```
'''
        Time complexity: O(E * log(V))
        Space complexity: O(V^2)

        where E is the number of edges in the graph and
        V is the number of vertices in the graph
'''
import sys
class Graph:

    def __init__(self,nVertices):
        self.nVertices = nVertices
        self.adjMatrix = [ [ 0 for i in range(nVertices)] for j in range(nVertices)]

    def addEdge(self,v1,v2,wt):

        self.adjMatrix[v1][v2] = wt
        self.adjMatrix[v2][v1] = wt

    def __bfsHelper(s,visited):
        q = queue.Queue()
        q.put(s)
        visited[s] = True
        while q.empty() is False:
            u = q.get()
            print(u)
            for v in range(self.nVertices):
                if self.adjMatrix[u][v] > 0 and visited[v] is False:
                    q.put(v)
                    visited[v] = True

    def bfs():
        visited = [False for i in range(self.nVertices)]
        for i in range(self.nVertices):
            if visited[i] is False:
                self.__bfsHelper(i,visited)

    def __getMinVertex(self,visited,weight):
        minVertex = -1
        for i in range(self.nVertices):
            if(visited[i] is False and (minVertex == -1 or (weight[minVertex] > weight[i]))):
                minVertex = i
        return minVertex
    def prims(self):
        visited = [False for i in range(self.nVertices)]
        parent = [-1 for i in range(self.nVertices)]
        weight = [sys.maxsize for i in range(self.nVertices)]
```

```
        for i in range(self.nVertices - 1):
            minVertex = self.__getMinVertex(visited,weight)
            visited[minVertex] = True
            for j in range(self.nVertices):
                if(self.adjMatrix[minVertex][j] >0 and visited[j] is False):
                    if(weight[j] > self.adjMatrix[minVertex][j]):
                        weight[j] = self.adjMatrix[minVertex][j]
                        parent[j] = minVertex
        for i in range(1,self.nVertices):
            if parent[i] > i:
                print(str(i) + " " + str(parent[i]) + " " + str(weight[i]))
            else:
                print(str(parent[i]) + " " + str(i) + " " + str(weight[i]))
    def __getMinVertexD(self,visited,weight):
        minVertex = -1
        for i in range(self.nVertices):
            if(visited[i] is False and (minVertex == -1 or (weight[minVertex] > weight[i]))):
                minVertex = i
        return minVertex

    def djikstra(self):

        visited = [False for i in range(self.nVertices)]
        dist = [sys.maxsize for i in range(self.nVertices)]
        dist[0] = 0
        for i in range(self.nVertices - 1):
            minVertex = self.__getMinVertexD(visited,dist)
            visited[minVertex] = True

            for j in range(self.nVertices):
                if (self.adjMatrix[minVertex][j] > 0 and visited[j] is False):
                    if(dist[j] > dist[minVertex] + self.adjMatrix[minVertex][j]):
                        dist[j] = dist[minVertex] + self.adjMatrix[minVertex][j]

        for i in range(self.nVertices):
            print(str(i) + " " + str(dist[i]))
    def removeEdge(self,v1,v2):
        if not self.containsEdge(v1,v2):
            return
        self.adjMatrix[v1][v2] = 0
        self.adjMatrix[v2][v2] = 0


    def containsEdge(self,v1,v2):
        return True if self.adjMatrix[v1][v2] > 0 else False

li = [int(ele) for ele in input().split()]
n = li[0]
E = li[1]
```

```
g = Graph(n)
for i in range(E):
    curr_edge = [int(ele) for ele in input().split()]
    g.addEdge(curr_edge[0],curr_edge[1],curr_edge[2])

g.djikstra()
```