CSEN703

Analysis and Design of Algorithms

Dr. Wael Abouelsaadat

Assignments 2 & 3

A2 + A3 Weight: 10%

Release Date: November 13th, 2018

A2 Due Date: November 29th, 2018

A3 Due Date: Dec 11th, 2018

Teams: to be done in teams of 4

Deliverable: TA will announce on MET website submission mechanism.

In this assignment, you are going to create a graph library for undirected graphs for use by other programmers. You will create the library using Java. The library should have the following class and methods as specified below. You are free to choose either an adjacency list or an adjacency matrix as the internal representation of the graph. You can add methods and attributes to the following classes but you must not change the signature or the attributes described below. Failing to do so, will result in breaking the auto-marker test cases which leads to 30% deduction of your grade. Include in your submission a README.TXT with your team member names/IDs.

Assignment 2 **[worth 5%]**

The implementation of the following classes and methods are included in A2. Note: the following code using Hungarian notation (you do not have to use it but make sure to name your variables properly).

```
public class Vertex{

      protected StringBuffer  _strUniqueID, //a unique id identifying vertex
                              _strData;     //data associated with vertex
      protected int           _nX,_nY;      //Coordinates of vertex on some
                                            //map. Assume 0,0 is
                                            //bottom left.
```

```
        public StringBuffer getUniqueID( ){
                return _strUniqueID;
        }

        public StringBuffer getData( ){
                return _strData;
        }

        public int getX( ){
                return _nX;
        }


        public int getY( ){
                return _nY;
        }

}


public class Edge{
        protected StringBuffer  _strUniqueID, //a unique id identifying edge
                                _strData;     //data associated with this edge.
                                              //Data could be name of edge or
                                              // any meaningful property for
                                              // an edge.
        protected int           _nEdgeCost;   // cost of traversing this edge

        public StringBuffer getUniqueID( ){
                return _strUniqueID;
        }

        public StringBuffer getData( ){
                return _strData;
        }

        public int getCost( ){
                return _nEdgeCost;
        }

}

// the following class could be used as the building block of a path where a
// path consists of path segments and each path segment consist of a
// vertex and associated edge with it.
public class PathSegment {
        protected Vertex  _vertex;   // the vertex in this path segment
        protected Edge    _edge;     // the edge associated with this vertex

        public Vertex getVertex( ){
              return _vertex;
        }

        public Edge getEdge( ){
              return _edge;
        }
}
```

```
public interface Visitor{
       public abstract void visit( Vertex v );
       public abstract void visit( Edge e );
}


// the following Exception should be your resort whenever an error occurs.
public class GraphException extends Exception{
         public GraphException( String strMessage ){
              super( strMessage );
         }
}



public class Graph{

              // returns the name you have given to this graph library [1 pt]
              // choose whatever name you like!
              public String getLibraryName( ){
              }

              // returns the current version number                   [1 pt]
              // read the following if you are wondering what this is ☺
              // https://en.wikipedia.org/wiki/Software_versioning
              public String getLibraryVersion( ){
              }

              // the following method adds a vertex to the graph      [2 pts]
              public void insertVertex(String strUniqueID,
                                       String strData,
                                       int    nX,
                                       int    nY) throws GraphException


              // inserts an edge between 2 specified vertices         [2 pts]
              public void insertEdge(String strVertex1UniqueID,
                                     String strVertex2UniqueID,
                                     String strEdgeUniqueID,
                                     String strEdgeData,
                                     int    nEdgeCost) throws GraphException


              // removes vertex and its incident edges                [1 pt]
              public void removeVertex(String strVertexUniqueID) throws
                                                          GraphException

              // removes an edge from the graph                       [1 pt]
              public void removeEdge(String strEdgeUniqueID) throws
                                                          GraphException

              // returns a vector of edges incident to vertex whose
              // id is strVertexUniqueID                              [1 pt]
              public Vector<Edge> incidentEdges(String strVertexUniqueID)
                                                    throws GraphException
```

```
// returns all vertices in the graph                    [1 pt]
public Vector<Vertex> vertices()throws GraphException


// returns all edges in the graph                       [1 pt]
public Vector<Edge> edges() throws GraphException


// returns an array of the two end vertices of the
// passed edge                                          [1 pt]
public Vertex[] endVertices(String strEdgeUniqueID)
                                  throws GraphException

// returns the vertex opposite of another vertex        [1 pt]
public Vertex opposite(String strVertexUniqueID,
                       String strEdgeUniqueID) throws
                                                GraphException

// performs depth first search starting from passed vertex
// visitor is called on each vertex and edge visited.  [12 pts]
public void dfs(String  strStartVertexUniqueID,
                Visitor visitor) throws GraphException


// performs breadth first search starting from passed vertex
// visitor is called on each vertex and edge visited.  [17 pts]
public void bfs(String  strStartVertexUniqueID,
                Visitor visitor) throws GraphException


// returns a path between start vertex and end vertex
// if exists using dfs.                                 [18 pts]
public Vector<PathSegment> pathDFS(
                     String strStartVertexUniqueID,
                     String strEndVertexUniqueID)
                                        throws GraphException


// finds the closest pair of vertices using divide and conquer
// algorithm. Use X and Y attributes in each vertex.   [30 pts]
public Vertex[] closestPair() throws GraphException
```

