

CS542-02 PROJECT

LINK STATE ROUTING PROTOCOL IMPLEMENTATION USING JAVA AND JAPPLET

**Anilraj Chennuru (A20328762)
Dheeraj Prajwal B V (A20329437)
Praninder Gupta V (A20330107)
Sahith Kumar Andel (A20329614)**

Introduction

Link state protocols, sometimes called shortest path first or distributed database protocols, are built around a well-known algorithm from graph theory, E. W. Dijkstra's shortest path algorithm.

Every switching node in the network performs the link-state protocol.

Each router originates information about itself, its directly connected links, the state of those links (hence the name), and any directly connected neighbors. This information is passed around from router to router, each router making a copy of it, but never changing it. The ultimate objective is that every router has identical information about the network, and each router will independently calculate its own best paths. That is, all routers speaking the same routing protocol.

Code Implementation and Design:

Implementation:

The main purpose is to implement the Link-State routing protocol using a program. The program's main goals are as follows:

- Building a path from one router to another with the given topology matrix.
- Generating routing tables at each router, which are identical.
- Finding the optimal least cost path given the source and destination from the routing table generated with the help of the program's core logic.
- Using Dijkstra's algorithm to obtain the shortest path as well as the direction between two routers.

Java and Applets are used to obtain both the logical and graphical representation of this project.

In the code, we use 2-dimension arrays to store original routing table, the distance between routers, values of distance during shortest path calculation, final table after calculation. Integer values are used for routers representation; the numbers of routers are limited to 100 routers. The program interface displays an applet to the user where he has to browse for the topology matrix file, input the source and destination nodes and will be able to calculate the shortest path between routers and display it to the applet viewer with the help of a button.

Algorithm

Dijkstra's algorithm is called the single-source shortest path. It is also known as the single source shortest path problem. It computes length of the shortest path from the source to each of the remaining vertices in the graph.

What the algorithm actually does is:

Computes the shortest route from the source to all other vertices and there are no negative edge weights, we know that the shortest edge from the source to another vertex must be a shortest path. Now, for each iteration, it checks if going through that new vertex can improve our distance estimates. We know that all shortest paths contain subpaths that are also shortest paths. Thus, if a path is to be a shortest path, it must build off another shortest path. That's essentially what we are doing through each iteration, is building another shortest path. When we add in a vertex, we know the cost of the path from the source to that vertex. Adding that to an edge from that vertex to another, we get a new estimate for the weight of a path from the source to the new vertex.

The algorithm used in the programs is as follows:

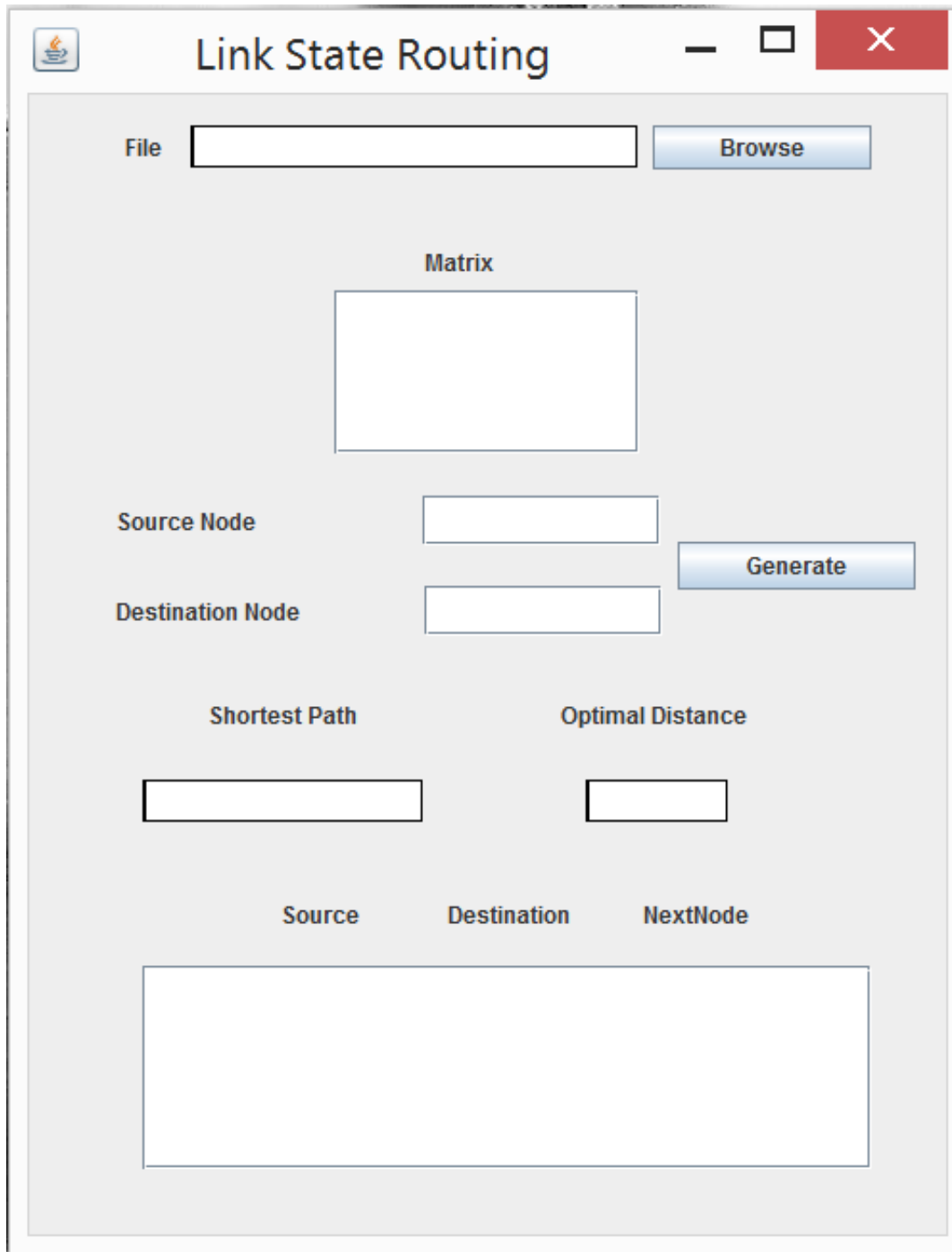
- Initially set min to a large value which is equivalent
- Store the Source and Destination routers
`src=x;`
`des=y;`
- Check whether min is greater than Distance to a particular node and the node is not visited
`if(min>distance[k][j] && visited[k][j]!=1)`
- If this is satisfied, min value is updated to the lesser cost distance
`min=distance[k][j];`
- Minimum distance node is assigned to nextnode.
`nextnode=j;`
- Visited from source to next node is changed to 1.
`visited[k][nextnode]=1;`
- If the given node is not visited then check whether the min value and cost for nextnode is less than distance
`if(visited[k][c]!=1)`
`if(min+matrix[nextnode][c]<distance[k][c])`
- If the condition satisfies the distance is updated with min + cost of the nextnode and pred stores the path traversed(Shortest path).
`distance[k][c]=min+matrix[nextnode][c];`
`pred[k][c]=nextnode;`

