IMPLEMENTATION OF LINK – STATE ROUTING PROTOCOL

Developed By: Suresh Hegde, Ashwath

1. INTRODUCTION

Link State is a routing protocol which is used in packet switching networks for computer communication. In the network that uses link state, every node will perform link - state; they construct a map of the connection between nodes in network by showing which nodes connected with which other nodes. After that each nodes can independently calculate the best path from it to other nodes in network.

Dijkstra's Algorithm is a graph search algorithm which solves the shortest path for the graph with non - negative edge. The algorithm is also used to find the path with lowest cost between one vertex and every other vertex in one graph.

In this project we develop a program to implement Link - State routing protocol. The program should simulate the process of generating routing tables for each router in a given network and compute optimal path with least cost between any two particular given routers. Dijkstra's algorithm will be used to calculate the direction as well as the shortest path between two routers.

We will use Java to implement the Dijkstra's algorithm. In the code, we use several two - dimensional arrays to store original routing table, the distance between routers, values of distance during shortest path calculation, final table after calculation. Some integer valuable are used for routers counting. The program interface display a menu to user who can choose to load matrix of routing table from file, the program also help users calculate the shortest path between any couple of routers and display it to on the screen. We implement Graphical User Interface to calculate shortest path using Dijkstra's Algorithm.

Routing protocol and its classification:

Routing protocol is response to the demand for dynamic routing tables in network. A routing protocol is a combination of rules and procedures that let routers in the internet inform other routers of change. It helps routers in the network know about the internet of their neighbour by sharing information.

In routing protocol we have interior protocol which handles intra-domain routing and exterior protocol which handles inter-domain routing protocol. Link-state is one of the intra-domain routing protocol.

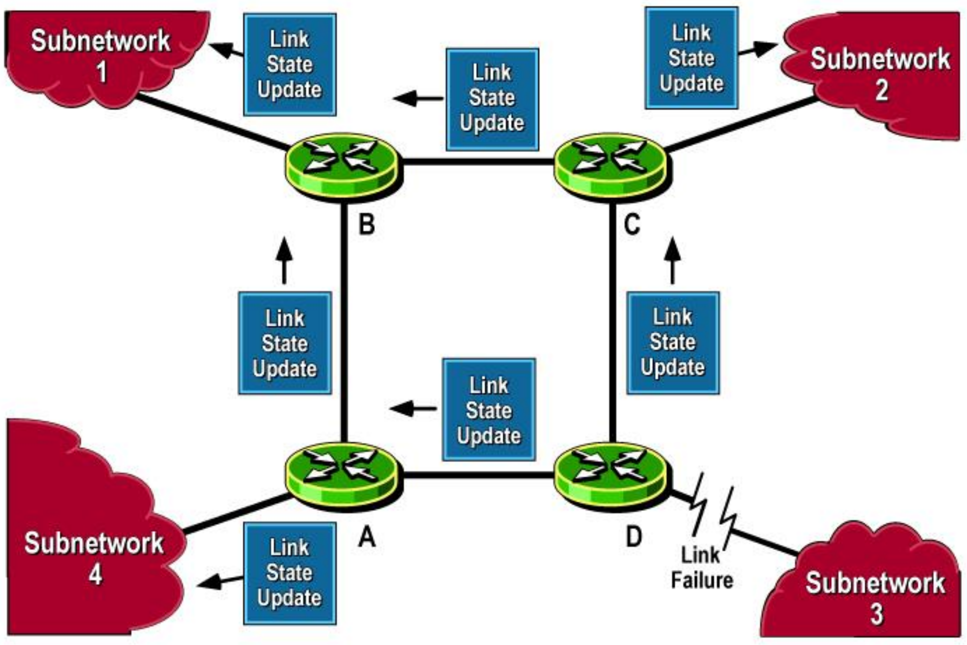


Figure 1: Representation of Link – State Routing Protocol

2. ALGORITHM

The method to calculate the optimal routing table and thus the shortest path using Dijkstra's algorithm and link state in project can be explained as follows:

We have a matrix of routing table that present the original value of metric between a router and its neighbour. At very first stage, each router does not know about connectivity of all others routers, each one just knows about its neighbour and its link, we use non-negative integer to present cost of a link and -1 for indirect connection. Router will uses the link-state routing table through four steps:

(i) Creating the state of link by each nodes,
(ii) Disseminating the link-state packet to other nodes,
(iii) Formatting shortest path for each node,
(iv) Calculating the routing table based on shortest path tree.

After each node has connectivity of other nodes it still does not have shortest path from itself to other one. At this time we use the Dijkstra's algorithm to calculate the shortest path:

(i) From table of n routers we will have n shortest path tree, each node will become the root in its shortest path tree.
(ii) Set the shortest path distance for all root neighbours to the cost between root and neighbour.
(iii) We continue search the nodes in the path select one with minimum shortest distances and add to the path.
(iv) Use the operation: $D_i = minimum(D_i, D_j + c_{i,j})$, we repeat above step until all nodes are added to the path.

(v) While having the shortest path tree we can create the routing table, we use 0 to represent the distance from root to root i.e. for same router say R1 → R1 has cost = 0. On the other hand it has positive cost values for two different nodes.

3. CODE

The code snippets of our Dijkstra project is as shown below.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
   try{
         if(source.getText().equals("")){
              System.out.println("Blank");
         }
         f3.srcRtr= Integer.parseInt(source.getText());
         if (f3.srcRtr <= 0 || f3.srcRtr > f3.b)
         {
            JOptionPane.showMessageDialog(this, "Invalid input : Source router number is
           between 1 and size of matrix","ERROR", JOptionPane.ERROR_MESSAGE);
            source.setText("");
         }
       }
       catch (Exception e) {
            JOptionPane.showMessageDialog(this, "Invalid Input","WRONG CHOICE",
           JOptionPane.OK_CANCEL_OPTION);
            source.setText("");
       }
      val = f3.srcRtr;
      f3.nSrc = 1;
      // coding is based on assumption routers index 0,1,2.., not 1,2,3
      f3.srcRtr = f3.srcRtr - 1;
      // clearing all values for next iteration.
      f3.minValNextPos = 0;
      f3.minValNxt = 0;
      f3.listCurrVal.clear();
      f3.upPar.clear();
      f3.visitArr.clear();
      f3.intrArr.clear();
      // initializing current list array
      addGivenRow(f3.srcRtr);
      // make source node as visited
      f3.visitArr.add(f3.srcRtr);
      // initilize the parent list
      updateParentList(f3.srcRtr);
      // print the Connection table
      printConnectionTable(f3.srcRtr, f3.mat);
      try {
         ct.read(new FileReader("temp.txt"), "here's a description of the file");
      } catch (FileNotFoundException ex) {
         Logger.getLogger(Form4.class.getName()).log(Level.SEVERE, null, ex);
      } catch (IOException ex) {
         Logger.getLogger(Form4.class.getName()).log(Level.SEVERE, null, ex);
      }
      source.setText("");
   }
```

```
public static void addGivenRow(int sourceRouter) {
                for (int j = 0; j < Fram3.b; j++) {
                        if (j == sourceRouter) {
                                Fram3.listCurrVal.add(0);
                        } else {
                                Fram3.listCurrVal.add(1000);
                        }
                }
}

public static void printConnectionTable(int sourceRouter, int[][] matrix) {
                // storing initial source
                Fram3.minValNextPos = sourceRouter;
                while (Fram3.visitArr.size() != Fram3.b) {
                        findNeibours(Fram3.minValNextPos);
                        findWeiAndUpdateList(Fram3.minValNextPos);
                        findMinValInRow();
                        setVisitedNodeInList();
                }
                connectionTableArray(sourceRouter);
                Fram3.getRes = 0;
                while (Fram3.getRes != 1) {
                        printTableValues();
                        checkInterfaceListZeros();
                }
                printTable();
}

public static void printTable()  {
                PrintWriter writer = null;
      try {
        writer = new PrintWriter("temp.txt", "UTF-8");
        writer.println("\tRouter "+ val +" Connection Table\n");
        writer.println("\n          DESTINATION" + "\t" + "            INTERFACE");;
        writer.println("         ============" + "\t" + "          ============");
        for (int k = 0; k < Fram3.b; k++)
        {

          writer.println("\n\t" + (k + 1) + "\t\t" + Fram3.intrArr.get(k));
        }
        writer.close();
      } catch (FileNotFoundException ex) {
        Logger.getLogger(Form4.class.getName()).log(Level.SEVERE, null, ex);
      } catch (UnsupportedEncodingException ex) {
        Logger.getLogger(Form4.class.getName()).log(Level.SEVERE, null, ex);
      } finally {
        writer.close();
      }
}

