements) to this problem - which is accounting for the wait time. To do this, when deciding to and updating the time value of adjacent vertices, the necessary time spent waiting before traveling to the adjacent vertice is calculated and added to the travel time. Calculating the wait time is simple, as the arrival time at the current vertice is known and the first & freq are given.

e) shortestTime Runtime Improvements: