CS542 Project

Link State Routing using Dijkstra's Algorithm Simulation

Members:

Raja Sekhar Reddy Venna (A20345234)

Manas Guduri (A20345155)

Sai Ravali Nunnuru (A20354346)

Gautam Mishra (A20345311)

INTRODUCTION

Link State Routing Protocol is one of the main classes of Routing Protocols used in **Packet Switching** networks for computer communications. Links State protocols include Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (IS-IS).

LOW LEVEL DESIGN

The link state protocol is performed by every switching node in the network, means the nodes that are prepared to forward packets, are so called routers. The basic concept of link state routing is that every node constructs a map of connectivity to the network, in the form of a graph, showing which nodes are connected to which nodes. Each node then independently calculates the next best logical path from it to every possible destination in the network.

Link state protocols, Also called as distributed database protocols are built around a well-known algorithm from graph theory, **Dijkstra's Shortest Path Algorithm.**

Implementation:

The main goals of the implementation of Link State Routing Protocol are as follows.

- Getting all the shortest path(s) from a given source node to destination node with the given network topology (as specified in adjacency matrix).
- Generating routing table (connection table) for each node (router) in the network.
- Using Dijkstra's algorithm to obtain the shortest path as well as the direction between two nodes.
- Simulating the points mentioned above using Graphical User Interface.
- Functionality to modify an edge i.e. modifying link weight of a existing edge between two nodes and also to add a new link/edge between two nodes. In both the cases, the shortest path(s) has to be updated.
- Functionality to add a node to the existing topology and consequently find a new connection table, and shortest path(s) from a source node to a destination node.
- Display all the possible paths between two directly connected nodes.

Technologies Used:

Java is used for implementation of the core algorithm and Java Swings has been used to build graphical representation of the project.

Representing the network topology:

We have used 2-D matrix array to store the original network topology table. Whenever a node is added or removed and a link is added or modified, we make a change to this 2-D matrix. This matrix is only used for displaying purpose.

We perform routing and all other operations using a HashMap structure which stores the network topology. Each entry in the HashMap structure is a <Key, Value> pair where Key is name of the node and Value is the Vertex object of that node. This will be better understood after reading the modules section as mentioned below.

Algorithm:

Dijkstra's Algorithm calculates the shortest path from the source to each of the remaining vertices in the graph. For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, if the nodes of the graph represent routers and edge path costs represent driving distances between pairs of routers connected by a direct link, Dijkstra's algorithm can be used to find the shortest route between one routers. As a result, the shortest path algorithm is widely used in network routing protocols, most notably IS-IS and Open Shortest Path First (OSPF).

The Pseudo code for the Dijkstra's Algorithm implemented by us is as follows.

```
Graph := Set of all nodes in the network.
function route(source):
      V := Graph.get(source)          // Retrieve the source node from the graph
      V.distance = 0;                 // Initialize distance to 0; start of path

      Q : = new Queue()               // Initializing the queue
      add V to Q
      previous := undefined

      while Q is not Empty:           // Main Loop
            u := node in Q with smallest distance

            for each neighbor X of U:
                  alt := distance of U + distance between U and V
                  remove U from Q

                  if (alt < distance of X):       // Relax (U,X)
                        distance of V := alt
                        previous of V := U
                        add X to the previous paths of U
                  else if (alt = distance of X):
                        add X to the previous paths of U
```

Modules:

The modules of the project are as follows -

1. **Algorithm**
   a. ***Edge:*** This class defines an edge between two nodes. It stores the distance between two nodes.
   b. ***Vertex:*** This class represents a node in the network. It also stores the information of its neighboring nodes as well as the cost in a path up to that node.
   c. ***Graph:*** This sub-module is responsible for reading the text file containing the network topology in matrix information and building the adjacency matrix and the edges (Edge object) for the network.
   d. ***Dijkstras:*** This is the core sub-module of the Algorithm module. It maintains graph information and is responsible for performing routing and getting all the shortest paths between two nodes as well as getting all the paths between two directly connected nodes.
2. **Design**

a. ***Main Window:*** This sub-module has all the code to display the GUI for the project. It makes use of Java Swings to display appropriate GUI components. This sub-module is responsible for accepting user input in all forms and displaying results. It makes use of the Algorithm module to calculate the shortest paths, print routing tables and other required information.
b. ***Graph Stream:*** This sub-module displays the network graph and paths in a natural way is in graph of connected nodes.

HIGH LEVEL DESIGN

GUI Design:

Swing is a GUI widget toolkit for Java. It is part of Oracle's Java Foundation Classes (JFC) — an API for providing a graphical user interface (GUI) for Java programs.

- It provides a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT).
- It provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform.
- It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.
- Swing is light-weight and platform-independent. Additionally, this framework provides a layer of abstraction between the code structure and graphic presentation of a Swing-based GUI.

Graph Stream:

Graph Stream is a graph handling Java library that focuses on the dynamics aspects of graphs. Its main focus is on the modeling of dynamic interaction networks of various sizes. The goal of the library is to provide a way to represent graphs and work on it. We have Graph Stream to display the nodes and edges of the network in a natural way.

Test Cases:

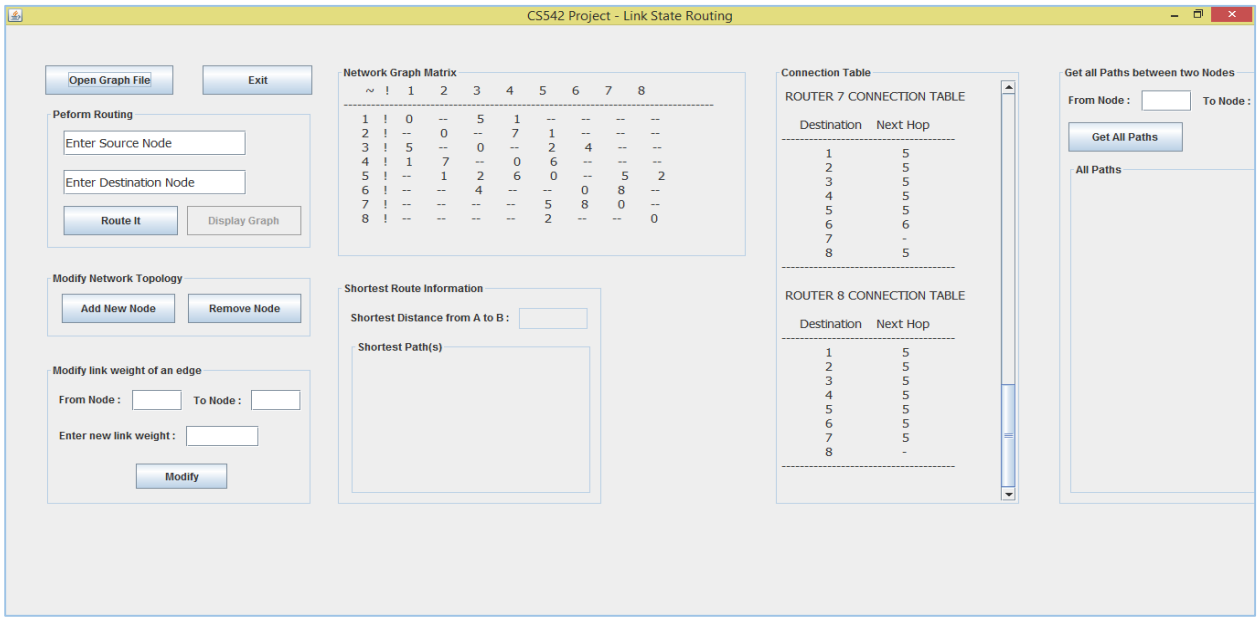
The tests were performed on a graph of 8x8 matrix which is as follows –

0	-1	5	1	-1	-1	-1	-1
-1	0	-1	7	1	-1	-1	-1
5	-1	0	-1	2	4	-1	-1
1	7	-1	0	6	-1	-1	-1
-1	1	2	6	0	-1	5	2
-1	-1	4	-1	-1	0	8	-1
-1	-1	-1	-1	5	8	0	-1
-1	-1	-1	-1	2	-1	-1	0

Test Case 1:

Test Case for loading 8*8 matrix

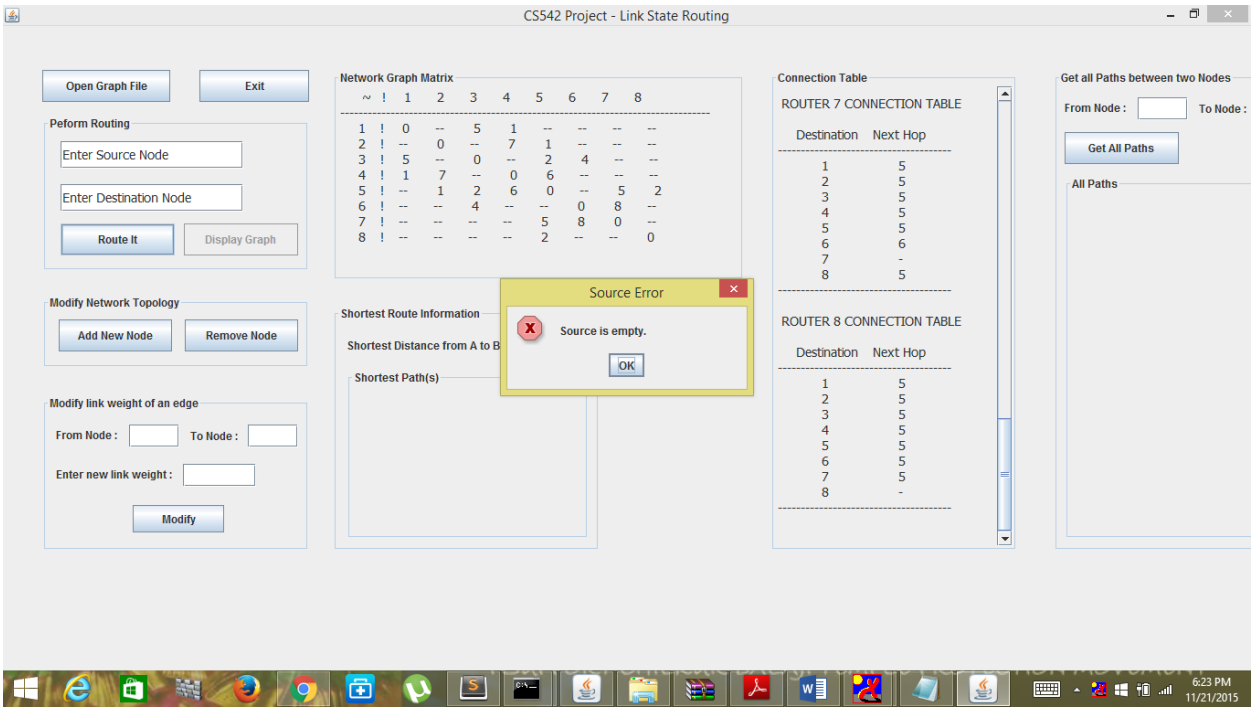
	Test Condition	Expected O/P	Actual O/P	Result
Test Case 1	Loading the text file as an input	Should read the text file, load the graph and display matrix in the Network Graph Matrix block and all the buttons should be enabled. Connection Table of all the nodes should be displayed in Connection Table block.	File loaded successfully and all the buttons are enabled. Matrix is displayed in the Network Graph Matrix block. Connection Table of all the nodes is displayed in the Connection Table block.	Pass



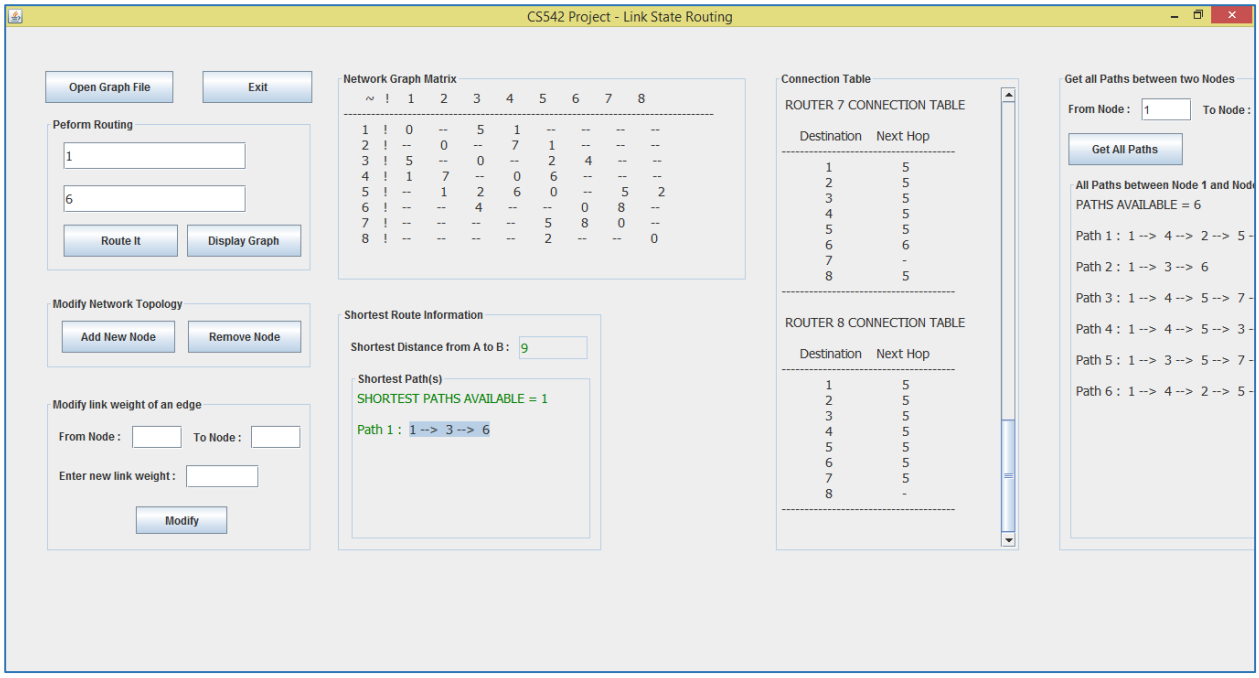
Test Case 2:

Finding the shortest path from Source to Destination

	Test Condition	Expected O/P	Actual O/P	Result
Test Case 1	To check if Source and Destination is not specified and we click on Route IT button	It should display an Error Message.	It displays an Error Message saying Source is Empty.	Pass
Test Case 2	Finding the shortest path from 1st node to 6th	It should display the path as 1 --> 3 --> 6	The path shown is 1 --> 3 --> 6	Pass



Test Case 1



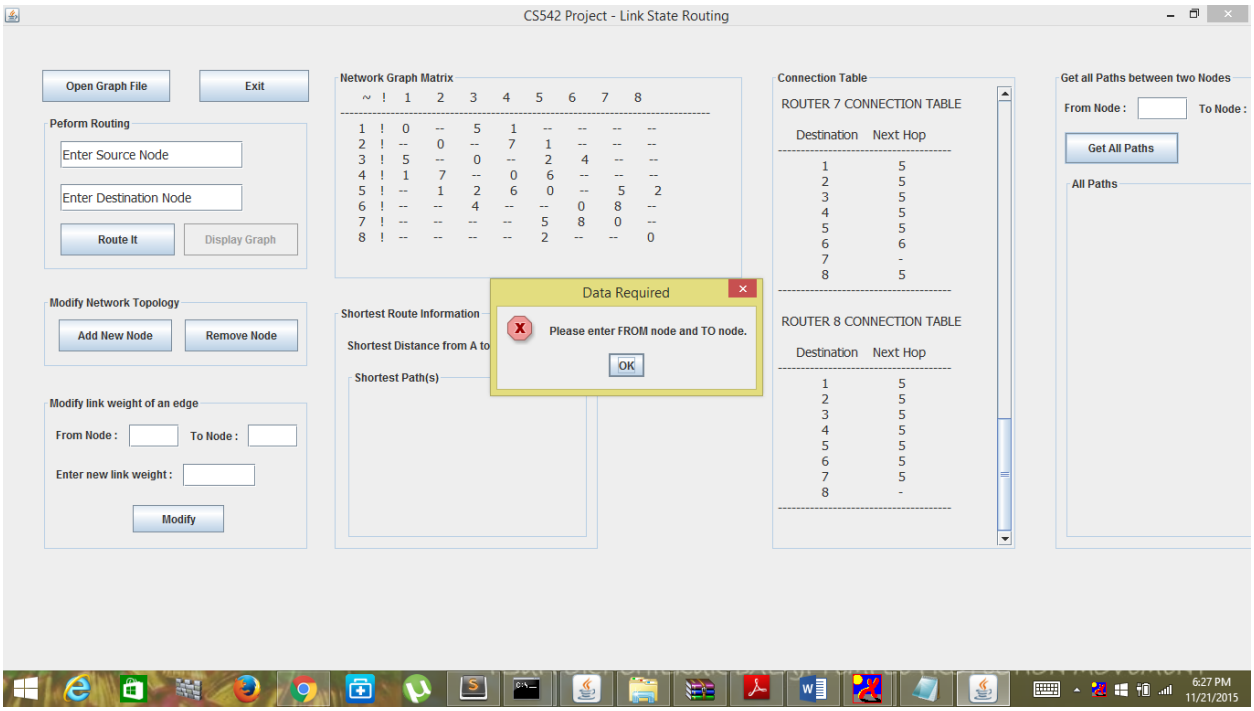
Test Case 2

Test Case #3

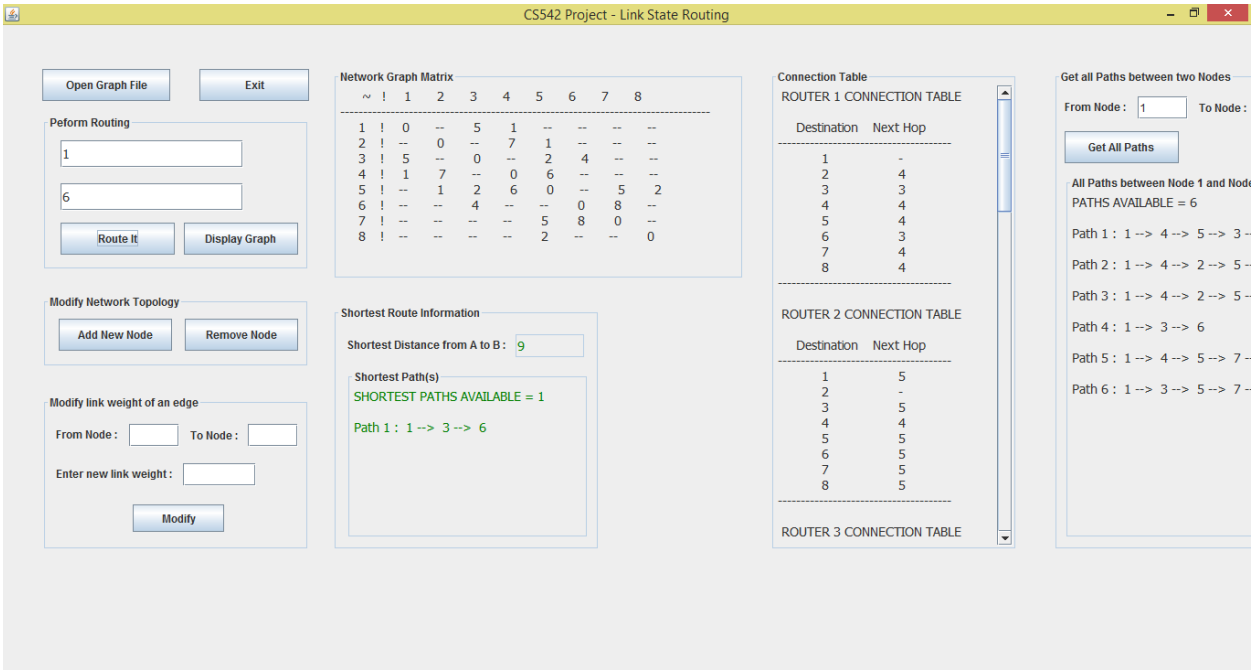
Showing all possible paths from a given Source to Destination

	Test Condition	Expected O/P	Actual O/P	Result
Test Case 1	To check if From node and To node is not specified and we click on Get All Paths.	It should display an Error Message.	It displays an Error Message saying From node or To node is empty.	Pass
Test Case 2	Finding the all possible paths from 1st node to 6th. Not necessarily the shortest path	It should display the path as Path 1 : 1 --> 4 --> 2 --> 5 --> 3 --> 6 Path 2 : 1 --> 3 --> 6 Path 3 : 1 --> 4 --> 5 --> 7 --> 6 Path 4 : 1 --> 4 --> 5 --> 3 --> 6 Path 5 : 1 --> 3 --> 5 --> 7 --> 6 Path 6 : 1 --> 4 --> 2 --> 5 --> 7 --> 6	It displays all the path as Path 1 : 1 --> 4 --> 2 --> 5 --> 3 --> 6 Path 2 : 1 --> 3 --> 6 Path 3 : 1 --> 4 --> 5 --> 7 --> 6 Path 4 : 1 --> 4 --> 5 --> 3 --> 6 Path 5 : 1 --> 3 --> 5 --> 7 --> 6 Path 6 : 1 --> 4 --> 2 --> 5 --> 7 --> 6	Pass

It displays all the paths in the "Get All Paths between Two Nodes" panel towards right end.



