Comp 482 Project 3 (35 pts)

Due: Dec 7 (12 noon for CANVAS submission; in class for hard copy submission)

1. **GENERAL DIRECTIONS:** This is an individual project. In this project you will create a Graph class to implement several Graph algorithms for directed weighted graphs. Neatness counts and so does indented code that is easy to read with helpful variable names. Your Graph class methods must match the specifications in the class specification below. In particular, all specified public methods must have the same signatures as specified. Otherwise the instructor test cases will not work. You may add any extra private methods and private fields as needed.

2. Create and test a general Graph class based on the specifications below. The Graph class should create a directed weighted graph with n vertices and m edges with weights. Weights may be negative. Use an *adjacency list* graph representation. There will be one Graph constructor which will read the graph data from a file. Create a printGraph() method to print out your graph.

   The Graph class should also implement the following directed graph algorithms from the text or class:
   - BFS-based algorithm for finding shortest paths (using edge count as path length)
   - Dijkstra's algorithm for finding shortest paths with positive weights
   - Bellman Ford algorithm for finding shortest paths with positive or negative weights, as long as there is no negative weight cycle.
   - printShortestPaths which prints the shortest paths using an SPPacket.

   - isStronglyConnected returns true if every vertex is reachable from any other vertex. That is, for each pair of vertices x and y , there is a path from x to y and an path from y to x. Otherwise isStronglyConnected returns false.

3. Notes on adjacency list representation. You must use the data structure ArrayList<EdgeNode>[] adjList; The EdgeNode class is specified at the end of the project description.

4. Notes Inputting the graph data from a file
   a. Read number of vertices and the edge pairs from an input file. The filename is passed in as a parameter to the constructor. The vertices will be numbered from 0 to n-1 where n is the number of vertices. The input file will be formatted as seen below (without the comments). Use Scanner with nextInt() to read the integers.

```
//number of vertices   Vertices are labeled 0 .. nVertices-1
4
// v  u  w   means there is an edge from vertex v to vertex u with weight w
0 1 2
2  0 3
```

```
1  2 -1
2  3 1
0  3  4
```

b. printGraph() prints the adjacency list, along with the number of vertices, the number of edges in the graph.. Label. The print out for the graph above should look like

```
Graph: nVertices = 4  nEdges = 5
Adjacency Lists
v= 0  [(0,1,2), (0,3,4)]
v= 1  [(1,2,-1)]
v= 2  [(2,0,3), (2,3,1)]
v= 3  []
```

5. Notes on *public SPPacket bfsShortestPaths(int start)*

   Find shortest paths (measured in number of edges, not edge weights) from start to all other vertices reachable from start. **Do this by modifying the breadth first search algorithm**. Find the distance and parent arrays. Return the SPPacket which is a data structure that holds the start vertex and the distance and parent arrays. The SPPacket class is specified at the end of the project description.

6. Notes on *public SPPacket dijkstraShortestPaths (int start)*

   Implement Dijkstra algorithm from text or class. Find the distance and parent arrays. Return the SPPacket.

7. Notes on *public SPPacket bellmanFordShortestPaths (int start)*

   Implement the Bellman Ford algorithm from text or class. Find the distance and parent arrays. Return null if a negative cycle is detected; otherwise return the SPPacket.

8. Notes on *public static void printShortestPaths(SPPacket spp)* This method prints shortest paths from source to all other reachable vertices using the SPPacket returned from one of our shortest path methods. Use path printing format from class that looks like this:

```
Shortest Paths from vertex 1 to vertex
0:  [1, 2, 0]  Path weight = 2
1:  [1]  Path weight = 0
2:  [1, 2]  Path weight = -1
3:  [1, 2, 3]  Path weight = 0
```

9. Test your Graph class thoroughly by creating your own text files with many different directed graphs. All output should be clearly labeled.

10. Hand in as hard copy (stapled in this order)
    a. Cover Sheet
    b. Report: Careful runtime analysis of your isStronglyConnected method. Express answer as a function of the |V| and |E|. Typewrite your analysis. Include your isStronglyConnected method in the report.

    c. Graph.java file containing the Graph class , EdgeNode class , SPPacket class and any other helper classes you created. It should also contain the main program used to run instructors's test cases.

    d. Labeled input and output from Instructors test case (posted Dec 1).
    e. Your name must be embedded in all electronic files (not just written by hand)

11. Submit your Graph.java class to Canvas by 12 noon on Dec 7. Your name must be embedded in this file.

Outline of Graph class

```
public class Graph
{

  //-----------------------------------------------------
   private ArrayList<EdgeNode>[]  adjList;
   private int nVertices;
   private int nEdges;
   private String fileName;

/******************  Constructor*********************/
   public Graph ( String inputFileName)
   { }

 /******************Print graph method**************/

   public void printGraph()
   {  }

  /******************* BFS Shortest paths  ******************/
   public SPPacket   bfsShortestPaths ( int start)
   {  }

 /********************Dijkstra's Shortest Path Algorithm*** */

   public SPPacket  dijkstraShortestPaths (int start )
   { }

/********************Bellman Ford Shortest Paths **************/
   public SPPacket bellmanFordShortestPaths(int start)
   {}

/************************Prints shortest paths************************/
public void printShortestPaths( SPPacket spp)
{}

/*****************isStronglyConnected*************************/
public boolean isStronglyConnected()
{}
}//end Graph class
```

```
//place the EdgeNode class and the SPPacket class inside the Graph.java file
/*****************************************/
class EdgeNode
{
  int vertex1;
  int vertex2;
  int weight;

  public EdgeNode ( int v1, int v2, int w)
  { vertex1 = v1; vertex2 = v2; weight = w; }

  public String toString()
  {  }
}
 /*********************************************/
class SPPacket
{
  int[] d;    //distance array
  int[] parent;   //parent path array
  int source; //source vertex

  public SPPacket( int start, int[] dist, int[] pp)
  {}

  public int[] getDistance()
  {}

  public int[] getParent()
  {}

  public int getSource()
  {}

  public String toString()
  { }

}
```