RYERSON UNIVERSITY

Faculty of Engineering, Architecture and Science

Department of Electrical and Computer Engineering
Program: Computer Engineering

| | |
|---|---|
| Course Number | **COE618** |
| Course Title | **Object Oriented Engineering Analysis and Design** |
| Semester/Year | **Winter 2013** |

| | |
|---|---|
| Instructor | **Olivia Das** |

| | |
|---|---|
| **Project Title** | **Find the shortest distance on user interface from free creating graph – implement by Dijkstra's Algorithm** |

| | |
|---|---|
| Submission Date | **April 12, 2013** |
| Due Date | **April 15, 2013** |

| | |
|---|---|
| Student Name | **Yu Sui** |
| Student ID | 12700 |
| Signature* | |

(Note: Remove the first 4 digits from your student ID)

**By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary*

*Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: www.ryerson.ca/senate/current/pol60.pdf.*

## Table of Contents

## Introduction

Our objective was to create a program for generating driving directions that uses a graph to represent a street map, and computes a shortest path to find directions from one point to another. We wanted to create a navigation system which allows us to create a map and from that map any user could choose a starting point and an ending point and the computer will find the shortest path which should be taken to get there.

Our motivation was that it can have many applications and many real life problems can be solved from graphs also graphs give more intuitive solutions to the users.   Ideally it would have been better if we could connect our software to network so that it could be accessed anywhere online just like Google maps however this was outside of the scope of this course.

### Problem description

We met many challenges for the development of the program and it was crucial to find solutions to overcome each of those challenges and fulfill the requirements. One of the main problems was:

How to computes the shortest distance between two points?
Assuming we have a graph already, do we need to try all other paths to make sure that none of them is shorter?

## Design

The solution was to apply: Dijkstra's Algorithm

1. This Algorithm allows us to find the shortest path from one node to another node in a given graph.

2. First of all, we were required to implement two building blocks: a class for node and a class for edge. For node class, we created a name for each node and made sure no duplicate node can be created.  Edge is a class containing two nodes and a given distance. Both of classes were implemented using ArrayList  as it has the ability of dynamic allocation

3. The Dijkstra's algorithm partitioned the nodes in two distinct ArrayList, the list of unsettled nodes and the list of settled nodes. Initially all nodes are unsettled and algorithm ends once all nodes are in the settled nodes.

4. The user can select any starting point on the created graph. After selection, the starting point will be added to the unsettled list and extract minimum distance of this list. However since the starting point is the only element in unsettled list right now, we remove starting point from the unsettled list and add it into to the settled list.

5. After this process, we will relax the neighbors of starting point and add them into unsettled list. Then we follow the previous procedure and extract a minimum distance from them. Then this node is selected and we try to find whether a shortest distance for this node exists. If it does

exist, we update the distance of this node and put this distance into the PATH list. This process is repeated until the unsettled list becomes empty.

## Requirements & Specifications

**Include any techniques/principles/patterns that you have used in your design.**
We used composite design patterns.

**Will the graph be *mutable* or immutable?**
Yes .The graph is mutable we made a Hash Map containing all of the nodes and paths, the label and distance for the path, including the node for the starting point and node for the ending point.

**Will it be possible to change the label of an edge?**
Yes we can, because we can modify the name and the distance of each edge.

**Will the graph allow edges without labels?**
No, each edge must be connected by two nodes. Each edge has a label.

**Will edge labels be strings or generic objects?**
Edge label is generic object, not string. Each edge object is formed by two nodes and weight whose type is double. For each node object, their label is a string.

**Will nodes be required to only satisfy the interface of java.lang.Object?**
No, nodes can either be inherited or composited with other classes

**Will you design a Java interface for nodes?**
No, we did not used java interface for the nodes

**Will the graph be implemented as a single class, or will there be a separate Java interface for the Graph specification, and a class for the implementation?**
The graph was implemented as a single class but it inherits the methods from the create graph class.