Test Case 1

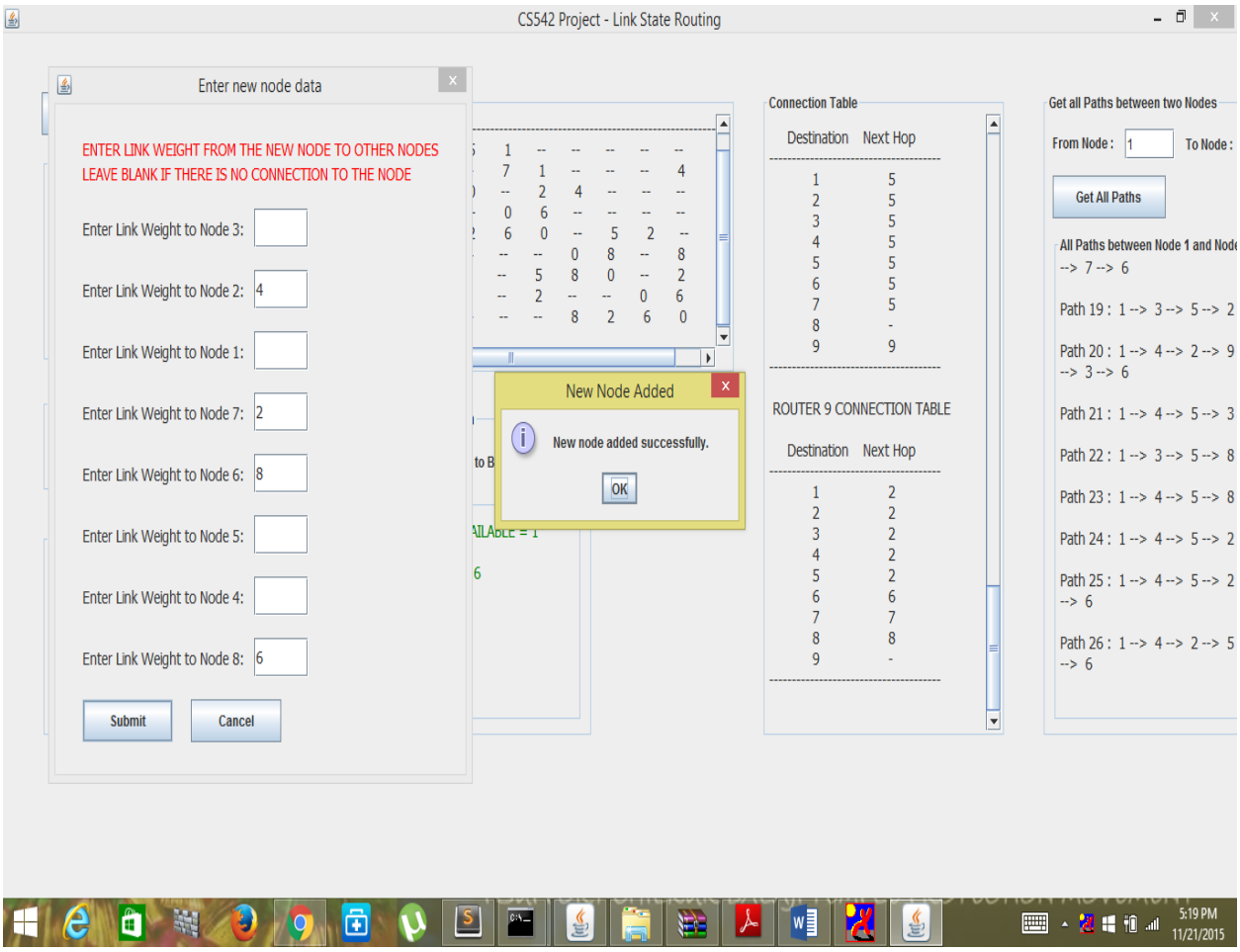


Test Case 2

Test Case #4:

Adding a new node to the existing topology.

	Test Condition	Expected O/P	Actual O/P	Result
Test Case 1	Adding new Node to the topology.	It should display the new node in the Network Graph Matrix with weights associated with connected nodes. It modifies all the paths between source and destination if, new node effects the existing paths.	New node added successfully and modifies the Network Graph. All the paths available are also changed.	Pass



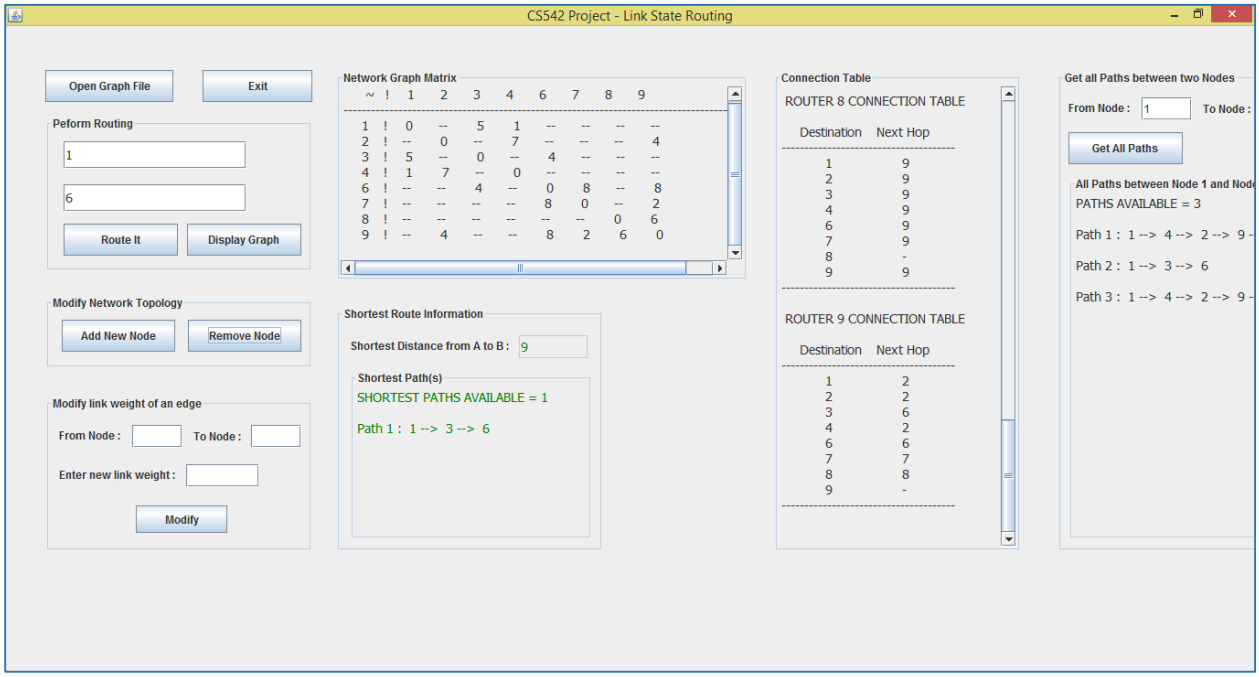
There are 26 paths available between 1st and 6th after adding 9th node to the network.

Network Graph Matrix is also modified as we can observe in the above screenshot.

Test Case #5:

Removing a Node from the existing topology

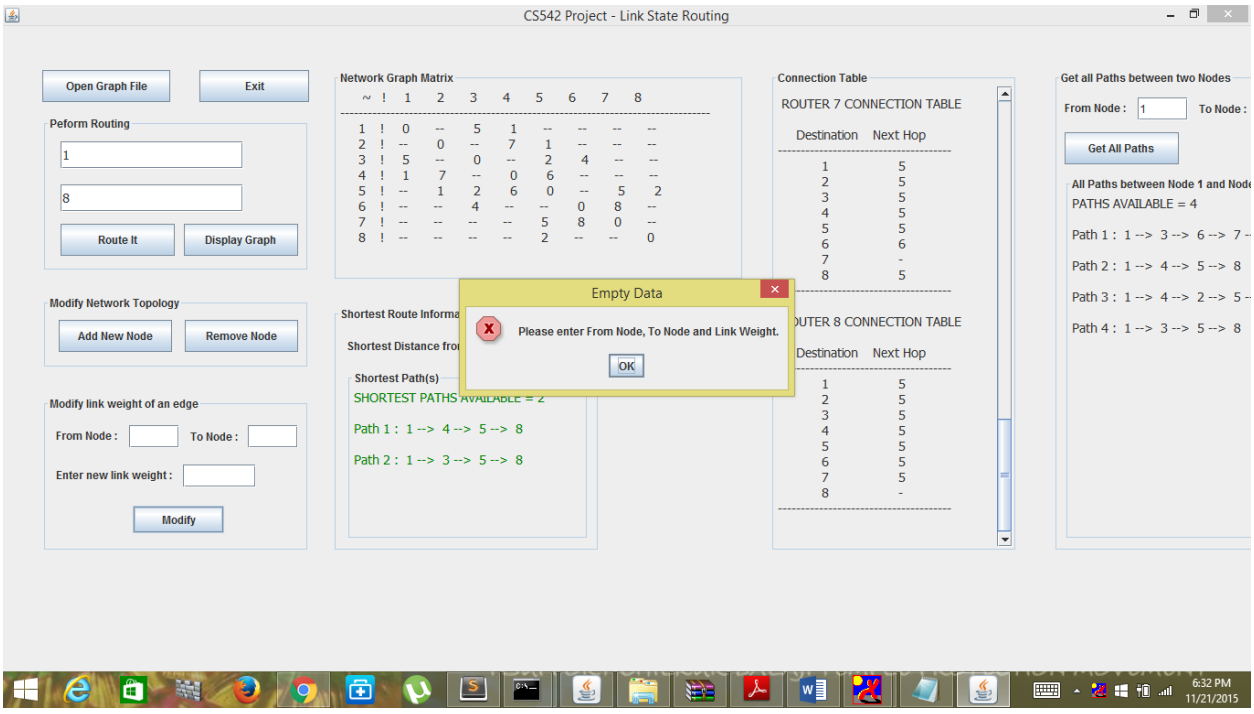
	Test Condition	Expected O/P	Actual O/P	Result
Test Case 1	Removing a Node from the existing topology.	It should removes the node in the Network Graph Matrix with weights associated with connected nodes. It may modify all the paths between source and destination, if they are associated with the removing node	Node removed successfully and modifies the Network Graph. All the paths available are also changed.	Pass



