

Task-3

In task-1:

```
def Dijkstra(dict1, dict2, source):  
    wei = [float('inf')] * (len(dict2) + 1)  
    par = [None] * (len(dict2) + 1)  
    priorty = {}
```

```
    vis = [False] * (len(dict2) + 1)
```

```
    wei[source] = 0
```

```
    priorty[wei[source]] = source
```

```
    while priorty != {}:
```

```
        m = min(priorty.keys())
```

```
        mins = priorty.pop(m)
```

```
        if vis[mins]:
```

```
            continue
```

```
        vis[mins] = True
```

$O(V) +$

$O(\log V)$

$\rightarrow O(V)$

for nei in $dict[mini]$:

check = $wei[mini] + dict[mini, nei]$

if check < $wei[nei]$:

$wei[nei] = \text{check}$

$par[nei] = mini$

$priority_q[wei[nei]] = nei$

$\rightarrow O(\log V)$

\therefore Time complexity of this algorithm
is $O(M \log N)$ and where for the adjacency
list it is $O(N + E)$

\therefore Total time complexity = $O(M \log N)$
 $+ O(N + E)$

In task-2

```
def Dijkstra (dict1, dict2, source):  
    wei = [float('inf')] * (len(dict1) + 1)  
    par = [None] * (len(dict1) + 1)  
    pri_que = {}  
    visit = [False] * (len(dict1) + 1)  
  
    wei[source] = 0  
    pri_que[wei[source]] = source  
  
    while pri_que != {}:  
        m = min(pri_que.keys())  
        visit[m] = pri_que.pop(m)  
        if visit[m] != True:  
            continue  
        visit[m] = True
```

$O(V) +$
 $O(\log(V))$
 $= O(V)$

for val in dict2[word]:

check = wei[word] + dict1[word, val]

if check < wei[val]:

wei[val] = check

par[val] = word

parent = par[par[val]] = val

path = []

x = len(par) - 1

path.append(x)

while par[x] != None:

path.append(par[x])

x = par[x]

path.reverse()

$O(E \log V)$

$O(V)$

So, here for adjacency list it is $O(M+N)$
and for the algorithm it is $O(M \log N)$

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

If the number of litans in each road is exactly 1, then the algorithm we can use is Breadth First Search which is BFS.