**Will edges be objects in their own right?**
No, the edge class is a child of the create graph class and will inherit the parameters of the latter

**Will it be possible to find the successor of a node from the node alone, or will the graph be needed too?**
Yes, after a node is removed from unsettled list and added to a settled list, its neighbors will in turn be added to the unsettled list while the shortest distance is calculated using the Dijkstra's algorithm. Once the node with the shortest distance is found, it is added to the PATH list and the distance is updated.

**Can a node belong to multiple graphs?**
Yes. A node can be connected to 2 or more nodes.

**Should path-finding operations be included as methods of the graph, or should they be implemented in client code on top of the graph?**
Path-finding operations were implemented in the client code on top of the graph.

## Modules description and explain how these modules are decomposed.

Graph class is formed by node and edge classes. Algorithm class use the given graph to find the shortest distance of start point on this graph. Manager has authority to add or delete graph. User can login in to the system created by manger and find the shortest distance by any given points

**Your project must include at least 8 classes. Write specifications for all the methods of any two classes.**
We totally have 11 classes in this project. Take edge and node class as examples

**Node:**

```
public class Node {
   //Overview: Node is immutable with a String type instance;

   private String Node_name;

   @Override
   public String toString() {
      // Effect: Object method, return Node name;
      return Node_name;
   }

   @Override
   public boolean equals(Object obj) {
      // Effect: make sure whether add node with same name, if exists same name, return false;
      // Otherwise, return true
      if (obj == null) {
         return false;
      }
      if (getClass() != obj.getClass()) {
         return false;
      }
      final Node other = (Node) obj;
      if ((this.Node_name == null) ? (other.Node_name != null) : !
this.Node_name.equals(other.Node_name)) {
         return false;
      }
      return true;
   }

   @Override
   public int hashCode() {
      // Effect: return hashcode of Node
      int hash = 5;
      return hash;
   }
```

```
    public void setNode_name(String Node_name) {
        //Effect: Modify the node name
        this.Node_name = Node_name;
    }



    public String getNode_name() {
        //Effect: return node name;
        return Node_name;
    }

    public Node( String Node_name) {
        // Effect: Initializes this
        this.Node_name = Node_name;
    }

}
```

**Edge:**

```
public class Edge {
    // Overview: Edge is immuable class which has four instances
    // String for edge name, Two nodes and double distance
    private String Edge_name;
    private Node from_node;
    private Node to_node;
    private double distance;

    @Override
    public boolean equals(Object obj) {
        //Effect: check wheter exists the same edge. if exists, return false. otherwise, return true
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Edge other = (Edge) obj;
        if (this.from_node != other.from_node && (this.from_node == null || !
this.from_node.equals(other.from_node))) {
            return false;
        }
        if (this.to_node != other.to_node && (this.to_node == null || !
this.to_node.equals(other.to_node))) {
            return false;
        }
        return true;
    }



    @Override
    public int hashCode() {
        // Effect: return the hashcode
        int hash = 3;
        return hash;
    }

    @Override
    public String toString() {
        // Effect: object method which return a string containning the edge name, beginning node,
destination node and weight
        return "Edge{" + "Edge_name= " + Edge_name + " ,from_node= " + from_node +
" ,to_node= " + to_node + " ,distance= " + distance +" "+'}';
    }



    public void setEdge_name(String Edge_name) {
        //Effect: modify the name of edge
        this.Edge_name = Edge_name;
    }

    public void setDistance(double distance) {
        //Effect: modify the weight
        this.distance = distance;
    }

    public void setFrom_node(Node from_node) {
        //Effect: modify the beignning node
        this.from_node = from_node;
    }

    public void setTo_node(Node to_node) {
        // Effect: modify the destination node
        this.to_node = to_node;
    }

    public String getEdge_name() {
        // Effect: return the name of edge
        return Edge_name;
    }

    public double getDistance() {
        //Effect: return the weight of two nodes
        return distance;
    }

    public Node getFrom_node() {
        // Effect: return the beginning node
        return from_node;
    }

    public Node getTo_node() {
        //Effect: return the destination node
        return to_node;
    }

    public Edge(String Edge_name, Node from_node, Node to_node, double distance) {
        //Effect: Initializes this
        this.Edge_name = Edge_name;
        this.from_node = from_node;
        this.to_node = to_node;
        this.distance = distance;
    }

}
```