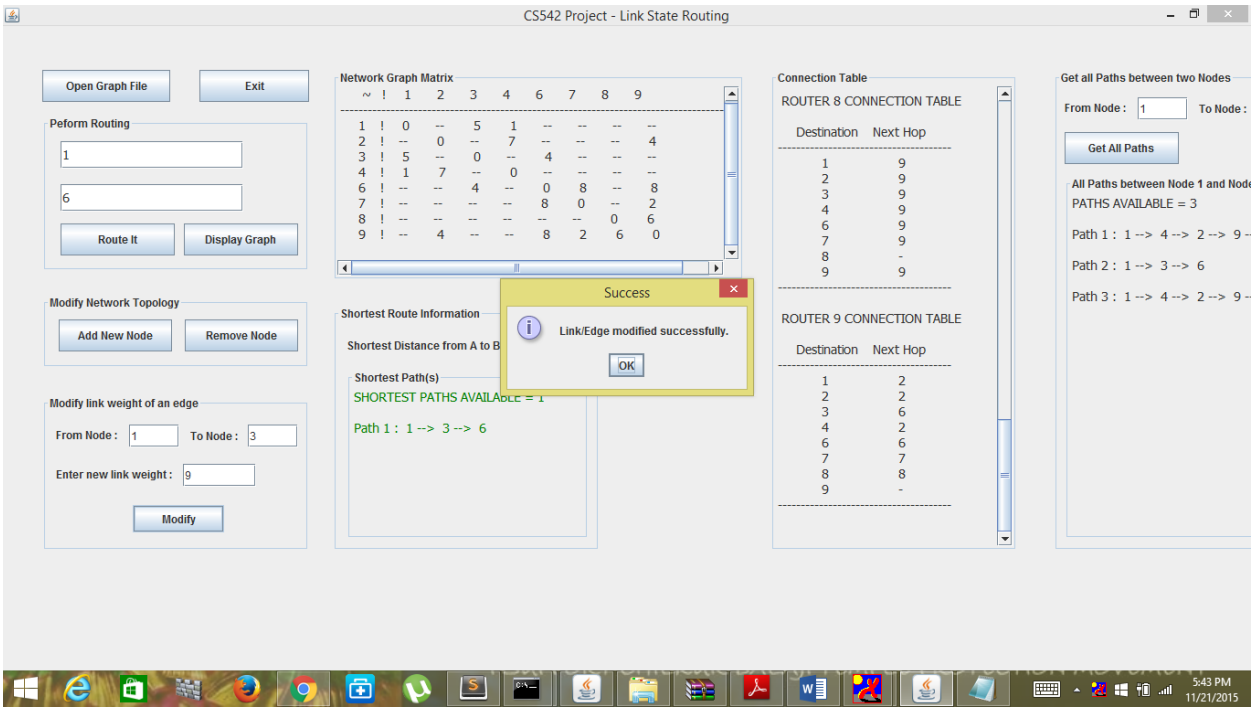
Test Case #6:

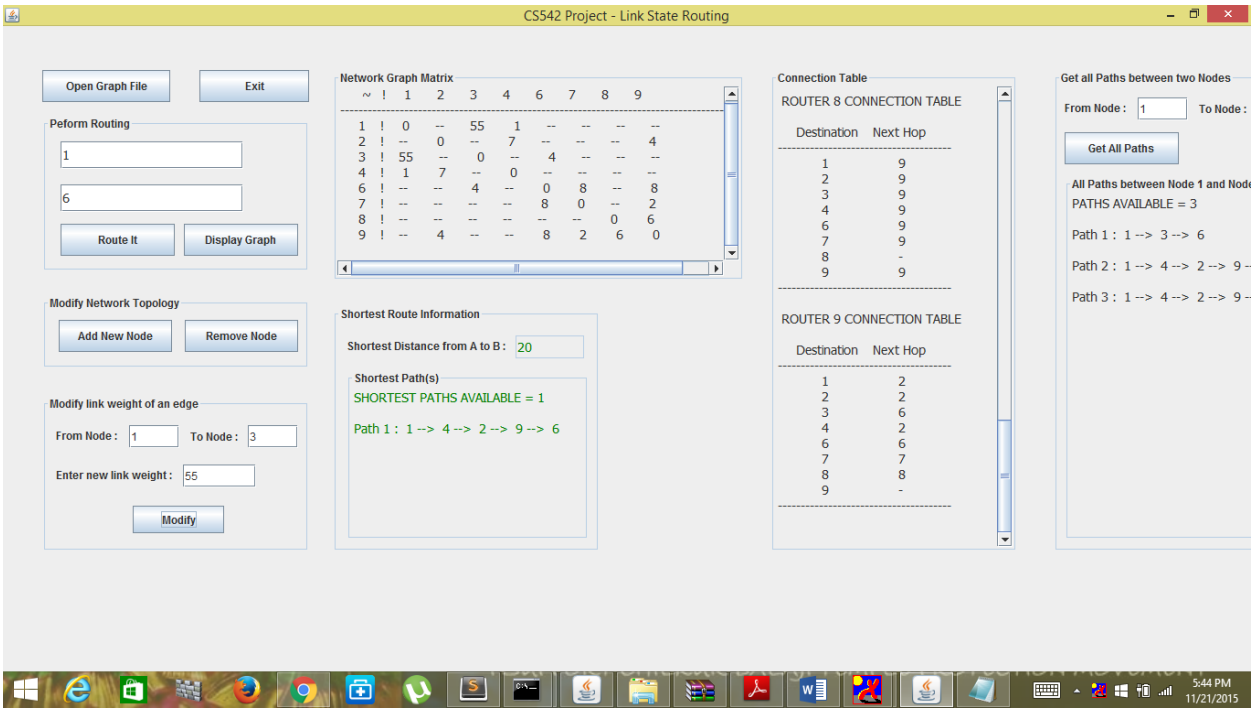
Modify the link weight between two nodes

	Test Condition	Expected O/P	Actual O/P	Result
Test Case 1	Click on Modify button without entering From, To node and link weight.	It should give an error message.	It gives an error message.	Pass
Test Case 2	Modifying the existing weight between two nodes	It should say that edge modified successfully and should modifies the node weight in the Network Graph Matrix It may modify all the shortest paths between source and destination, if they are associated with the modifying node	Message "Node weight modified successfully" and modifies the Network Graph. Shortest path have been changed.	Pass



Test Case 1


CS542 Project - Link State Routing
Open Graph File
Exit
Peform Routing
1
6
Route It
Display Graph
Modify Network Topology
Add New Node
Remove Node
Modify link weight of an edge
From Node : 1
To Node : 3
Enter new link weight : 9
Modify
Network Graph Matrix
Shortest Route Information
Shortest Distance from A to B
Shortest Path(s)
SHORTEST PATHS AVAILABLE = 1
Path 1 : 1 --> 3 --> 6
Success
Link/Edge modified successfully.
OK
Connection Table
ROUTER 8 CONNECTION TABLE
Destination Next Hop
ROUTER 9 CONNECTION TABLE
Destination Next Hop
Get all Paths between two Nodes
From Node : 1
To Node :
Get All Paths
PATHS AVAILABLE = 3
Path 1 : 1 --> 4 --> 2 --> 9 --
Path 2 : 1 --> 3 --> 6
Path 3 : 1 --> 4 --> 2 --> 9 --
5:43 PM
11/21/2015

