Stacy Murray in collaboration with Polina Chernomaz

Experiments Scheduling

(a) Optimal Substructure
    a. Count the number of consecutive steps each student had and keep track of it
    b. The student that has the most consecutive steps will do those steps
    c. Update the table with these values
    d. The steps that have been claimed no longer need to be checked so decrement the number of steps
    e. Move on to the student with the largest amount of consecutive steps
    f. Continue this until there are no more steps

(b) Greedy
    a. The student that has the most steps in order will do those steps
    b. The steps that have already been assigned no longer need to be done so decrease the amount of steps remaining. Only check the steps that have not been assigned.
    c. This will be repeated for the rest of the steps until there are no more steps

(c)

(d) The runtime complexity should be O(m*n)

(e) Greedy Solution: Student 1 does A steps, Student 2 does B steps, Student 3 does C steps for a total of D switches

OPT : Student 1 does A' steps , Student 2 does B' steps , Student 3 does C' steps for a total of D' switches

Suppose D < D'

There are three possible cases

Case 1: If D' < D cannot be true because the in the greedy solution the student with the most consecutive steps will be assigned those steps.

Case 2: If D' > D then the optimal solution is not optimal and the student with the most consecutive steps was not assigned all of the steps they could have completed. We reach a contradiction that there is a solution that is more optimal

Case 3: if D' = D, then D' can just replace D in the optimal solution. If the number of switches are the same then the greedy solution is no worse than the optimal solution so it is optimal

Example if your experiment has 8 steps <1, 2, 3, 4, 5, 6, 7, 8>

- Student 1: <5, 7, 8>
- Student 2: <2, 3, 4, 5, 6>
- Student 3: <1, 5, 7, 8>

- Student 4: <1, 3, 4, 8>
  Your optimal solutions could be any one of these:

    - Student 1 does no steps, Student 2 does <2, 3, 4, 5, 6>, Student 3 does <1, 7, 8>, Student 4 does no steps --> 2 switches (student 3 does step 1, student 2 does up to 6, student 3 picks up again to do 7 and 8)
    - Student 1 does <7, 8>, Student 2 does <2, 3, 4, 5, 6>, Student 3 does <1>, Student 4 does nothing --> 2 switches
    - Student 1 does no steps, Student 2 does <2, 3, 4, 5, 6>, Student 3 does <7, 8>, Student 4 does <1> --> 2 switches

The optimal solution would have only 2 switches since the goal is to have the least amount of switches possible.

Based on the greedy algorithm:

- No steps have been assigned , we need to count the consecutive steps
- Student 2 has the most consecutive steps at 5 consecutive steps (2,3,4,5,6) so student 2 will do these steps
- Steps 1,7 and 8 are remaining
- Students 1 and 2 have the most consecutive steps at 2. Either of these students can do these steps. Student 1 can do steps 7,8.
- Only step 1 is remaining , since student two who has the most consecutive steps does not have step one , a switch in students has to be made so either student 3 or student 4 can do this step. Student 3 will do step one.
- Start at Student 3 for step 1, go to Student 2 for steps 2, 3, 4, 5, 6 and back to Student 3 for steps 7, 8. This solution has 2 switches, the same as the optimal solution.

Since two students can have the same amount of consecutive steps, either student would be able to be assigned those steps. The greedy solution and the optimal solution may assign different students the same steps but they will both do so while simultaneously minimizing the number of switches. In the end they will both return the same number of switches.

Public , Public Transit

(a) Using Dijkstras algorithm as a template for this question , the algorithm to solve this question is as follows

1. Mark the first station with a current distance of 0 and the rest with infinity
2. Initialize array that marks all stations as not visited
3. Set the non-visited station with the smallest current time as the current station, U.
4. For each neighboring station of the current station, add the current time it takes to get to U from the source + the wait time for the next train + the time it takes the train to get between U

and V. If it is smaller than the current distance of the neighboring station, set this as the new current distance of V.

5. Mark the current station U as visited
6. If there are non-visited stations, refer back to step 2

(b) The time complexity would be $O(V^2)$

(c) The algorithm being implemented is Dijkstras algorithm

(d) We modified the code to include the wait time for the next train at the station. To calculate the wait time we used the start time, the frequency of the train and the time of the first train. You have to make sure that the total time passed after train i stops at u is greater than or equal to the startTime to be able to leave the station.

(e) The time complexity can be reduced to O(E lg V) if the input graph is represented using adjacency list with the help of a binary heap. It can be reduced to O(E+VlogV) with the implementation of a Fibonacci heap.

We used GeeksForGeeks to reference Dijkstras algorithm.