## Design considerations

There are many aspects to consider in the design of a piece of software. The importance of each should reflect the goals the software is trying to achieve. Some of these aspects are:

- **Compatibility / Platform-independence**

The software being built in java and has the ability to operate with other operating systems like Linux, Windows, Gnome, IOS.

- **Extensibility / Reusability**

In java new classes can be added to the software with slight or no modification to the underlying software architecture. This is a great advantage especially for those kinds of web based programs required frequent updates and it won’t be easy having to start everything from scratch if we want to include new features.

- **Fault-tolerance/Robustness**

Our software was able to operate under stress and was able to recover from invalid input of the user. The implementation of the file in file out (FIFO) gives us the advantage of not having of speed and portability of saving the data in a file which can be read from most operating systems

- **Object-oriented / Modularity**

Java encourages the creation of modular programs and so the resulting software comprises of well defined, independent components that leads to better maintainability. The components could be then implemented and tested in isolation before being integrated to form a desired software system. This allows division of work in a software development project.

- **Usability**

We decided to use Jframes from java as the user interface which requires the use to click on each box and input the data but we realized that direct input from the keyboard into the terminal might have been more efficient and suitable for this purpose.

Also we had to design a way to display the graph we created in java, we decided to use java Applet. Using function like paint() and repaint(), we were able to reproduce the graph. I showed the edges and the nodes and the distance between them. Our main idea was to implement a graphic user interface. Then the user will be able to use the map and from the input of his mouse, he will be able to choose the starting point and the ending point and the computer will find the shortest distance in between them. However we neither had the knowledge nor the time to be able to implement this.

## Testing

What testing technique have you used? Explain how did you test your classes.
We created a graph on paper and we calculated the shortest path ourselves from the methods we were taught in COE 428. Then we compared the result to the exact same graph which we now incorporated in our software's database. As the results were identical, we concluded the algorithm was working.

Test cases:

I. Store the same name for nodes (node_to and node_from)
It works! We were successful in testing if we can create two nodes with the same name and see if they connect to itself.

II. Use same name for edges
No, we won't allow to create two edges with a same name.

III. Check if we can create without a label?
No, when we create a node or an edge, we must have label for each of them.

IV. Check if we can create a new road with distance zero?
No we can't. Our code does not allow the creation of a new path with distance zero.

V. Multiple path for same starting and ending point
No, we won't create multiple path for same starting and ending node. For example, if we create A to B with one path, we can't create an edge from B to A.

VI. No edges without nodes
No, we could not; the way the code was implemented does not allow us to create edges without nodes.

## Conclusion:

We discovered that usually best path problems are computationally harder than any path problems and that software used for navigation with GPS requires a lot more software in order to work. We were required to apply the knowledge of previous courses together with our course in order to solve the problem. Overall we can say we were successful in meeting the required specification for the project.

Yet we understand that much more could be done to make the program more user-friendly and also we maybe if we had the time, we could have look for a way to read collect data from picture drawn from Microsoft paint and use that picture to create map and find shortest distance between two points