The looping continues for all the rows thereby obtaining the shortest path and the min distance to reach the destination through this path is computed.

This is the dijkstra's logic used in this program for computing the shortest path from a source router to a destination router, this algorithm can be modified to fit any number of routers.

Design

The program has a GUI implemented using APPLET .The APPLET design is as given below.



The image shows a Java applet window titled "Link State Routing". It features a standard window header with a Java icon, a title bar, and control buttons (minimize, maximize, close). The main content area is light gray and contains several interactive elements: a "File" label followed by a text input field and a "Browse" button; a "Matrix" label above a large empty rectangular text area; "Source Node" and "Destination Node" labels each followed by a text input field, with a "Generate" button to the right of the "Destination Node" field; two labels, "Shortest Path" and "Optimal Distance", each followed by a text input field; and a table with three columns labeled "Source", "Destination", and "NextNode", with a large empty rectangular text area below the column headers.

- The Applet has a Browse button, which will enable to choose a file from the local system. It also has a TextArea for displaying the Inputted Topology matrix.
- The Applet accepts the Source node and the Destination node from the user and has a button Generate to compute the protocol and obtain the shortest path as well as the Optimal Distance between the given source and destination routers which is being displayed in the respective TextAreas.

- The Applet also has a TextArea to display the Routing Table, which has the source, destination and the nextnode (Interface).
- The Applet code is controlled by the main Java Program to enable Button Functionalities like button click events through ActionListener interfaces and also to set and get the text from the respective TextFields and TextAreas.

Program Code Sample

1) To determine the dimensions of the topology matrix i.e. Number of routers (rows and columns).

Dijkstra (String Fpath)

```
{
    Scanner count = new Scanner (new File(Fpath));

    While (count.hasNextLine())
    {

        if (count.hasNextInt())
        {
            rows ++;
            cols =1;
        }

        String s=count.nextLine()+" ";
        Scanner count1=new Scanner(s);
        count1.useDelimiter(" ");
        while(count1.hasNext())
        {

            Display=Display+count1.next()+" ";
            if(count1.hasNextInt())
                cols++;

        }
        Display=Display+"\n";
    }
}
```

2. Reading the topology matrix, after browsing from the local system.

```
if(rows==cols)
{
    Scanner sc = new Scanner(new File(Fpath));
    for(int i=0;i<rows&&sc.hasNextLine();i++)    // checks whether there is a next line.
    {
        int j=0;
```

```

        String str=sc.nextLine()+" ";
// Reads the next line.
        Scanner sc1=new Scanner(str);
        sc1.useDelimiter(" ");
// The scanner class uses the delimiter space to read input vales
        while(sc1.hasNext()&&j<cols)
// checks whether there is a number.
        {
            matrix[i][j] = Integer.parseInt(sc1.next());//Stores the values in an array.
            visited[i][j]=0;
//Visited matrix represents the value 0 initially.
            pred[i][j]=i;
//pred matrix is used to store the traversed(shortest) path
            if(matrix[i][j]==-1)

                matrix[i][j]=999;
//It initializes matrix elements with "0" to 999.
            System.out.println(matrix[i][j]);
            j++;
        }
        System.out.println();
    }
}