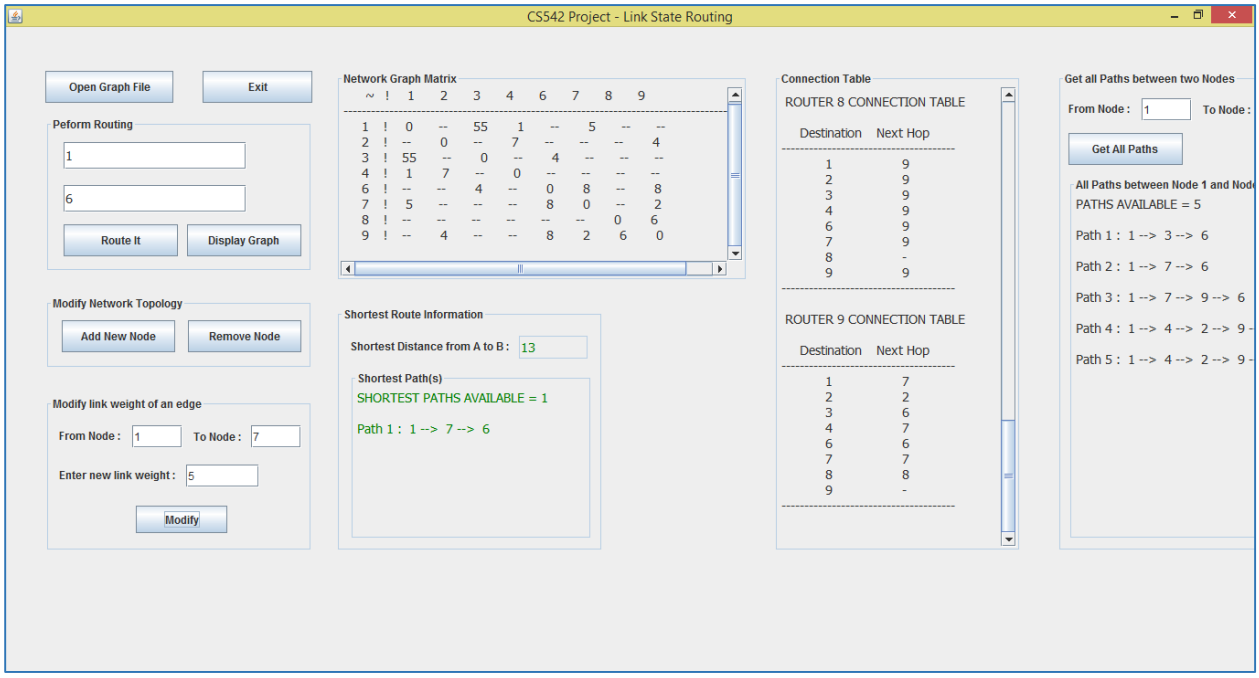

CS542 Project - Link State Routing
Open Graph File
Exit
Peform Routing
1
6
Route It
Display Graph
Modify Network Topology
Add New Node
Remove Node
Modify link weight of an edge
From Node : 1
To Node : 3
Enter new link weight : 55
Modify
Network Graph Matrix
Shortest Route Information
Shortest Distance from A to B : 20
Shortest Path(s)
SHORTEST PATHS AVAILABLE = 1
Path 1 : 1 --> 4 --> 2 --> 9 --> 6
Connection Table
ROUTER 8 CONNECTION TABLE
Destination Next Hop
ROUTER 9 CONNECTION TABLE
Destination Next Hop
Get all Paths between two Nodes
From Node : 1
To Node :
Get All Paths
PATHS AVAILABLE = 3
Path 1 : 1 --> 3 --> 6
Path 2 : 1 --> 4 --> 2 --> 9 --
Path 3 : 1 --> 4 --> 2 --> 9 --
5:44 PM
11/21/2015

Test Case 2

Test Case #7:

Test case for adding new link between two nodes.

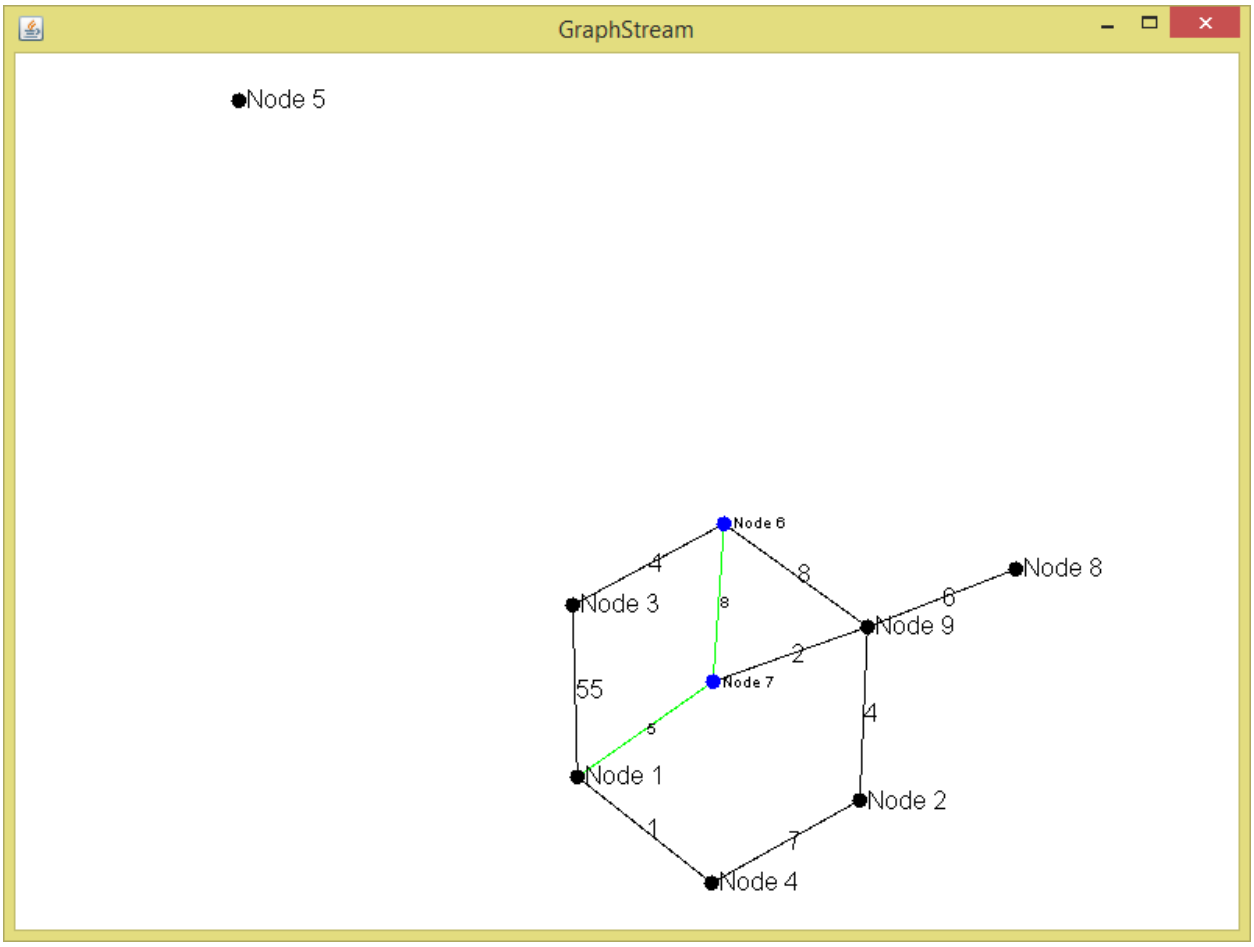
	Test Condition	Expected O/P	Actual O/P	Result
Test Case 1	Adding the weight between two nodes	It should modifies the node weight in the Network Graph Matrix It may modify all the paths and possibly the shortest path between source and destination, if they are associated with the modifying node	Node weight modified successfully and modifies the Network Graph. Shortest path and all the paths between two nodes have been changed.	Pass



Test Case #8

Displaying the Graph Stream for the Shortest Path

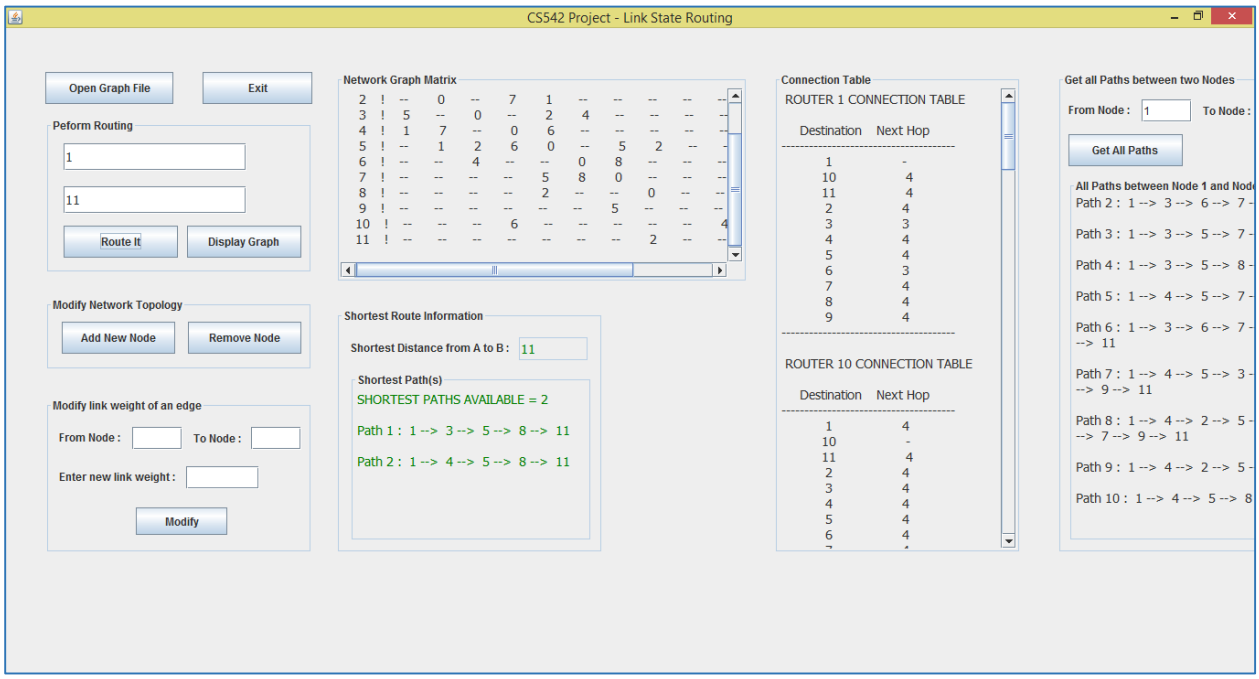
	Test Condition	Expected O/P	Actual O/P	Result
Test Case	After entering Source and Destination node and clicking on Route It button, click on Display Graph.	A new window should appear showing the graph.	A new window is shown which displays the graph and the path.	Pass



Test Case #9:

Loading 15x15 matrix to the interface

	Test Condition	Expected O/P	Actual O/P	Result
Test Case	Load the 15*15 matrix to the interface. Enter Source node and destination node and click on Route It.	Matrix should be displayed in Network Graph Matrix. Routing should be done and Shortest Paths should be displayed.	Matrix loaded to the interface successfully. Shortest paths are displayed between node 1 and node 11.	Pass



Project Scheduling:

CS542 Project — Project Schedule

	Start Week	Nov 1, 2015

Week	1	2	3	4	Notes
Starting	Nov 1	Nov 5	Nov 10	Nov 20	
Phase 1 - Design					Initial System Design went for one week
Phase 2 - Coding					Coding for two weeks
Phase 3 - Testing					Documentation and Testing was done simultaneously went for one week.
Phase 4 - Documentation					

Work Load:

Sr. No.	Member	Responsibility
1	Raja Sekhar Reddy Venna A20345234	Implemented Dijkstra's basic algorithm and created a basic GUI design. Added the functionality of modifying/adding a link. Low Level Design Document
2	Manas Guduri A20345155	Implemented Edge.java, Vertex.java, part of Graph.java module High Level Design Document
3	Sai Ravali Nunnuru A20354346	Implemented part of Graph.java module. Added functionality to display graph and all paths between two nodes in GUI. Prepared the Presentation (PPT) and User Manual
4	Gautam Mishra A20345311	Modified the Dijkstra's algorithm to evaluate multiple shortest paths (if exist) between two nodes. Added the functionality of Adding a node and Removing a node. Test Cases