## Appendix 1: UML diagrams

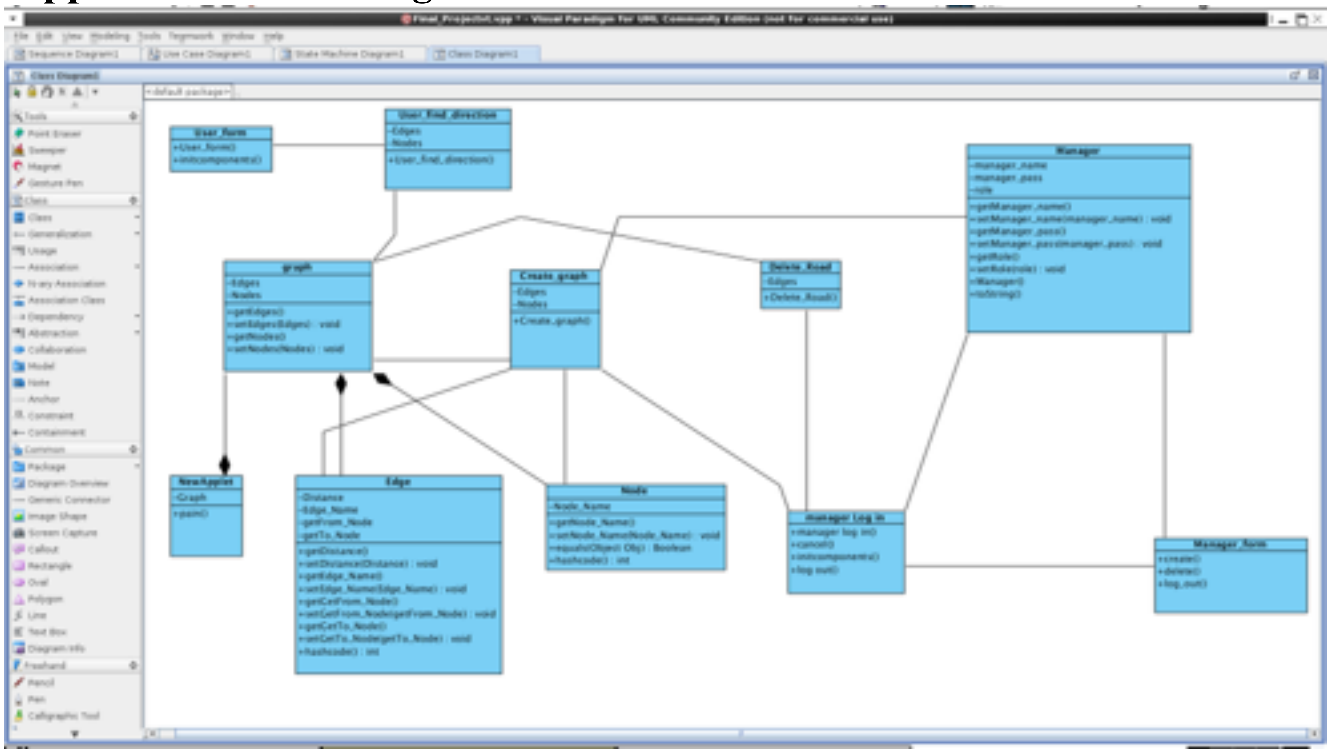

*Figure 1 : Uml Diagram for Classes*

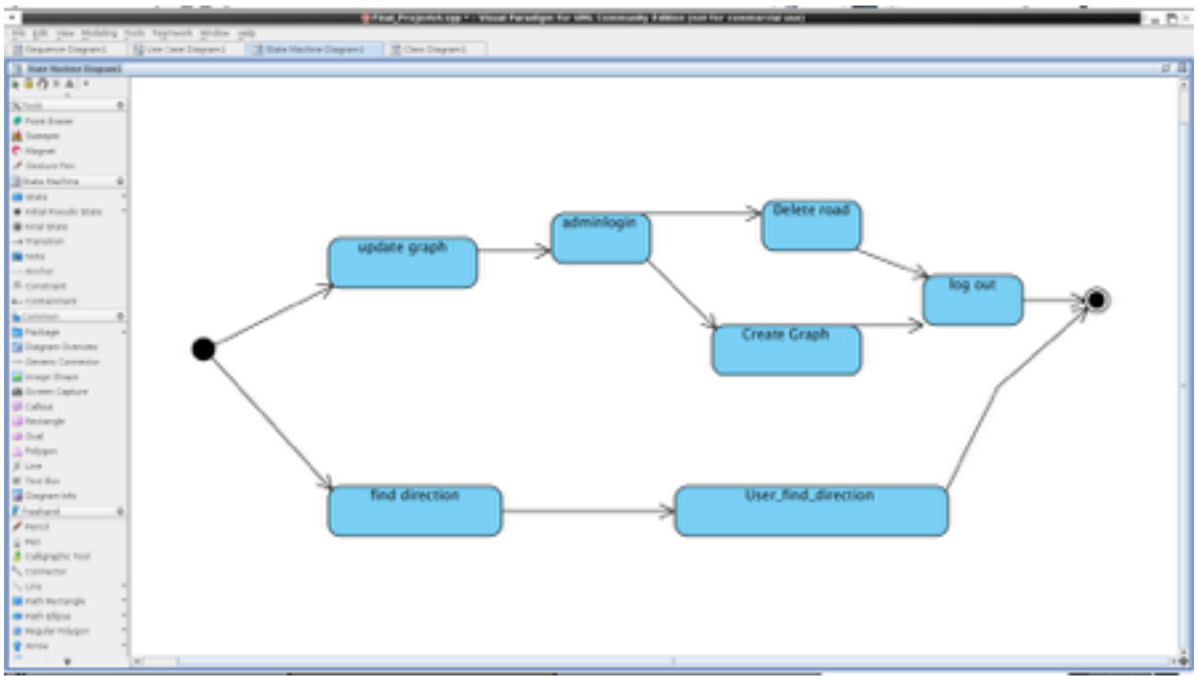

*Figure 2 : Uml State diagram*

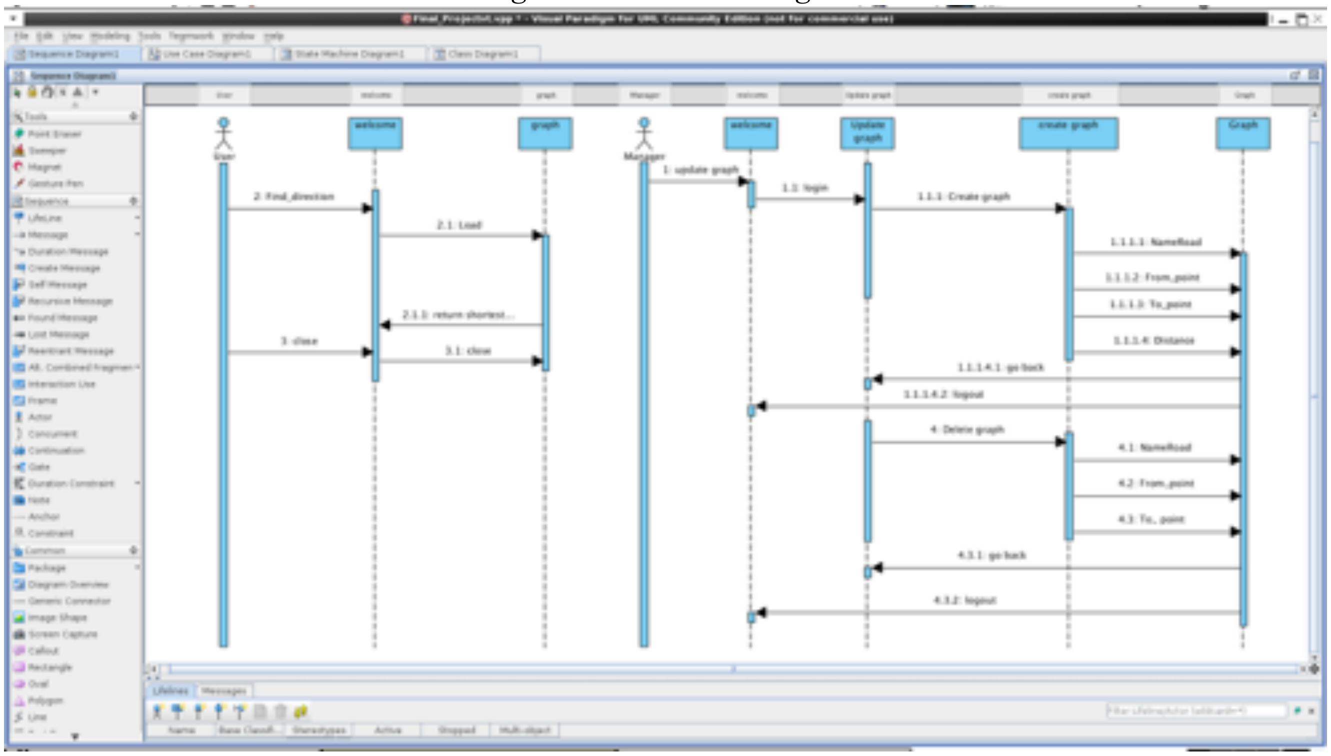

*Figure 3 : Uml Sequence diagrams*

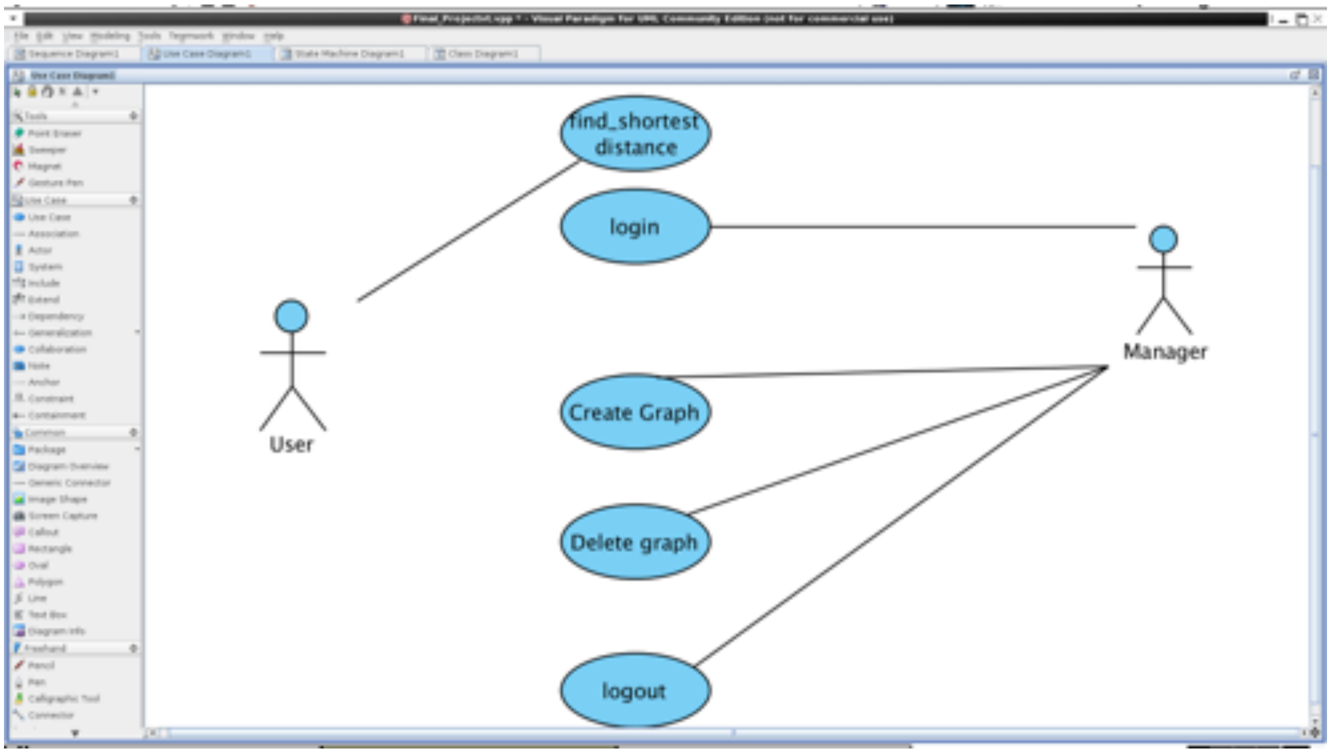

*Figure 4 : Uml diagrams for use case*

## Appendix 2: Javadoc for all the classes

Node and Edge classes were on above page

***Algorithm***

```
import java.util.*;
import java.io.*;
import javax.swing.JOptionPane;

public class Apply_ALgorithm  {
    private ArrayList<Node> Nodes;
    private ArrayList<Edge>Edges;
    private ArrayList<Node> pre_Nodes;
    private ArrayList<Node> post_Nodes;
    private Map<Node, Node> Node_before;
    private Map <Node,Double> distance;


// Constructor to initialize sets of Nodes and Edges
   public Apply_ALgorithm(Graph graph) {
       this.Nodes = new ArrayList<Node>(graph.getNodes());
       this.Edges = new ArrayList<Edge>(graph.getEdges());

   }
   // find the distance between two nodes
   public double getDistance(Node begin, Node end) {
   for (Edge edge : Edges) {
     if (edge.getFrom_node().equals(begin)
         && edge.getTo_node().equals(end)||edge.getFrom_node().equals(end)
         && edge.getTo_node().equals(begin) ){
         //System.out.println(edge.getDistance());
       return edge.getDistance();
     }
   }

   System.err.println("cannot find the distance");
   return -1;
   }
   // to define a node is processed or not
   public boolean is_process(Node node) {