Assignment 3 **[worth 5%]**

The implementation of the following methods in the Graph class are included in A3.

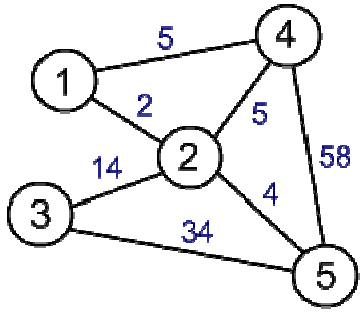
```
        // finds a minimum spanning tree using kruskal greedy algorithm
        // and returns the path to achieve that. Use Edge._nEdgeCost
        // attribute in finding the min span tree              [30 pts]
        public Vector<PathSegment> minSpanningTree()
                                        throws GraphException


        // finds shortest paths using bellman ford dynamic programming
        // algorithm and returns all such paths starting from given
        // vertex. Use Edge._nEdgeCost attribute in finding the
        // shortest path                                       [35 pts]
        public Vector<Vector<PathSegment>> findShortestPathBF(
                                    String strStartVertexUniqueID)
                                        throws GraphException



        // finds all shortest paths using Floyd-Warshall dynamic
        // programming algorithm and returns all such paths. Use
        // Edge._nEdgeCost attribute in finding the shortest path
        //                                                     [35 pts]
        public Vector<Vector<PathSegment>> findAllShortestPathsFW( )
                                            throws GraphException
}
```

Test Cases

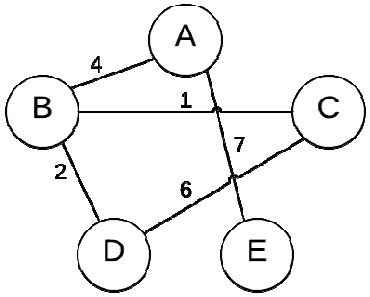
For your reference, the following graphs will be used as part of the test cases run against your code. As an example, to run test case 1 below we will have:

```
public class GradingVisitor implements Visitor{
       protected String _strResult = new String();

       public void visit( Vertex v ){
              _strResult += "v=" + v.getUniqueID( ) + " ";
       }

       public void visit( Edge e );
              _strResult += "e=" + v.getUniqueID( ) + " " ;
       }

       public String getResult( ){
              return _strResult;
       }
}
```

