CS 542 LINK-STATE ROUTING

PREPARED BY:
ARUN SACHITHANANDAM -A 20325521
KARTHIK NARISETTI –A 20338139
SHREYAS SALIGRAMA CHANDRAKANTHA -A20330984
SECTION – 02

Arun, Karthik, Shreyas

Introduction:

Routing is the process of selecting best paths in a network. Routing is performed for many kinds of networks, including the telephone network (circuit switching), electronic data networks and transportation networks.

In packet switching networks, routing directs packet forwarding (the transit of logically addressed network packets from their source toward their ultimate destination) through intermediate nodes. Intermediate nodes are typically network hardware devices such as routers, bridges, gateways, firewalls, or switches. General-purpose computers can also forward packets and perform routing, though they are not specialized hardware and may suffer from limited performance. The routing process usually directs forwarding on the basis of routing tables which maintain a record of the routes to various network destinations. Thus, constructing routing tables, which are held in the router's memory, is very important for efficient routing. Most routing algorithms use only one network path at a time. Multipath routing techniques enable the use of multiple alternative paths.

Routing, in a more narrow sense of the term, is often contrasted with bridging in its assumption that network addresses are structured and that similar addresses imply proximity within the network. Structured addresses allow a single routing table entry to represent the route to a group of devices. In large networks, structured addressing (routing, in the narrow sense) outperforms unstructured addressing (bridging).

Linked State Routing Protocol

A link-state routing protocol is one of the two main classes of routing protocols used in packet switching networks for computer communications, the other is the distance-vector routing protocol. Examples of link-state routing protocols include open shortest path first (OSPF) and intermediate system to intermediate system (IS-IS).

The link-state protocol is performed by every switching node in the network (i.e., nodes that are prepared to forward packets; in the Internet, these are called routers). The basic concept of link-state routing is that every node constructs a map of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes. Each node then independently calculates the next best logical path from it to every possible destination in the network. The collection of best paths will then form the node's routing table.

This contrasts with distance-vector routing protocols, which work by having each node share its routing table with its neighbors. In a link-state protocol the only information passed between nodes is connectivity related.

Link-state algorithms are sometimes characterized informally as each router 'telling the world about its neighbors. Instead of having each node share its routing table with its neighbors, which DVRT does, link-state routing protocol the only information passed between nodes is connectivity related. The basic concept of link-state routing is that every node constructs a map of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes. Each node then independently calculates the next best logical path from it to every possible destination in the network. The collection of best paths will then form the node's routing table. Thus in link state routing, if each node in the domain has the entire topology of the domain— the list of nodes and links, how they are connected including the type, cost), and the condition of the links the node can use the Dijkstra’s algorithm to build a routing table.

Dijkstra's Shortest Path Algorithm

Dijkstra's algorithm, is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree. This algorithm is often used in routing and as a subroutine in other graph algorithms.

It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path algorithm is widely used in network routing protocols, most notably IS-IS and OSPF (Open Shortest Path First).

Dijkstra's original algorithm does not use a min-priority queue and runs in time O(|V|^2) (where |V| is the number of vertices). The idea of this algorithm is also given in the implementation based on a min-priority queue implemented by a Fibonacci heap and running in O(|E|+|V|log|V|) where |E| is the number of edges. This is asymptotically the fastest known single-source shortest-path algorithm for arbitrary directed graphs with unbounded non-negative weights.

Pseudo code for Dijsktra's algorithm:

```
// Let v1 be the origin vertex,
//   and initialize W and ShortDist[u] as
   W := {v1}
   ShortDist[v1] :=0
   FOR each u in V - {v1}
      ShortDist[u] := T[v1,u]

// Now repeatedly enlarge W
//   until W includes all verticies in V
   WHILE W <> V

      // Find the vertex w in V - W at the minimum distance
      //   from v1
      MinDist := INFINITE
      FOR each v in V - W
         IF ShortDist[v] < MinDist
```

```
        MinDist = ShortDist[v]
        w := v
      END {if}
    END {for}

    // Add w to W
    W := W U {w}

    // Update the shortest distance to vertices in V - W
    FOR each u in V - W
      ShortDist[u] := Min(ShorDist[u],ShortDist[w] + T[w,u])
  END {while}
```

Application Design

The objective of the application is to performs the following tasks

- To simulate the process of generating connection table for each router in a given network.
- To compute the optimal path with least cost between any two specific routers.

The application is designed in Java.

The application provides the following options to the user

1. Enter a file name to automatically load a file
2. Manually enter a distance matrix
3. Build a routing table for a router
4. Find the shortest path between two routers
5. Exit

The topology (graph) matrix can either be loaded from a file or the user could wish to enter it manually. When the matrix is loaded from the file, the program runs the commands for getting the matrix from the file as mentioned by the user and it constructs the input matrix by reading each row one by one as a string. The value read is stored in the Loaded_Matrix. If any unexpected character is encountered then an appropriate error message is thrown.

After we input the topology matrix to the application, the user enters the router number for which the routing table is to be displayed. If the routers are not directly connected then the input from the topology matrix to that pair would be -1.To build a routing table for a router, the function find_Print_Routing_Table is used to build the routing table for a particular node. If the user enters an invalid router number an error message is displayed.

To find the shortest path between the routers, the function perform_Dijkstra , based on Dijkstras algorithm is used. The input entered by the user is parsed and the function computes the shortest path between the two nodes. As per the algorithm, Variables.Distance.BW_Nodes[source][source] is set to zero and visited[source] is set to true. The algorithm iterates for every node in the topology, the shortest path of node k is calculated, and dmin is the path length.

Variables Used

// Global variables

static int Random_Large_Number = 32784; Used to compare with any other edge weight

static int Router_Count = 0; Used to hold router count

// For initial load

static int[][] Loaded_Matrix; // Used to hold the the matrix loaded either manually or from file

// For building routing table

static int[][] Prev_Node; // Used to hold previous nodes in a path
static int[][] Next_Node; // Used to hold the next nodes in the path

// For Applying Dijkstra's Algorithm
static int[][] Weight_Of_Edges; // Used to hold all the edge weights
static int[][] Distance_BW_Nodes; // Used to calculate distance between both nodes

static boolean fileflag = false;//Initially is set to false, the value is set true when the topology matrix is loaded to the application.

Implementation of Dijkstra's algorithm

- The source number is passed to the function and it computes the shortest path between two nodes.
- The values from the input topology matrix is read and assigned to the variables. A 2 dimensional array int[][] Weight_Of_Edges ,is used to hold all the edge weights.
- A 2 dimensional array int[][] Distance_BW_Nodes, is used to calculate distance between both nodes.
- boolean fileflag = false, is Initially is set to false, the value is set true when the topology matrix is loaded to the application.
- Variables.Distance_BW_Nodes[source][source] is set to zero and visited[source] is set to true.
- Iterate and find a node k which is least cost to source.
- Calculate the shortest path of node k, and dmin is the path length.

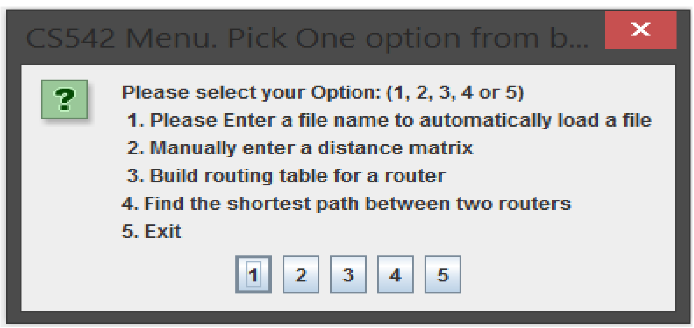
```
public class Dijkstra {

        public static void perform_Dijkstra(int source)
        {
           //The number of source node is passed to this method
           boolean[] visited = new boolean[Variables.Random_Large_Number];      //visited flag

           //initialization of visited[] and Prev_Node[][]

           for (int i = 0; i < Variables.Router_Count; i++)
           {
              //Distance_BW_Nodes[source][i] = Weight_Of_Edges[source][i];
              visited[i] = false;
```

```
    if (Variables.Distance_BW_Nodes[source][i] > 999)
    {
        Variables.Prev_Node[source][i] = -1;
    }
    else
    {
        Variables.Prev_Node[source][i] = source;
    }
}

 //the source node is the first one we visit, and the path length is 0

Variables.Distance_BW_Nodes[source][source] = 0;
visited[source] = true;

//we need n-1 iterations of the algorithm

for (int count = 1; count <= Variables.Router_Count - 1; count++)
{
    //find a node k which is least cost to source

    int k = -1;
    int dmin = Variables.Random_Large_Number;
    for (int i = 0; i < Variables.Router_Count; i++)
    {
        if (!visited[i] && Variables.Distance_BW_Nodes[source][i] < dmin)
        {
            k = i;
            dmin = Variables.Distance_BW_Nodes[source][i];
        }
    }

    //the shortest path of node k is calculated, and dmin is the path length

    Variables.Distance_BW_Nodes[source][k] = dmin;
    visited[k] = true;

    //adjust all other Distance_BW_Nodess with k as the intermediate node

    for (int i = 0; i < Variables.Router_Count; i++)
    {
```

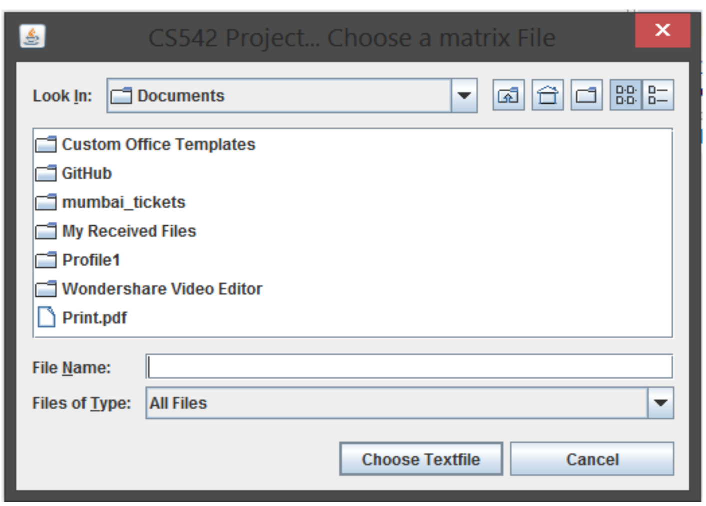
```
        if(!visited[i]&&Variables.Distance_BW_Nodes[k][i]<Variables.Random_Large_Number)
                {
                 dmin=Variables.Distance_BW_Nodes[source][k]                              +
                Variables.Weight_Of_Edges[k][i];
                }

                if(dmin<Variables.Distance_BW_Nodes[source][i]                         &&
Variables.Distance_BW_Nodes[k][i] < Variables.Random_Large_Number)
                {
                Variables.Distance_BW_Nodes[source][i] = dmin;
                Variables.Prev_Node[source][i] = k;
                }
              }

            }
          }

}
```

