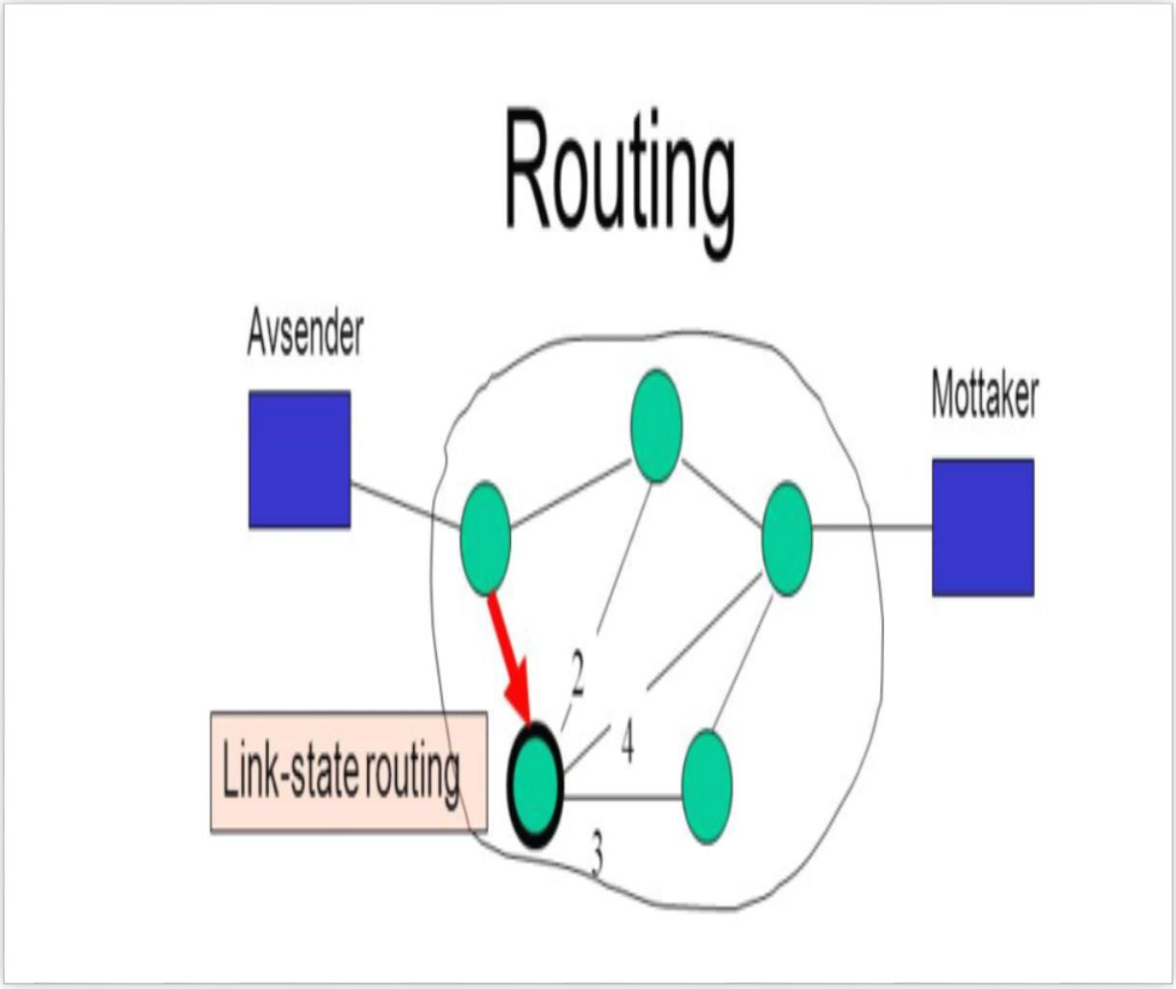

CS 542 LINK-STATE ROUTING

Chethan Kumar Appaji | Computer Networks_Section 02| 11-21-2015

Link State Routing protocol

INDEX

Introduction

Routing:

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).

Link State Routing Protocol:

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.

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.