```

3. Applying Dijkstra's algorithm for a given topology matrix.

```

Dimplementation(int x,int y)
{
    /*Scanner s=new Scanner (System.in);
    System.out.println("enter the source");
    src=s.nextInt();
    System.out.println("enter the destination");
    des=s.nextInt();*/

    src=x;
    des=y;
    if(rows!=cols)
    {
        System.out.println("Incorrect matrix format");
        return -1;
    }
    else
    {
        for(int k=0;k<rows;k++)
        {
            for(int i=0;i<rows;i++)
            {
                min=999;
                // Intially sets the minimum value to largest value 999.
                for(int j=0;j<cols;j++)
                {

```

```

        if(min>distance[k][j] && visited[k][j]!=1)
// Checks whether min is less than Distance to a particular node and visited is not equal to 0.
        {
            min=distance[k][j];
// If true min value is updated until minimum value is updated.
            nextnode=j;
// minimum distance node is assigned to nextnode.
        }
    }

    visited[k][nextnode]=1;
// Visited from source to nextnode is changed to 1.

    for(int c=0;c<rows;c++)
    {
        if(visited[k][c]!=1)

        {
            if(min+matrix[nextnode][c]<distance[k][c])
// Checks min value and cost for nextnode is less than actual distance
            {
                distance[k][c]=min+matrix[nextnode][c];
// Distance for that node is updated with min plus cost to nextnode
                pred[k][c]=nextnode;
// pred stores the shortest path
                c++;
            }
        }
    }
}
return 0;
}
}
}

```

4.)

a. Printing the path for a given source and destination.

```

printPath ()
{
    if(rows!=cols)
        return "No path Exists";
    System.out.print("pred 0 1 2 3 4 5"+"\\n  ");
    for(int k=0;k<rows;k++)
    {
        System.out.print(pred[src][k]+"-");
    }

    System.out.println();
}

```

```

int j;
System.out.print("path = " );
j=des;
String Path= " "+des;
do
{

    j=pred[src][j];
    // pred matrix has the path from destination to source.
    //System.out.print("-" + j);
    Path=Path+"-"+j;
    // Path is converted to string

}while(j!=src);
StringBuffer a = new StringBuffer(Path);
// reverse the string path
String s=a.reverse().toString();
// Convertes the string buffer to string.
System.out.print(s);
return(s);
// Returns the string s.
}

```

b. Printing the optimal distance for a given source and destination.

```

printDis ()
{
    If (rows==cols)
    {
        System.out.println("\nsrc-des\tDistance");
        System.out.println(src+"---"+des+"\t   "+distance[src][des]);
        return distance[src][des];
        // returns the distance from source to destination
    }
    else
        return -1;
}

```

c. Printing the routing table with source, destination and next node as parameters.

```

printRoute()
{
    for(int i=0;i<rows;i++)
    {
        int j=i;
        do
        {
            route[i]=j;
            // Route array is used to store the next node from source to destination

```



```

        j=pred[src][j];

        }while(j!=src);
    }
    System.out.println("Source---Destination---Nextnode");
    String rout="";
    for(int i=0;i<rows;i++)
    {
        rout=rout+"\t"+src+"\t"+" "+i+"\t "+route[i]+"\n";
        System.out.println(src+"\t"+" "+i+"\t\t"+" "+route[i]);
// appends all the Source destination and nextnode and
    }
    return rout;

        // returns the rout.
    }

```

5. GUI with JAVA Applet.

```

MyApplet () {

    JFrame frame = new JFrame("Link State Routing");
    Container c = (Container)frame.getContentPane();

    JLabel lblFile = new JLabel("File");

    c.add(lblFile);

    ta_browse = new JTextArea();
    c.add(ta_browse);

    btn_browse = new JButton("Browse");
    c.add(btn_browse);

    ButtonBro bb = new ButtonBro();
    btn_browse.addActionListener(bb);

    lbl_error = new JLabel();
    c.add(lbl_error);

    JLabel lbl_source = new JLabel ("Source Node");
    c.add(lbl_source);

    JScrollPane scrollPane_1 = new JScrollPane();

    ta_matrix = new JTextArea();
    scrollPane_1.setViewportView(ta_matrix);

    JLabel lblMatrix = new JLabel("Matrix");
    c.add(lblMatrix);

```

```

JLabel lbl_destination = new JLabel ("Destination Node");
c.add(lbl_destination);

tf_source = new JTextField();
c.add(tf_source);

tf_destination = new JTextField();
c.add(tf_destination);

btn_generate = new JButton("Generate");
c.add(btn_generate);
btn_generate.addActionListener(this);

JLabel lbl_path = new JLabel("Shortest Path");
c.add(lbl_path);

JLabel lbl_distance = new JLabel("Optimal Distance");
c.add(lbl_distance);

ta_path = new JTextArea();
c.add(ta_path);

ta_distance = new JTextArea();
c.add(ta_distance);

JLabel lbl_rtable1 = new JLabel("Source");
c.add(lbl_rtable1);

JLabel lbl_rtable2 = new JLabel("Destination");
c.add(lbl_rtable2);

JLabel lbl_rtable3 = new JLabel("NextNode");
c.add(lbl_rtable3);

JScrollPane scrollPane = new JScrollPane();
frame.getContentPane().add(scrollPane);

ta_rtable = new JTextArea();
scrollPane.setViewportViewView(ta_rtable);

}