// used in finding next node.
public static void checkInterfaceListZeros() {
                if (Fram3.intrArr.contains(0)) {
                        Fram3.getRes = 0;
```

```
			} else {
				Fram3.getRes = 1;
			}
}

// finding next node, from parentlist
public static void printTableValues() {
			int number = 0;
			int i = 0;
			int checkNumber = 0;
			for (int m = 0; m < Fram3.b; m++) {
				if (Fram3.intrArr.get(m) == 0) {
					number = Fram3.upPar.get(m);
					i = 0;
					while (i != 1) {
						if (Fram3.intrArr.get(number) == (number + 1)) {
							Fram3.intrArr.remove(m);
							Fram3.intrArr.add(m, number + 1);
							i = 1;
						} else {
							number = Fram3.intrArr.get(number);
							if (number == 0) {
								Fram3.intrArr.remove(m);
								Fram3.intrArr.add(m, 0);
								i = 1;
							} else {
								number--;
					}
						}
					}

				}
			}
}

// used in finding next node from the parent list
public static void connectionTableArray(int sourceRouter) {
			for (int k = 0; k < Fram3.b; k++) {
				if (sourceRouter == Fram3.upPar.get(k)) {
					Fram3.intrArr.add(k + 1);
				} else {
					if (-1 == Fram3.upPar.get(k))
						Fram3.intrArr.add(-1);
					else
						Fram3.intrArr.add(0);
				}
			}
}

public static void findWeiAndUpdateList(int minValNextRouterPosition) {
			int y = 0;
			int nextMin = 0;
			for (int j = 0; j < Fram3.neighLst.size(); j++) {
			y = Fram3.neighLst.get(j);
			nextMin = Fram3.mat[minValNextRouterPosition][y] + Fram3.minValNxt;
```

```
				if (nextMin < Fram3.listCurrVal.get(y)) {
					Fram3.listCurrVal.add(y, nextMin);
					Fram3.listCurrVal.remove(y + 1);
					Fram3.upPar.add(y, minValNextRouterPosition);
					Fram3.upPar.remove(y + 1);
				}
			}
}

// finding neibours of current node
public static void findNeibours(int minValNextRouterPosition) {
			int x;
			Fram3.neighLst.clear();
			for (int j = 0; j < Fram3.b; j++) {
				x = Fram3.mat[minValNextRouterPosition][j];
				if (Fram3.visitArr.contains(j)) {
					// nothing , don't add the visited node;
				} else {
					if ((x != -1) && (x > 0)) {
						Fram3.neighLst.add(j);
					}
				}
			}
}

// finding next min value,position
public static void findMinValInRow() {
			Integer i = 0;
			int min = 1000;
			for (Integer x : Fram3.listCurrVal) {
				if ((x != -1) && (x > 0)) {
					if (x <= min && !(Fram3.visitArr.contains(i))) {
						Fram3.minValNxt = x;
						Fram3.minValNextPos = i;
						min = x;
					}
				}
				i++;
			}
}

// adding visited node in list
public static void setVisitedNodeInList() {
			Fram3.visitArr.add(Fram3.minValNextPos);
}

// parent list contain -1 initially
public static void updateParentList(int sourceRouter) {
			for (int j = 0; j < Fram3.b; j++) {
				Fram3.upPar.add(-1);
			}
}
```

4. DESIGN AND WORKFLOW

This application consists of 5 separate java files that encompasses distinct set of functionalities of the program.

Firstly the start.java file has the first page of the application that welcomes the user. The start button in this page has to be clicked, which when invokes the link.java class.

This link.java class provides the user with the 4 cases, which the user has to select. The four functionalities are,

1. Input network topology file
2. Build a connection table
3. Shortest path to destination router
4. Exit

The user is prompted to select his option. Upon pressing the execute button the code flows to a separate class based on the user input.

When the user presses 1 in the previous menu, Fram3.java is invoked. This file is used to receive and retrieve the matrix inside a topology text file. The matrix retrieved is then displayed on the application.

Upon entering the 2nd option, Form4.java class is invoked. This java class has methods to use the matrix of the previous menu and compute the routing table. The user has to input the source router value with which this file computes the shortest path to every other node using Dijkstra's Routing Algorithm

The 3rd option when entered, the Form5.java is invoked which implements the functionality of finding the shortest path with the least cost available. This requires the user to enter the destination router number, using which the class compute the cost & the complete shortest path using Dijkstra's algorithm

The 4th option when pressed breaks the code flow hence exiting the application.

5. PROOF FOR DIJKSTRA'S ALGORITHM IMPLEMENTED IN THIS PROJECT

To check the correctness of the implemented Dijkstra's algorithm we first manually iterate through the well – known, standard Dijkstra's algorithm and compared the standard results with the results we got from our code for the same topology matrix or input. The results were found to be same and were accurate.

We would also like to prove the implemented Dijkstra's algorithm to find the shortest path between the source and the destination mathematically using the concept of mathematical induction, which is explained at its best as shown below.

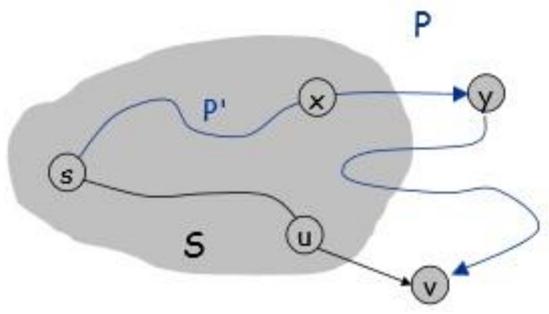
Invariant: For each node $u \in S, d(u)$ is the length of shortest path from the source (s) to destination (d) (s – u path).

Proof by Induction on $|S|$

Base Case: $|S| = 1$ is trivial.

Inductive Hypothesis: Assume true for $|S| = k \geq 1$.

- Let *v* be the next node added to *S*, and let *u-v* be the chosen edge.
- The shortest *s-u path* plus *(u, v)* is an *s-v path* of length $\pi(v)$.
- Consider any *s-v path P*. We will see that it's no shorter than $\pi(v)$.
- Let *x-y* be the first edge in *P* that leaves *S*, and let *P'* be the subpath to *x*.
- *P* is already too long as soon as it leaves *S*.



$$l(P) \underset{\substack{\uparrow\\ \text{Non-negative}\\ \text{weights}}}{\geq} l(P') + l(x,y) \underset{\substack{\uparrow\\ \text{Inductive}\\ \text{Hypothesis}}}{\geq} d(x) + l(x,y) \underset{\substack{\uparrow\\ defn\\ of\ \pi(y)}}{\geq} \pi(y) \underset{\substack{\uparrow\\ \text{Dijkstra choose}\\ \text{v instead of y}}}{\geq} \pi(v)$$

6. USER MANUAL

For the execution of this project (Dijkstra) requires the following steps to be performed.

1. Download & Install the latest version of NetBeans IDE.
2. Import the project Dijkstra into the NetBeans IDE.
3. Build the project by selecting Run → Build Project (F11).
4. After Build has successfully completed, run the project Dijkstra.

Steps to run the JAR executable

1. Double click on the Djikstra.jar file and follow the instructions.

7. PROBLEMS

1. The input topology text file should not contain any extra spaces or line feeds. Extra spaces may create exception.

8. TESTING AND IT'S RESULTS

The Link – State Routing Protocol implemented in this project using Dijkstra's algorithm has been tested under various cases and is found to be successful. This project has been tested for various number of routers like 5 routers, 10 routers, 12 routers, 15 routers, 20 routers etc. The successful and correct results were obtained during testing and the results are tabulated as below.

- **Instance 1:** Number of routers = 5
  Input File: 5_Routers.txt
  Original Matrix:

```
0 2 5 1 -1
2 0 8 7 9
5 8 0 -1 4
1 7 -1 0 2
-1 9 4 2 0
```

  Routing or Connection Table:

	R1	**R2**	**R3**	**R4**	**R5**
R1	-1	2	3	4	4
R2	1	-1	1	1	1
R3	1	1	-1	5	5
R4	1	1	1	-1	5
R5	4	4	3	4	-1

The shortest path from the router 1 to router 5 is: 1-4-5
The total cost is: 3

The shortest path from the router 3 to router 1 is: 3-1
The total cost is: 5

The shortest path from the router 5 to router 4 is: 5-4
The total cost is: 2

- **Instance 2:** Number of routers = 10
  Input File: 10_Routers.txt
  Original Matrix:

```
0 2 3 1 -1 3 2 7 2 9
2 0 -1 3 -1 7 -1 4 2 4
3 -1 0 9 8 7 -1 3 4 -1
1 3 9 0 4 5 7 2 1 6
-1 -1 8 4 0 2 -4 3 7 2
3 7 7 5 2 0 -1 8 -1 3
```

2 -1 -1 7 -1 -1 0 3 4 7

7 4 3 2 3 8 3 0 2 5

2 2 4 1 7 -1 4 2 0 -1

9 4 -1 6 2 3 7 5 -1 0

Routing or Connection Table:

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**
R1	-1	2	3	4	4	6	7	4	9	2
R2	1	-1	1	4	10	1	1	8	9	10
R3	1	1	-1	1	8	1	1	8	9	8
R4	1	2	1	-1	5	1	1	8	9	10
R5	6	10	8	4	-1	6	8	8	8	10
R6	1	1	1	1	5	-1	1	5	1	10
R7	1	1	1	1	8	1	-1	8	9	10
R8	4	2	3	4	5	5	7	-1	9	10
R9	1	2	3	4	4	1	7	8	-1	2
R10	6	2	8	4	5	6	7	8	2	-1

The shortest path from the router 10 to router 5 is: 10-5
The total cost is: 2

The shortest path from the router 3 to router 7 is: 3-1-7
The total cost is: 5

The shortest path from the router 4 to router 1 is: 4-1
The total cost is: 1

- **Instance 3:** Number of routers = 12
  Input File: 12_Routers.txt
  Original Matrix:

0 2 3 1 -1 3 2 7 8 9 -1 1

2 0 -1 3 -1 7 -1 4 2 4 3 5

3 -1 0 9 8 7 -1 3 4 -1 1 7

1 3 9 0 4 5 7 2 1 6 3 -1

-1 -1 8 4 0 2 -1 3 7 2 8 -1

3 7 7 5 2 0 -1 8 -1 3 4 5

2 -1 -1 7 -1 -1 0 3 4 7 9 -1

7 4 3 2 3 8 3 0 2 5 3 2

2 2 4 1 7 -1 4 2 0 -1 7 1

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

-1 3 1 3 8 4 9 3 7 2 0 1

1 5 7 -1 -1 5 -1 2 1 8 1 0

Routing or Connection Table:

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**R11**	**R12**
R1	-1	2	3	4	4	6	7	12	12	12	12	12
R2	1	-1	11	4	10	1	1	8	9	10	11	9
R3	1	11	-1	11	11	11	1	8	11	11	11	11
R4	1	2	1	-1	5	1	1	8	9	11	11	9
R5	6	10	10	4	-1	6	8	8	8	10	10	8
R6	1	1	11	1	5	-1	1	5	1	10	11	1
R7	1	1	1	1	8	1	-1	8	9	1	1	1
R8	12	2	3	4	5	5	7	-1	9	10	11	12
R9	1	2	12	4	4	1	7	8	-1	12	12	12
R10	11	2	11	11	5	6	11	8	11	-1	11	11
R11	12	2	3	4	10	6	12	8	12	10	-1	12
R12	1	9	11	9	8	1	1	8	9	11	11	-1

The shortest path from the router 12 to router 9 is: 12-9
The total cost is: 1

The shortest path from the router 6 to router 12 is: 6-1-12
The total cost is: 4

The shortest path from the router 8 to router 1 is: 8-12-1
The total cost is: 3

- **Instance 4:** Number of routers = 15
  Input File: 15_Routers.txt
  Original Matrix:

```
0 3 -1 4 -1 2 9 8 7 -1 9 2 3 1 3

3 0 -1 -1 2 3 9 8 7 -1 2 -1 5 3 1

-1 -1 0 -1 1 1 4 9 2 -1 3 -1 5 2 8

4 -1 -1 0 -1 2 3 7 2 8 7 2 6 5 -1

-1 2 1 -1 0 9 3 8 -1 -1 -1 2 3 -1 3

2 3 1 2 9 0 2 9 3 2 6 8 -1 -1 3

9 9 4 3 3 2 0 2 8 6 3 4 -1 -1 2

8 8 9 7 8 9 2 0 2 3 -1 7 8 -1 -1

7 7 2 2 -1 3 8 2 0 -1 -1 8 2 1 -1

-1 -1 -1 8 -1 2 6 3 -1 0 7 6 5 2 1

9 2 3 7 -1 6 3 -1 -1 7 0 2 -1 1 1

2 -1 -1 2 2 8 4 7 8 6 2 0 7 9 -1

3 5 5 6 3 -1 -1 8 2 5 -1 7 0 -1 6

1 3 2 5 -1 -1 -1 -1 1 2 1 9 -1 0 -1

3 1 8 -1 3 3 2 -1 -1 1 1 -1 6 -1 0
```

Routing or Connection Table:

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
R1	-1	2	14	4	12	6	6	14	14	14	14	12	13	14	15
R2	1	-1	5	6	5	6	15	15	14	15	11	11	13	14	15
R3	6	5	-1	6	5	6	6	9	9	6	11	5	5	14	6
R4	1	6	6	-1	12	6	7	9	9	6	12	12	9	9	6
R5	12	2	3	12	-1	3	7	3	3	3	3	12	13	3	15
R6	1	2	3	4	3	-1	7	7	9	10	3	3	3	3	15
R7	6	15	6	4	5	6	-1	8	8	15	11	12	5	11	15
R8	9	7	9	9	7	7	7	-1	9	10	9	7	9	9	7
R9	14	14	3	4	3	6	8	8	-1	14	14	14	13	14	14
R10	14	15	6	6	15	6	15	8	14	-1	15	15	13	14	15
R11	14	2	3	12	15	15	7	14	14	15	-1	12	14	14	15
R12	1	11	5	4	5	4	7	4	4	11	11	-1	5	11	11
R13	1	2	9	9	5	9	5	9	9	10	9	5	-1	9	9
R14	1	2	3	9	3	1	11	9	9	10	11	11	9	-1	11
R15	1	2	11	7	5	6	7	10	11	10	11	11	11	11	-1

The shortest path from the router 1 to router 7 is: 1-6-7
The total cost is: 4

The shortest path from the router 8 to router 6 is: 8-7-6
The total cost is: 4

The shortest path from the router 13 to router 11 is: 13-9-14-11
The total cost is: 4

- **Instance 5:** Number of routers = 15
  Input File: 15_Routers.txt
  Original Matrix:

```
0 -1 3 1 -1 9 3 7 7 1 2 2 4 6 9 4 -1 7 1 3
-1 0 3 9 1 4 1 3 5 1 2 8 -1 5 6 1 5 1 3 5
3 3 0 7 7 2 1 8 6 2 1 5 5 6 1 -1 8 7 4 5
1 9 7 0 7 3 3 4 1 1 1 6 6 1 8 1 8 5 7 1
-1 1 7 7 0 7 1 5 3 6 4 9 2 1 1 8 6 7 4 2
9 4 2 3 7 0 1 1 9 2 2 5 8 5 8 6 9 2 9 8
3 1 1 3 1 1 0 4 3 5 4 8 2 1 1 8 9 7 4 8
7 3 8 4 5 1 4 0 2 7 8 1 1 8 9 4 4 1 5 6
7 5 6 1 3 9 3 2 0 7 3 6 1 8 5 2 9 7 1 1
1 1 2 1 6 2 5 7 7 0 9 4 1 7 1 1 6 3 4 5
2 2 1 1 4 2 4 8 3 9 0 -1 5 1 2 7 5 4 1 8
```

