Task - 3

We know in general time complexity dijkstra is O(ElogV) for adjacency list.

In task-1 and task-2 we have implemented dijkstra algorithm. Here, N places is vertices and M roads is edges.

Each vertex will be connected with n-1 vertices. So we can say M represents N-1 edges connected to each vertex. When mean heap is used, finding, poping and updating in it becomes O(log N) + O(1) or O(logN). So we can say when all the vertices of vertex is updated it takes O(M log N) time.

So, total time stands O (NM log N)

But here N is no vertices and M is maximum number of edges attached to single nodes. Here $O(NM \log N)$ or $O(M \log N)$ both are correct, but if we consider tighter bound, $O(NM) = M$, here $M \log N$ is a tighter estimation.

So finally the time complexity becomes $O(M \log N)$ for both task 1 and 2.

If the number of titans in each road is exactly 1, it means the weight are same for each roads. $O(N + M)$ alogrith mentioned in the quation is BFS. Using BFS this problem can be solved.

The algorithm is BFS, pseudocode is

```
visited = [ ]* no. of places
queue = [ ]
BFS (visited, graph, node, end Point)
 Do visited (append, node)
 Do queue (append, node)
 while queue not empty
  Do m -> pop
  Print m
  if m = endpoint
     break
  For each neigh. of m in graph
   if neigh not in visited
     visited. append (neigh)
     queue. append (neigh)
```

Now to find the shortest path, the above BFS pseudocode need to be modify a little bit. We use BFS after starting from source and stop it when we reach the destination.

Modification is that when we visit the node we need to store every previous node in an array name previous. So that to get the path untill we find the destination we will loop through the previous array. And the time complexity of BFS algorith will be O(N+M)

Input

Vertex | Edge

3 4

1 2

2 5

3 4

4 5

1 3 → destination

here, 1 = source

It is given that no. titan is exactly 1 in each road i.e weight is same so for every road weight is 1