```

6. The above mentioned view modules are controlled by the following snippet shown below.

```

ActionsPerformed(ActionEvent args0) {

    int a = Integer.parseInt(tf_source.getText());
    int b= Integer.parseInt(tf_destination.getText());
    String Fpath=ta_browse.getText();
    if(Fpath=="")

```

```

        lbl_error.setText("Please Input the file");
    else
    {
        //Dijkstra dj= new Dijkstra(ta_browse.getText());
        if(dj.Dimplementation(a,b)==0)
        {
            lbl_error.setText("");
            String s=dj.printPath();
            ta_path.setText(s);
            ta_distance.setText(""+dj.printDis());
            ta_rtable.setText(dj.printRoute());
        }
        else
            lbl_error.setText("Incorrect matrix file");
    }
}

```

Instructions to compile and run:

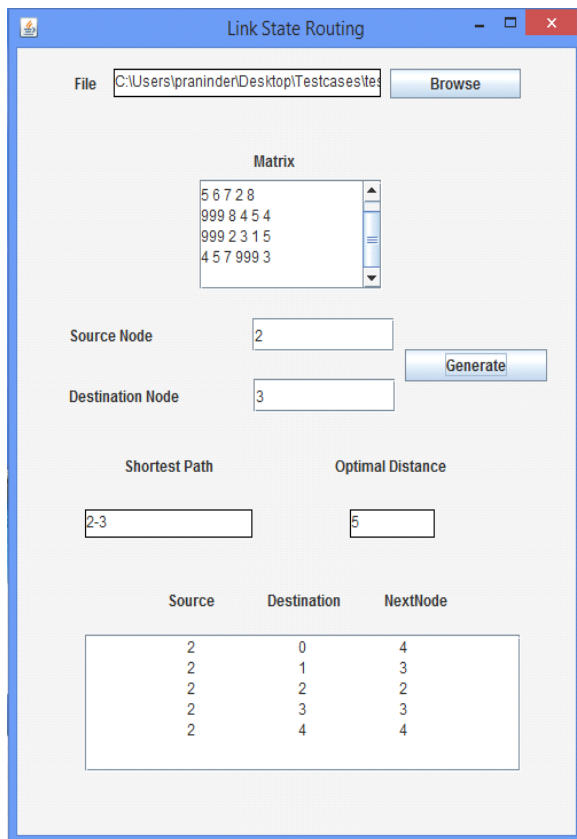
1. Install standard jre (java runtime environment) software preferably 1.6 version or above and set the path in the environment variables (for windows).
2. The next step after installing and setting the path for java, download the jar file.
3. Run the jar file.
4. After executing the jar file, browse the appropriate topology file.
5. Enter the source and destination and press the generate button to find the shortest path between the nodes and the distance.
6. Finally, Displaying the routing table with source node and various destinations nodes via the next node (interfaces) on the Applet viewer.

Test Cases:

Test cases for 5 *5 Asymmetric matrix:

	Test Condition	Expected O/P	Actual O/P
Test Case 1:	Src=2, Des=3	Path=2-3,Distance=5	Path=2-3,Distance =5
Test Case 2:	Src=5, Des=2	Error must be Handled	Node not exists
Test Case 3:	Src= ,Des=	Error must be Handled	Incorrect Node.
Test Case 4:			
Test Case 5:			

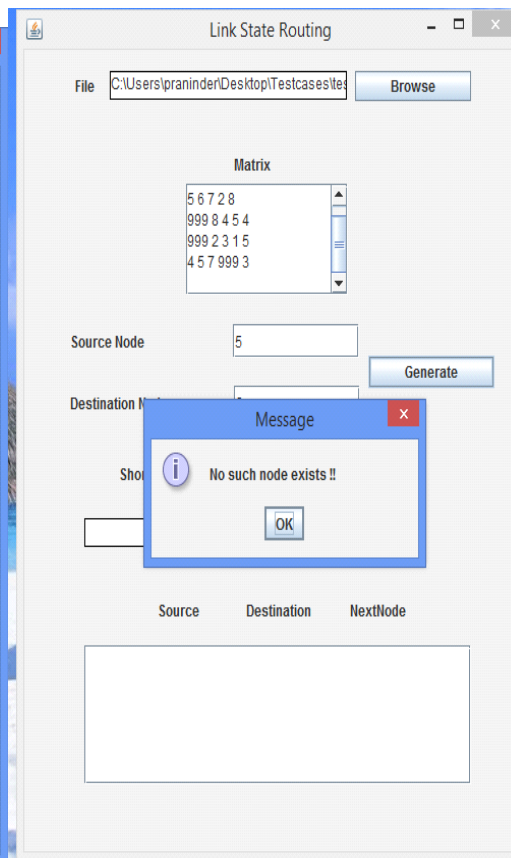
Test Case 1:



The screenshot shows the 'Link State Routing' application window. The 'File' field contains the path 'C:\Users\praninder\Desktop\Testcases\tes'. The 'Matrix' field displays a 5x5 grid of numbers: 5 6 7 2 8, 999 8 4 5 4, 999 2 3 1 5, 4 5 7 999 3. The 'Source Node' is set to 2 and the 'Destination Node' is set to 3. The 'Generate' button is visible. Below the matrix, the 'Shortest Path' is displayed as '2-3' and the 'Optimal Distance' is displayed as '5'. At the bottom, a table shows the next hop for each node:

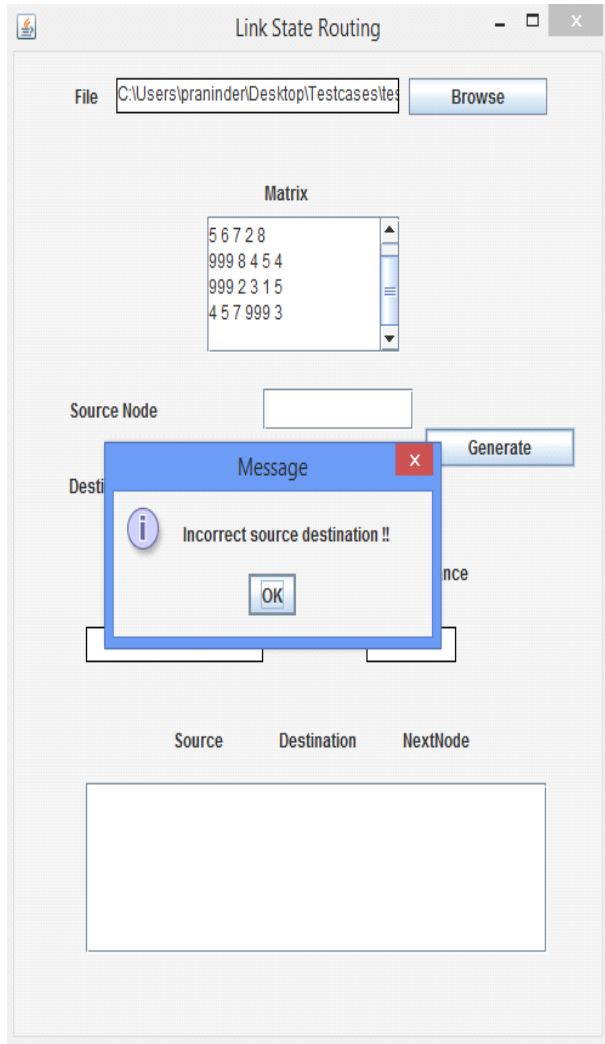
Source	Destination	NextNode
2	0	4
2	1	3
2	2	2
2	3	3
2	4	4

Test Case 2:



The screenshot shows the 'Link State Routing' application window. The 'File' field contains the path 'C:\Users\praninder\Desktop\Testcases\tes'. The 'Matrix' field displays a 5x5 grid of numbers: 5 6 7 2 8, 999 8 4 5 4, 999 2 3 1 5, 4 5 7 999 3. The 'Source Node' is set to 5 and the 'Destination Node' is set to 2. The 'Generate' button is visible. A 'Message' dialog box is displayed in the foreground with the text 'No such node exists !!' and an 'OK' button.

Test Case 3:



Test Cases for 6 x 6 Symmetric Matrix:

	Test Condition	Expected O/P	Actual O/P
Test Case 1:	Src=2, Des=3	Path=2-4-3,Distance=2	Path=2-4-3,distance=2
Test Case 2:	Src=7, Des=3	Error must be handled	No such node exists
Test Case 3:	Src= ,Des=	Error must be handled	Incorrect source destination

Link State Routing

File:

Matrix

0	2	5	1	0	0
2	0	3	2	0	0
5	3	0	3	1	5
1	2	3	0	1	0
0	0	1	1	0	2

Source Node:

Destination:

Message

No such node exists !!

Source Destination NextNode

Link State Routing

File:

Matrix

5	3	0	3	1	5
1	2	3	0	1	-1
-1	-1	1	1	0	2
-1	-1	5	-1	2	0

Source Node:

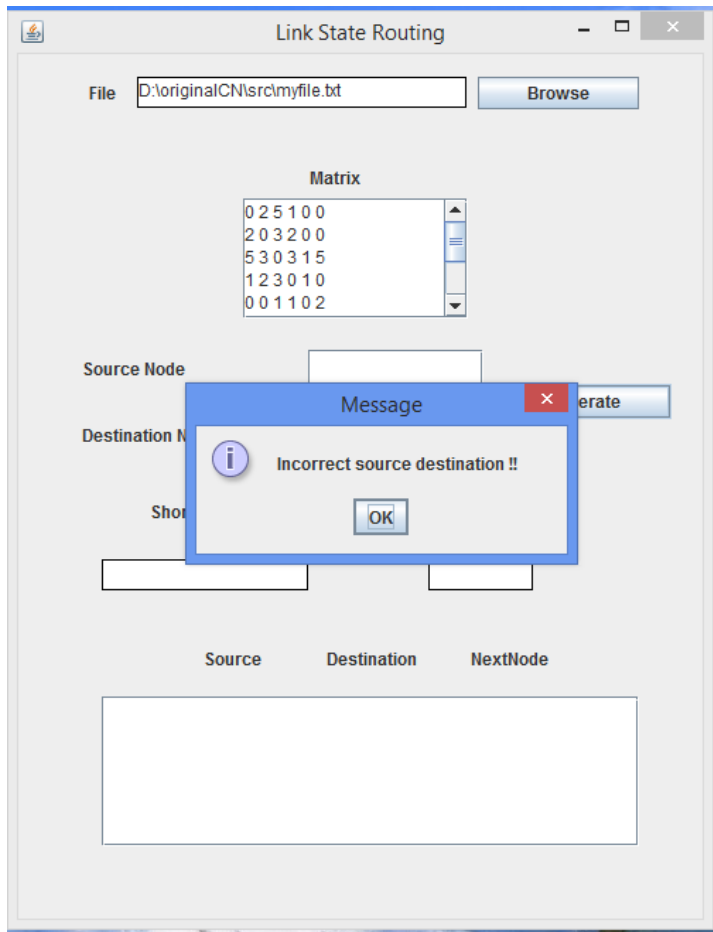
Destination Node:

Shortest Path:

Optimal Distance:

Source Destination NextNode

2	0	4
2	1	1
2	2	2
2	3	4
2	4	4
2	5	4



Test cases for 10 x10 Asymmetric Matrix:

	Test Condition	Expected O/P	Actual O/P
Test Case 1:	Src=1, Des=	Path=2-3,Distance=5	Path=2-3
Test Case 2:	Src=4, Des=12	Error must be handled	Node does not exit
Test Case 3:	Src= ,Des=	Error must be handled	Incorrect source destination

Link State Routing

File

C:\Users\praninder\Desktop\Testcases\tes

Browse

Matrix

7	3	5	2	2	8	9	2	1	5
999	999	3	7	4	3	9	2	6	4
999	999	999	4	7	2	9	6	3	9
999	999	999	999	7	4	9	3	7	

