CSCI3200 Spring 2019 LAB4 – Adjacency Matrix based Graph

Objective: The objective of this lab is to help the students have a better understanding of how can we represent and operate on a graph whose implementation is based on an adjacency matrix.

In this LAB, you can find two files named AdjacencyMatrixGraph.java and testClass.java. You are required to provide the implementation of the methods that listed in AdjacencyMatrixGraph.java. AdjacencyMatrixGraph.java provides methods for implementation of a graph based on adjacency matrix. You can use testClass.java as test sample to test your adjacency matrix based graph. The edge class defines an edge that can be used in methods requiring the edge object.

Here are the list of required methods:

1) **public AdjacencyMatrixGraph(int[][] adjMatrix)** : The constructor. Will initialize the adjacencyMatrix with passed parameter adjMatrix. You need to copy all the contents from adjMatrix to adjacencyMatrix instead of simply change the references.
2) **public boolean isDirected( )**:check if the graph is a directed graph or not. Return true if the answer is yes, false, otherwise.
3) **public int numVertices()** : return the number of vertices in a graph.
4) **public int numEdges()**: return the number of edges in a graph, make sure to distinguish the count for directed or undirected graph.
5) **public boolean isComplete():** return true if the graph is a complete graph.
6) **public int outDegree(int v)**: check the outDegree of a vertex whose index is v.
7) **public int inDegree(int v)**: check the inDegree of a vertex whose index is v. For undirected graph, the inDegree of vertex v is equal to its outDegree.
8) **public ArrayList<Integer> neighbors(int v)**: get the list of neighbors for the vertex whose index is v. The list of neighbors will be returned as an instance of ArrayList.
9) **public ArrayList<Edge> edgeSet( ):** get the list of edges in the graph.
10) **public boolean isNeighbors(int u, int v)**: //check if vertex u and v are neighbor.
11) **public boolean insertEdge(int u, int v, double weight)**: insert an edge between two vertices, return false if there is an edge between them already.
12) **public boolean removeEdge(int u, int v)**: remove an edge between two vertices, return false if there is no edge between them
13) Graph traversal: choose **ONE** of the following algorithms to implement

a. **public ArrayList<Edge> DFS(int start)** : get the traversal sequence of the graph by using DFS from start index vertex and keep the **edge sequence** in an ArrayList

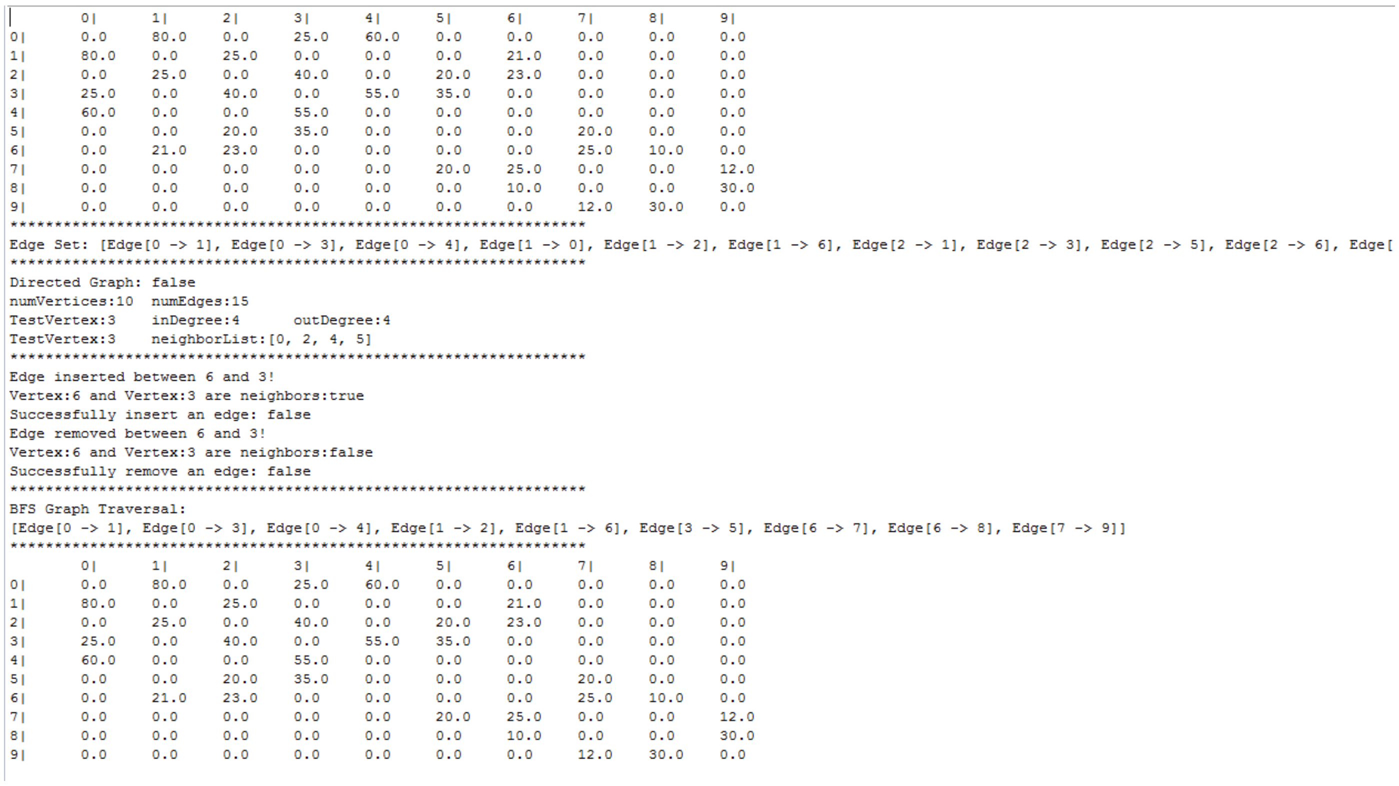
b. **public ArrayList<Edge>BFS(int start)**:get the traversal sequence of the graph by using BFS from start index vertex and keep the **edge sequence** in an ArrayList