   return post_Nodes.contains(node);
 }
// Find a list of neighbor(adjacentNode) of a processing Node that have not processed yet

public  ArrayList<Node> getNeighbors (Node node) {
   ArrayList<Node> neighbors = new ArrayList<Node>();
   for (Edge edge : Edges) {
     if (edge.getFrom_node().equals(node)&& !is_process(edge.getTo_node())) {
       neighbors.add(edge.getTo_node());
     }
     if (edge.getTo_node().equals(node)&& !is_process(edge.getFrom_node())) {
       neighbors.add(edge.getFrom_node());


     }


   }

     for(Node n:neighbors){



       System.out.println("Node"+n.toString());}
   return neighbors;
 }
///////////////////////////////////////////////////////////////////////
// get Distance from a node to a node in a graph
```

```
public void find_Distances(Node node) {
   ArrayList<Node> adjacentNodes = getNeighbors(node);
   for (Node target : adjacentNodes) {
     if (getShortestDistance(target) > getShortestDistance(node)
         + getDistance(node, target)) {
       distance.put(target,getShortestDistance(node)
           +getDistance(node, target));
       Node_before.put(target,node);
       pre_Nodes.add(target);
     }
   }

  }
/////////////////////////////////////////////////////
// find shortest distance from root to node destination
public double getShortestDistance(Node destination) {
   Double  d = distance.get(destination);
   if (d == null) {
     return Double.MAX_VALUE;
   } else {
       //System.out.println(d);
     return d;
   }
  }
