Lab 4

Name: Toka Alaa Elgindy **ID: 14**

Name: Nada Salama Mohammed **ID: 55**

Introduction:

In this assignment, you're required to implement two shortest paths algorithms which are Dijkstra and Bellman-Ford.

Input Graph Structure:

Input file will contain several lines that describe a directed graph structure as follows. First line contains two integers V and E which determine number of vertices and edges respectively. This line is followed by E lines describing the edges in the graph. Each of the E lines contain 3 numbers: i, j, w separated by a single space, meaning that there is a weighted edge from vertex i to vertex j ($0 \le i, j \le V - 1$), and the weight of the edge is w, where w may be negative or positive.

Data structure:

1) Vertex class which contains index, distance from source and adjacent list.
2) IEdge interface to access source vertex, destination vertex and the weight.
3) The graph is represented with array of vertex.
4) Array list to return ordered processed vertices.
5) Priority queue.

Assumption:

- Dijkstra algorithm requires all edge weights to be nonnegative.
  This is too restrictive, since it suffices to outlaw negative weight cycles.
- Bellman-Ford algorithm can handle negative edge weights.
  It even can detect negative weight cycles if they exist.

Algorithm used:

1- Bellman-Ford algorithm

```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

2- Dijkstra algorithm

- input: *G* = (V,E,w) and source node s

// initialization

- d[s] := 0
- d[v] := infinity for all other nodes v
- initialize priority queue Q to contain all nodes using d values as keys

- while Q is not empty do
  - u := extract-min(Q)
  - for each neighbor v of u do
    - if d[u] + w(u,v) < d[v] then // relax
      - d[v] := d[u] + w(u,v)
      - decrease-key(Q,v,d[v])
      - parent(v) := u