// SCHEDULING PROBLEM

Given: **n** number of steps, **m** number of students that give you a list of steps (sorted) they can participate in. Assume there's a lookup table where you can find if student X signed up for step Y in O(1), so no need to factor that into your runtime.
Find: An optimal way to schedule students to steps such that there is the **least amount of switching** as possible.

(a) Describe the optimal substructure of this problem

From the current step, choosing the student who can do the most consecutive steps will minimize the number of switches we have to make.

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

Start at 1. Set current step = 1.
While the current step is not the last step,
    Check how far each student can go from the current step.
    Schedule student who can go farthest – the greedy choice.
    Set current step to the last step of this student's consecutive sequence of steps.

(c) Code your greedy algorithm

See java file.

(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 your scheduling algorithm.

Worst case is if at the first step:
For every student, we check how many steps they can do: O(m)
    Every student can do all n steps: O(n)
Total is O(mn)

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

Invariant: Choosing the greedy solution, the longest consecutive sequence, will minimize switching.

Proof: Assume the opposite, that there exists some optimal solution OPT whereby we minimize switching without taking the longest consecutive sequence, and that this is better (i.e., has less switches) than our greedy solution, GREEDY. We can use the copy and paste method.

OPT and GREEDY start at the same step, step 1. Let step x be the first step when OPT differs from GREEDY. Then, OPT would have switched to a new student, because GREEDY does not switch unless the student cannot go any further. We can replace this switch by copying and pasting GREEDY here, so OPT has one less switch, making it even more optimal. We then continue comparing them until they differ again. We repeat the copy & paste if OPT switches before GREEDY.

If OPT and GREEDY switch at the same time, but OPT chooses a different student than GREEDY, we know that GREEDY chooses the student with the longest sequence. Therefore, OPT must have chosen a student with a shorter sequence and therefore more switches in the same amount of steps. We can copy and paste GREEDY's choice onto OPT to get a longer sequence without switches.

We continue to compare difference in OPT and GREEDY and copy and paste GREEDY onto OPT, since any differences reveal OPT to be switching sooner than necessary or choosing a sequence that would require them to later switch sooner than necessary. We eventually end up with OPT = GREEDY. This is a contradiction. Therefore, GREEDY must be an optimal solution.

Fastest Route

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

You can take the shortest edge from vertex S until you reach vertex T.
You have to only visit unvisited vertices.
For every current station, you check all un-visited vertices.
You have to keep track of which vertices you visited using a boolean visited array.
and make sure that your "findNextToProcess" method that finds the shortest edge does not lead you to an unvisited vertex.

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

Worst case we need to traverse every vertex, so it would be the same as the complexity of a shortest path problem. If we use Prim's algorithm with an adjacency matrix, this would be O(|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?

Prim's algorithm.
It takes a starting point and finds the shortest edge (time) from that point, then continues taking the shortest path from this and each subsequent point until all points have been 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.

I used the numVertices and processed array formats to help keep track of the vertices.
I use a shortest path algorithm where each edge value is the time it takes to get to adjacent vertices.
I added variables startTime, nextTrain, minTime, nextStation, and thisTripEndsAt to keep track of the total length of the next trip, taking starting times and frequency into account.

I modified the code so that we do not have to visit every station in order to complete the algorithm:

```
while (u != T && visited[u] != true) {
```

Within this while loop, I added a for loop to iterate through each vertex, checking if it had been and could be visited or not. If it was possible to go there (and hadn't been previously visited), I used the variables mentioned above to store the shortest time to another unvisited station.

At the end of the for loop we are left with the shortest time to the next station.

Before the while loop resumes my algorithm takes the shortest trip to the next station and updates the variable u to be the next station.

(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?

Current complexity is O(|V|^2) + 2V . // 2V is for set times and processed arrays and print times array
You could use a hash table or hash map to avoid having to iterate through the arrays.

(g) Code! In the file FastestRoutePublicTransit.java, in the method "myShortestTravelTime", implement the algorithm you described in part (a) using your answers to (d). Don't need to implement the optimal data structure.

See java file.

(g) Extra credit (15 points): I haven't set up the test cases for "myShortestTravelTime", which takes in 3 matrices. Set up those three matrices to make a test case for your myShortestTravelTime method. Make a call to your method from main passing in the test case you set up.

See java file.