//-------------------------------------------------------------
// to find a minimal distance in the Array_List of Nodes
private Node minimum_distance( ArrayList<Node> Nodes) {
      Node minimum = null;
   for (Node node : Nodes) {
     if (minimum == null) {
       minimum = node;
     } else {
       if (getShortestDistance(node) < getShortestDistance(minimum)) {
         minimum = node;
       }
     }
   }
   return minimum;
  }
// perform the alogrithm
public void ex(Node source) {
   pre_Nodes = new ArrayList<Node>();
   post_Nodes = new ArrayList<Node>();

   distance = new HashMap<Node, Double>();
```

```
    Node_before = new HashMap <Node, Node>();
    distance.put(source,0.0);
    pre_Nodes.add(source);
    while (pre_Nodes.size() > 0) {
      Node node = minimum_distance(pre_Nodes);
      post_Nodes.add(node);
      pre_Nodes.remove(node);
      find_Distances(node);
    }
}
    ////////////////////////////////////////////////
    // find a path to the destination:
    public ArrayList<Node> findPath(Node target) {
    ArrayList<Node> path = new ArrayList<Node>();
    Node tmp = target;
    // Check if a path exists
    if (Node_before.get(tmp) == null) {
      System.out.println("cannot find a path \n");
      //return null;
    }
    path.add(tmp);
    while (Node_before.get(tmp) != null) {
      tmp = Node_before.get(tmp);
      path.add(tmp);
    }
    // Put it into the correct order
    Collections.reverse(path);
    System.out.println("Shortest path from source is \n");
     String out = new String();
    for(Node node:path){

