Given a graph and a source vertex *src* in graph, find shortest paths from *src* to all vertices in the given graph. The graph may contain negative weight edges.

Dijkstra's algorithm is a Greedy algorithm and time complexity is $O(|V|log|V|)$ with the use of Fibonacci heap. Dijkstra doesnot work for graphs with negative weight edges. Bellman-Ford works for such graphs. Bellman-Ford is also simpler than Dijkstra and suites well for distributed system. Time complexity of Bellman-Ford is $O(VE)$.

procedure Bellman-Ford:

Input: Graph G and a source vertex *src*

Output: Shortest distance to all vertices from *src*. If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is reported.

1). Initialize distances from source to all vertices as infinite and distance to source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.

2). Calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph:

1. Do following for each edge (u,v):

   if dist[v] > dist[u] + weight of edge uv, then update dist[v]:

   dist[v] = dist[u] + weight of edge uv

3). This step reports if there is a negative weight cycle in graph. Do following for each edge (u,v):

1. if dist[v] > dist[u] + weight(u, v), then "Graph contains negative weight cycle"

```
# python program for Bellman-Ford's single source
# shortest path algorithm

from collections import defaultdict

# class to represent a graph
class Graph:
    def __init__(self, vertices):
        self.V = vertices # number of vertices
        self.graph = [] # default dictionary to store
graph

    # function to add an edge to graph
    def addEdge(self, u, v, w):
```

```
        self.graph.append([u, v, w])

    # utility function used to print the solution
    def printArr(self, dist):
        print("Vertex Distance from Source")
        for i in range(self.V):
            print("%d \t\t %d" % (i, dist[i]))

    # the main function
    def BellmanFord(self, src):
        # step 1: initialize distances from src to all
other         # vertices as Infinite
        dist = [float("Inf")] * self.V
        dist[src] = 0

        # step 2, relax all edges |V|-1 times. A simple
# shortest path from src to any other vertex can
        # have at most |V|-1 edges
        for i in range(self.V-1):
            # update dist value and parent index of the
adjacent
            # vertices of the picked vertix
            # consider those vertices which are sill in
queue
            for u, v, w in self.graph:
                if dist[u] != float("Inf") and dist[u] +
w < dist[v]:
                    dist[v] = dist[u] + w
        # check for negative-weight cycles.
        for u, v, w in self.graph:
            if dist[u] != float("Inf") and dist[u] + w <
dist[v]:
                print("Graph contains negative weight
cycle")
                return
        self.printArr(dist)
```