Source Node

1

Generate

Destination Node

9

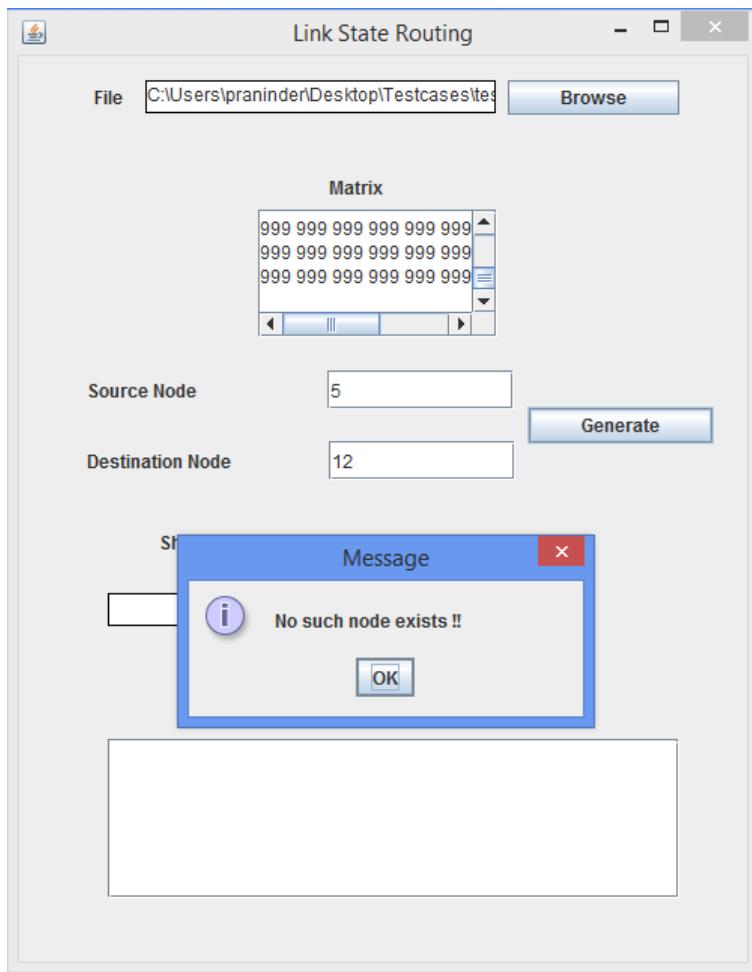
Shortest Path

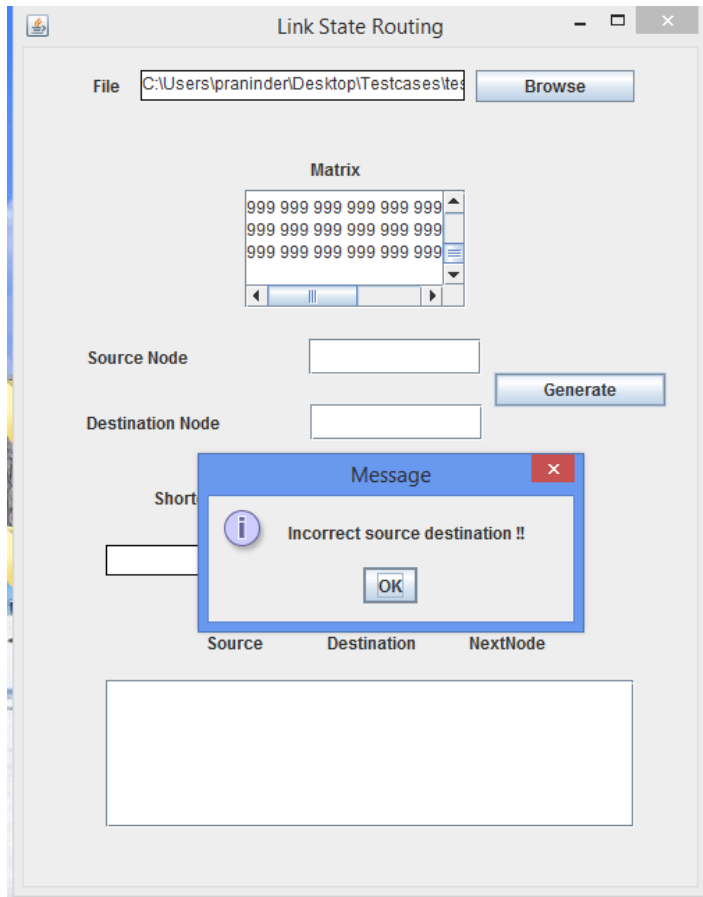
1-9

Optimal Distance

4

Source	Destination	NextNode
1	0	0
1	1	1
1	2	2
1	3	3
1	4	4
1	5	5
1	6	6





Test Cases for 12 x 12 Symmetric Matrix:

	Test Condition	Expected O/P	Actual O/P
Test Case 1:	Src=17, Des=17	Path=17-17,Distance=0	same
Test Case 2:	Src=5, Des=14	Error must be handled	No such node exists
Test Case 3:	Src= ,Des=	Error must be handled	Incorrect source destination

Link State Routing

File:

Matrix

11	4	4	9	7	1	8	6	0	1	6	6
7	1	7	7	4	5	3	2	1	0	5	5
12	2	9	1	2	3	8	7	6	5	0	3
4	3	1	5	1	2	7	5	6	5	3	0

Source Node:

Destination Node:

Shortest Path:

Optimal Distance:

Source	Destination	NextNode
6	7	2
6	8	9
6	9	9
6	10	2
6	11	2

Link State Routing

File:

Matrix

11	4	4	9	7	1	8	6	0	1	6	6
7	1	7	7	4	5	3	2	1	0	5	5
12	2	9	1	2	3	8	7	6	5	0	3
4	3	1	5	1	2	7	5	6	5	3	0

Source Node:

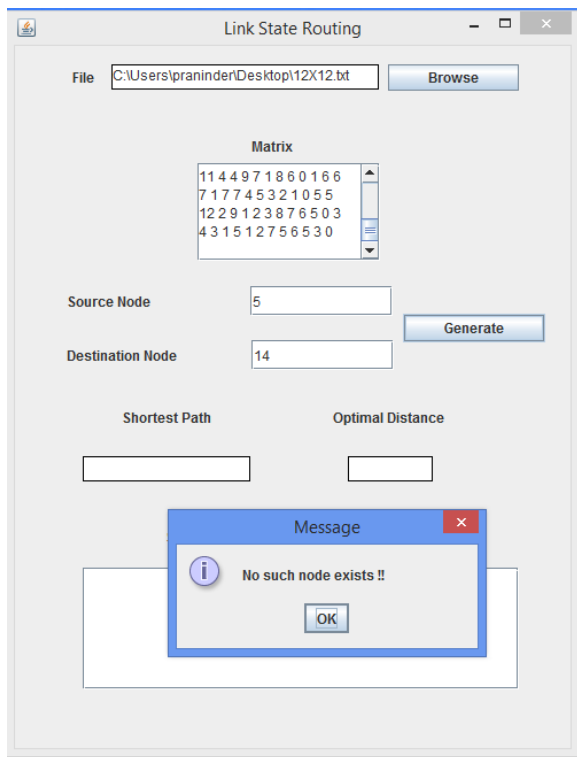
Destination Node:

Shortest Path:

Optimal Distance:

Message

Incorrect source destination !!



Link State Routing

File

Matrix

11	4	4	9	7	1	8	6	0	1	6	6
7	1	7	7	4	5	3	2	1	0	5	5
1	2	9	1	2	3	8	7	6	5	0	3
4	3	1	5	1	2	7	5	6	5	3	0