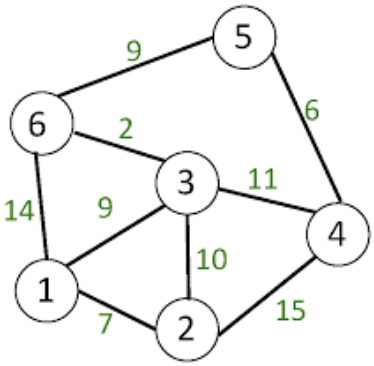
```
public class Grading{

        public int runTestCase1( ){
              int nMark=0;
              Graph g = new Graph( );
              GradingVisitor gVisitor = new GradingVisitor( );
              g.insertVertex("1", "1" );
              g.insertVertex("2", "2" );
              g.insertVertex("3", "3" );
              g.insertVertex("4", "4" );
              g.insertVertex("5", "5" );
              g.insertEdge("1","4","88","88",5);
              g.insertEdge("1","2","2","2", 2);
              g.insertEdge("2", "3","14","14",14);
              g.insertEdge("2", "4","99","99",5);
              g.insertEdge("2", "5","4","4",4);
              g.insertEdge("4", "5 ", "58", "58", 58);
              g.insertEdge("3", "5 ", "34", "34", 34);
              g.dfs("1",gVisitor );
              if( gVisitor.getResult().equalsIgnoreCase("blah"))
                  nMark+= 12;
      }

      public static void main( String[] args ){
              int nTotalMark=0;
              Grading grading = new Grading( );
              nTotalMark += Grading.runTestCase1( );

      }
}
```