Interface

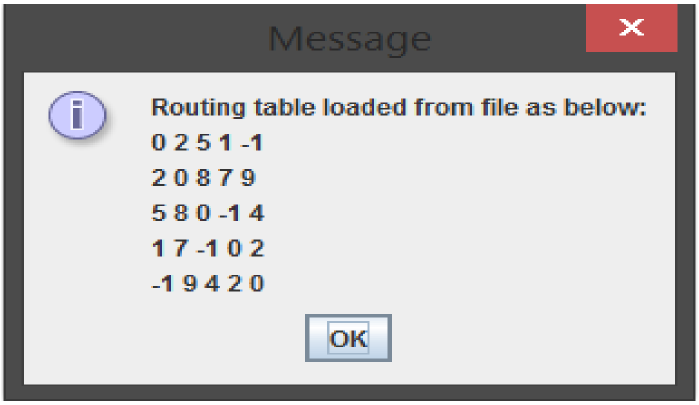
Main Menu



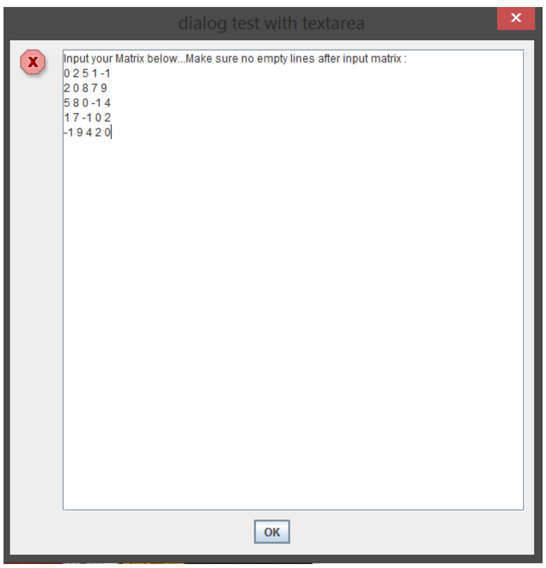
On selecting Option 1,to load the distance matrix from a text file



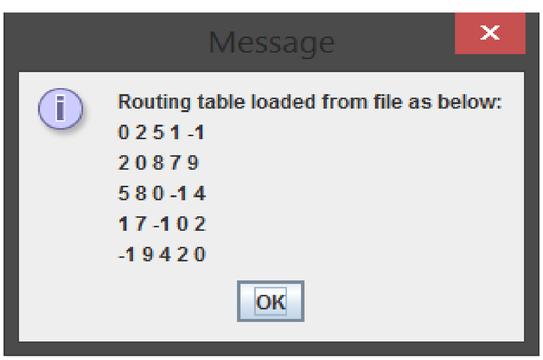
Matrix is sucessfully loaded



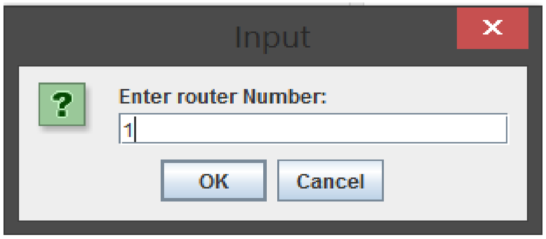
On selecting Option 2,to enter the distance matrix manually



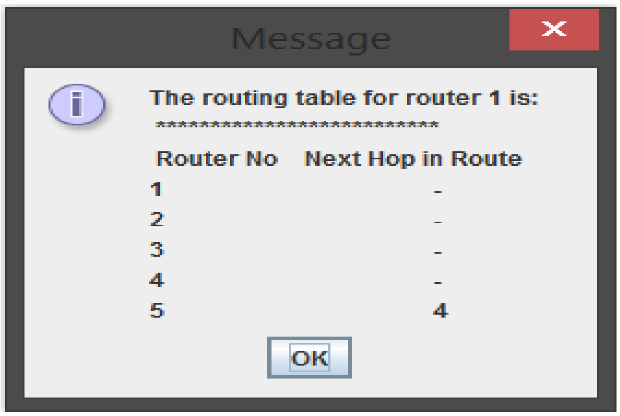
Matrix is sucesfully loaded



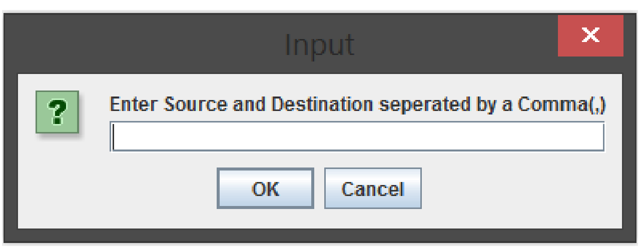
On selecting option 3,to build the routing table



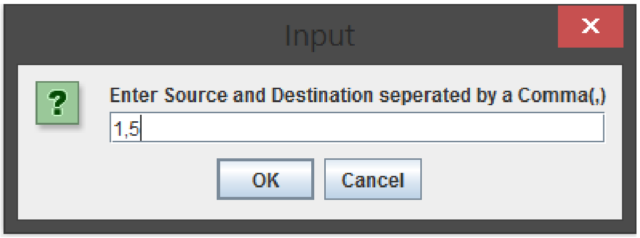
Routing table for router 1

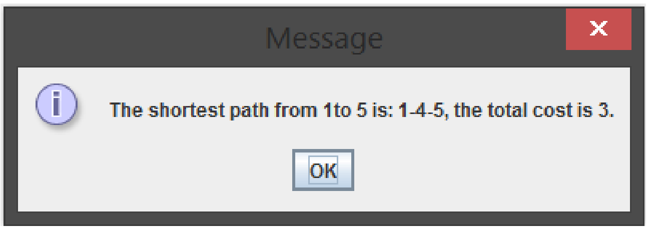


On selecting option 4,to find the shortest path between routers



Finding the shortest path between routers 1 and 5





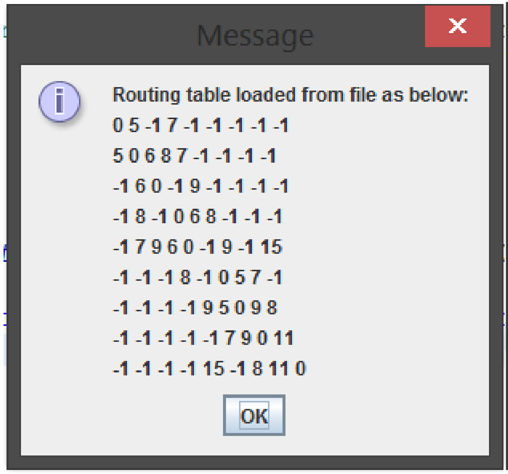
On selecting option 5, the application exits

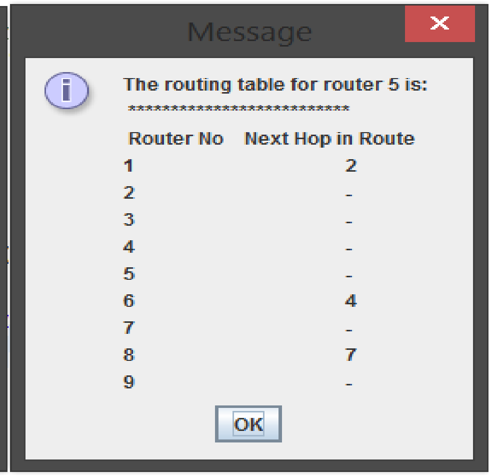
TEST CASES

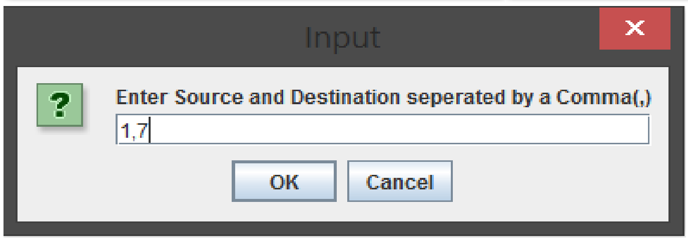
We have tested the functionality of the application with a wide range of inputs, with the number of routers in the topology ranging from 5 to 16.

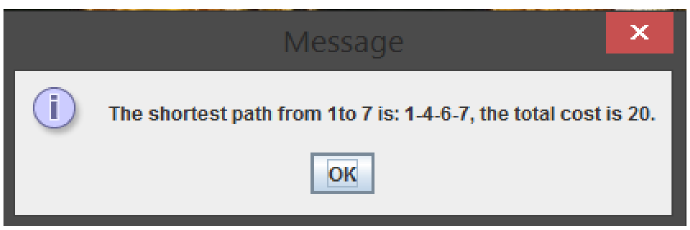
Test case 1:

(9*9) Input topology matrix



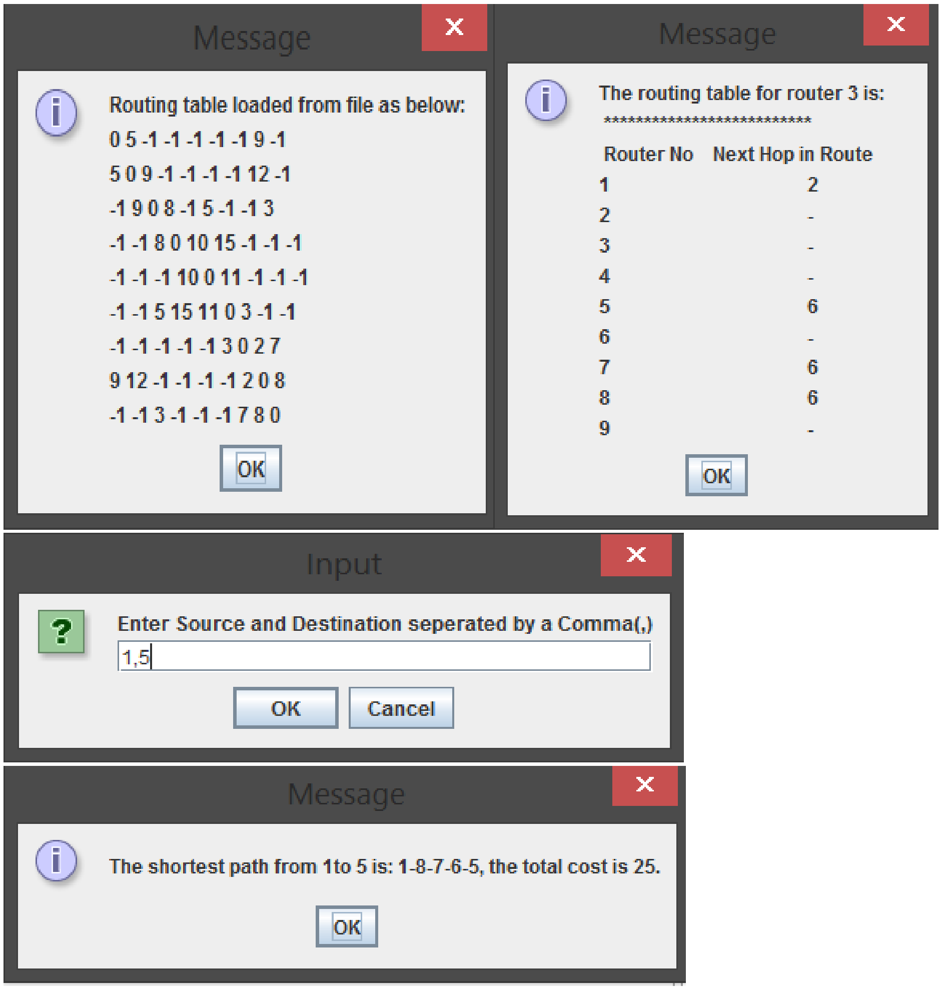






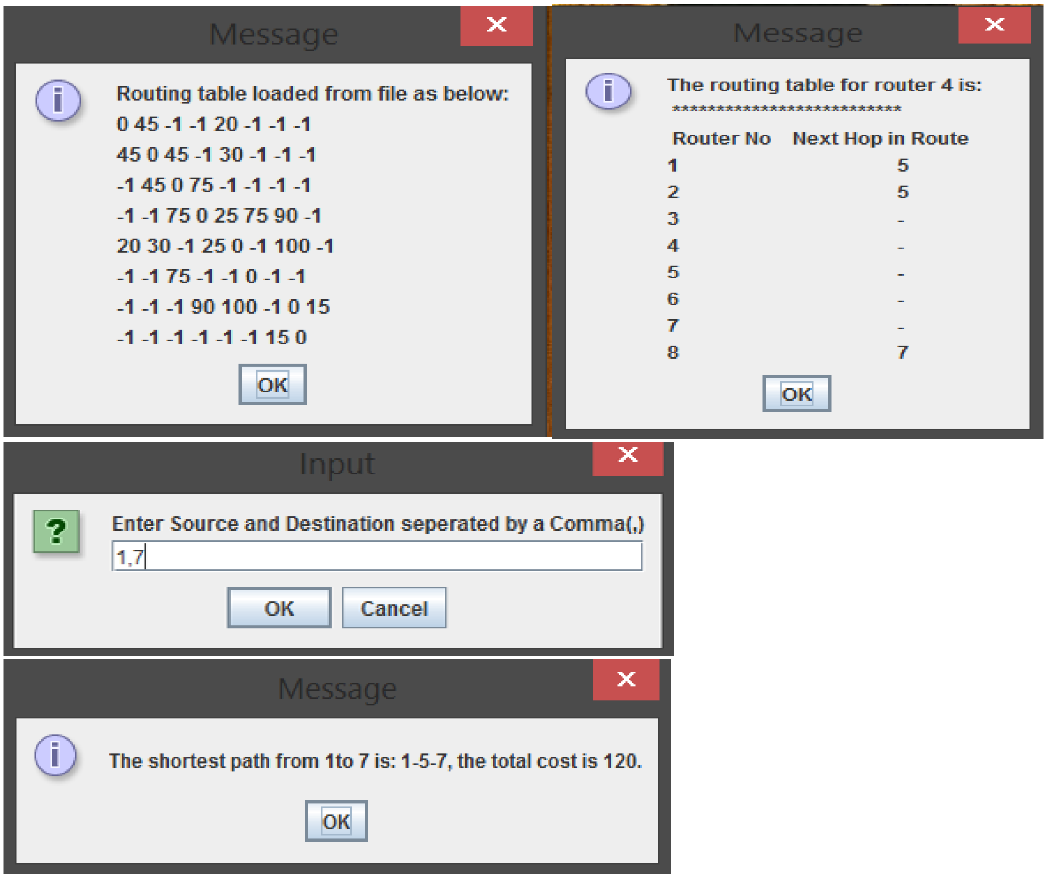
Test case 2:

(9*9) Input topology matrix



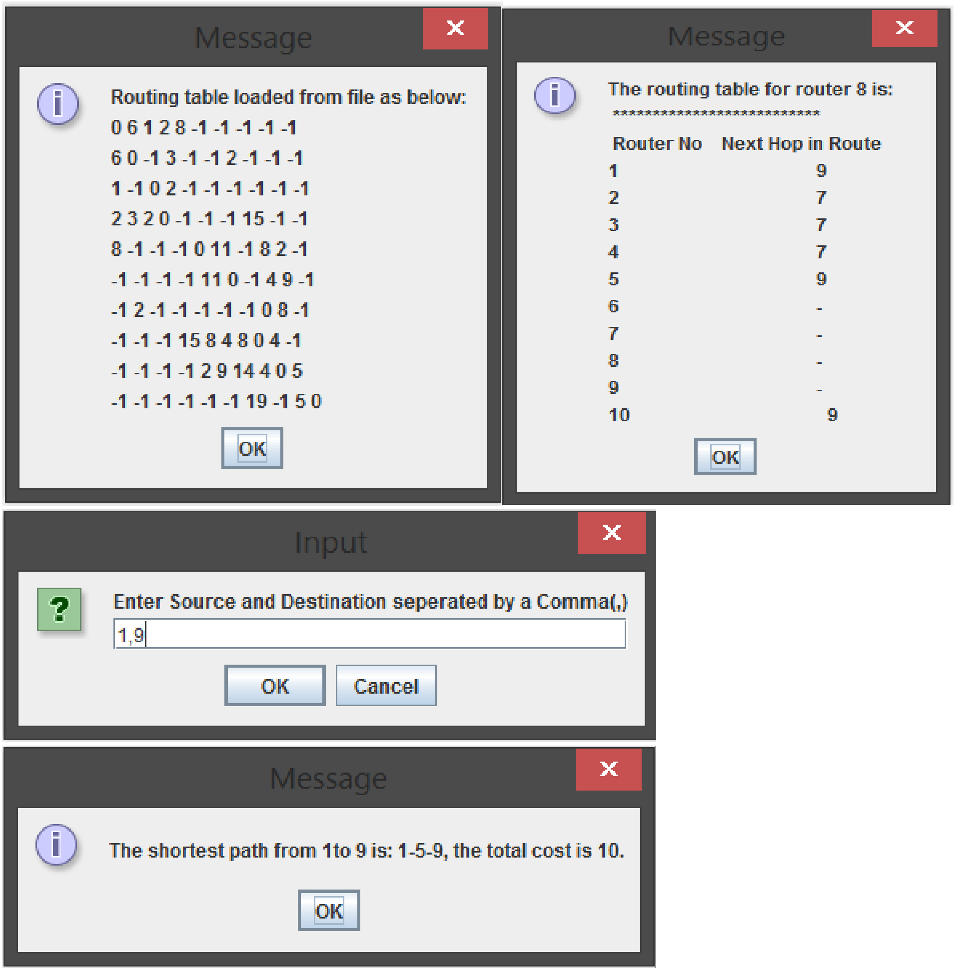
Test case 3:

(8*8) Input topology matrix



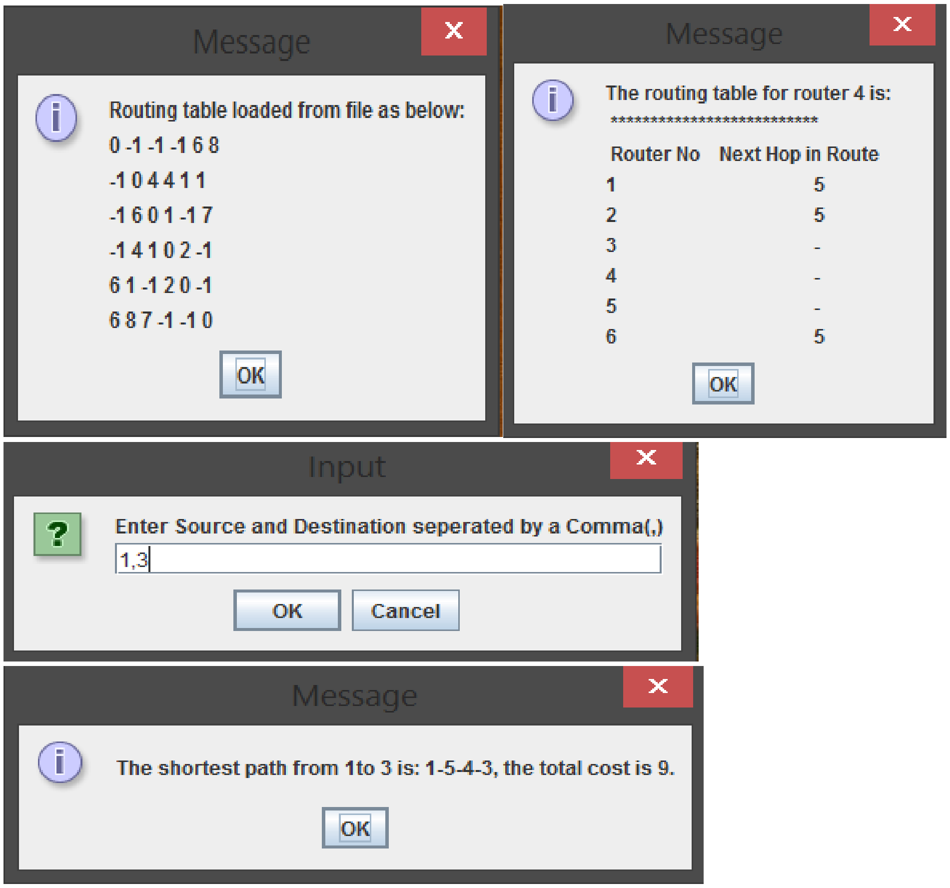
Test case 4:

(10*10) Input topology matrix



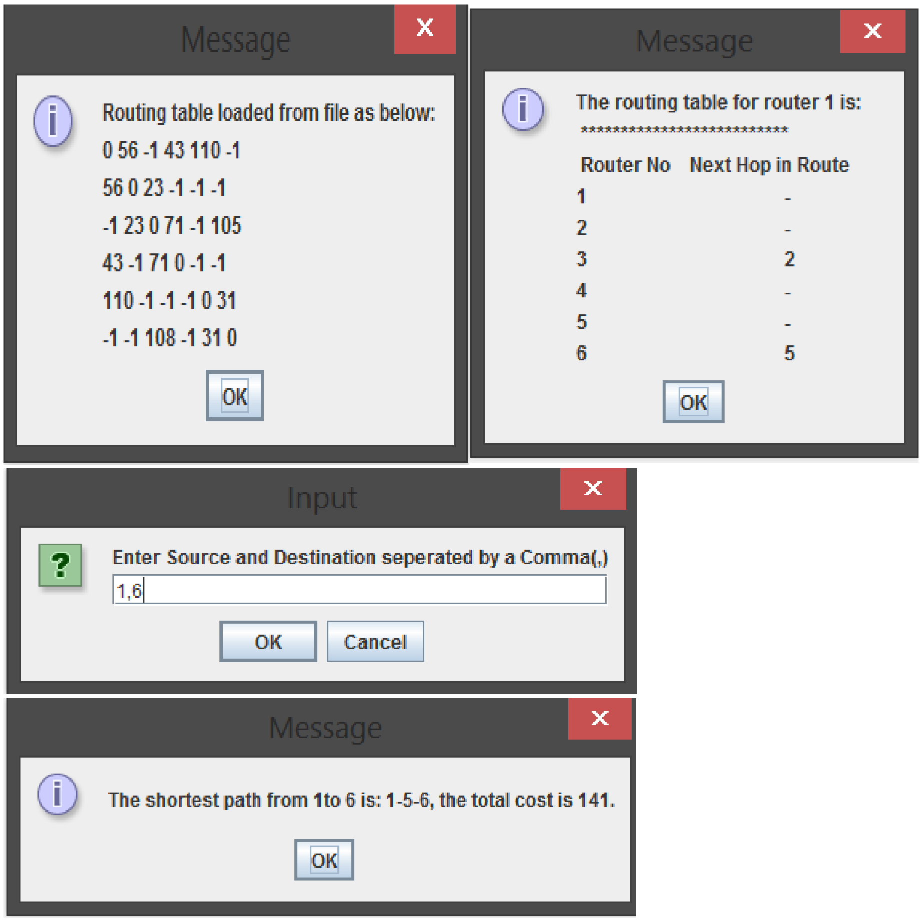
Test case 5:

(6*6) Input topology matrix



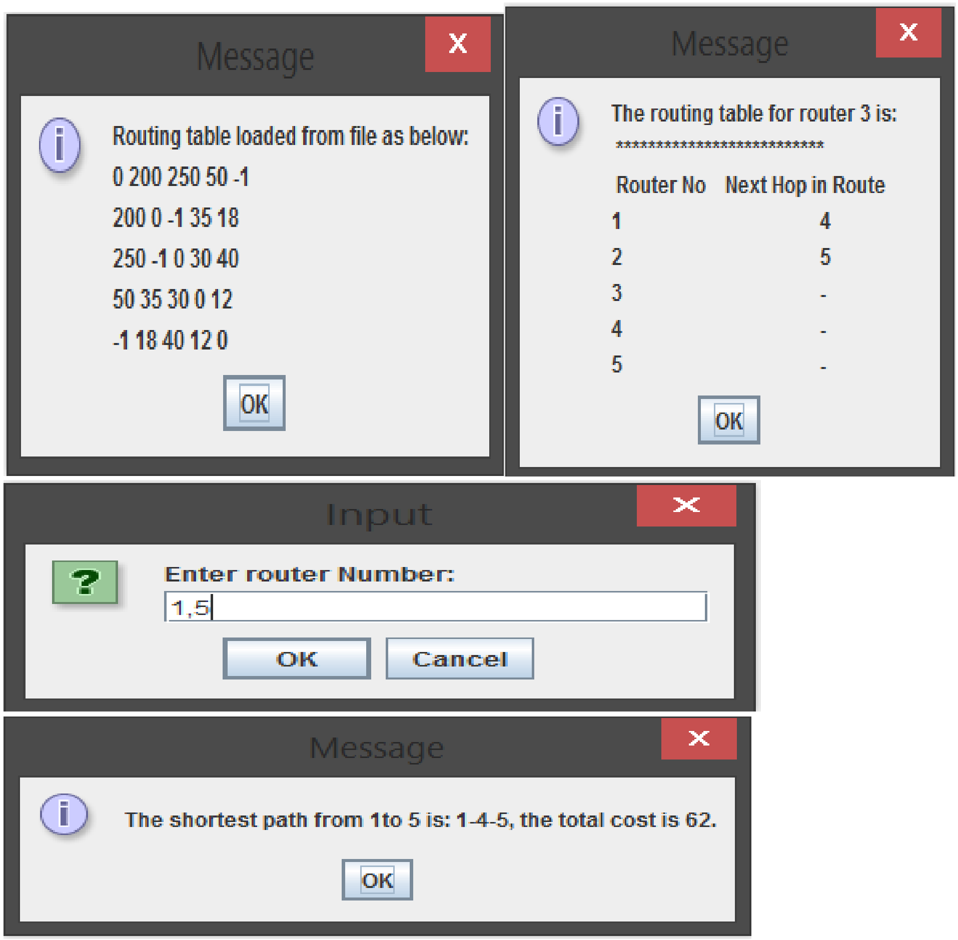
Test case 6:

(6*6) Input topology matrix


Message
Routing table loaded from file as below:
0 56 -1 43 110 -1
56 0 23 -1 -1 -1
-1 23 0 71 -1 105
43 -1 71 0 -1 -1
110 -1 -1 -1 0 31
-1 -1 108 -1 31 0
OK
Message
The routing table for router 1 is:
****
Router No Next Hop in Route
1 -
2 -
3 2
4 -
5 -
6 5
OK
Input
Enter Source and Destination seperated by a Comma(,)
1,6
OK
Cancel
Message
The shortest path from 1to 6 is: 1-5-6, the total cost is 141.
OK

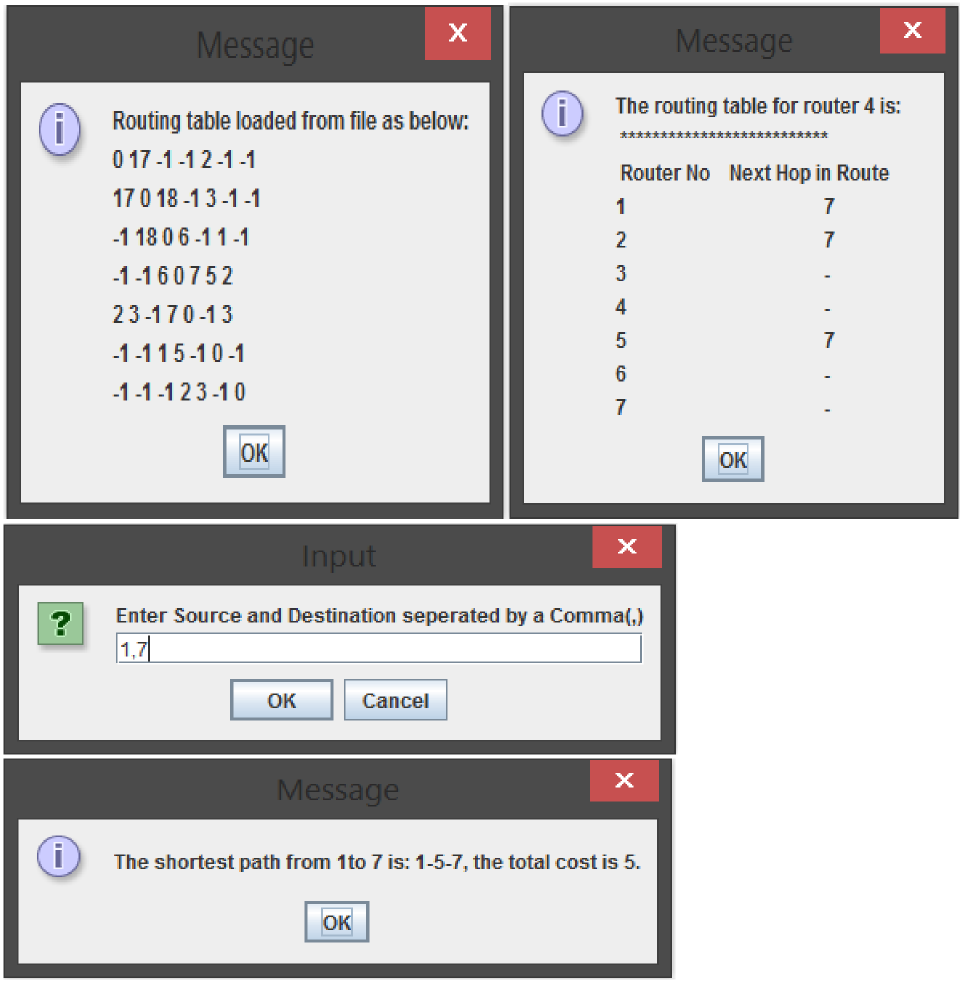
Test case 7:

(5*5) Input topology matrix



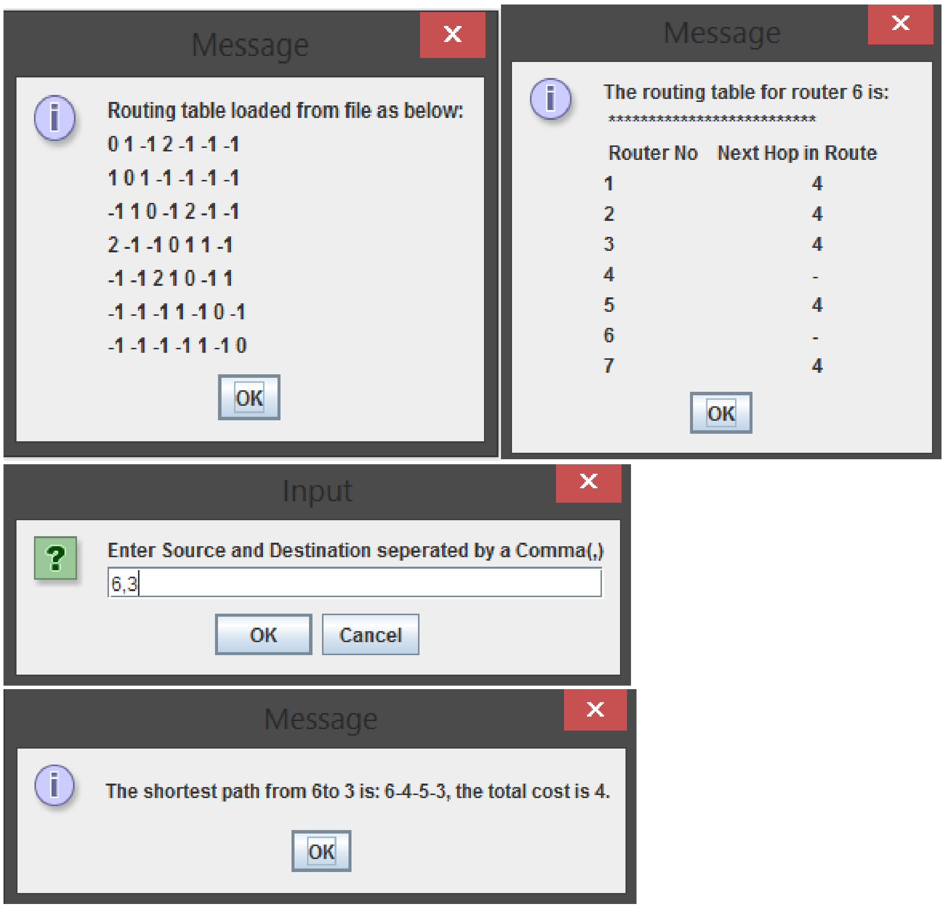
Test case 8:

(7*7) Input topology matrix



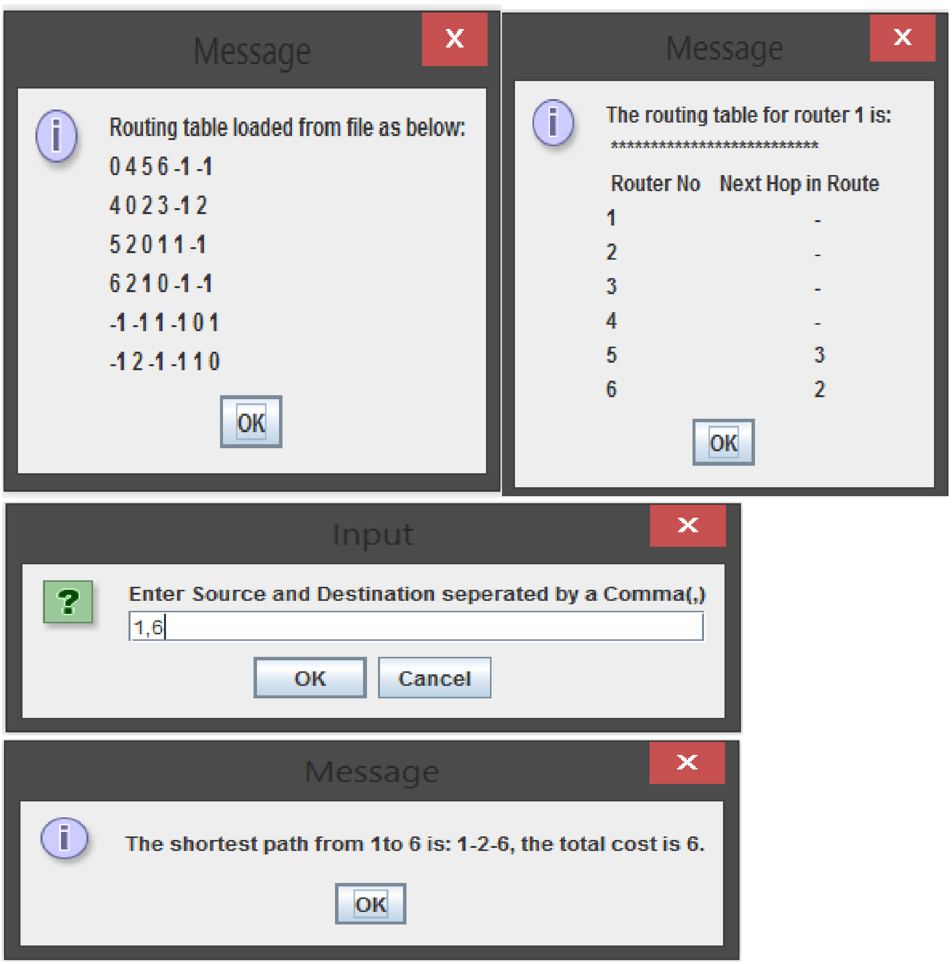
Test case 9:

(7*7) Input topology matrix



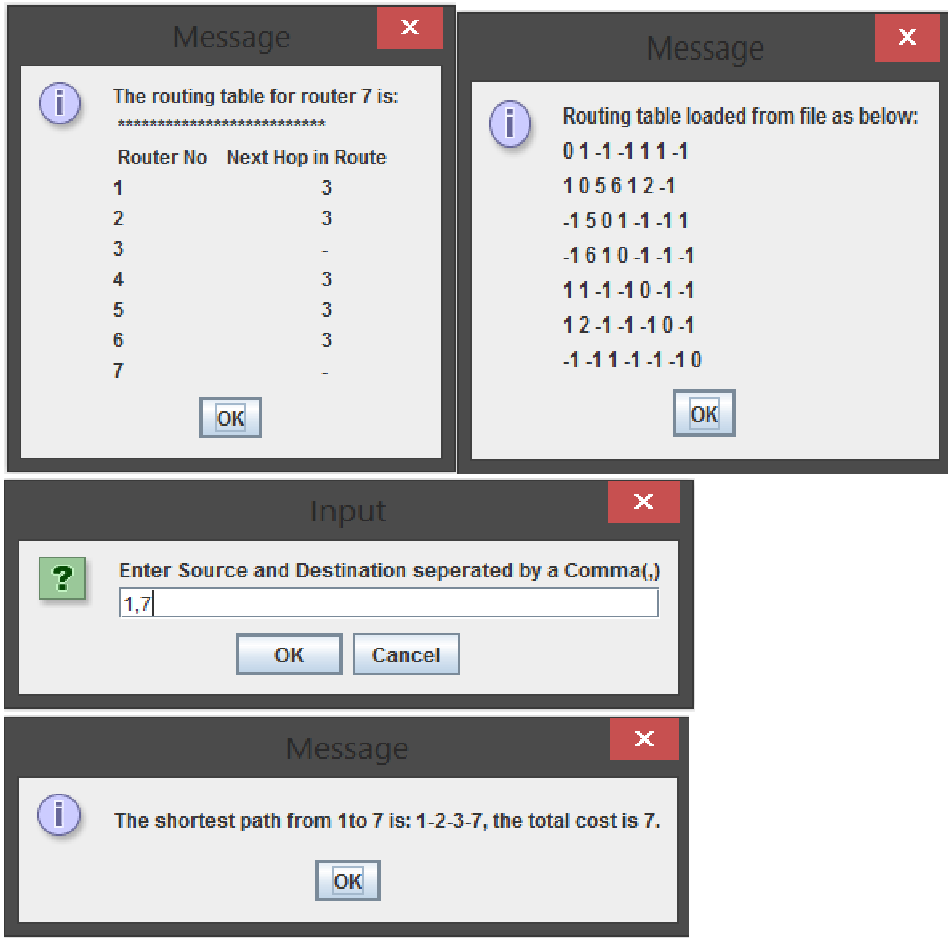
Test case 10:

(6*6) Input topology matrix



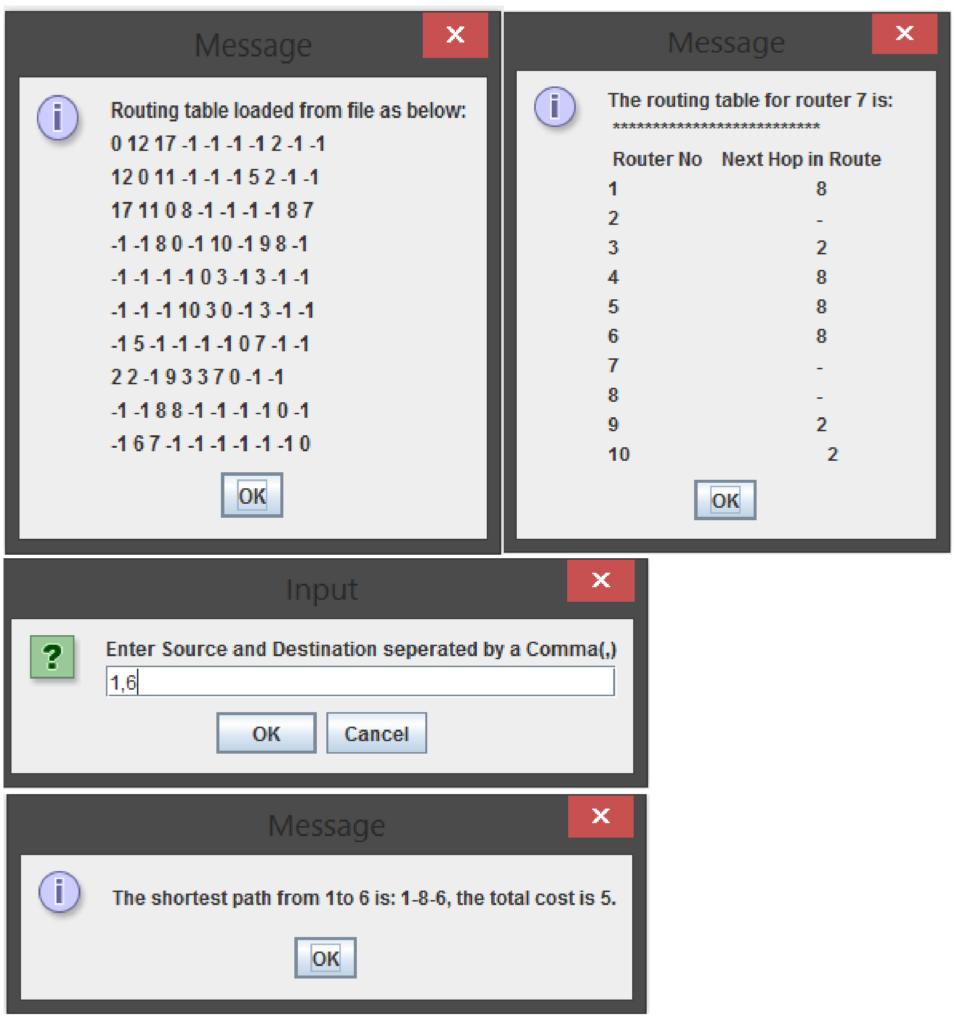
Test case 11:

(7*7) Input topology matrix



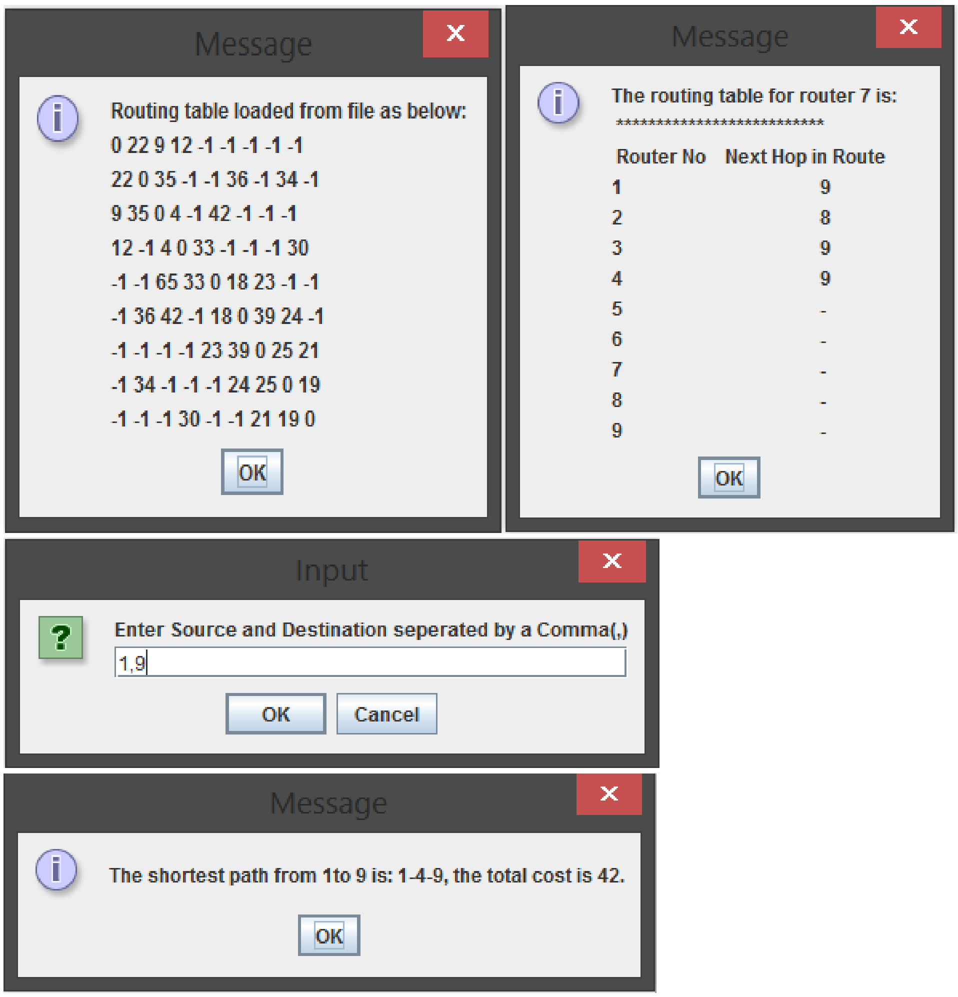
Test case 12:

(10*10) Input topology matrix



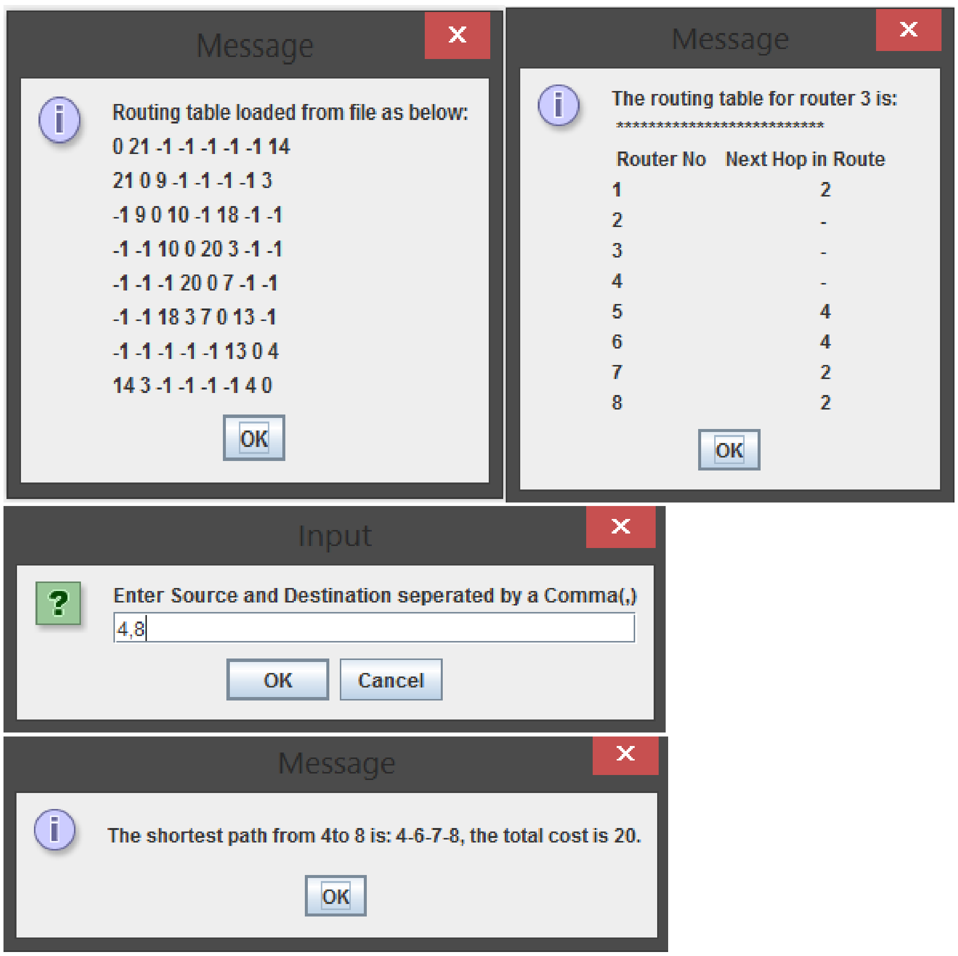
Test case 13:

(9*9) Input topology matrix



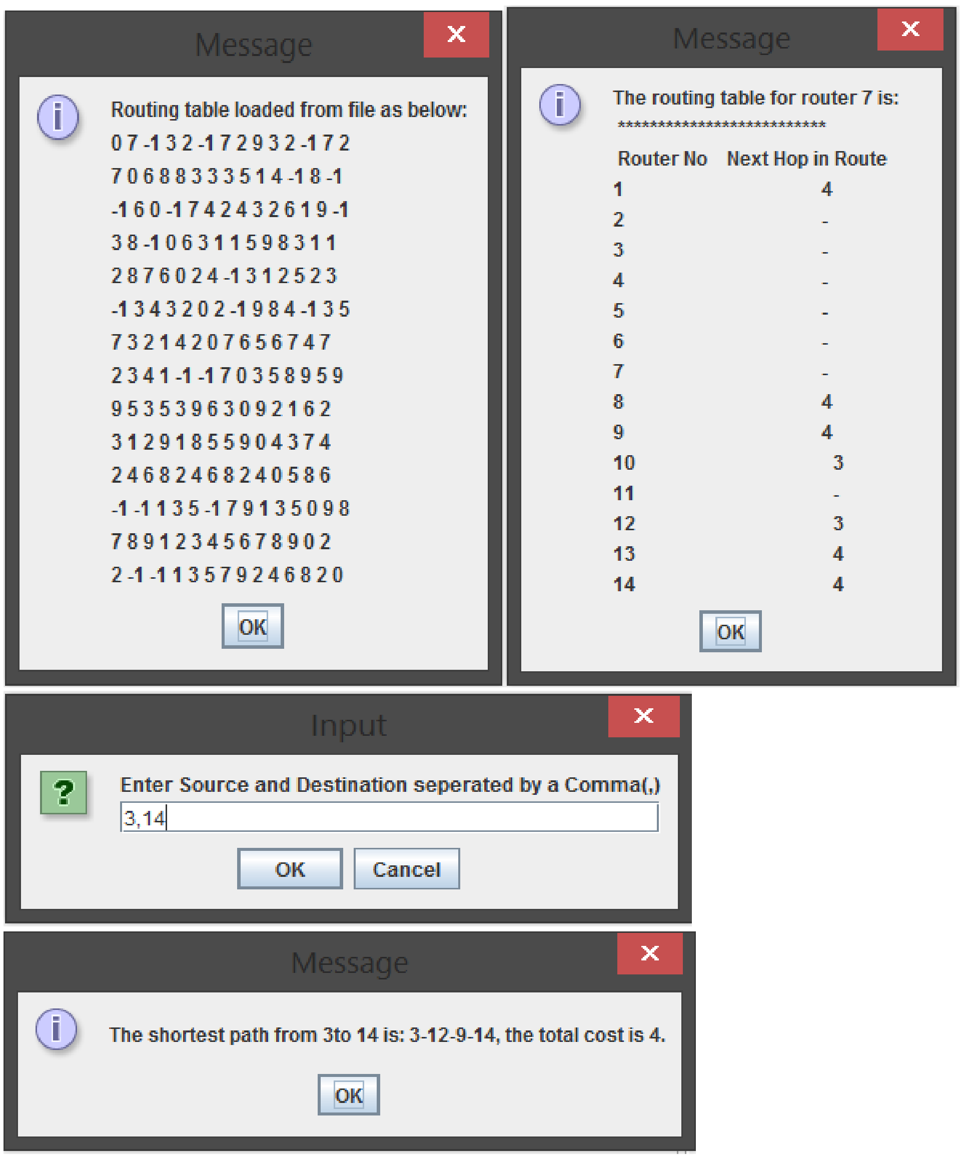
Test case 14:

(8*8) Input topology matrix



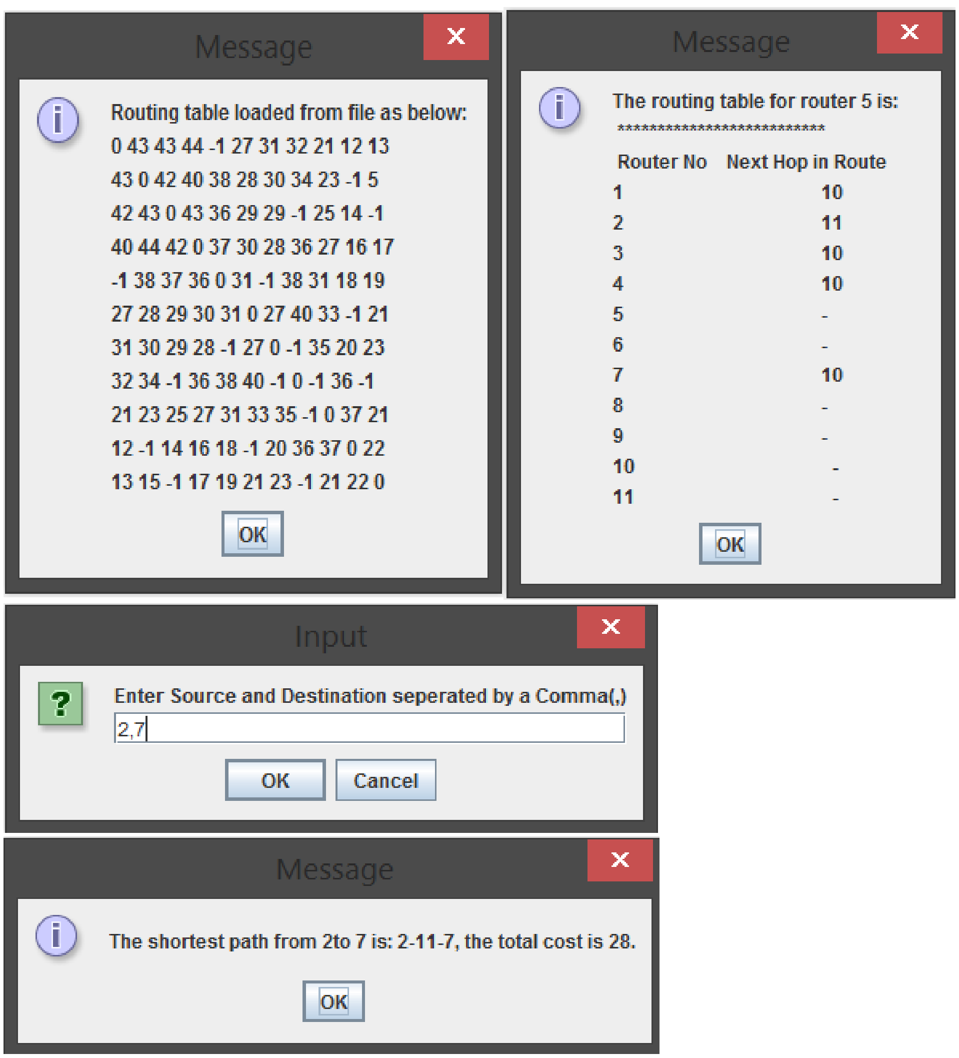
Test case 15:

(14*14) Input topology matrix



Test case 16:

(11*11) Input topology matrix

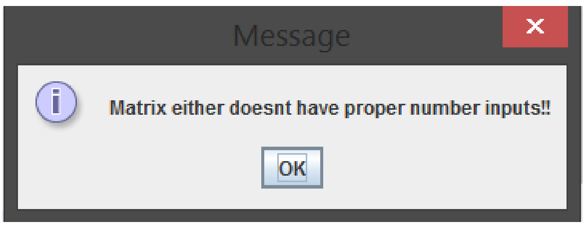


Error Handling

We have ensured that the application throws an appropriate error message when an error in encountered.

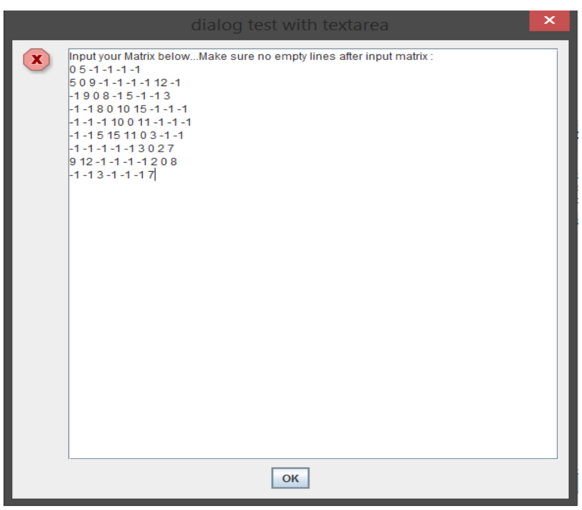
The following are the error conditions that are being taken care of

1) When we are loading a file format other than a text file.

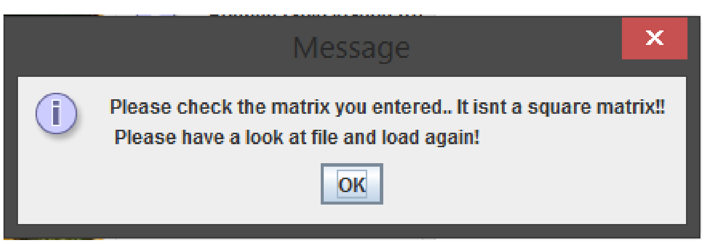


When there is an invalid input to the application from the text file or if the entered matrix is not a square matrix, an appropriate error message is displayed.

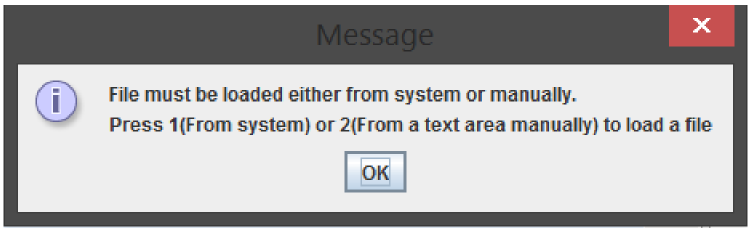
Consider the following example,



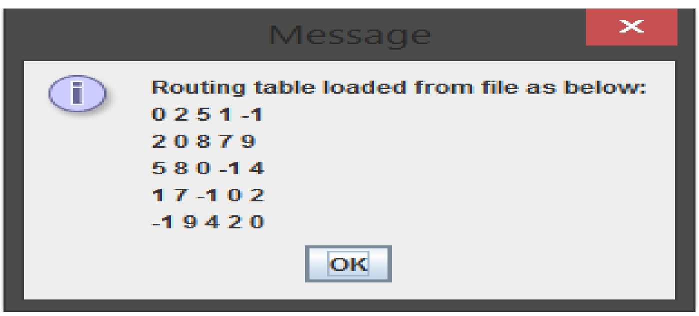
Since the entered topology matrix is not a square matrix an appropriate error message is thrown.