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.

```
1  function Dijkstra(Graph, source):
2
3      create vertex set Q
4
5      for each vertex v in Graph:            // Initialization
6          dist[v] ← INFINITY                 // Unknown distance from source to v
7          prev[v] ← UNDEFINED                // Previous node in optimal path from source
8          add v to Q                         // All nodes initially in Q (unvisited nodes)
9
10     dist[source] ← 0                       // Distance from source to source
11
12     while Q is not empty:
13         u ← vertex in Q with min dist[u]   // Source node will be selected first
14         remove u from Q
15
16         for each neighbor v of u:          // where v is still in Q.
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]:              // A shorter path to v has been found
19                 dist[v] ← alt
20                 prev[v] ← u
21
22     return dist[], prev[]
```

If we are only interested in a shortest path between vertices *source* and *target*, we can terminate the search after line 13 if *u* = *target*. Now we can read the shortest path from *source* to *target* by reverse iteration:

```
1  S ← empty sequence
2  u ← target
3  while prev[u] is defined:              // Construct the shortest path with a stack S
4      insert u at the beginning of S     // Push the vertex onto the stack
5      u ← prev[u]                        // Traverse from target to source
6  insert u at the beginning of S         // Push the source onto the stack
```

Application Design

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.

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. **Create a Network Topology**
2. **Build a Connection Table**
3. **Shortest Path to Destination Router**
4. **Modify a topology**
5. **Exit**

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.

Variables Used

```
static int choice, row, column;

// To take input from the command prompt
static Scanner scan = new Scanner(System.in);
static int[][] fileData=null;

// Variables to store intermediate values
static int[] temp;
static String fName;
static int gotResult;
static int nextMinvalue;
static int nextMinvaluePosition;
static int sourceNode;
static int destNode;
static int nodeToInsert;
static int[] inArray;
static int nodeToDelete;
static int conn = 0;
static int shpth =0;
static boolean breakWhileLoop = false;
static boolean sourceflag = false;
static boolean topologyflag = false;

// "nodesVisited" keep track of the nodes that are visited
static ArrayList<Integer> nodesVisited = new ArrayList<Integer>();

// "listOfMinEdge" keep track of minimum value edges between nodes to   find
        shortest path
static ArrayList<Integer> listOfMinEdge = new ArrayList<Integer>();

// "listofAdjNodes" keep track of adjacent neighbors of the node
static ArrayList<Integer> listofAdjNodes = new ArrayList<Integer>();

// "listofParentNodes" that keep track of parents of the node
static ArrayList<Integer> listofParentNodes = new ArrayList<Integer>();

// calculating the next node
static ArrayList<Integer> listOfNextNode = new ArrayList<Integer>();
```

Implementation & Instructions to run the code

Interface:

Main Menu:

```
**********************************************************
|      ......Link State Routing Simulator......          |
**********************************************************
|1.| Create a Network Topology.                          |
|2.| Build a Connection Table.                           |
|3.| Shortest Path to Destination Router.                |
|4.| Modify a Topology.                                  |
|5.| Exit.                                               |
**********************************************************

Enter your choice:
```

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

```
Enter your choice:1
Enter the Inputfile:

Linktest.txt
File exists

Reading file

Content of original topology matrix is:

Row:5     Column:5
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
```

Step 2: On selecting Option 2, to enter the source node and Display Connection table

```
**********************************************************
|      ......Link State Routing Simulator......          |
**********************************************************
|1.| Create a Network Topology.                          |
|2.| Build a Connection Table.                           |
|3.| Shortest Path to Destination Router.                |
|4.| Modify a Topology.                                  |
|5.| Exit.                                               |
**********************************************************

Enter your choice:2
Enter the source router

Enter the source router
3
The Connection table for Source Node
-----------     --------
Destination      NextHop
-----------     --------

1                1

2                1

3                -1

4                5

5                5



----------------------------------------

The Connection table for all the nodes are
*****************************************************************************************************
|              |      1       |      2       |      3       |      4       |      5       |
*****************************************************************************************************
|      1       |      1       |      1       |      -1      |      5       |      5       |
|      2       |      1       |      1       |      -1      |      5       |      5       |
|      3       |      1       |      1       |      -1      |      5       |      5       |
|      4       |      1       |      1       |      -1      |      5       |      5       |
|      5       |      1       |      1       |      -1      |      5       |      5       |
```

Step 3: On selecting Option 3, to enter the destination node and Display shortest Path and Cost

```
**************************************************
|      ......Link State Routing Simulator......     |
**************************************************
|1.| Create a Network Topology.                     |
|2.| Build a Connection Table.                      |
|3.| Shortest Path to Destination Router.           |
|4.| Modify a Topology.                             |
|5.| Exit.                                          |
**************************************************

Enter your choice:3

Select the destination router: 5

The shortest path from the router 3 to router 5 is : 3-5 || The total cost is: 4
**************************************************
```