        out = "Node"+node;
        System.out.println("Node  "+node.toString());
    }
    double total_distance = 0;
    String t="Shortest path from: "+ path.get(0) +" is \n";
    String r = " ";
    for(int i = 0; i <path.size()-1;i++){
        Edge tem1 = new Edge("tmp",path.get(i),path.get(i+1),0);
        Edge tem2 = new Edge("tmp",path.get(i+1),path.get(i),0);
         for(int n = 0; n <Edges.size();n++){
           if(Edges.get(n).equals(tem1)||Edges.get(n).equals(tem2)){
           r = Edges.get(n).getEdge_name();
           }
         }
    total_distance = total_distance + getDistance(path.get(i),path.get(i+1));
    t = t+" From Node "+path.get(i).toString()+ " to node " +path.get(i+1).toString() +
            " --via road-- "+ r+
            " --Distance : "+ getDistance(path.get(i),path.get(i+1))+ " km"
            + ""
            + "\n";
    /* System.out.println("From Node  "+path.get(i).toString()+ "  to node  " +path.get(i
+1).toString() +
            "   Distance : "+ getDistance(path.get(i),path.get(i+1))+ " km"
            + ""
            + "/n"); */

    }

    t = t+"total_distance is: "+ total_distance+" km \n";
    JOptionPane.showMessageDialog(null, t+"\n");
    return path;
  }
  ///////////////////////////////////////////////
    // to print a path from a departure point to the destination:
    public String findPath1(Node destination) {
     ArrayList<Node> path1 = new ArrayList<Node>();
    Node tmp = destination;
    // Check if a path exists
    if (Node_before.get(tmp) == null) {
