Khrithik . S . Anand

1BM 18CS046

```
class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def print_solution(self, dist):
        print("Vertex \t Distance from Source")
        for node in range(self.V):
            print(node, "\t", dist[node])

    def min_distance(self, dist, spSet):
        min = 9999
        for v in range(self.V):
            if dist[v] < min and sptSet[v] == False:
                min = dist[v]
                min_index = v
        return min_index

    def add_edge(self, src, dest, weight):
        self.graph[src][dest] = self.graph[dest][src] = weight

    def dijkstra(self, src):
        dist = [9999] * self.V
        dist[src] = 0
        sptSet = [False] * self.V
        for cout in range(self.V):
            u = self.min_distance(dist, sptSet)
            sptSet[u] = True
            for v in range(self.V):
                if self.graph[u][v] > 0 and sptSet[v] == False and
                dist[v] > dist[u] + self.graph[u][v]:
                -> dist[v] = dist[u] + self.graph[u][v]
        self.print_solution (dist)

g = Graph (int (input ("Enter number of nodes in the topology:")))
c = int (input (" Enter number of edges:"))

for i in range (c)
    src, dest, cost = [int(_) for _ in input ("Enter [SRC][DEST]
                                    [WEIGHT]: ").split(' ')]
    g.add_edge (src, dest, cost)

src = int(input (" Enter [SRC] to find cost:"))

g.dijkstra (src)
```