```
2 8 5 6 9 5 8 1 6 4 -1 0 3 3 6 4 4 4 8 6
4 1 5 6 2 8 2 1 1 1 5 3 0 -1 1 1 5 7 1 8
6 5 6 1 1 5 1 8 8 7 1 3 -1 0 7 7 1 9 2 8
9 6 1 8 1 8 1 9 5 1 2 6 1 7 0 2 2 1 2 3
4 1 -1 1 8 6 8 4 2 1 7 4 1 7 2 0 -1 1 8 7
-1 5 8 8 6 9 9 4 9 6 5 4 5 1 2 -1 0 2 4 1
7 -1 7 5 7 2 7 1 7 3 4 4 7 9 1 1 2 0 2 9
1 3 4 7 4 9 4 5 1 4 1 8 1 2 2 8 4 2 0 -1
3 3 5 5 1 2 8 8 6 1 5 8 6 8 8 3 7 1 9 -1 0
```

Routing or Connection Table:

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**R11**	**R12**	**R13**	**R14**	**R15**	**R16**	**R17**	**R18**	**R19**	**R20**
R1	-1	10	3	4	10	10	7	19	19	10	11	12	19	4	10	10	4	19	19	4
R2	10	-1	7	16	5	7	7	18	16	10	11	18	16	7	18	16	18	18	19	5
R3	1	7	-1	11	15	6	7	15	11	10	11	15	15	11	15	15	15	15	11	11
R4	1	16	11	-1	14	6	14	9	9	10	11	1	16	14	10	16	20	16	11	20
R5	15	2	15	14	-1	7	7	15	9	15	14	14	13	14	15	2	14	15	15	20
R6	10	7	3	4	7	-1	7	8	8	10	11	8	8	7	7	18	7	18	8	7
R7	1	2	3	14	5	6	-1	6	9	15	14	6	13	14	15	2	14	15	15	5
R8	12	13	6	18	13	6	6	-1	9	13	6	12	13	6	18	18	18	18	13	9
R9	19	13	13	4	5	8	7	8	-1	13	19	8	13	4	13	16	20	19	19	20
R10	1	2	3	4	15	6	15	13	13	-1	4	1	13	4	15	16	15	16	13	4
R11	1	2	3	4	14	6	14	19	19	4	-1	14	19	14	15	4	14	19	19	4
R12	1	8	8	1	8	8	8	8	8	8	8	-1	8	14	8	8	17	8	8	8
R13	19	2	15	16	5	8	7	8	9	10	19	8	-1	19	15	16	15	16	19	9
R14	4	7	11	4	5	7	7	7	4	4	11	12	7	-1	7	4	17	17	19	17