     return  "cannot find a path \n";

    }
    path1.add(tmp);
    while (Node_before.get(tmp) != null) {
      tmp = Node_before.get(tmp);
      path1.add(tmp);
    }
    // Put it into the correct order
    Collections.reverse(path1);
    System.out.println("Shost test path from source is \n");
     String out = new String();
    for(Node node:path1){

        out = "Node"+node;
        System.out.println("Node"+node);
    }
    return out;
  }
}
```

}

***<u>Manager:</u>***

```
public class manager {
    private String m_name;
    private String m_pass;
    private String m_role;

    public manager(String m_name, String m_pass, String m_role) {
        this.m_name = m_name;
        this.m_pass = m_pass;
        this.m_role = m_role;
    }

    public void setM_name(String m_name) {
        this.m_name = m_name;
    }

    public void setM_pass(String m_pass) {
        this.m_pass = m_pass;
    }

    public void setM_role(String m_role) {
        this.m_role = m_role;
    }

    public String getM_name() {
        return m_name;
    }

    public String getM_pass() {
        return m_pass;
    }

    public String getM_role() {
        return m_role;
    }

   // to print out and save a manager profile
    @Override
    public String toString() {
        return "manager{" + "m_name= " + m_name + " ,m_pass= " + m_pass + " ,m_role="
+ m_role + '}';
    }
}
```

***The remaining classes are implemented by java applet.***

## *References :*

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5564518&tag=1