Task 03

ID: 19101465

prob-1 (only Dijkstra part)

```
import queue
import math
def Dijkstra(Graph, source):
    dist
    dist = ['null', 0]
    visited = []
    for i in range(0, len(Graph)+1):
        visited.append(false)                    } O(N)
    Q = queue.PriorityQueue()  ⟶ O(1)
    for i in range(2, len(Graph)+1):
        dist.append(float('inf'))                } O(N)
    prev = []
    for i in range(0, len(Graph)+1):
        prev.append(~~float('inf')~~) ('null')   } O(N)
    j = 1
    for i in Graph
    Q.put([dist[j], i])                          } O N(log N)
    j = j+1
[while not Q.empty():
        u = Q.get()[1] → O(log N)
    if visited[int(u)]:
        continue
```

```
visited[int(u)] = True
for x in Graph[u]:  → O(M)
    v = x[0]
    alt = dist[int(u)] + int(u[1])
    If alt < dist[int(v)]:
        dist[int(v)] = alt
        prev['int(v)'] = u
        Q.put((dist['int(v)'], v))
                    ↳ O(log N)
return dist
```

O(N)

Time complexity = $O(N) + O(N \log N) + O(N \log N) + O(N \log N) + O(N) + O(M \log N)$

$= O(N) + O(N \log N) + O(M \log N)$

$= O(N \log N) + O(M \log N)$

$= O(N \log N + M \log N)$

Problem 2 (Only Dijkstra part)

```
'import queue
'import math
def Dijkstra(Graph, source):
    dist = ['null', 0]
    visited = []
    for i in range(0, len(Graph)+1):
        visited.append(False)                    } O(N)
    Q = queue.PriorityQueue()
    for i in range(2, len(Graph)+1):
        dist.append(float('inf'))                } O(N)
    prev = []
    for i in range(0, len(Graph)+1):             } O(N)
        prev.append('null')
    j = 1
    for i in Graph:                              → O(N)
        Q.put((dist[j], i))                      O(N log N)
        j = j + 1
```

```
while not Q.empty():  → O(N)
    u = Q.get()[1]  → O(log N)
    if visited[int(u)]:
        continue
    visited[int(u)] = true
    for x in Graph[u]:  → O(M)
        v = x[0]
        alt = dist[int(u)] + int(x[1])
        if alt < dist[int(v)]:
            dist[int(v)] = alt
            prev[int(v)] = u
            Q.put([dist[int(v)], v])
                → O(log N)
```

O(N log N) + O(N) + O(M log N)

```
m = []
j = len(Graph)
m.append(str(j))
while j != 1:
    y = prev[j]
    m.append(y)
    j = int(y)
return m
```

Time complexity = $O(N) + O(N \log N) + O(N) + O(M \log N)$

$= O(N \log N) + O(M \log N)$

$= O(N \log N + M \log N)$

If the number of titans in each road is exactly 1, there is an $O(N+M)$ algorithm to solve this problem. That is BFS (Breadth first search).

In the first line of input there will be the number of vertex and then the number in the second line will be the same weight for each edge. Then the rest number of lines will show the connections of the graph. So, the input will be for the graph in problem 1 =>

```
5
1
1 2 4
2 1 3 5
3 2 4 5
4 1 3
5 2 3
```