R15	10	13	3	10	5	7	7	18	13	10	11	18	13	7	-1	16	17	18	19	20
R16	10	2	10	4	2	18	2	18	9	10	4	18	13	4	15	-1	18	18	13	4
R17	20	14	15	20	14	14	14	18	20	15	14	12	15	14	15	18	-1	18	14	20
R18	19	16	15	16	15	6	15	8	16	16	15	8	16	17	15	16	17	-1	19	17
R19	1	13	11	11	13	11	13	13	9	13	11	1	13	14	15	13	9	18	-1	9
R20	4	4	4	4	5	4	17	9	9	4	4	4	9	17	15	4	17	17	9	-1

The shortest path from the router 6 to router 2 is: 6-7-2
The total cost is: 2

The shortest path from the router 17 to router 19 is: 17-14-19
The total cost is: 3

The shortest path from the router 14 to router 4 is: 14-4
The total cost is: 1

8. SNAPSHOTS

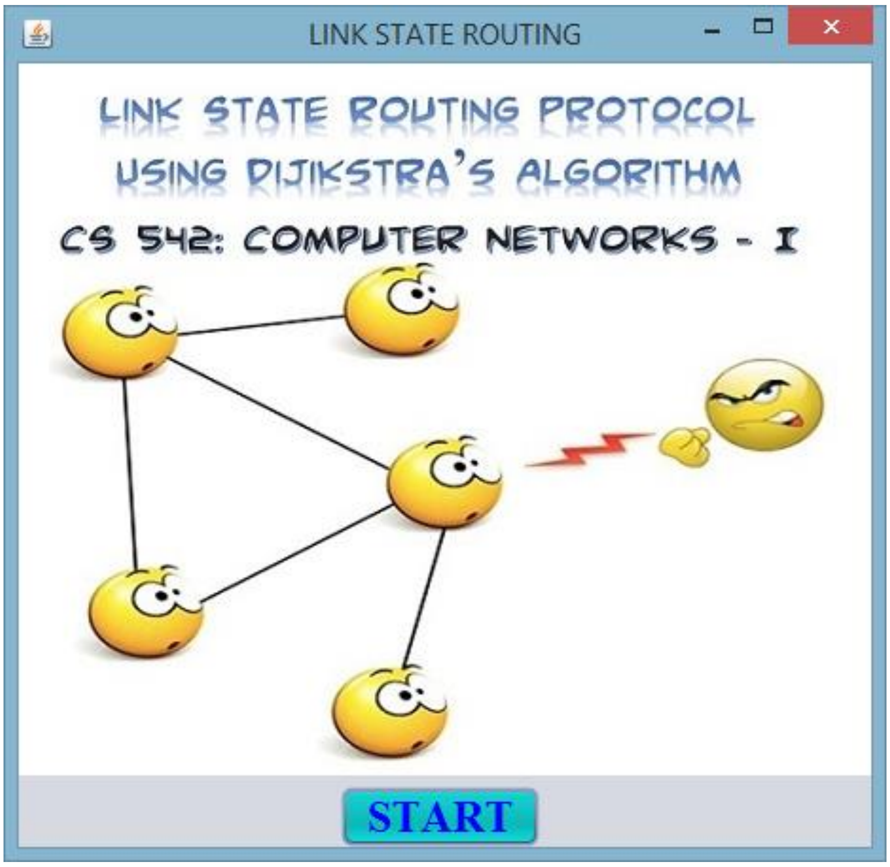
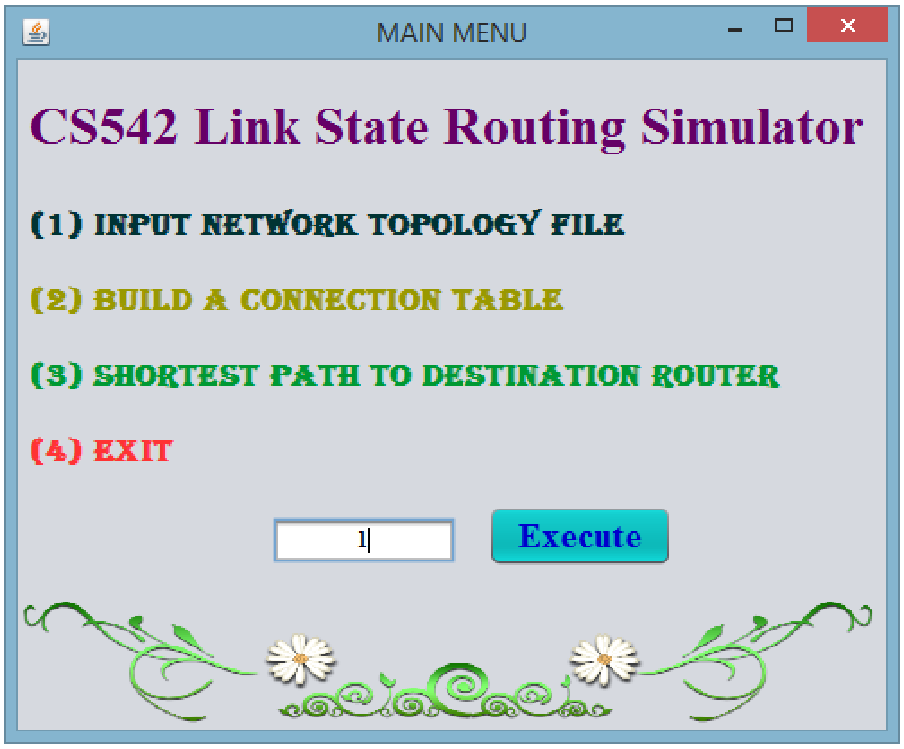

LINK STATE ROUTING
LINK STATE ROUTING PROTOCOL
USING DIJIKSTRA'S ALGORITHM
CS 542: COMPUTER NETWORKS - I
START

Figure: Start Screen of the Project.



CS542 Link State Routing Simulator

(1) INPUT NETWORK TOPOLOGY FILE

(2) BUILD A CONNECTION TABLE

(3) SHORTEST PATH TO DESTINATION ROUTER

(4) EXIT

1

Figure: Window indicating the Selection of Option 1.

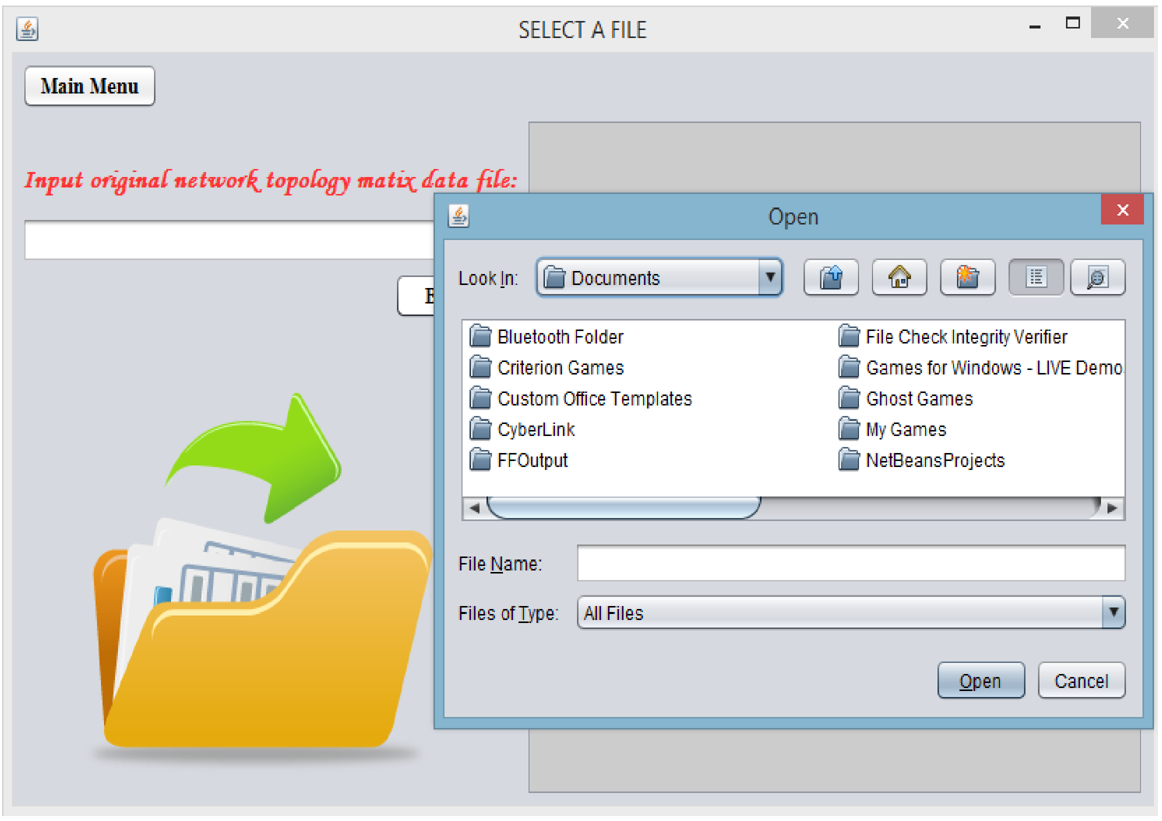
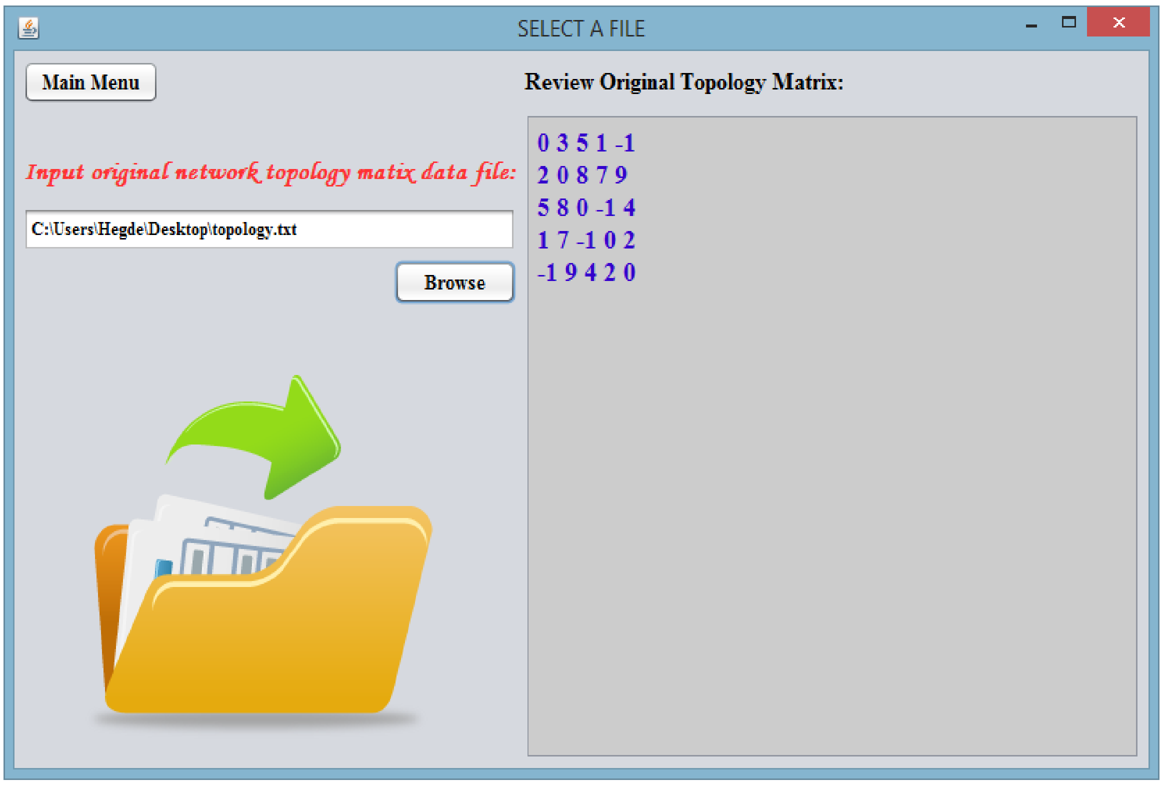


Figure: Browse or Select of a File for Computation.


SELECT A FILE
Main Menu
Review Original Topology Matrix:
Input original network topology matix data file:
Browse

C:\Users\Hegde\Desktop\topology.txt

