Polina Chernomaz in collaboration with Stacy Murray

1. Experiments Scheduling

(a) Optimal Substructure – optimized solution to current subproblem depends on optimal solutions for previous problems.
   i. Count the number of consecutive steps each student signed-up for and store it.
   ii. The student with the maximum consecutive steps will be assigned those steps.
   iii. Fill steps scheduled for student in table.
   iv. Decrement the number of steps left by the number of steps assigned to student.
   v. Schedule the next student with maximum consecutive steps to those steps.
   vi. Repeat iii – iv until there are no more steps.

(b) Greedy – solves the problem by making the choice that seems best at the particular moment.
   i. Find the student with the maximum consecutive steps in order and assign the steps.
   ii. Mark the assigned steps in table and decrement steps left by number of steps just assigned. Look for the next student with most consecutive steps starting from the next step needed.
   iii. Repeat ii – iii until there are no more steps.

(d) The runtime complexity of our greedy algorithm is O(m*n)

for m students, n steps

(e) Let us prove this by contradiction. Say that there is an optimal solution that uses fewer switches than the greedy solution returns.

Assume different solutions.
Greedy Solution: Student 1 does p_1 steps, Student 2 does p_2 steps, Student 3 does p_3 steps, …, Student n does p_n steps for a total of S switches.
OPT : Student 1 does q_1 steps , Student 2 does q_2 steps , Student 3 does q_3 steps, …, Student n does q_n steps for a total of S' switches.

Suppose that S' < S.

There are three possible cases:

- Case 1: S' < S cannot be true because the greedy algorithm will pick the student with the most consecutive steps so long as they are not already assigned.
- Case 2: If S' > S, OPT is not optimal and the student with the maximum consecutive steps was not assigned all of the steps they could have helped out with. We reach a contradiction that there exists an OPT that is a more optimal solution than our greedy solution.

- Case 3: if S' = S, then S', then we could just replace S' with S in the optimal solution. If the number of switches is the same, then the greedy solution is no worse than the optimal solution, so it is optimal.

If we reach Case 2 scheduling students to steps, we know that the greedy solution is better than the OPT solution and we reach a contradiction.

2. Public, Public Transit

(a) Adapting Dijkstra's algorithm to this question, the algorithm solution to this problem is:
1. Initialize a processed[] array that marks all stations as not visited and all elements of times[] array to infinity.
2. Set time at starting station 0 in times[] array.
3. Pick the shortest distance station not visited yet and assign it to U. Mark station U as processed/visited.
Increase startTime by the time of the station processed/visited.
4. For all adjacent stations of U, update time value of adjacent station V with (wait time for the next train + the shortest time to U from source + length from station U to V) if it is less than time at station V.
5. Repeat steps 3 to 4 until there is no station left to be visited.

(b) The complexity of our proposed solution in (a) is $O(V^2)$.

(c) The method "shortestTime" is implementing Dijkstra's algorithm.

(d) We would modify the existing code to check whether the time at station V should be updated. The waiting time accounts for waiting for the next train. To implement waiting time, we calculated the time of the first train + (the number of trains * frequency of the train stopping at U) and if it is greater than or equal to the startTime, you will be able to leave the station. If (wait time for the next train + the shortest time to U from source + length from station U to V) is less than time at station V, you can replace time at station V.

(e) The current complexity of "shortestTime" given V vertices and E edges is $O(V^2)$. We can make the "shortestTime" implementation faster by performing the algorithm on a binary heap, yielding a complexity of O(E log V). If we perform the algorithm on a fibonnaci heap, the complexity can be O(E + V log V).

Source: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/