Name: Udoy Saha

ID : 2130 1095

Ans. to the prob. no.: 01

[a]

My starting location = P

My friend's starting location = q

Meeting location = V

We will use Dijkstra algorithm here to compute the time taken. The algorithm will work like the following:

1. Run Dijkstra with the source as P.
2. Run Dijkstra with the source as q.
3. Output the maximum between the ~~distance~~ time taken to reach V from ①, and the time taken to reach V from ②.

This algorithm will give us our desired result, which is the required time to reach V, from P and q.

Running time = $O(n^2)$ [Dijkstra]

[b]

For this task, we need to modify our previous algorithm in the following way:

1. Run Dijkstra with the source as P.
2. Run Dijkstra with the source as q.
3. Construct a new array, where,

   array [i] = max (time taken to reach i from ①, time taken to reach i from ②).

   If it is not possible to reach to i from ① or ②, the distance of array [i] = ∞.

4. For which i, the new array has the minimum time is the target vertex t.

Following this way, we can find out the targeted vertex t.

—o— x —o—

Ans. to the prob. no: 03

[a]

The following algorithm will help us to solve the problem.

1] Take the input in ~~two sep~~ a ~~no make~~ min-heap in the format (duration, task_no, deadline)

2] Apply heapsort, and every time take the task_no in a reserved array.

3] The reserved array contains the tasks in the desired order.

So, we simply sort the tasks in ascending order corresponding to their duration.

[b]

We have applied a greedy solution in our algorithm. The greedy approach

is to take the work which has less duration to be done. We use this strategy, because if we ~~use~~ do the work with less duration at first, the (duration − x) will most likely be positive. This strategy helps us to get to the optimized result by taking the work with less duration each time. If we do otherwise, the tasks with less duration would decrease the profit largely.

C

- Our step ① pushes n values in heap. So, it has $O(n \cdot \log n)$ runtime.
- Our step ② pops n values from heap. So, it has $O(n \cdot \log n)$ runtime.
- And, step ③ has $O(n)$ runtime to print.

So, overall runtime = $O(n \cdot \log n)$

Ans to the ques no:- 2

Our graph $G = (V, E)$ is an undirected weighted graph.

If, $l \in V$ and l is a leaf in the constructed MST, it should not connect more than 1 connected component. It might be connected with several vertices of a single connected component.

Now, our algorithm will be :-

1. From the adjacency list / matrix, remove all the edges connected with l, except the edge with minimum weight. (This will ensure there is only 1 edge with the vertex l, and will make it a leaf node).
2. Run Prim's / Kruskal's algorithm on the updated graph and get MST.

The running time of this algorithm:

- Our first step would take $O(k)$ time, where k is the number of edges the special vertex l has.
- Our second step would take $O(|V| \cdot |E|)$ time.

So, the overall runtime of our algorithm is $O(|V| \cdot |E|)$.