```
0 3 5 1 -1
2 0 8 7 9
5 8 0 -1 4
1 7 -1 0 2
-1 9 4 2 0
```

Figure: Window displaying the selected input file.

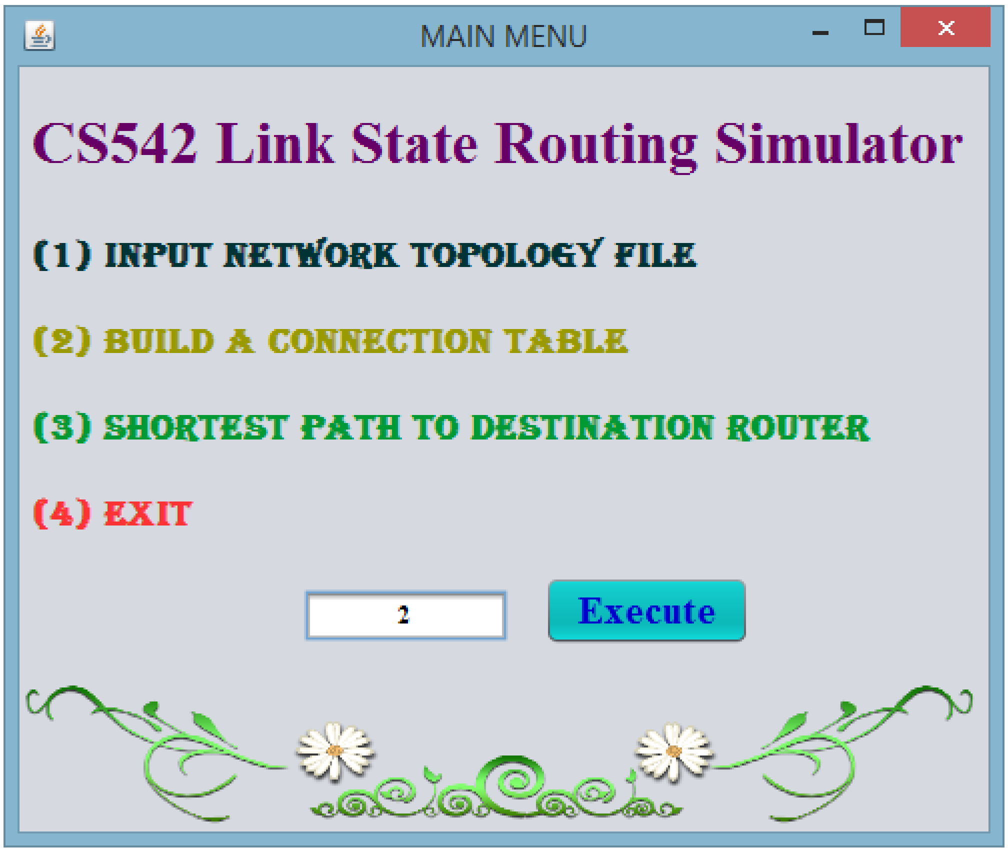

MAIN MENU
CS542 Link State Routing Simulator
(1) INPUT NETWORK TOPOLOGY FILE
(2) BUILD A CONNECTION TABLE
(3) SHORTEST PATH TO DESTINATION ROUTER
(4) EXIT
2
Execute

Figure: Window indicating the Selection of Option 2.

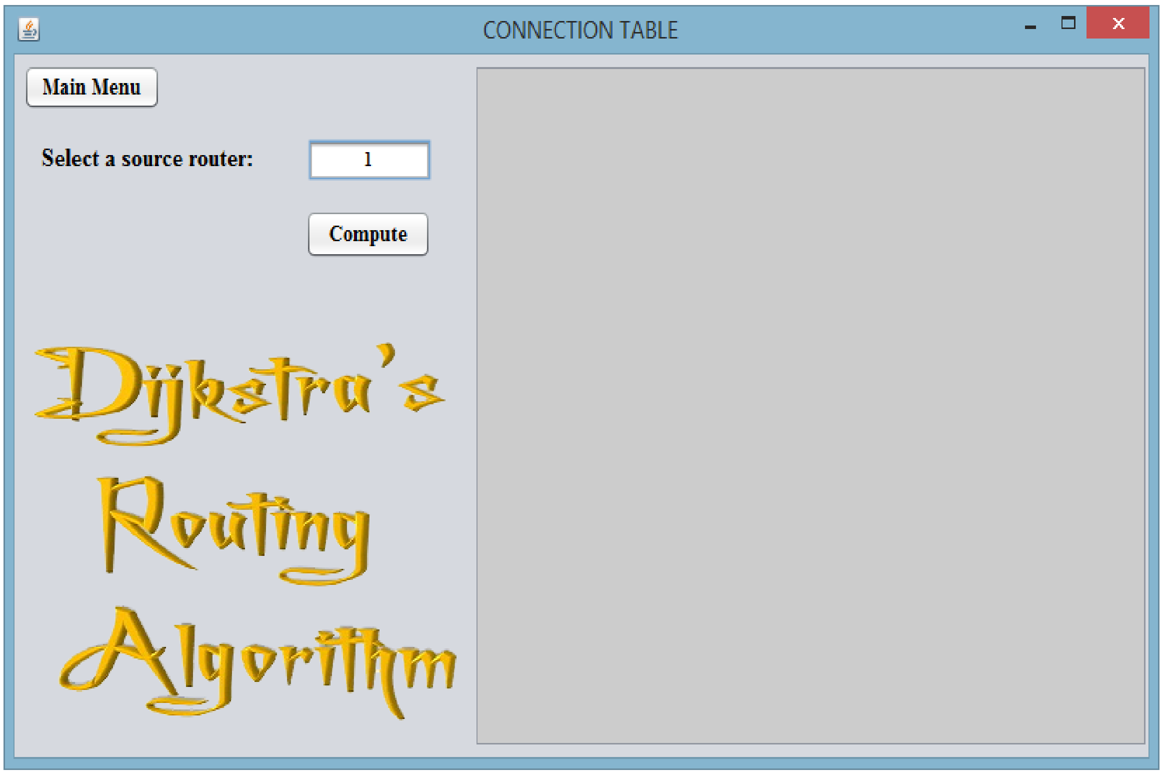
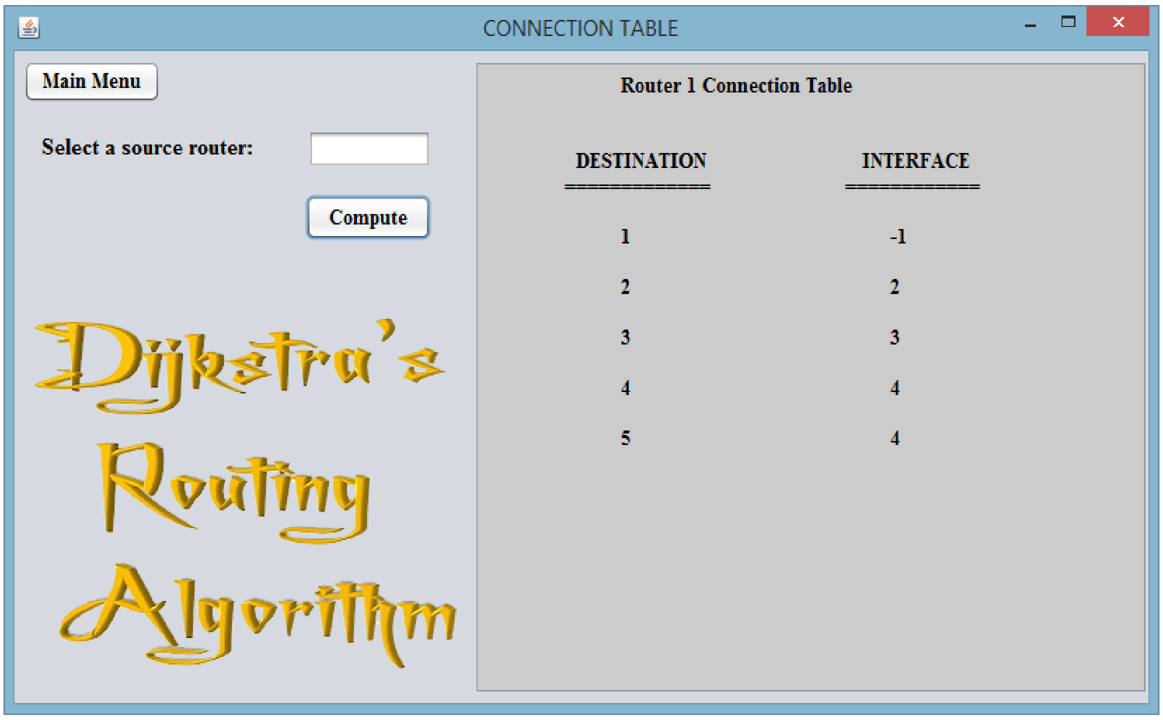


Figure: Window indicating the Selection of a source router.


CONNECTION TABLE

Main Menu

Select a source router:

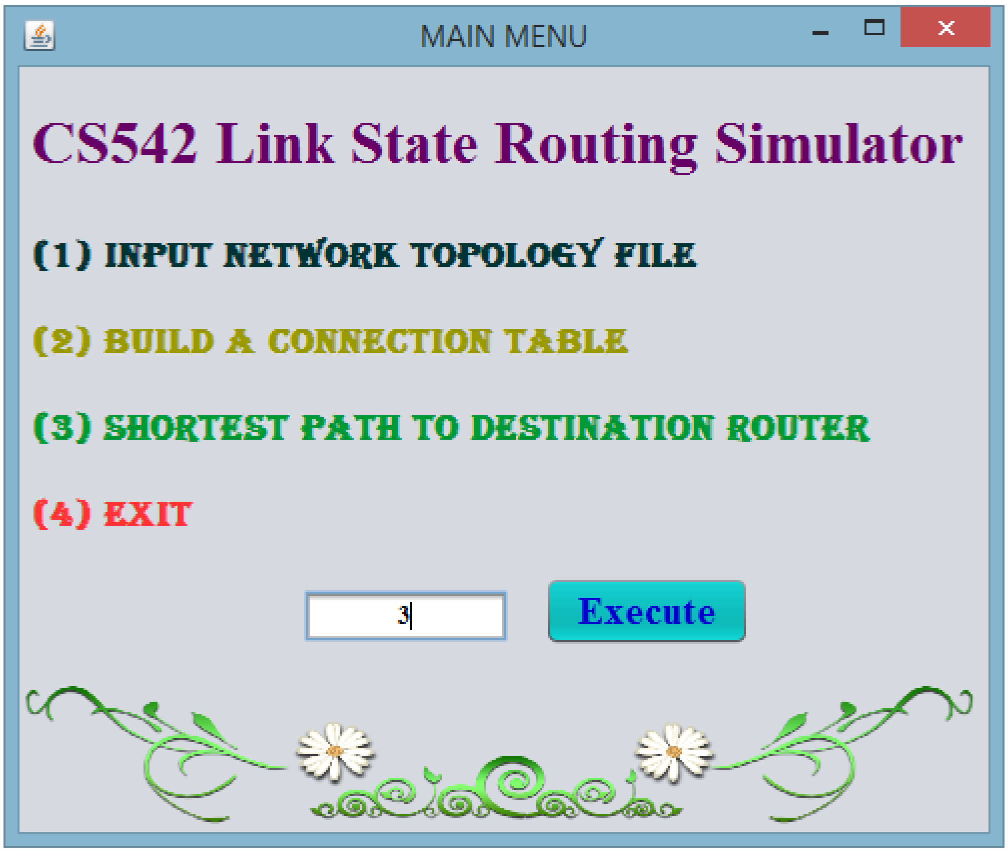
Compute

Dijkstra's Routing Algorithm

Router 1 Connection Table

DESTINATION	INTERFACE
1	-1
2	2
3	3
4	4
5	4

Figure: Display of the Router's Connection Table.


MAIN MENU

CS542 Link State Routing Simulator

(1) INPUT NETWORK TOPOLOGY FILE

(2) BUILD A CONNECTION TABLE

(3) SHORTEST PATH TO DESTINATION ROUTER

(4) EXIT

3

Execute

Figure: Window indicating the Selection of Option 3.

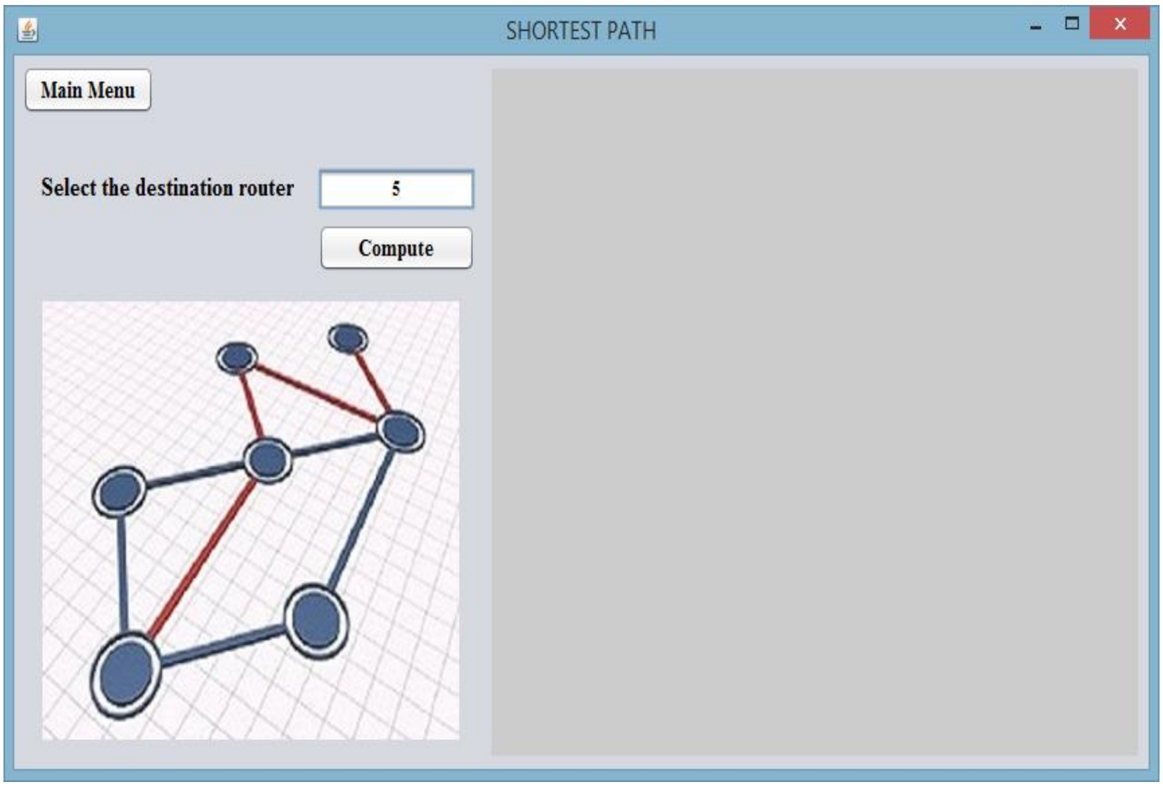
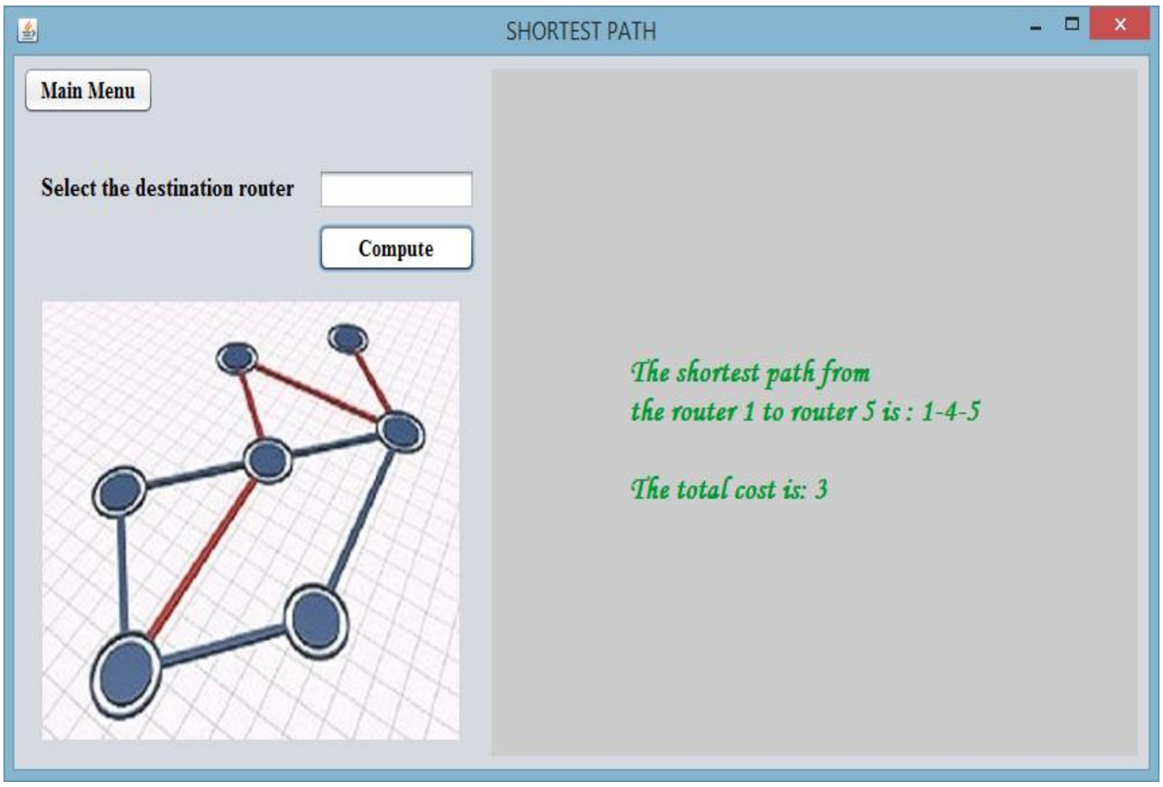


Figure: Window indicating the Selection of destination router.



The shortest path from
the router 1 to router 5 is : 1-4-5

The total cost is: 3

Figure: Displaying the Shortest Path from source to destination and the Total Cost.