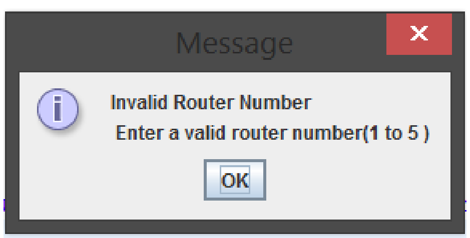
When the user selects the options to build a routing table for a router or to find the shortest path between two routers without providing input to the application, an appropriate error message is displayed.



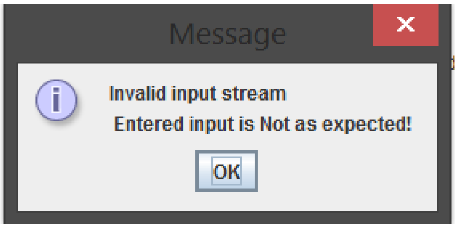
While building the routing table, if the user enters a router number that is greater than the number of routers



For instance,if the user has given the input to find the routing table of the 7th router,when the total number of routers in the topology matrix is 5,an appropriate error message is displayed.



While finding the shortest path between two routers, if the user inputs an invalid router number or format other than (source, destination), an appropriate error message is displayed.


Message
Invalid input stream
Entered input is Not as expected!
OK

Instructions to run the code

1. Keep the input.txt file ready so that it can be loaded to the application format.

- Ensure that a .txt file format is chosen and a square matrix is entered with the values separated by a single space

2. The user can also manually enter the topology matrix in the text area.

- Ensure that a square matrix is entered and there is no empty lines after the input matrix.

3. An executable file cs542LSR is provided and a minimum JRE version of 1.7.0 is required in order to run the application.

4. A menu option is displayed to the user to perform the required operations.

5. On selecting option 5 the application exits.

6. The input topology matrix for the test cases are present in the folder titled “graphs”.

Source code with comments

CS542prj.Java

- This class has the main method which triggers the menu options.

```
package cs542Prj;

import javax.swing.*;

import org.jgraph.JGraph;
import org.jgraph.graph.DefaultEdge;
import org.jgraph.graph.DefaultGraphCell;
import org.jgraph.graph.GraphConstants;
import org.jgrapht.ListenableGraph;
import org.jgrapht.ext.JGraphModelAdapter;
import org.jgrapht.graph.ListenableUndirectedGraph;
import org.jgrapht.graph.SimpleDirectedWeightedGraph;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.Rectangle2D;
import java.lang.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
import java.io.*;

@SuppressWarnings("unused")

// This class is for calculating shortest path with Link state routing (Dijkstra's Algorithm) to do 2
tasks
// 1) To find the Next_Node hop in shortest path
// 2) To find the shortest path between two specified hops

public class CS542Prj
{
/*	private static final Color	DEFAULT_BG_COLOR = Color.decode( "#FAFBFF" );
```

```
    private static final Dimension DEFAULT_SIZE = new Dimension( 530, 320 );

    public JGraphModelAdapter m_jgAdapter;*/

// Start of main method
public static void main(String args[])
{
        SimpleDirectedWeightedGraph g = new SimpleDirectedWeightedGraph(
DefaultEdge.class );

//      m_jgAdapter = new JGraphModelAdapter( g );
//  create a visualization using JGraph, via an adapter
//  JGraph jgraph = new JGraph( m_jgAdapter );
//  adjustDisplaySettings( jgraph );
//  getContentPane(  ).add( jgraph );
//  resize( DEFAULT_SIZE );

    // add some sample data (graph manipulated via JGraphT)


        // Define Local Variables

    File fileName = null; // File name
    int i, j, k; // Counters
    int option = 0; // For holding menu option
    int routerNum; // Local variable to track router number

    // Display a menu to enter options
    JOptionPane.showMessageDialog(null,"Link State routing using Dijkstra's Algorithm\nby Arun
S,Shreyas and Karthik N\n");
    //System.out.println("");

    do
    {
        ArrayList<String> optionList = new ArrayList<String>();
        optionList.add("1");
        optionList.add("2");
        optionList.add("3");
        optionList.add("4");
        optionList.add("5");
        Object[] options = optionList.toArray();
```

```
        // Display Menu
            option = JOptionPane.showOptionDialog(
              null,
              "Please select your Option: (1, 2, 3, 4 or 5)\n 1. Please Enter a file name to automatically
load a file \n "
              + "2. Manually enter a distance matrix \n "
              + "3. Build routing table for a router \n"
              + "4. Find the shortest path between two routers \n"
              + "5. Exit \n",
              "CS542 Menu. Pick One option from below Menu!",
              JOptionPane.YES_NO_OPTION,
              JOptionPane.QUESTION_MESSAGE,
              null,
              options,
              optionList.get(0));


        //Will not execute any option unless file is loaded.
        if (Variables.fileflag == false && option+1 != 1 && option+1 != 2 && option+1 != 5) {
            JOptionPane.showMessageDialog(null, "File must be loaded either from system or
manually.\nPress 1(From system) or 2(From a text area manually) to load a file ");
           continue;
        }

        switch (option+1) {
           case 1:
              //Load a file. File name is specified by user from JFileChoser.
              try
              {
                      Variables.fileflag = false;  //set the flag in case user chose a different file
                      Variables.Router_Count = 0;

                      // Choose a file from FileSystem
                      JFileChooser fc=new JFileChooser();
                      fc.setDialogTitle("CS542 Project... Choose a matrix File");
                      fc.setApproveButtonText("Choose Textfile");
                      int returnVal=fc.showOpenDialog(new JFrame());

                      if (returnVal == JFileChooser.APPROVE_OPTION)
                      {
                         fileName = fc.getSelectedFile();
                      }
```

```
DataInputStream dis = new DataInputStream(new FileInputStream(fileName));
BufferedReader br = new BufferedReader(new InputStreamReader(dis));
String strLine;
Variables.fileflag = true;

// Input of the routing table
// 1. The number of rows is the number of routers
// 2. The -1 edge will be assigned with Random_Large_Number weight

while ((strLine = br.readLine()) != null)
{
    Variables.Router_Count++;
    g.addVertex( "node"+Variables.Router_Count );
    System.out.println(g);
}
dis.close();
br.close();

if (Variables.Router_Count == 0)
{
    JOptionPane.showMessageDialog(null, "Nothing Loaded from file!!");
    break;
}


Variables.Loaded_Matrix = new int[Variables.Router_Count][Variables.Router_Count];
Variables.Prev_Node = new int[Variables.Router_Count][Variables.Router_Count];
Variables.Next_Node = new int[Variables.Router_Count][Variables.Router_Count];
//two arrays which stores the Prev_Node and Next_Node_Node node of source
Variables.Weight_Of_Edges = new int[Variables.Router_Count][Variables.Router_Count];
Variables.Distance_BW_Nodes = new int[Variables.Router_Count][Variables.Router_Count];

j = -1;

dis = new DataInputStream(new FileInputStream(fileName));
br = new BufferedReader(new InputStreamReader(dis));

while ((strLine = br.readLine()) != null)
```

```
{

   j++;
   String[] line = new String[Variables.Router_Count];
   line = strLine.split(" ");
   for (k = 0; k < Variables.Router_Count; k++)
     Variables.Loaded_Matrix[j][k] = Integer.parseInt(line[k]);
}

dis.close();
br.close();

// Print Routing table entered
String mat_Print = "Routing table loaded from file as below: \n",mat_Print1="";
for (j = 0; j < Variables.Router_Count; j++)
{
   for (k = 0; k < Variables.Router_Count; k++)
   {
     mat_Print1 = mat_Print1+ Variables.Loaded_Matrix[j][k] + " ";
     //System.out.println(Variables.Loaded_Matrix[j][k]);
     if ( Variables.Loaded_Matrix[j][k] != -1 && Variables.Loaded_Matrix[j][k] != 0)
     {
               //System.out.println("node"+j+1+"node"+k+1);
     g.addEdge( "node"+(j+1),"node"+(k+1) );
     System.out.println(g);
     }
   }
   mat_Print1 = mat_Print1+ "\n";
}

//positionVertexAt( "node1", 130, 40 );
//positionVertexAt( "node2", 60, 200 );
//positionVertexAt( "node3", 310, 230 );
//positionVertexAt( "node4", 380, 70 );



JOptionPane.showMessageDialog(null, mat_Print+mat_Print1);

Routing_Methods.init_Distance_Matrix();

// Build routing table for all the nodes
```

```
            for (j = 0; j < Variables.Router_Count; j++)
            {
                Dijkstra.perform_Dijkstra(j);
            }

            // Finding next
            for (j = 0; j < Variables.Router_Count; j++)
            {
                Routing_Methods.find_print_Routing_Table(j);
            }

          }
          catch (IOException | NumberFormatException| NullPointerException e)
          {
                // Avoid exceptions if any... If not then print
                System.err.println(e.getMessage());
                JOptionPane.showMessageDialog(null, "Matrix either doesnt have proper
number inputs!!");
            return;
          }
          catch (ArrayIndexOutOfBoundsException e)
          {
                // Avoid exceptions if any... If not then print
                JOptionPane.showMessageDialog(null, "Please check the matrix you entered.. It
isnt a square matrix!! \n Please have a look at file and load again!");
            return;
          }

          break;

        case 2:

          //Load a user input matrix from a text area.
          try
          {
                Variables.fileflag = false;  //set the flag in case user chose a different file
                Variables.Router_Count = 0;

                JTextArea textArea = new JTextArea("Input your Matrix below...Make sure no
empty lines after input matrix :\n");
                JScrollPane scrollPane = new JScrollPane(textArea);
                textArea.setLineWrap(true);
```

```
        textArea.setWrapStyleWord(true);
        scrollPane.setPreferredSize( new Dimension( 500, 500 ) );
        JOptionPane.showMessageDialog(null, scrollPane, "dialog test with textarea",
                        JOptionPane.YES_NO_OPTION);
        String[] ta = textArea.getText().split("\n");
    Variables.fileflag = true;

    String[] lines = Arrays.copyOfRange(ta, 1, ta.length);


    // Input of the routing table
    // 1. The number of rows is the number of routers
    // 2. The -1 edge will be assigned with Random_Large_Number weight

    Variables.Router_Count = lines.length;

    if (Variables.Router_Count == 0)
    {
        JOptionPane.showMessageDialog(null, "Nothing Loaded from text area!!");
        break;
    }

    Variables.Loaded_Matrix = new
int[Variables.Router_Count][Variables.Router_Count];
    Variables.Prev_Node = new int[Variables.Router_Count][Variables.Router_Count];
    Variables.Next_Node = new int[Variables.Router_Count][Variables.Router_Count];
//two arrays which stores the Prev_Node and Next_Node_Node node of source
    Variables.Weight_Of_Edges = new
int[Variables.Router_Count][Variables.Router_Count];
    Variables.Distance_BW_Nodes = new
int[Variables.Router_Count][Variables.Router_Count];

    j = -1;


    for (String line : lines)
    {

      j++;
      String[] lines1 = new String[Variables.Router_Count];
      lines1 = line.split(" ");
      for (k = 0; k < Variables.Router_Count; k++)
```

```
            {
              Variables.Loaded_Matrix[j][k] = Integer.parseInt(lines1[k]);
            }
          }


          String mat_Print = "Routing table loaded from file as below: \n",mat_Print1="";
          for (j = 0; j < Variables.Router_Count; j++)
          {
            for (k = 0; k < Variables.Router_Count; k++)
            {
              mat_Print1 = mat_Print1+ Variables.Loaded_Matrix[j][k] + " ";
            }
            mat_Print1 = mat_Print1+ "\n";
          }

          JOptionPane.showMessageDialog(null, mat_Print+mat_Print1);

          Routing_Methods.init_Distance_Matrix();

          // Build routing table for all the nodes
          for (j = 0; j < Variables.Router_Count; j++)
          {
              Dijkstra.perform_Dijkstra(j);
          }

          // Finding next
          for (j = 0; j < Variables.Router_Count; j++)
          {
              Routing_Methods.find_print_Routing_Table(j);
          }

        }
        catch ( NumberFormatException| NullPointerException e)
        {
              // Avoid exceptions if any... If not then print
              System.err.println(e.getMessage());
              JOptionPane.showMessageDialog(null, "Matrix either doesnt have proper
number inputs!!");
          return;
        }
        catch (ArrayIndexOutOfBoundsException e)
```

```
            {
                    // Avoid exceptions if any... If not then print
                    JOptionPane.showMessageDialog(null, "Please check the matrix you entered.. It
isnt a square matrix!! \n Please have a look at file and load again!");
               return;
            }

            break;



         case 3:
            // Read the Router number from JOptionPane
          try
          {
          String router_No = JOptionPane.showInputDialog ( "Enter router Number: " );
          //String case3_1,case3_2;
            routerNum = Routing_Methods.Check_for_Proper_RN(router_No); //check if the
number is valid
            System.out.println(routerNum);
                    if (routerNum == 999)
                    {
                              System.out.println("The routing table for router..Cant be build\n");
                              break;
                    }
                    else
                    {


                              // Print it hop by hop
                              Routing_Methods.print_Routing_Table(routerNum-1);
                    }

          }
          catch (NumberFormatException e)
          {
               System.out.println("Entered is a wrong input!!");
            }

            break;

         case 4:
```

```
            //Compute and print minimum path between 2 mentioned routers
            int source = 0,
             dest = 0;
            try
            {
                    String Src_Dest = JOptionPane.showInputDialog ( "Enter Source and Destination
seperated by a Comma(,)" );
               String[] temp = Src_Dest.split(",");

               source = Integer.parseInt(temp[0]) - 1;
               dest = Integer.parseInt(temp[1]) - 1;

               Routing_Methods.findPath(source, dest);
            }

            // Catch if not a number
            catch (NumberFormatException e)
            {
                    JOptionPane.showMessageDialog(null, "Invalid input stream \n Entered input is
Not as expected!");
            }

            // Catch if number is in or out of array
            catch (ArrayIndexOutOfBoundsException e)
            {
                    JOptionPane.showMessageDialog(null, "Enter 2 VALID Node numbers (Source and
Destination) seperated by a Comma! \n \t\t Neither Less Nor More!!");
            }

            break;
         case 5:
            System.exit(0);
            break;
         default:
            break;
      }
   }
   while (option != 5);
}

/*private void adjustDisplaySettings( JGraph jg ) {
   jg.setPreferredSize( DEFAULT_SIZE );
```

```
    Color  c       = DEFAULT_BG_COLOR;
    String colorStr = null;

    try {
        colorStr = getParameter( "bgcolor" );
    }
     catch( Exception e ) {}

    if( colorStr != null ) {
        c = Color.decode( colorStr );
    }

    jg.setBackground( c );
}

private void positionVertexAt( Object vertex, int x, int y )
{
    DefaultGraphCell cell = m_jgAdapter.getVertexCell( vertex );
    Map              attr = cell.getAttributes(  );
    Rectangle2D       b    = GraphConstants.getBounds( attr );

    GraphConstants.setBounds( attr, new Rectangle( x, y, b.OUT_RIGHT, b.OUT_TOP ) );

    Map cellAttr = new HashMap(  );
    cellAttr.put( cell, attr );
    m_jgAdapter.edit( cellAttr, null, null, null );
}*/
}
```

