CIS2168 006 Assignment 8
MatrixGraph

1. Objectives

This assignment will help you to:

- Learn how to program using matrix graph
- Understand how graph works
- Enhance your skills in programming using array and multiple classes

2. Description

All page numbers below are for our textbook.

You will complete the implementation of graphs as illustrated in Figure 10.14 in Page 553 in Chapter 10. You will write classes Edge, MatrixGraph, and some method in AbstractGraph. You will also write some methods in class DepthFirstSearch in Listing 10.4.

The graph vertices are represented using integers, starting with 0. The graph may or may not be directed. The graph may or may not be weighted.

You will implement the following classes and methods:

2.1 Class **`Edge`** listed in Table 10.1 on Page 549

Your implementation must meet these requirements:

- Implement Edge as an independent class stored in its own file.
- Must implement all data fields and methods listed in Table 10.1 as stated in the Attribute, Purpose, and Behavior columns
- For hashCode(), use the following hashing method. Generate the hash code of the edge by left-shifting the source vertex number of the edge 16 bits and then exclusive or with the destination vertex number of the edge. Refer to this site for the left-shift and exclusive-or operators in Java: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html
- For toString(), return a string representation of the edge in the format of "[(src, dst)]". For example, for the edge from 1 to 2, return "[(1, 2)]"

2.2 Method **`loadEdgeFromFile`** in class **`AbstractGraph`** in Listing 10.2 in Pages 554-555

As stated in the book, this method load the edges of a graph (ListGraph or MatrixGraph) from the data in an input file. The edges are loaded into the calling object of AbstractGraph. The parameter (Scanner scan) is already open and connected to the input file before the method is called. The file contains only edges with one edge per line. Each line contain 2 or 3 numbers, separated by one or more spaces. The first is the source. The second is the destination. There third, optional, is the weight of an edge.

Note: the input file does not contain the number of vertices and the number of edges. So you need to keep reading until there is no data to read.

2.3 Class **`MatrixGraph`** in Figure 10.14 on Page 553 and described in Page 558

Your implementation must meet these requirements:

- Must define the data field named **`edges`** as a 2-D array of **`double`**, as described in Page 558.

- Class **`MatrixGraph`** must be a subclass of AbstractGraph.
- Must include a constructor equivalent to the constructor of ListGraph listed in Table 10.3 on Page 556. The constructor takes the number of vertices and a boolean value about whether graph is directed or not, and sets the number of rows (vertices) in this array.
    - public MatrixGraph(int numV, boolean directed)
- Must define 4 public methods equivalent to those in ListGraph in Table 10.3. Specifically these 4 methods:
    - public void insert(Edge edge)
    - public boolean isEdge(int source, int dest)
    - public Edge getEdge(int source, int dest)
    - public Iterator<Edge> edgeIterator(int source)
- To help you to implement method edgeIterator(), a private class **`Iter`** is provided to you. It's in the file Iter.txt. You need to copy and paste it into your Class **`MatrixGraph`**. It's a private class. Use it as you did with the private Node class in LinkedList.

2.4 (Bonus) New method **`depthFirstSearchNR`** in class DepthFirstSearch

Add a new method depthFirstSearchNR to class DepthFirstSearch. It is the non-recursive implementation of the depth first search of a graph. It starts the depth first search from the vertex: current. You must use the following header.

- public void depthFirstSearch(int current)

Hint:
Use a stack to save the parent of the current vertex when you start to search one of its adjacent vertices. In the non-recursive implementation of breadth first search in Listing 10.3 on Pages 564-565, a queue is used. Here in depth first search, you use a stack.

2.5 Driver class **`MatrixGraphTest`**

Your class must implement these operations:

- Create two Edge objects
- Use the Edge objects to call methods toString(), hashCode(), equals and display the results
- Create a list graph
- Populate the list graph using the method loadEdgesFromFile()
- Print the adjacency list of the list graph (content of data field edges)
    - **Hint: you may want to add a method in ListGraph to complete this task.**
- Create a matrix graph and populate it with some data
- Print the adjacency matrix of the matrix graph
- Perform a depth first search of the matrix graph using the methods lectured in class and display the discovery Order and finish order
- If you implemented the bonus method, call it and display the discovery Order and finish order.

