Algorithm – 11

——Single-Source Shortest Paths

A. Problem Description

In a shortest-paths problem, we are givin a weighted, directed graph G = (V, E), with weight function w: E -> R mapping edges to real-valued weights. The weight w(p) of path p={v0, v1, …, vn} is the sum of the weights of its constituent edges:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

We define the shortest-path weight D(u, v) from u to v by

$$D(u,v) = \begin{cases} \min\{w(p) : u \to p\} \\ \infty \end{cases}$$

A shortest path from vertex u to vertex v is then defined as any path p with weight w(p) = D(u, v).

B. Description of the algorithm

DIJKSTRA(G, w, s)

INITIALIZE-SINGLE-SOURCE(G, s)

S = {NULL}   // make the set S empty

Q = G.V   // put all vertexs into G

```
while Q != NULL

    // extract the vertex who has the smallest d-value

    u = EXTRACT-MIN(Q)

    S = S + {u}

    // if there exists one vertex that can be directly reached from u
    and the distance[s -> u -> x] is shorter than that stored in x.d,
    then change the x.d

    for each vertex v in G.Adj[u]

        RELEX(u, v, w)     // With 'DECREASE-KEY(Q) inside
```

C. Time complexity

Step "INITIALIZE-SINGLE-SOURCE(G, s)" ➔θ(V)

Step "Q = G.V" ➔θ(V)

Step "while" ➔θ(V)

Therefore, the total time T is

D. Code[Python]

```
#!/usr/bin/python
# Filename: Dijkstra.py

inf = float('inf')

class vertex:
  def __init__(self, start, end, value = inf , distance = inf, isTaken = 
False):
    self.start = start
```

```
    self.end = end
    self.value = value
    self.distance = distance
    self.isTaken = isTaken

def Dijkstra(S, G, n, source):
  G[source][source].distance = 0
  for i in range(0, n):
    for j in range(0, n):
      if G[source][j].isTaken == False:
        index = j
        break
    for j in range(0, n):
      if G[source][j].distance < G[source][index].distance and
G[source][j].isTaken == False:
        index = j
    G[source][index].isTaken = True   # delete vertex[index] from G
    S.append(G[source][index])        # insert vertex[index] into S
    for j in range(0, n):
      if G[source][j].isTaken == False and G[index][j].value < inf:
        availableDistance = G[source][index].distance + G[index]
[j].value
        originalDistance = G[source][j].distance
        G[source][j].distance = availableDistance if availableDistance
< originalDistance else originalDistance
```