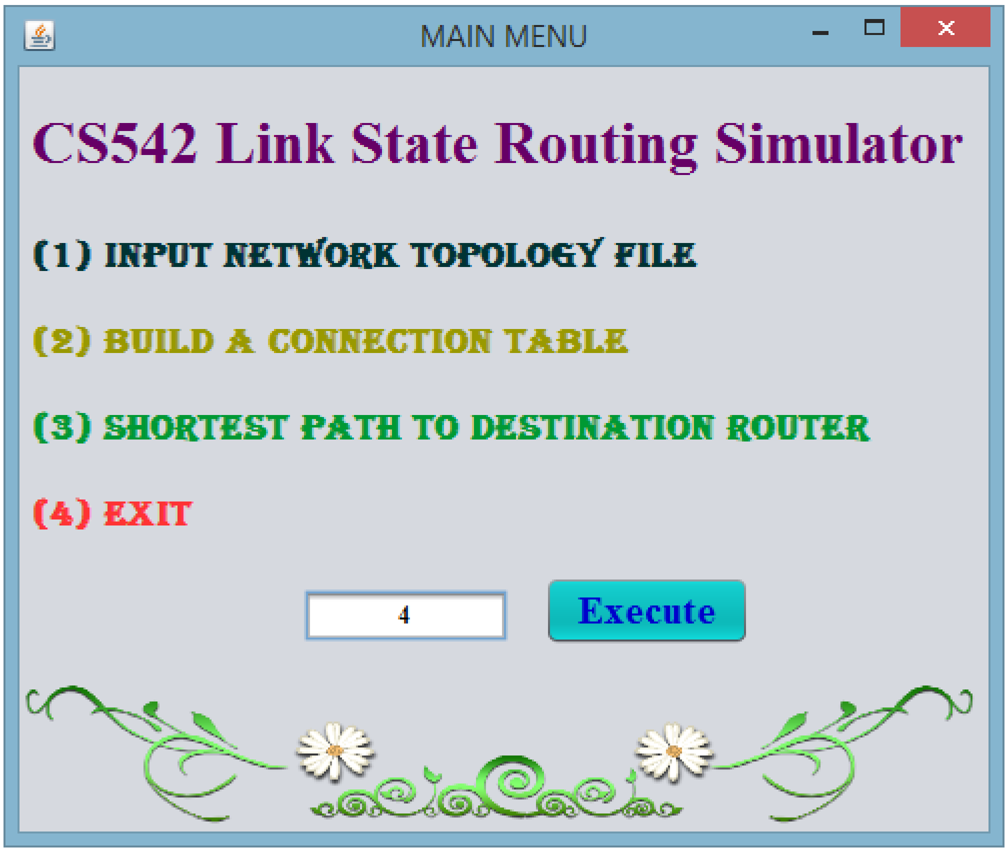
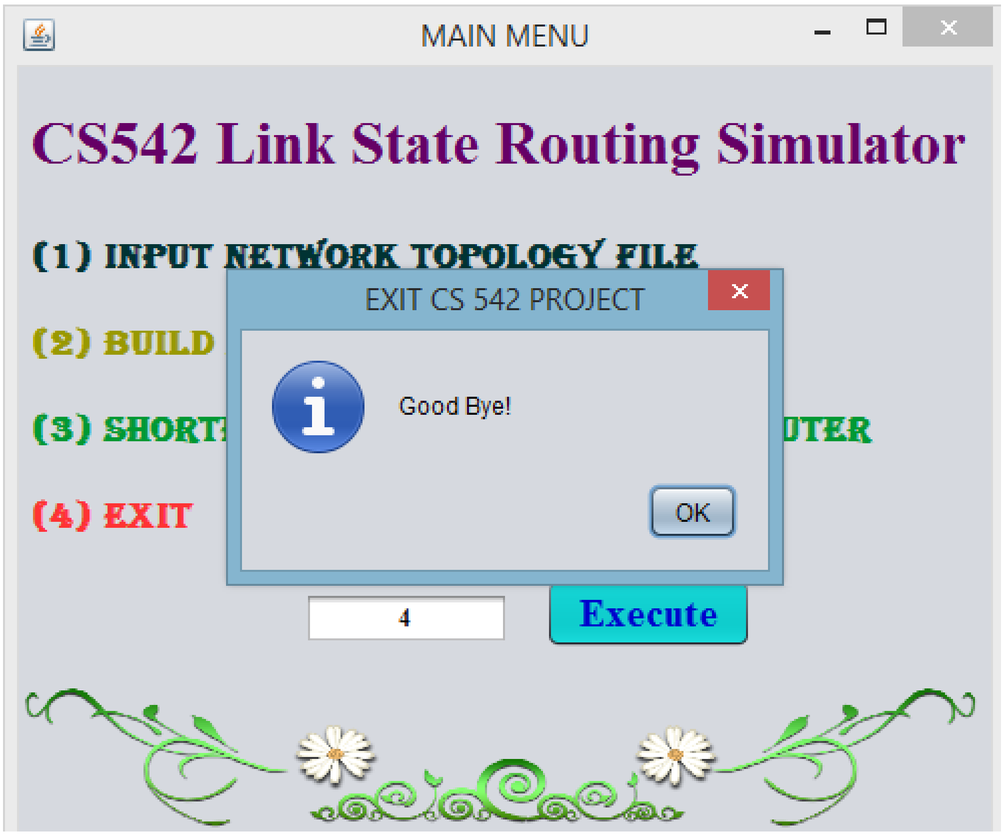

MAIN MENU
CS542 Link State Routing Simulator
(1) INPUT NETWORK TOPOLOGY FILE
(2) BUILD A CONNECTION TABLE
(3) SHORTEST PATH TO DESTINATION ROUTER
(4) EXIT
4
Execute

Figure: Window indicating the Selection of Option 4.



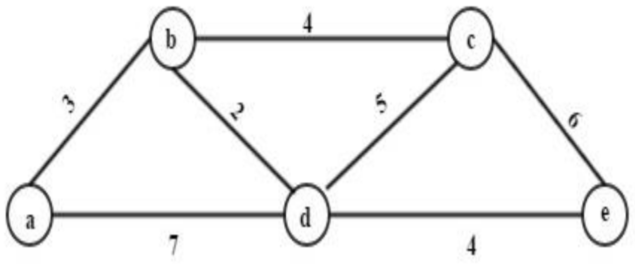
Good Bye!

Figure: Exit of the Project Dialog.

9. CASE STUDY

We consider a case study to illustrate the working of the Dijkstra’s Algorithm. This case study will briefly explain each and every step the algorithm follows to calculate the shortest path from source to the destination.

Let us now consider the following example:



The above weighted graph has 5 vertices from ‘a’ to ‘e’. The value between the two vertices is known as the edge cost between two vertices. For example the edge cost between ‘a’ and ‘d’ is 7. Now let us first construct the topology matrix for the above weighted graph.

0	3	-1	7	-1
3	0	4	2	-1
-1	4	0	5	6
7	2	5	0	4
-1	-1	6	4	0

Now let’s find out the routing table or the connection table. Suppose, our source router is ‘a’. The connection table for source router ‘a’ is as follows:

DESTINATION	INTERFACE
a	-1
b	b
c	b
d	b
e	b