You must use a text-based menu to let user select different operations.

3. Implementation Requirements

- You MUST implement your classes as describe here and in the textbook in Chapter 10. You cannot use boolean array to represent edges in MatrxGraph.
- Your Edge class must contain all these components:

Data Field	Attribute
`private int dest`	The destination vertex for an edge.
`private int source`	The source vertex for an edge.
`private double weight`	The weight.
Constructor	**Purpose**
`public Edge(int source, int dest)`	Constructs an `Edge` from `source` to `dest`. Sets the `weight` to 1.0.
`public Edge(int source, int dest, double w)`	Constructs an `Edge` from `source` to `dest`. Sets the `weight` to `w`.
Method	**Behavior**
`public boolean equals(Object o)`	Compares two edges for equality. Edges are equal if their source and destination vertices are the same. The weight is not considered.
`public int getDest()`	Returns the destination vertex.
`public int getSource()`	Returns the source vertex.
`public double getWeight()`	Returns the weight.
`public int hashCode()`	Returns the hash code for an edge. The hash code depends only on the source and destination.
`public String toString()`	Returns a string representation of the edge.

- Your AbstractGraph class must contain these component. You must include all existing methods I gave. You must implement the method loadEdgesFromFile.

Data Field	Attribute
`private boolean directed`	**`true`** if this is a directed graph.
`private int numV`	The number of vertices.
Constructor	**Purpose**
`public AbstractGraph(int numV, boolean directed)`	Constructs an empty graph with the specified number of vertices and with the specified `directed` flag. If `directed` is **`true`**, this is a directed graph.
Method	**Behavior**
`public int getNumV()`	Gets the number of vertices.
`public boolean isDirected()`	Returns **`true`** if the graph is a directed graph.
`public void loadEdgesFromFile(Scanner scan)`	Loads edges from a data file.
`public static Graph createGraph (Scanner scan, boolean isDirected, String type)`	Factory method to create a graph and load the data from an input file.

- Your MatrixGraph class must contain these component: data field, constructor, and methods:
  - double[][] edges
  - public MatrixGraph(int numV, boolean directed)
  - public void insert(Edge edge)
  - public boolean isEdge(int source, int dest)

- public Edge getEdge(int source, int dest)
- public Iterator<Edge> edgeIterator(int source)

These components equivalent to those in ListGraph:

Data Field	Attribute
`private List<Edge>[] edges`	An array of `Lists` to contain the edges that originate with each vertex.
Constructor	**Purpose**
`public ListGraph(int numV, boolean directed)`	Constructs a graph with the specified number of vertices and directionality.
Method	**Behavior**
`public Iterator<Edge> edgeIterator(int source)`	Returns an iterator to the edges that originate from a given vertex.
`public Edge getEdge(int source, int dest)`	Gets the edge between two vertices.
`public void insert(Edge e)`	Inserts a new edge into the graph.
`public boolean isEdge(int source, int dest)`	Determines whether an edge exists from vertex `source` to `dest`.

- **You earn bonus points only if you get all required methods correctly implemented.**

4. Major Steps and more Hints

Major steps:

- Understand the related classes I gave you in previous lectures (Lec#12 – Lec#13). I also attached them to this assignment.
- Implement Edge class and test it by adding related code in MatrixGraphTest class
- Implement AbstractGraph and test it by adding related code in MatrixGraphTest class
- Implement MatrixGraph and test it by adding related code in MatrixGraphTest class

More Hints:

- You need to include these classes and interfaces in your Java project:
  - Graph, Edge, AbstractGraph, ListGraph, MatrixGraph, DepthFirstSearch, MatrixGraphTest

5. Submission Requirements & Grading

This assignment is **due by 11:50PM, Wednesday, December 2, 2015**.