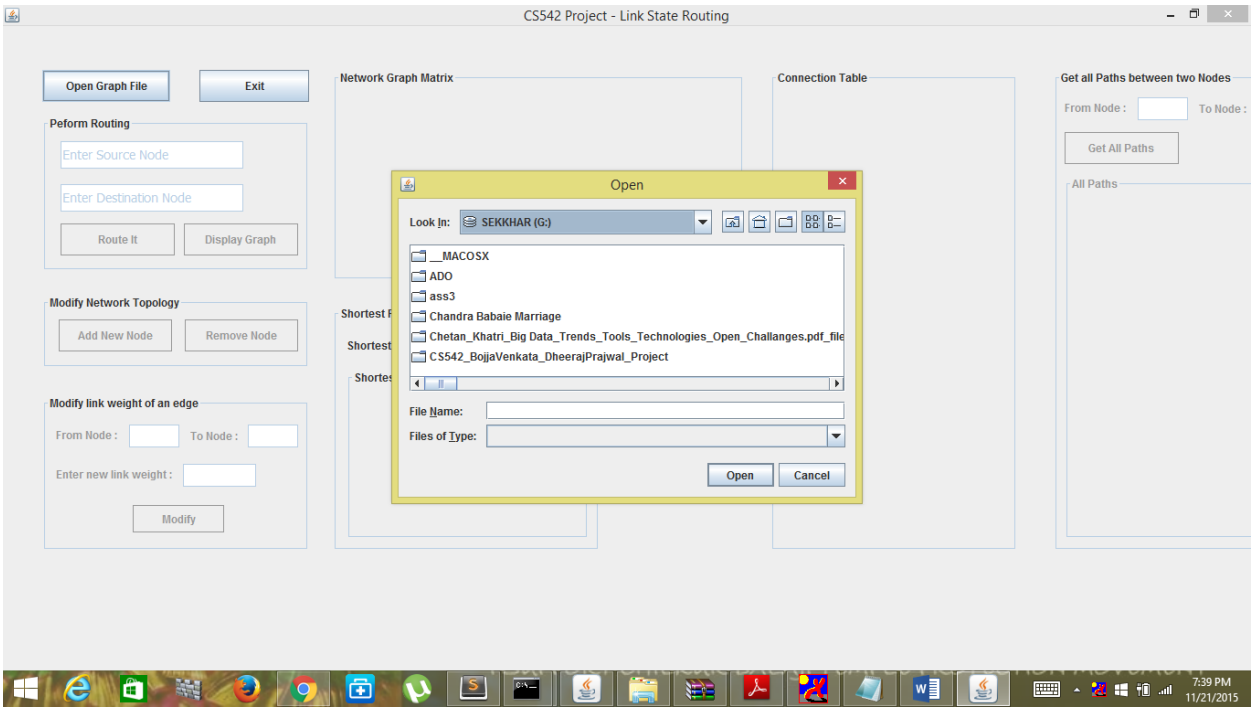
USER MANUAL

The input file containing the network topology must be text file. The file should contain the adjacency matrix representing the network topology. Each row on a new line. Each value in a row should be separated by WHITESPACE. If an edge is not present between two nodes, the distance should be kept -1 else keep it to the required distance. Example is given below:

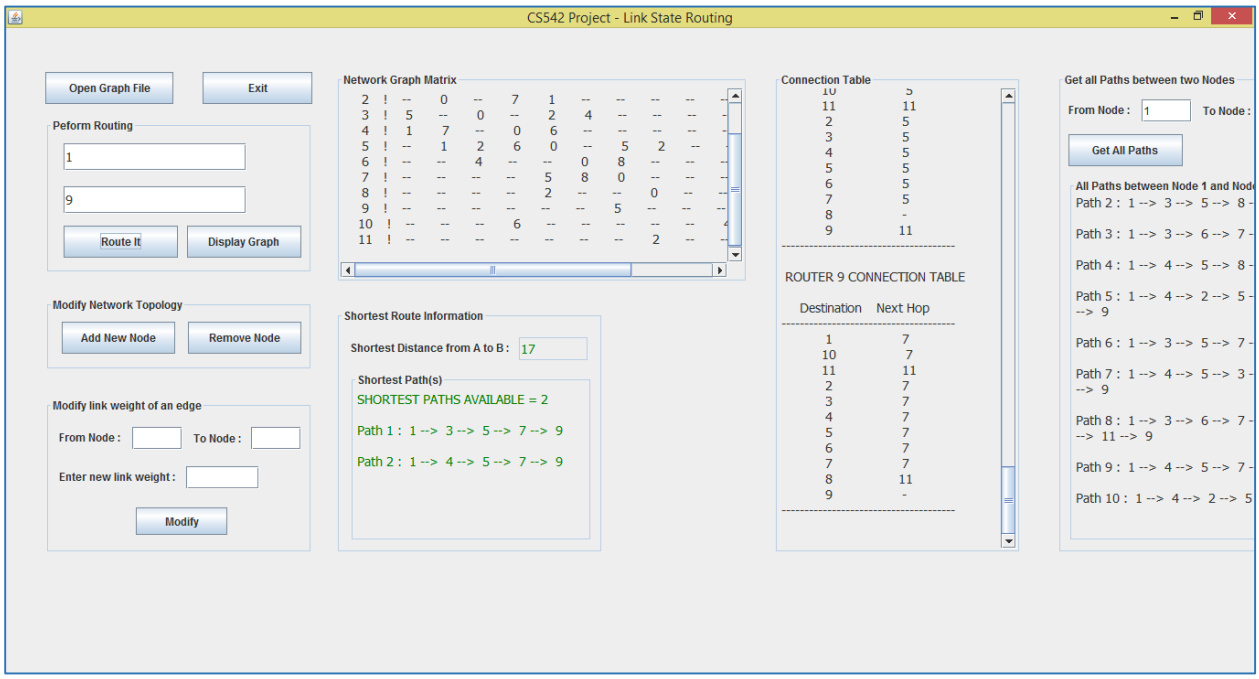
0 -1 5 1 -1 -1 -1 -1
-1 0 -1 7 1 -1 -1 -1
5 -1 0 -1 2 4 -1 -1
1 7 -1 0 6 -1 -1 -1
-1 1 2 6 0 -1 5 2
-1 -1 4 -1 -1 0 8 -1
-1 -1 -1 -1 5 8 0 -1
-1 -1 -1 -1 2 -1 -1 0

Instructions to compile and run:

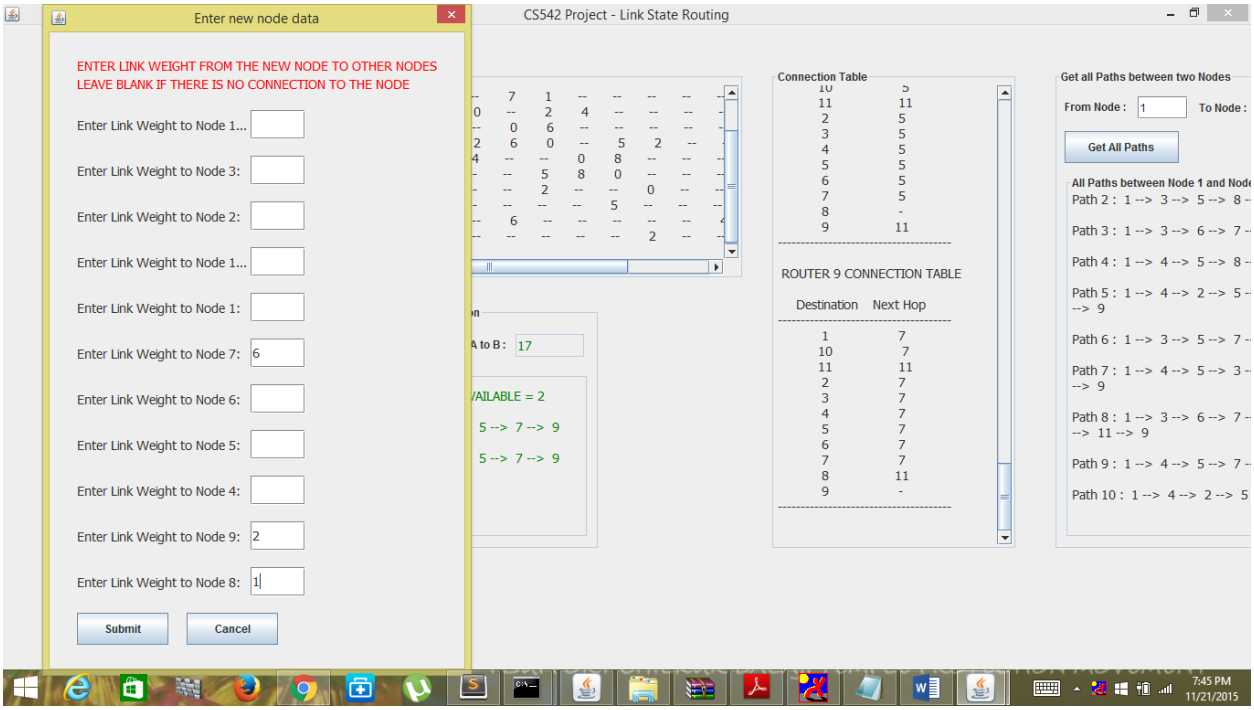
1. Install standard JRE (Java Runtime Environment) software preferably 1.7 version or above and set the path in the environment variables (for windows).
2. After installing and set the path for java, download the jar file.
3. Run the jar file.
4. After executing the jar file, click on "Open Graph File" button to browse and load the appropriate topology file (Input text file).

