Sihang Li

(a) Optimal Substructure

(1.) If the current student signed up for the current step, shedule that student for current step. Go to the next step. Reduce the total number of steps left.

(2) Else if the current student didn't sign up for the current step. Find the student that has the most consecutive 1's. Set that student as the current student. Go back to the first student if the current step equals to the total step but the schedule has not been filled. Else go to the next student.

(3) Repeat (1) and (2) until the total number of steps left is zero.

(b) Greedy Algorithm:

Always choose the student that signed up for the current step. If there isn't one, choose the student that has the most number of consecutive 1's.

- while (n>0)

  if (signUpTable[currentStudent][currentStep]=1), schedule it

  else find the student with max consecutive, currentStudent = that student

(d) Time complexity = Total # of steps * Cost of each step

= numSteps = n * numSteps = n * numStudents = m

= O(n) * O(mn)

= $O(mn^2)$

(e) Proof by contradiction

Assume there exists an OPT solution using fewest switches.

ALG: $s_1, s_2, \ldots s_k$

OPT: $s_1', s_2', \ldots s_L'$ $L<k$

Let's say i is where ALG and OPT have different # of switches.

By design, ALG schedules the current student to current step if that student signed up for it or the student that has the most consecutive 1's.

We replace s_i' with s_i and it won't worsen the OPT. Modify OPT and ALG, all the s_i' are the same up to s_{i-1}, we can use same cut & paste logic... up to s_L and s_L'

OPT claims that s_i is the last switch our alg stops only when we scheduled all steps. so there must be students missing an OPT. Now we reached contradiction. No valid OPT that uses fewer switches. Therefore our ALG yields the OPT solution

Public Transit

(a) We can use Dijkstra's shortest path algorithm with a slight modification for finding the shortest path between starting station and target station.

(b) Time complexity will be $O(E+V) \times O(\log V)$

(c) It implements Dijkstra's shortest path algorithm

(d) Since the original Dijkstra's shortest path only handles one piece of data per edge, in order to get the travel time between start station and target station, we need an array to store the time / edge weights of the shortest path between these two vertices.