Source Node

Destination Node

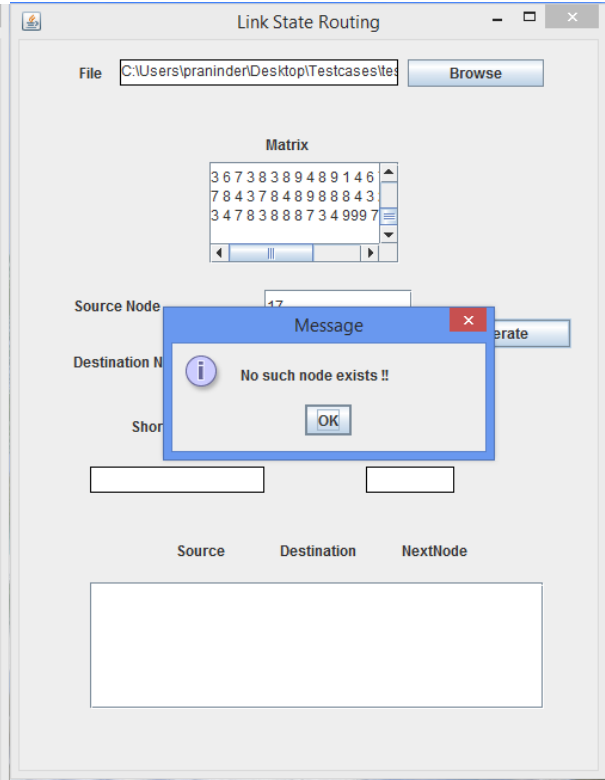
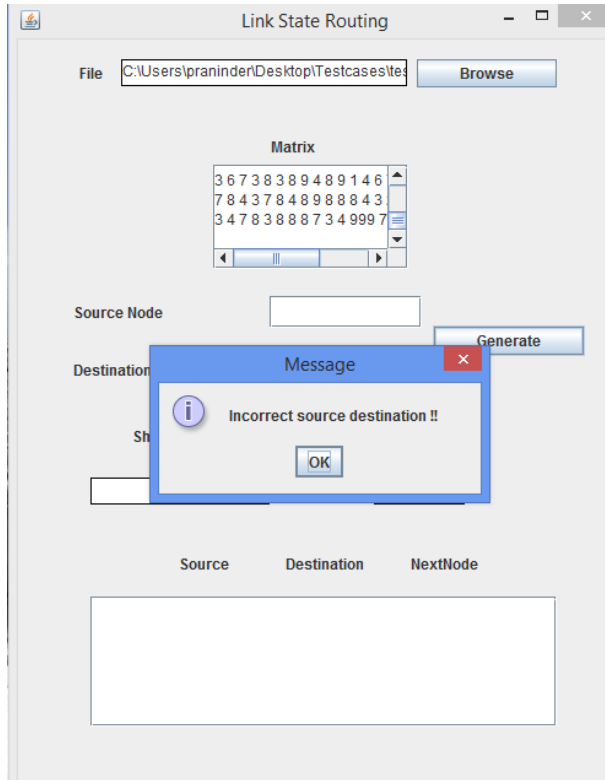
Shortest Path


Optimal Distance

Source	Destination	NextNode
7	0	2
7	7	7
7	8	9
7	9	9
7	10	2
7	11	2




Test Cases for 17 x 17 Asymmetric matrix:

	Test Condition	Expected O/P	Actual O/P
Test Case 1:	Src=16, Des=2	Path=16-4-2,Distance=5	Path=2-3
Test Case 2:	Src=19, Des=14	Error must be handled	No such node exists
Test Case 3:	Src= ,Des=	Error must be handled	Incorrect source destination
Test Case 4:	Src=16,des=16	Path=16-16,Des=0	Path=16-16,Des=0





Link State Routing



File

C:\Users\praninder\Desktop\Testcases\tes

Browse

Matrix

3	6	7	3	8	3	8	9	4	8	9	1	4	6
7	8	4	3	7	8	4	8	9	8	8	8	4	3
3	4	7	8	3	8	8	8	7	3	4	9	9	7

Source Node

16

Generate

Destination Node

16

Shortest Path

16-16

Optimal Distance

0

Source	Destination	NextNode
16	11	14
16	12	14
16	13	13
16	14	14
16	15	15
16	16	16

Link State Routing

File

C:\Users\praninder\Desktop\Testcases\tes

Browse

Matrix

3	6	7	3	8	3	8	9	4	8	9	1	4	6
7	8	4	3	7	8	4	8	9	8	8	8	4	3
3	4	7	8	3	8	8	8	7	3	4	9	9	7

Source Node

16

Generate

Destination Node

2

Shortest Path

16-4-2

Optimal Distance

5

Source	Destination	NextNode
16	12	14
16	13	13
16	14	14
16	15	15
16	16	16

References

[1]Dijkstra's original paper:

E. W. Dijkstra. (1959) A Note on Two Problems in Connection with Graphs. Numerische Mathematik, 1.269-271.

[2]MIT OpenCourseware, 6.046J Introduction to Algorithms.

< <http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/CourseHome/>> Accessed 4/25/09

[3]Department of Mathematics, University of Melbourne. Dijkstra's

Algorithm.<<http://www.ms.unimelb.edu.au/~moshe/620-261/dijkstra/dijkstra.html> > Accessed 4/25/09

[4]http://en.wikipedia.org/wiki/Linkstate_routing_protocol

[5]https://www.google.com/?gws_rd=ssl

[6]<https://docs.oracle.com/javase/tutorial/deployment/applet/getStarted.html>

[7] <http://reference.wolfram.com/language/Combinatorica/ref/Dijkstra.html>

[8]<http://www.danscourses.com/CCNA-2/link-state-routing-protocols.html>