In the above routing or connection table ‘-1’ indicates that the path does not exist, for example in the table ‘INTERFACE’ of router ‘a’ is ‘-1’ which indicates that there exists no any direct path from router ‘a’ to the router ‘a’ itself.

Similarly, the routing table or connection table for other routers can also be found and are as follows:

Connection table for router ‘b’ is as shown below:

DESTINATION	INTERFACE
a	a
b	-1
c	c
d	d
e	d

Connection table for router ‘c’ is as shown below:

DESTINATION	INTERFACE
a	b
b	b
c	-1
d	d
e	e

Connection table for router ‘d’ is as shown below:

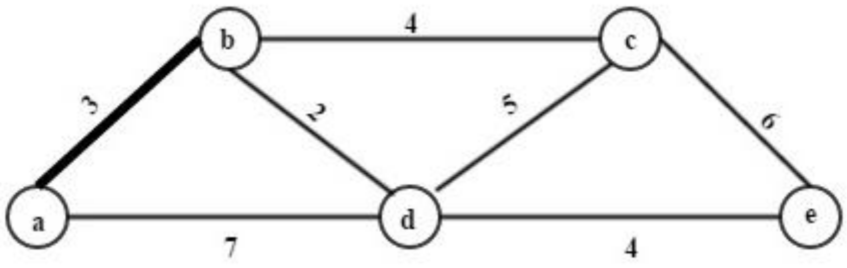
DESTINATION	INTERFACE
a	b
b	b
c	c
d	-1
e	e

Connection table for router ‘e’ is as shown below:

DESTINATION	INTERFACE
a	d
b	d
c	c
d	d
e	-1

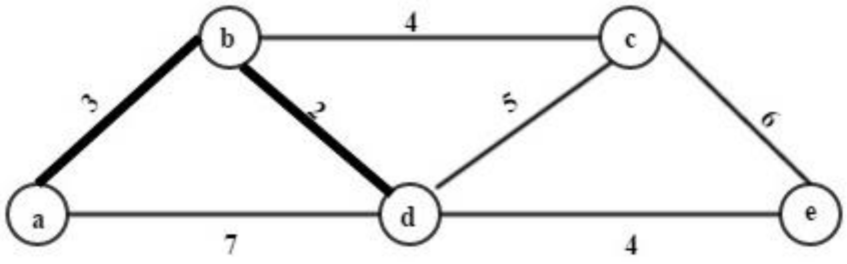
At last we have to find the shortest path from source to destination and the total cost which is as follows:

The shortest path from the router ‘a’ to router ‘b’ is: a – b



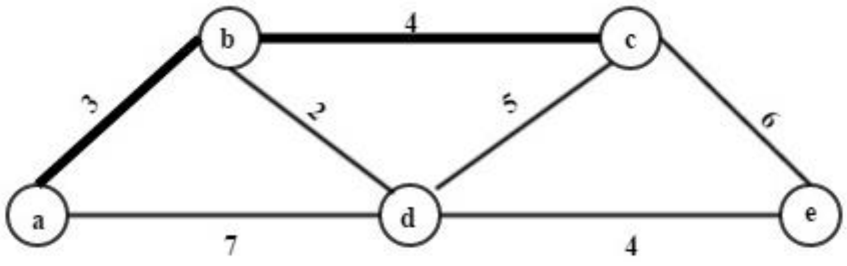
The total cost is: 3

The shortest path from the router ‘a’ to router ‘d’ is: a – b – d



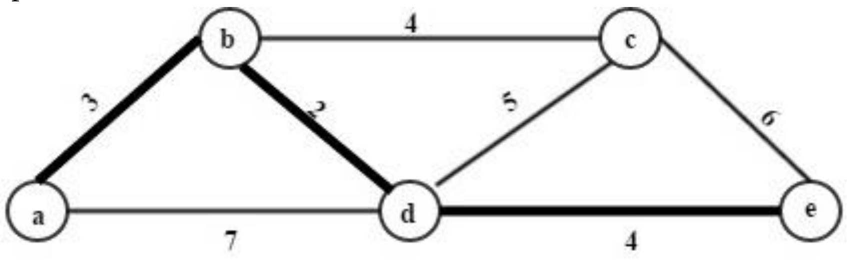
The total cost is: 5

The shortest path from the router ‘a’ to router ‘c’ is: a – b – c



The total cost is: 7

The shortest path from the router ‘a’ to router ‘e’ is: a – b – d –e



The total cost is: 9