Mini Project 2: Finding Shortest Path using Dijkstra's Algorithm. **(Total Marks: 25)**

Task:

In this project you are required to implement the Dijkstra's Algorithm which is a Breadth First Search (BFS) algorithm for finding shortest path from a starting vertex (***src***) to every other vertex in the graph. You are provided with skeleton code that implements a directed weighted graph represented by an Adjacency Matrix. Your task is to write a function

```
'void find_shortest_paths( int * graph, int size, int src, int * dist_array)'
```

that will compute the shortest path from ***src*** to all the other vertices in the graph using Dijkstra's Algorithm.

Deadline: 29th December 2019

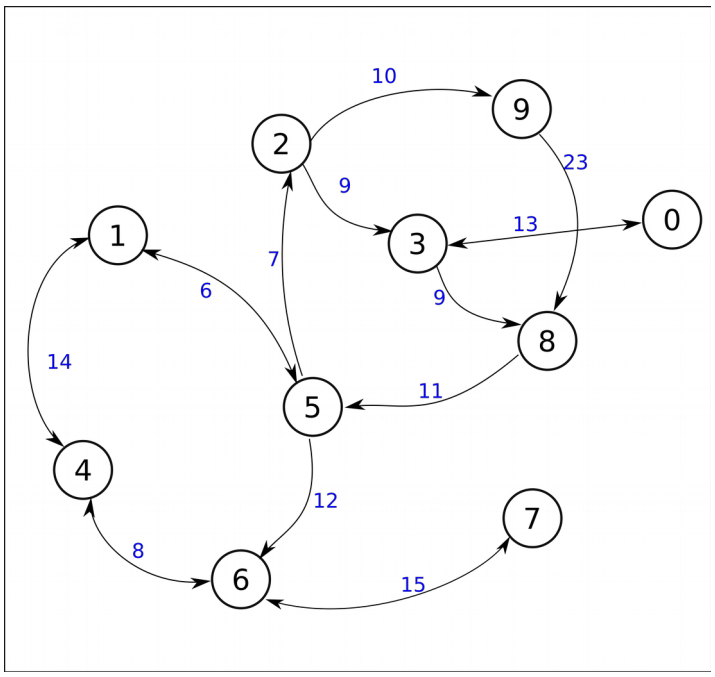

Figure 1: A Directed Weighted Graph

Dijkstra's Algorithm:

Dijkstra's algorithm finds a shortest path tree from a single source node, by building a set of nodes that have minimum distance from the source.

The graph has the following:

- Vertices, or nodes, denoted in the algorithm by ***v*** or ***u***;
- Weighted edges that connect two nodes: (***u,v***) denotes an edge, and ***w***(***u,v***) denotes its weight. In the figure below, the weight for each edge is written in gray.

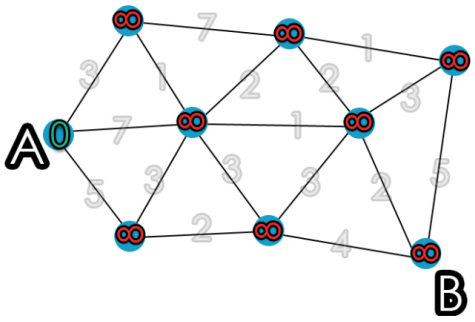

Figure 2: Starting point of Dijkstra's Algorithm

This is done by initializing three values:

dist, an array of distances from the source node ***s*** to each node in the graph, initialized the following way: ***dist***(***s***) = 0; and for all other nodes ***v***, ***dist***(***v***) = ∞. This is done at the beginning because as the algorithm proceeds, the ***dist(v)*** from the source to each node ***v*** in the graph will be recalculated and finalized when the shortest distance to ***v*** is found.

S, an empty set, to indicate which nodes the algorithm has visited. At the end of the algorithm's run, ***S*** will contain all the nodes of the graph.

Algorithm:

The algorithm proceeds as follows:

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the unvisited set ***S***.
2. Assign to every node a tentative distance value: set it to zero for our initial node ***u*** and to infinity (or -1) for all other nodes. Set the initial node as current.
3. For the current node ***u***, consider all of its unvisited neighbours and calculate their tentative distances (***dist***(***u***) + ***w***(***u,v***)) through the current node ***u***.
    - if ***dist***(***u***) + ***w***(***u,v***) < ***dist***(***v***), assign ***dist***(***v***) = ***dist***(***u***) + ***w***(***u,v***)
4. When we are done considering all of the unvisited neighbours of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
5. Select the unvisited node that has the smallest distance ***dist***(***v***), set it as the new current node ***u***, and go back to step 3.

The algorithm has visited all nodes in the graph and found the smallest distance to each node. ***dist*** now contains the shortest path tree from source ***s***.

Note: The weight of an edge ***w***(***u,v***) is taken from the value associated with (***u,v***) on the Adjacency Matrix.

Restrictions:

- You will implement the Dijkstra's Algorithm using C language.
- This is **not** a group activity and each student **must** work independently.
- You are allowed to consult books or Internet resources but should mention the resources in your reports.
- You are not allowed to use classes and objects or other purely object oriented programming constructs.

Deliverables:

1. Working code uploaded to the portal.
2. Report describing your implementation, the problems you faced, and the limitations of your code.

******** End of document ********