Step 4: On selecting Option 4, Modify a topology. Display options to Add and delete a node

```
**********************************************************
|      ......Link State Routing Simulator......     |
**********************************************************
|1.| Create a Network Topology.                     |
|2.| Build a Connection Table.                      |
|3.| Shortest Path to Destination Router.           |
|4.| Modify a Topology.                             |
|5.| Exit.                                          |
**********************************************************

Enter your choice:4
Modifying the Topology
Please select your option of Modification
(1)Add a Node
(2)Delete a Node
```

Step 5: On selecting sub-option 1, Add a node. Enter the node to be added and its values and display new matrix.

```
Modifying the Topology
Please select your option of Modification
(1)Add a Node
(2)Delete a Node
1
Enter the place you want to add the new node
2
Enter the routing values for the new node
3
4
1
6
2
1

The Total Number of Routers present after addition of a node are:6
The following is the Routing Table after Addition of a node

Content of original topology matrix is:

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

Step 6: On selecting sub-option 2, Delete a node. Enter the node to be deleted and display new matrix.

```
Modifying the Topology
Please select your option of Modification
(1)Add a Node
(2)Delete a Node
2
Enter The Node number to be deleted
2
The Total Number of Routers present after deletion of a node are:5
The following is the Routing Table after Deletion of a node

Content of original topology matrix is:

Row:5     Column:5
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
```

Step 7: On selecting Option 5, Exit from Main Menu

```
*******************************************************
|      ......Link State Routing Simulator......      |
*******************************************************
|1.| Create a Network Topology.                       |
|2.| Build a Connection Table.                        |
|3.| Shortest Path to Destination Router.             |
|4.| Modify a Topology.                               |
|5.| Exit.                                            |
*******************************************************

Enter your choice:5

Exit CS542 project. Good Bye...
```

Source code with comments

```
public static void main(String[] args) throws IOException{

                while(true){
                        System.out.println();

        System.out.println("*************************************************
****");
                        System.out.println("|      ......Link State Routing Simulator......      |");

        System.out.println("*************************************************
****");
                        System.out.println("|1.| Create a Network Topology.                       |");
                        System.out.println("|2.| Build a Connection Table.                        |");
                        System.out.println("|3.| Shortest Path to Destination Router.             |");
                        System.out.println("|4.| Modify a Topology.                               |");
                        System.out.println("|5.| Exit.                                            |");

        System.out.println("*************************************************
****");
                        System.out.printf("\nEnter your choice:",choice);


                        scan = new Scanner(System.in);
                        try {
```

```
                choice = scan.nextInt();

                if((choice<=0 || choice>5))
                {
                        System.err.println("Incorrect choice : Please select
options 1 to 5");
                        System.out.println();
                }
        } catch (Exception e) {
                System.err.println("Incorrect choice : Please select options
1 to 5");
                System.out.println();
        }


        switch(choice){

        case 1:
                try {
                        fName= readFile();
                        File file = new File(fName);
                        file = new File(fName);
                        if (file.exists()){
                                System.out.println("File exists");
                                }
                                else{
                                System.err.print("Specified      file      doesn
exists\n");
                                break;
                                }
                } catch (Exception e) {
                        e.getStackTrace();
                }

                try{

                fileData = readFileContent(fName);
                }
                catch (Exception e) {
                        e.getStackTrace();
                }

                topologyflag = true;
                dispTopology(fileData);
```

```
                    break;

                case 2:
                    if(topologyflag==false){
                        System.out.println();
                        System.err.println("Network topology input required before displaying Connection table");
                    }
                    else{
                        System.out.println("Enter the source router");

                        try {
                            sourceNode = scan.nextInt();
                        } catch (Exception e) {
                            System.out.println("Row:"+row+" "+"Column:"+column);
                            System.err.println("\n Incorrect Node : Source Node must be between 1 and "+column);
                            break;
                        }

                        if (sourceNode <= 0 || sourceNode > column) {
                            System.out.print("Row:"+row+" "+"Column:"+column);
                            System.out.print("\nIncorrect Node : Source Node must be between 1 and "+column);
                            System.out.println();
                            break;
                        }

                        sourceflag = true;
                        buildConnectionTable();
                    }
                    break;

                case 3:
                    if (sourceflag == false) {
                        System.err.print("Source router must be entered in choice 2 before Destination router\n");
                        break;
                    }
                    System.out.print("\nSelect the destination router: ");

                    try {
                        destNode = scan.nextInt();
```

```
                    } catch (Exception e) {
                        System.err.print("\n Incorrect Node : Destination router number is between 1 and "+column);
                        break;
                    }

                    if (destNode <= 0 || destNode > column) {
                        System.err
                                .print("\n Incorrect Node : Destination Router number is between 1 and "+column);
                        break;
                    }