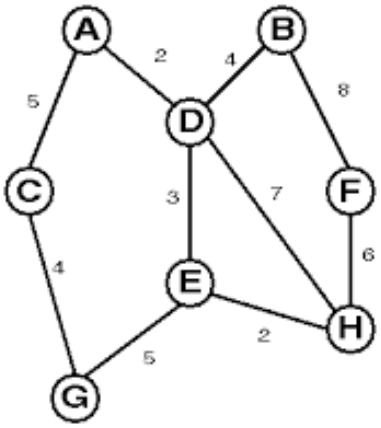
Test case 1



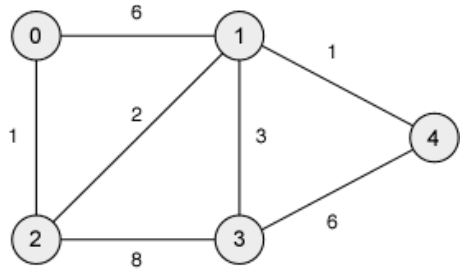
Test case 2



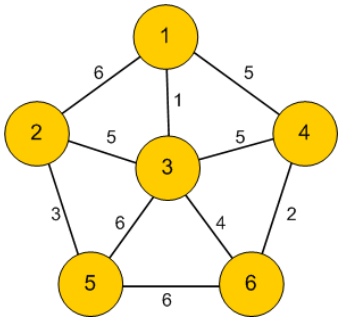
Test case 3



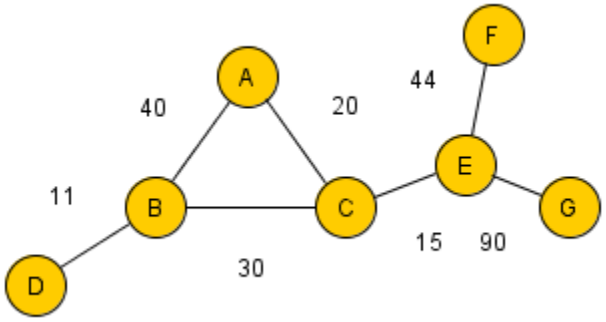
Test case 4



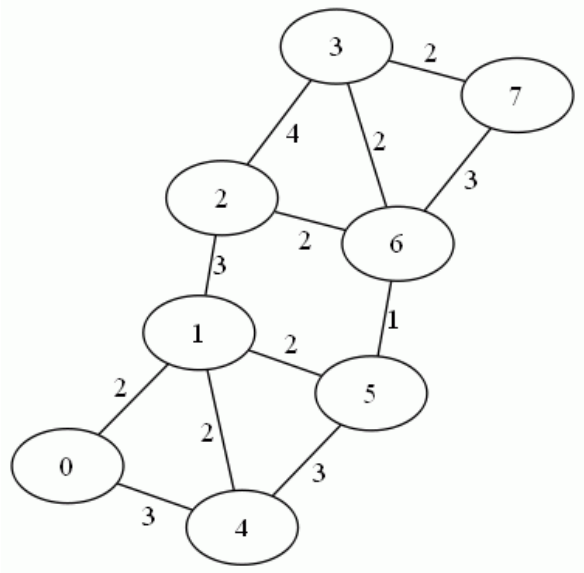
Test case 5



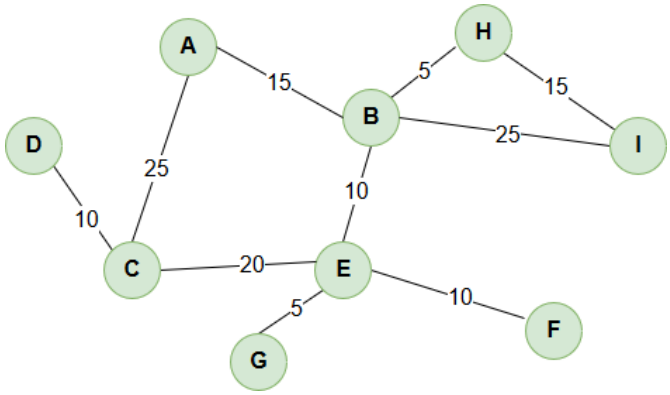
Test case 6



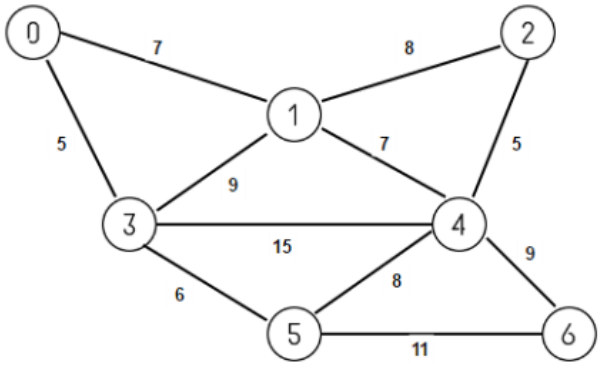
Test case 7



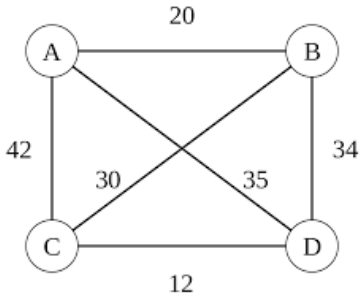
Test case 8



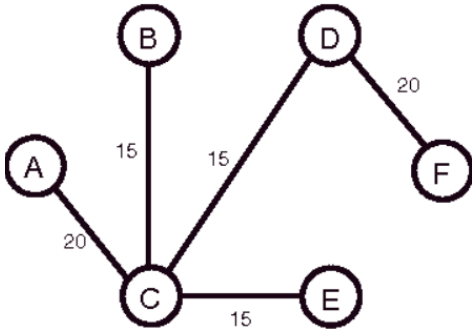
Test case 9



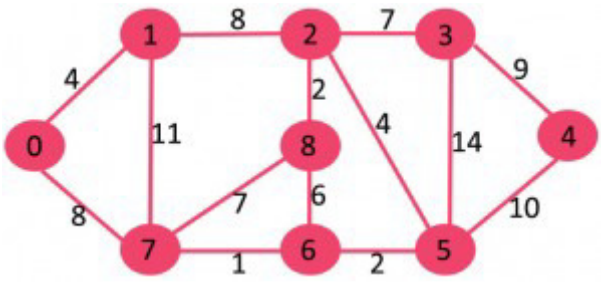
Test case 10



Test case 11



Test case 12



Test case 13