Kirsten Pevidal

Write ups for Algorithm Assignment: Greedy and Graphs

- ★ Experiments Scheduling
  - **Describe the optimal substructure of this problem**
    - If the currentStudent is signed-up for the currentStep, schedule currentStudent for currentStep. Then, reduce remainingSteps to do then go to next step
      if the currentStudent is NOT signed-up for the currentStep, look for a student with the most signed-up steps. If there are still remaining steps, search for a student that is signed-up for the current currentStep. Repeat steps until remainingSteps is zero.
  - **Describe the greedy algorithm that could find an optimal way to schedule the students**
    - Pick the student with the most consecutive steps and repeat until remainingSteps is zero. This gives us the least/minimum of switching needed to do all steps.
  - **Code your greedy algorithm in the file "PhysicsExperiment.java" under the "scheduleExperiments" method where it says "Your code here". Read through the documentation for that method. Note that I've already set up the lookup table automatically for every test case. Do not touch the other methods except possibly the main method to build your own test cases (but delete/comment out your own test cases in the submission). Run the code without your implementation**
    - See PhysicsExperiment.java
  - **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.**
    - O(n^3) --->the code has a while loop that continues until no more remainingSteps, inside is a for loop to find student with the most consecutive steps, and lastly, another loop to check consecutive steps

- **(e) In your PDF, based on your answer to part b, give a full proof that your greedy algorithm returns an optimal solution.**
  - Assume there exists an OPT Solution using fewest switches.
    Algorithm: S(1), S(2), ..., S(K)
    OPT: S(1'), S(2'),...., S(L) where L < K
    Say i is where alg and opt have different number of switches. By design, alg schedules either current student or student with most consecutive steps depending in the currentStep. We replace S(i') with S(i) and it will NOT worsen the OPT. Since S(i') =S(i+1), we can apply cut & paste technique.
    OPT claims that S(i) is the last student needed to complete all the steps with the least amount of switches. Our algorithm only stops if all all steps are scheduled(remainingSteps=0), This means OPT must have steps that were skipped or missed. This creates a contradiction, since an OPT that can schedule all the steps with the least number of switches DNE. Therefore, our algorithm yields the OPT.

★ Public, Public Transit

- **Describe an algorithm solution to this problem. Feel free to talk about how you would adapt an algorithm we covered in class.**
  - Use Dijkstra's algorithm + calculating the wait time for the next train
- **What is the complexity of your proposed solution in (a)?**
  - Dijkstra's Algorithm implementation has a time complexity of
    -> O(|v|^2 * log|v| + |v|^2)
- **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?**
  - It uses Dijkstra's Algorithm.
- **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.**
  - The shortestTime method will be used to include the additional waiting time to the total travel time.
- 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?

    - "The current runtime complexity of "shortestTime" is O (V2), 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."
      Source: https://www.geeksforgeeks.org/dijkstras-shortest-pathalgorithm-greedy-algo-7/

- **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 FastestRoutePublicTransit.java
- **Extra credit (15 points): I haven't set up the test cases for "myShortestTravelTime", which takes in 3 matrices. Set up those three matrices (first, freq, length) 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 main method of FastestRoutePublicTransit.java