1. Experiments Scheduling
    a. Describe the optimal substructure of this problem

       Assume there are n total steps and m students, the optimal substructure is that for each m student we are able to find most consecutive step he signed up for and fits in the total n step, we will record their steps and who done these steps down, and total number of steps completed, and compare with given total steps, until the end when there are no more total steps left, the optimal solution for this algorithm is found.

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

       Step1: keep track the number of steps we going to assign with given total number of steps. Keep running until no more steps left to assign.

       Step2: check the student, if it did not sign up from current step, search all student who can will do most consecutive steps from current step.

       Step3: we assign the step2 most consecutive step to that student and increase the steps by number of consecutive step the student can do. Then go back to step1.

    c. PhysicsExperiments.java

    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.

       The runtime complexity is O($m*n^2$), where m is the student, n is the step, one nested for loop in the while loop.

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

       Assume SOL is our solution
       SOL:S1,S2,S3,…SL
       Assume there is an optimal solution with least switch
       OPT: S1’,S2’,S3’,…SK’ where K < L

       Suppose the OPT == SOL from step 1 to step i, i < K < L
       S1,S2,S3,…Si-1 = S1’,S2’,S3’,…Si-1’

       By design of our algorithm, max number of consecutive step Si is from student i, so in OPT we replace Si’ with Si, similarly, we can replace until SL, we will get SL =

SK, and L = K, this contradicts with K < L, so our algorithm SOL is at least good as optimal solution OPT.

2. 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.

      Dijkstra's shortest path algorithm will used to solve this problem

      Step1: create two array, both has length of number of vertices, one is time[] initialized to max value, another one is processed[] initialized to false.

      Step2: find the next node to process if it has not yet processed

      Step3: calculate the shortest time from given adjacency matrix of the node that has not yet processed. Time calculated based on distance of node and the wait time, wait time is calculated with arrive time and first train time, wait time need to consider train frequency if we miss the train, arrive time is calculated with our start time and time to get to this node, we keep update the time[] to get the shortest time

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

      Time complexity is O(v^2), one nested for loop.

   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?

      This method is implemented with Dijkstra's shortest path algorithm, it calculates the shortest path from given source, and there is no negative weight.

   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.

      Modify the shortestTime method,
      ```
      Below if (!processed[v] && lengths[u][v]!=0 && times[u] != Integer.MAX_VALUE && times[u]+lengths[u][v] < times[v])
      ```
      Need to calculate arrival time, since we given start time and time to get to a station, and wait time if the first train pass because first train time greater than arrival time, we then keep compare and store the shortest time.

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 of “shortestTime” is O(V^2), change the adjacency matrix to adjancency list, then we are able to use binary heap which the time complexity would become O(ElogV).

f. FastestRoutePublicTransit.java
g. FastestRoutePublicTransit.java

Collaborates with Ning Wei