Varibles.java

- This class is used to keep track of the variables used

```
package cs542Prj;

public class Variables {

	// Global variables
	static int Random_Large_Number = 32784; // To compare with any other edge weight
	static int Router_Count = 0; // To hold router count

	// For initial load
	static int[][] Loaded_Matrix;  // To hold the the matrix loaded either manually or from file

	// For building routing table
	static int[][] Prev_Node;  // To hold previous nodes in a path
	static int[][] Next_Node;  // To hold the next nodes in the path

	// For Applying Dijkstra's Algorithm
	static int[][] Weight_Of_Edges; // To hold all the edge weights
	static int[][] Distance_BW_Nodes;  // To calculate distance between both nodes

	static boolean fileflag = false;
}
```

Routing_Methods.java

This class contains methods to find the next node and functions pertaining to build a routing table.

```
package cs542Prj;


import java.io.BufferedReader;
import java.io.IOException;

import javax.swing.JOptionPane;
```

```
@SuppressWarnings("unused")
public class Routing_Methods
{
	//Function which gets the Next_Node hop from the Source_Node router to Destination_Nodeination
	static void find_print_Routing_Table(int Source_Node)
	{
	   int i;
	   for (i = 0; i < Variables.Router_Count; i++)
	   {
	      int temp = Variables.Prev_Node[Source_Node][i];
	      if (temp == -1)
	      {
	          Variables.Next_Node[Source_Node][i] = -1;
	      }
	      else
	      {
	         while (temp != Source_Node)
	         {
	          Variables.Next_Node[Source_Node][i] = temp;
	            temp = Variables.Prev_Node[Source_Node][temp];
	         }
	      }
	   }
	}

	//print the routing table of #Source_Node router, indicating the Next_Node hop
	static void print_Routing_Table(int Source_Node)
	{
	          String sstr= "The routing table for router " + (Source_Node+1) + " is: \n ************************* \n Router No 	\tNext Hop in Route \n";
	          String str1="",str2="",str3="";
	   for (int i = 0; i < Variables.Router_Count; i++)
	   {
	      str1 = Integer.toString(i + 1) + "                     ";
	      int thisDis = Variables.Distance_BW_Nodes[Source_Node][i];
	      int thisWeight = Variables.Weight_Of_Edges[Source_Node][i];
	      if (thisDis == thisWeight)
	      {
	         str2 = "            " + "-"+"\n";
	      }
```

```
            else
            {
               str2 = "                " + Integer.toString(Variables.Next_Node[Source_Node][i] + 1) +
"\n";
            }
            str3 = str3+str1+str2;
         }
         //System.out.println();
         JOptionPane.showMessageDialog(null, sstr+str3);
         }

      //find the path of a Source_Node router to Destination_Node router
      static void findPath(int Source_Node, int Destination_Node)
      {
         int[] queue = new int[32784];
         //Find the path BW SOurce and Destination
         int i = 0;
         queue[i++] = Destination_Node;
         int temp = Variables.Prev_Node[Source_Node][Destination_Node];
         while (temp != Source_Node)
         {
            queue[i++] = temp;
            temp = Variables.Prev_Node[Source_Node][temp];
         }
         //print the path
         int j = 0;
         String strr1= "";
         for (j = i - 1; j > 0; j--)
         {
            strr1= strr1+(queue[j] + 1) + "-";
         }
         JOptionPane.showMessageDialog(null, "The shortest path from " + (Source_Node + 1)
+ "to " + (Destination_Node + 1) + " is: " +(Source_Node + 1) + "-" + strr1 + (Destination_Node +
1) + ", the total cost is " + Variables.Distance_BW_Nodes[Source_Node][Destination_Node] +
".\n");
      }

      // Check for proper router number whether it exceeds the number of routers calculated
based on matrix
      static int Check_for_Proper_RN(String line)
      {
         int rn = 0;
```

```
            try
            {
                // Check for more than router count condition
                if (Integer.parseInt(line) > Variables.Router_Count || Integer.parseInt(line) <= 0)
                {
                 JOptionPane.showMessageDialog(null, "Invalid Router Number \n Enter a valid
router number(1 to " + Variables.Router_Count + " ) ");
                   rn = 999;
                }
                else
                {
                rn = Integer.parseInt(line);
                }
            }
            catch (NumberFormatException|ArrayIndexOutOfBoundsException e)
            {
                 JOptionPane.showMessageDialog(null, "Invalid Router Number \n Entered input
is not a number!");
            }

            return rn;
        }

        //Initialize the Weight_Of_Edges and Distance_BW_Nodes according to the
Loaded_Matrix
        static void init_Distance_Matrix()
        {
            int j, k;
            // Loop for entire matrix reassigning the node weight as large number for all doesn't
have
            for (j = 0; j < Variables.Router_Count; j++)
            {
              for (k = 0; k < Variables.Router_Count; k++)
              {
                if (Variables.Loaded_Matrix[j][k] == -1)
                {
                 Variables.Weight_Of_Edges[j][k] = Variables.Random_Large_Number;
                }
                else
                {
                 Variables.Weight_Of_Edges[j][k] = Variables.Loaded_Matrix[j][k];
                }
```

```
            }
        }

        // Based on network topology re-making based on the weights to distance
        for (j = 0; j < Variables.Router_Count; j++)
        {
            for (k = 0; k < Variables.Router_Count; k++)
            {
                Variables.Distance_BW_Nodes[j][k] = Variables.Weight_Of_Edges[j][k];
            }
        }

    }
}
```

Dijkstra.java

- This class contains methods to find the shortest path between the two routers.

```
package cs542Prj;

public class Dijkstra {

        public static void perform_Dijkstra(int source)
        {
          //The number of source node is passed to this method
          boolean[] visited = new boolean[Variables.Random_Large_Number];      //visited flag

          //initialization of visited[] and Prev_Node[][]
          for (int i = 0; i < Variables.Router_Count; i++)
          {
            //Distance_BW_Nodes[source][i] = Weight_Of_Edges[source][i];
            visited[i] = false;

            if (Variables.Distance_BW_Nodes[source][i] > 999)
            {
                Variables.Prev_Node[source][i] = -1;
            }
            else
            {
                Variables.Prev_Node[source][i] = source;
            }
          }

           //the source node is the first one we visit, and the path length is 0
          Variables.Distance_BW_Nodes[source][source] = 0;
          visited[source] = true;
```

```
        //we need n-1 iterations of the algorithm
        for (int count = 1; count <= Variables.Router_Count - 1; count++)
        {
          //find a node k which is least cost to source
          int k = -1;
          int dmin = Variables.Random_Large_Number;
          for (int i = 0; i < Variables.Router_Count; i++)
          {
            if (!visited[i] && Variables.Distance_BW_Nodes[source][i] < dmin)
            {
              k = i;
              dmin = Variables.Distance_BW_Nodes[source][i];
            }
          }

          //the shortest path of node k is calculated, and dmin is the path length
          Variables.Distance_BW_Nodes[source][k] = dmin;
          visited[k] = true;

          //adjust all other Distance_BW_Nodes with k as the intermediate node
          for (int i = 0; i < Variables.Router_Count; i++)
          {
            if (!visited[i] && Variables.Distance_BW_Nodes[k][i] <
Variables.Random_Large_Number)
            {
              dmin = Variables.Distance_BW_Nodes[source][k] +
Variables.Weight_Of_Edges[k][i];
            }
            if (dmin < Variables.Distance_BW_Nodes[source][i] &&
Variables.Distance_BW_Nodes[k][i] < Variables.Random_Large_Number)
            {
             Variables.Distance_BW_Nodes[source][i] = dmin;
             Variables.Prev_Node[source][i] = k;
            }
          }

        }
      }

}
```

References:

http://web.eecs.umich.edu/~sugih/courses/eecs489/lectures/13-RoutingLS.pdf

http://en.wikipedia.org/wiki/RoutingMasterOpt/MyL09.pdf

http://en.wikipedia.org/wiki/Link-state_routing_protocol

http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

http://www.cc.gatech.edu/aimosaic/students/Psych-students/Shikano/Projects/AlgoNet/spa/code.html