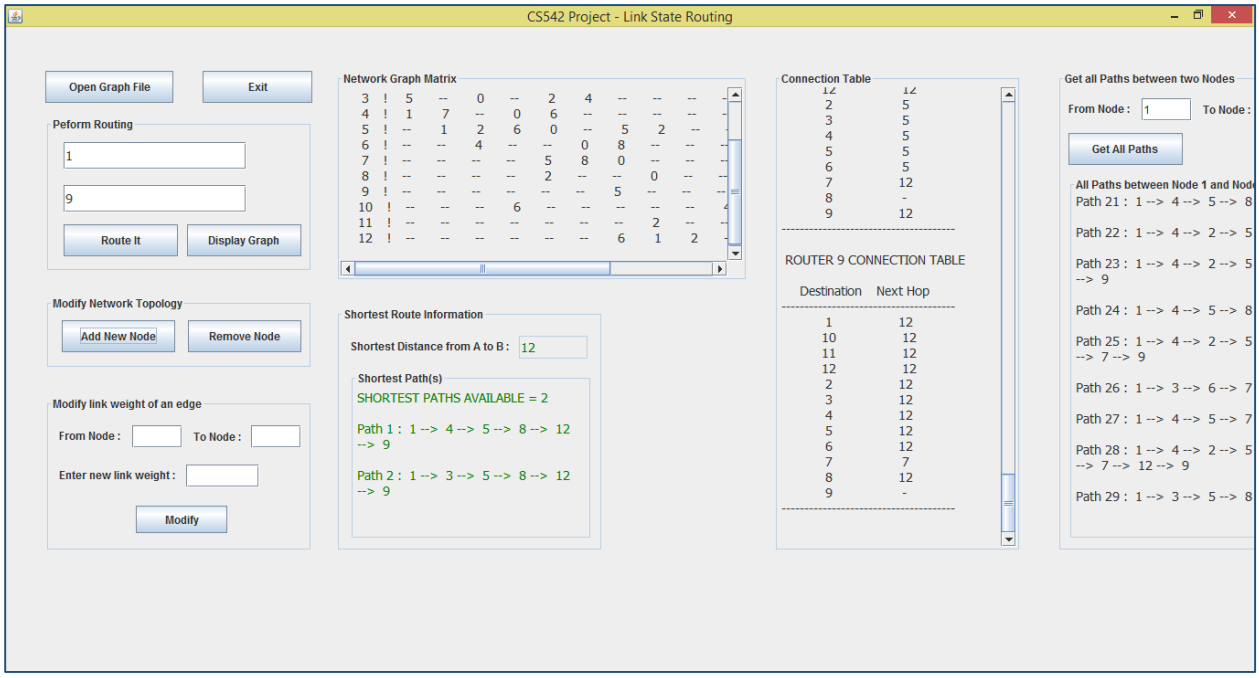


Enter the source and destination and press the "Route It" button to find the shortest path between the nodes and the distance. It will also displays all the possible paths between two nodes and Connection Table for each router



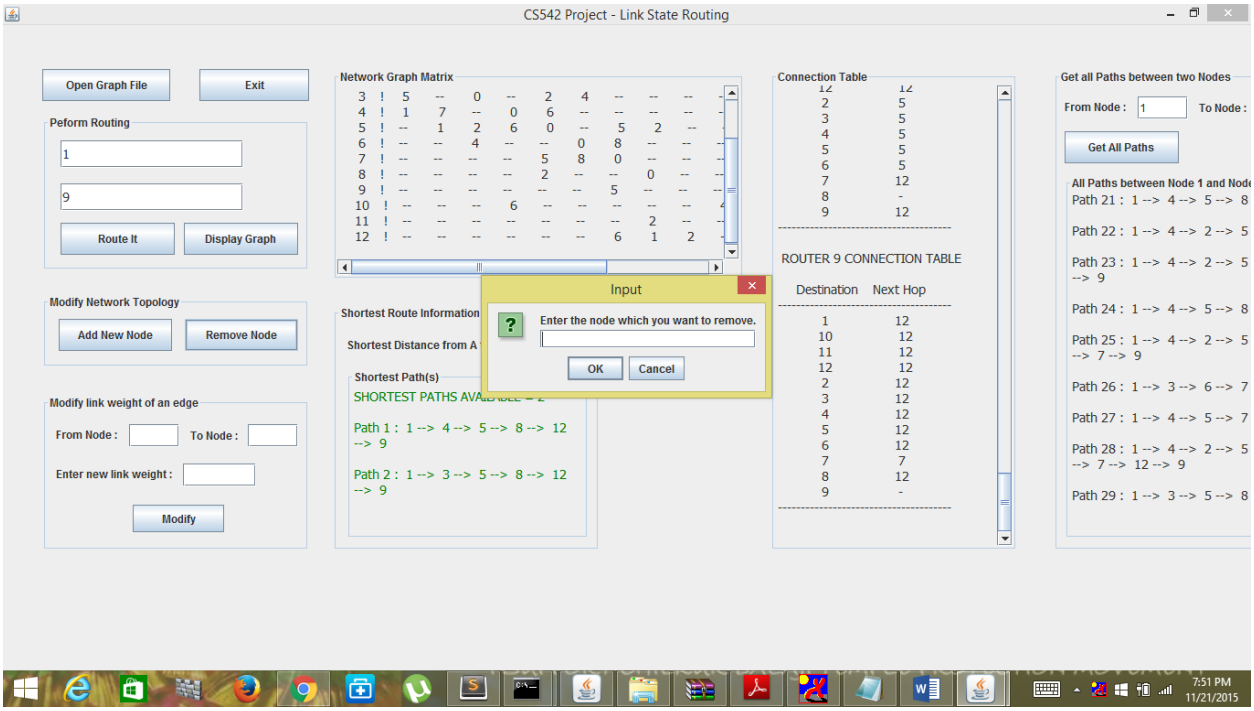
To add a new node for the topology, click on "Add New Node" button, it will pop up a window asking for the link weights between the nodes, once you enter the edge weights, it will add the node to Network Graph Matrix



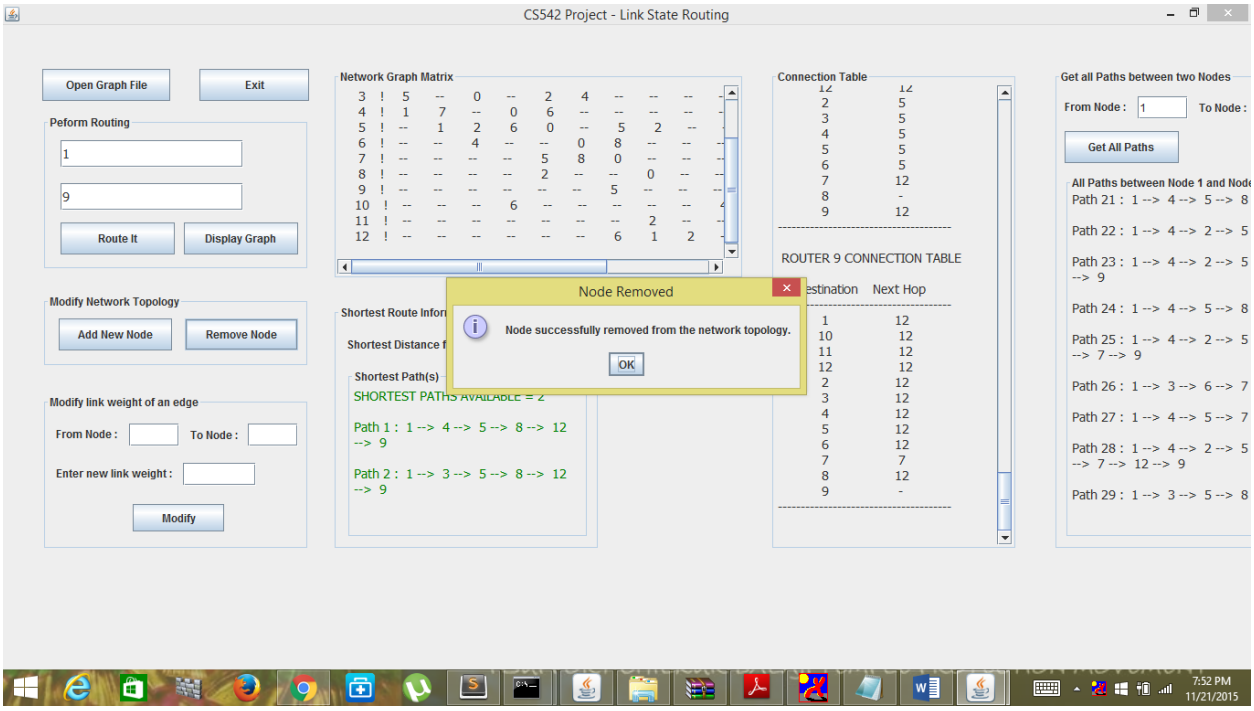


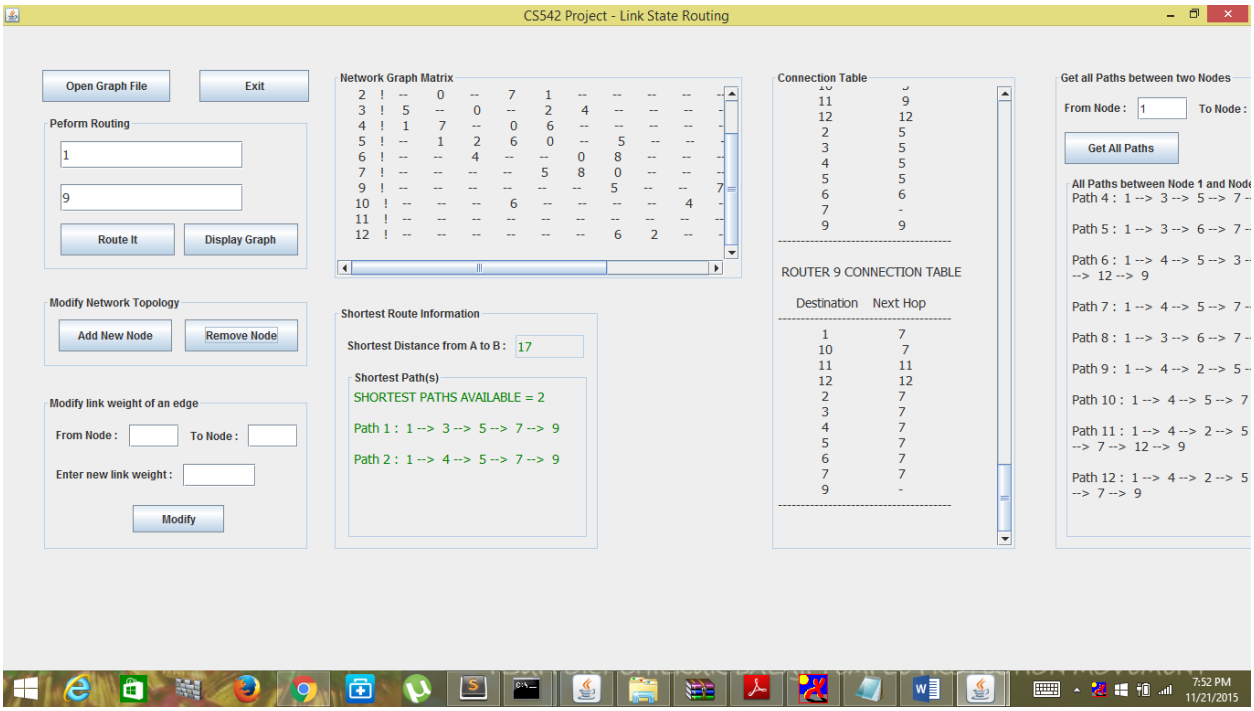
12th node added successfully, as you can see above.