c. **public ArrayList<Edge> MST(int start)**: Obtain the minimum spanning tree of an **undirected graph** and keep the **edge sequence** in an ArrayList

14) **public void printAdjMatrix()**:print out the adjacency matrix if the number of vertices in the graph is less than 20

Remark: For several methods required, you may have to handle directed and undirected situation differently.
For the test case provided, isComplete() method should return false.

If you use the testClass.java and chose BFS, below will be the test result:



```
|       0|      1|      2|      3|      4|      5|      6|      7|      8|      9|
0|      0.0     80.0    0.0     25.0    60.0    0.0     0.0     0.0     0.0     0.0
1|      80.0    0.0     25.0    0.0     0.0     0.0     21.0    0.0     0.0     0.0
2|      0.0     25.0    0.0     40.0    0.0     20.0    23.0    0.0     0.0     0.0
3|      25.0    0.0     40.0    0.0     55.0    35.0    0.0     0.0     0.0     0.0
4|      60.0    0.0     0.0     55.0    0.0     0.0     0.0     0.0     0.0     0.0
5|      0.0     0.0     20.0    35.0    0.0     0.0     0.0     20.0    0.0     0.0
6|      0.0     21.0    23.0    0.0     0.0     0.0     0.0     25.0    10.0    0.0
7|      0.0     0.0     0.0     0.0     0.0     20.0    25.0    0.0     0.0     12.0
8|      0.0     0.0     0.0     0.0     0.0     0.0     10.0    0.0     0.0     30.0
9|      0.0     0.0     0.0     0.0     0.0     0.0     0.0     12.0    30.0    0.0
*****************************************************************
Edge Set: [Edge[0 -> 1], Edge[0 -> 3], Edge[0 -> 4], Edge[1 -> 0], Edge[1 -> 2], Edge[1 -> 6], Edge[2 -> 1], Edge[2 -> 3], Edge[2 -> 5], Edge[2 -> 6], Edge[
*****************************************************************
Directed Graph: false
numVertices:10  numEdges:15
TestVertex:3    inDegree:4      outDegree:4
TestVertex:3    neighborList:[0, 2, 4, 5]
*****************************************************************
Edge inserted between 6 and 3!
Vertex:6 and Vertex:3 are neighbors:true
Successfully insert an edge: false
Edge removed between 6 and 3!
Vertex:6 and Vertex:3 are neighbors:false
Successfully remove an edge: false
*****************************************************************
BFS Graph Traversal:
[Edge[0 -> 1], Edge[0 -> 3], Edge[0 -> 4], Edge[1 -> 2], Edge[1 -> 6], Edge[3 -> 5], Edge[6 -> 7], Edge[6 -> 8], Edge[7 -> 9]]
*****************************************************************
        0|      1|      2|      3|      4|      5|      6|      7|      8|      9|
0|      0.0     80.0    0.0     25.0    60.0    0.0     0.0     0.0     0.0     0.0
1|      80.0    0.0     25.0    0.0     0.0     0.0     21.0    0.0     0.0     0.0
2|      0.0     25.0    0.0     40.0    0.0     20.0    23.0    0.0     0.0     0.0
3|      25.0    0.0     40.0    0.0     55.0    35.0    0.0     0.0     0.0     0.0
4|      60.0    0.0     0.0     55.0    0.0     0.0     0.0     0.0     0.0     0.0
5|      0.0     0.0     20.0    35.0    0.0     0.0     0.0     20.0    0.0     0.0
6|      0.0     21.0    23.0    0.0     0.0     0.0     0.0     25.0    10.0    0.0
7|      0.0     0.0     0.0     0.0     0.0     20.0    25.0    0.0     0.0     12.0
8|      0.0     0.0     0.0     0.0     0.0     0.0     10.0    0.0     0.0     30.0
9|      0.0     0.0     0.0     0.0     0.0     0.0     0.0     12.0    30.0    0.0
```