                    // finding shortest path and cost
                    dispShortestPathandCost();
                    break;

                case 4:
                    if(topologyflag==false){
                        System.out.println();
                        System.err.println("Initial network topology input required before modifying topology");
                    }
                    modifytopology();
                    break;

                case 5:
                    System.err.print("\nExit CS542 project. Good Bye... ");
                    breakWhileLoop = true;
                    break;
                }
                if (breakWhileLoop == true)
                    break;
                }

            }

        public static String readFile(){
            try{
            System.out.println("Enter the Inputfile:");
            fName = scan.next();
            }
            catch(Exception e){
                System.err.println("File reading Exception");
            }
```

```
        return fName;
        }

        public static int[][] readFileContent(String fileName) throws IOException {

                System.out.println();
                System.out.println("Reading file");

                String line = "";

                BufferedReader bf = new BufferedReader(new FileReader (fileName));
                line = bf.readLine();
                String[] matrixDataSize = line.split(" ");

                for (int i = 0; i < matrixDataSize.length; i++) {
                        column++;
                }
                bf.close();

                // declaring the size of matrix
                int[][] matrixData = new int[column][column];


                String lineData = "";

                BufferedReader bfData = new BufferedReader(new FileReader
(fileName));
        int lineCount = 0;
        String[] numbers;
        while ((lineData = bfData.readLine()) != null)
        {
           numbers= lineData.split(" ");
           row=0;
           for (int i = 0; i < numbers.length; i++)
           {
           try{
           matrixData[lineCount][i] = Integer.parseInt(numbers[i]);
           }
           catch(NumberFormatException nfe)
           {
           System.out.println("It was an invalid number");
           System.exit(1);
           }
           row++;
           }
```

```
   column = row;
   lineCount++;
}
bfData.close();
return matrixData;
   }

   public static void dispTopology(int[][] fileData) {
            System.out.println("\nContent of original topology matrix is:\n");
            System.out.println("Row:"+row+"     "+"Column:"+column);

            for (int i = 0; i < row; i++) {
                     for (int j = 0; j < column; j++) {
                              System.out.print(fileData[i][j] + " ");
                     }
                     System.out.println();
            }
            System.out.println();
   }

   public static void buildConnectionTable(){
            // routers index starts with 0,1,2.., rather than 1,2,3

            sourceNode = sourceNode - 1;
            // clearing all values for successive iteration.
            nextMinvaluePosition = 0;
            nextMinvalue = 0;
            listOfMinEdge.clear();
            listofParentNodes.clear();
            nodesVisited.clear();
            listOfNextNode.clear();
            // initializing current list array
            initCurrRow(sourceNode);
            // make source node as visited
            nodesVisited.add(sourceNode);
            // initialize the parent list
            initParentList(sourceNode);
            // print the Connection table
            dispConnectionTable(sourceNode);
   }

   // update listOfMinEdge ,used in calculating shortest path
            public static void initCurrRow(int sourceNode) {
                     int j=0;
                     while(j<column){
```

```
                    if(j != sourceNode){
                        listOfMinEdge.add(1000);
                    }
                    else{
                        listOfMinEdge.add(0);
                    }
                    j++;
                }
            }


            public static void initParentList(int sourceNode) {
                int j=0;
                while(j<column){
                    listofParentNodes.add(-1);
                    j++;
                }
            }

            // Finding next node to reach from curr node.
            public static void dispConnectionTable(int sourceNode) {

                // storing initial source
                nextMinvaluePosition = sourceNode;
                while (nodesVisited.size() != column) {
                    //    System.out.println("\n----------------------------------------
\n");

                    findNeighborNode(nextMinvaluePosition);
                    findAndUpdateParList(nextMinvaluePosition);

                    Integer i = 0;
                    int min = 1000;
                    for (Integer x : listOfMinEdge) {

                        if ((x != -1) && (x > 0)) {
                            if (x <= min && !(nodesVisited.contains(i)))
{
                                nextMinvalue = x;
                                nextMinvaluePosition = i;
                                min = x;
                            }
                        }
                        i++;
                    }
```

```
                    nodesVisited.add(nextMinvaluePosition);
                    //    System.out.println("\n----------------------------------------
\n");

                }
                connectionTableArray(sourceNode);
                gotResult = 0;
                while (gotResult != 1) {
                    findTableContent();
                    if (listOfNextNode.contains(0)) {
                        gotResult = 0;
                    } else {
                        gotResult = 1;
                    }
                }
                dispTableContent();