To remove a node from the existing topology, click on Remove Node button, it will ask for which node to be removed



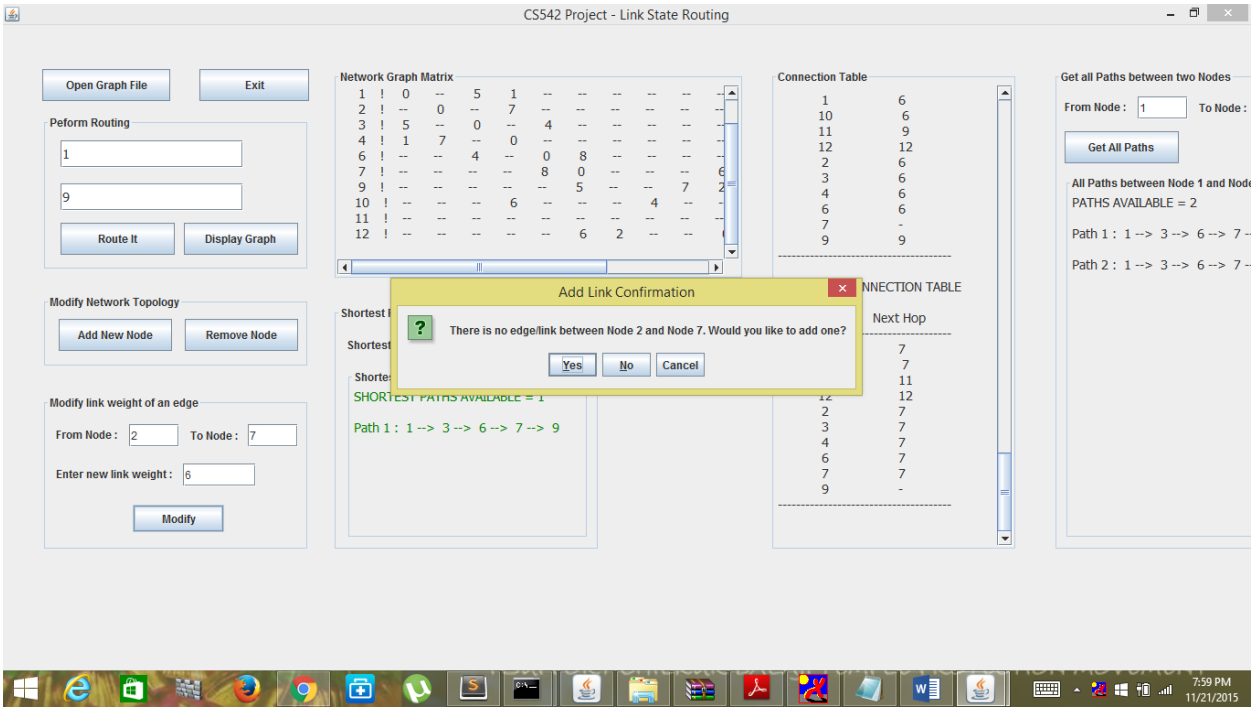
Once you enter the node to be removed, it will remove the node from the Network Graph Matrix

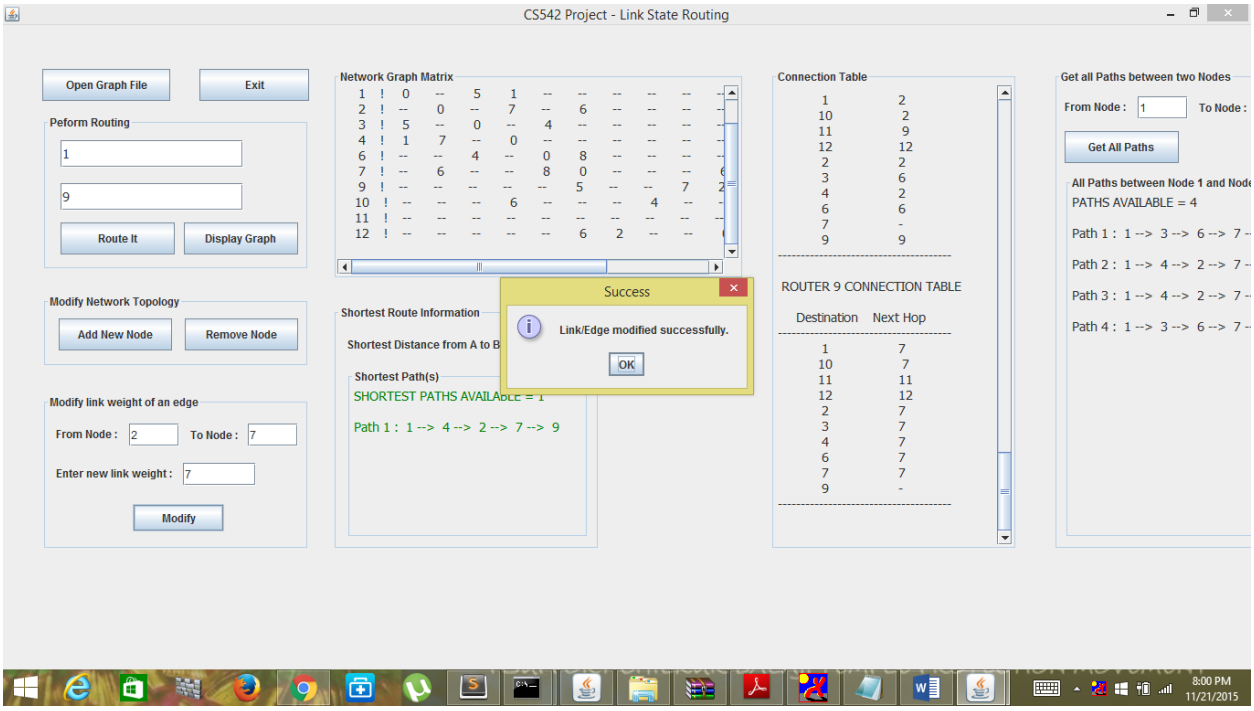




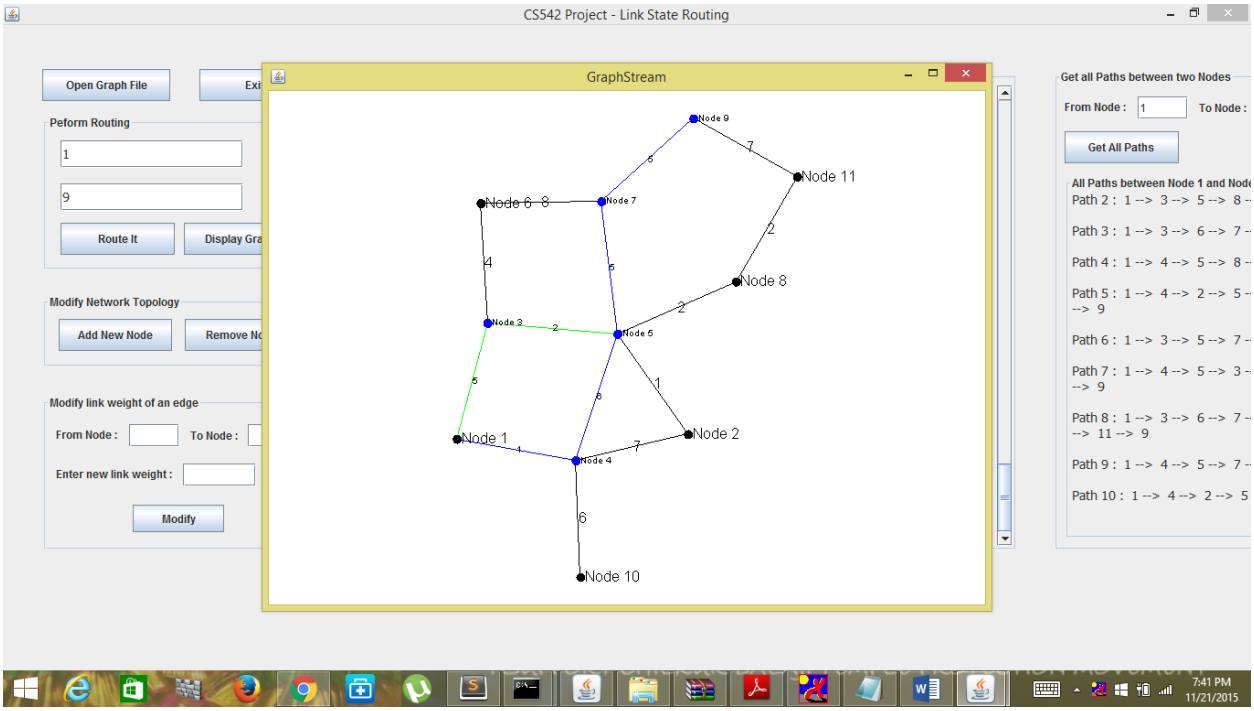
Node 8 has been deleted from the network as you can see above.

To modify a link weight or add new weight to the link (i.e. add a new edge/link), go to modify link weight of an edge and enter the nodes and weight and click on "Modify" button.





Display the Routing path from source node to destination nodes via the next hops (interfaces) using the Graph Stream.



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. http://en.wikepedia.org/wiki/Linkstate_routing_protocol
3. https://docs.oracle.com/javase/tutorial/deployment/swings/getstarted.html
4. https://reference.wolfam.com/language/Combinatorica/ref/Dijkstra.html
5. https://www.danscourses.com/CCNA-2/link-state-routing-protocols.html
6. http://graphstream-project.org/
7. http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html