        }
                public static void dispTableContent(){
                System.out.println("The Connection table for Source Node ");
                System.out.println("-----------" + "\t" + "--------");
                System.out.println("Destination" + "\t" + " NextHop");
                System.out.println("-----------" + "\t" + "--------");
                for (int k = 0; k < column; k++) {
                    System.out.println("\n"    +    (k    +    1)    +    "\t\t"    +    "
"+listOfNextNode.get(k));
                }
                System.out.println("\n");
                System.out.println("\n---------------------------------------- \n");


                System.out.println("The Connection table for all the nodes are ");
                for(int k=0;k<column;k++){
                    System.out.print("**********");
                }
                System.out.println();
                System.out.print("|\t \t|");
                for(int k=0;k<column;k++){
                    System.out.print("\t"+ (k+1) +"\t|");
                }
                System.out.print("\n");
                for(int k=0;k<column;k++){
                    System.out.print("**********");
                }
```

```
                System.out.print("\n");
                for(int i=0;i<column;i++){
                        allNodeRotingTable(i);
                        System.out.print("|\t"+ (i+1) +"\t|");
                        for(int j=0;j<column;j++){
                                System.out.print("\t"+temp[j]+"\t|");
                        }
                        System.out.print("\n");
                }
                System.out.print("\n");
                System.out.print("\n");

        }

                public static void allNodeRotingTable(int c)
                {
                        nextMinvaluePosition = 0;
                        nextMinvalue = 0;
                        listOfMinEdge.clear();
                        listofParentNodes.clear();
                        nodesVisited.clear();
                        listOfNextNode.clear();
                        // initializing current list array
                        initCurrRow(sourceNode);
                        // make source node as visited
                        nodesVisited.add(sourceNode);
                        // initialize the parent list
                        initParentList(sourceNode);

                        nextMinvaluePosition = sourceNode;
                        while (nodesVisited.size() != column) {
                                // System.out.println("\n----------------------------------
------- \n");

                                findNeighborNode(nextMinvaluePosition);
                                findAndUpdateParList(nextMinvaluePosition);

                                Integer i = 0;
                                int min = 1000;
                                for (Integer x : listOfMinEdge) {

                                        if ((x != -1) && (x > 0)) {
                                                if (x <= min
&& !(nodesVisited.contains(i))) {
                                                        nextMinvalue = x;
```

```
                                        nextMinvaluePosition = i;
                                        min = x;
                                    }
                                }
                                i++;
                            }

                            nodesVisited.add(nextMinvaluePosition);
                        }
                        connectionTableArray(sourceNode);
                        gotResult = 0;
                        while (gotResult != 1) {
                            findTableContent();
                            if (listOfNextNode.contains(0)) {
                                gotResult = 0;
                            } else {
                                gotResult = 1;
                            }
                        }
                        temp=new int[column];
                        for(int i=0; i<column;i++){
                            temp[i]=0;
                        }
                        // printing the connection table for the node
                        for (int k = 0; k < column; k++) {
                            temp[k]=listOfNextNode.get(k);
                        }

                    }

// finding next node from parentlist to identify the table content
public static void findTableContent() {
        int num = 0;
        int i = 0;

        for (int m = 0; m < column; m++) {
            if (listOfNextNode.get(m) == 0) {
                num = listofParentNodes.get(m);

                i = 0;
                while (i != 1) {
                    if (listOfNextNode.get(num) == (num + 1)) {
                        listOfNextNode.remove(m);
                        listOfNextNode.add(m, num + 1);
                        i = 1;
```

```
                                        } else {

                                                num = listOfNextNode.get(num);
                                                if (num == 0) {
                                                        listOfNextNode.remove(m);
                                                        listOfNextNode.add(m, 0);
                                                        i = 1;
                                                } else {
                                                        num--;
                                                }
                                        }
                                }

                        }
                }

        }

        // used in finding next node from the parent list
        public static void connectionTableArray(int sourceNode) {
                for (int k = 0; k < column; k++) {
                        if (sourceNode == listofParentNodes.get(k)) {
                                listOfNextNode.add(k + 1);
                        } else {
                                if (-1 == listofParentNodes.get(k))
                                        listOfNextNode.add(-1);
                                else
                                        listOfNextNode.add(0);
                        }
                }
        }


        // Finding final parent list for source node.
        public static void findAndUpdateParList(int nextMinvaluePosition) {
                int y = 0;

                int nextMin = 0;
                for (int j = 0; j < listofAdjNodes.size(); j++) {
                        y = listofAdjNodes.get(j);

                        nextMin = fileData[nextMinvaluePosition][y] +
nextMinvalue;

                        if (nextMin < listOfMinEdge.get(y)) {
```

```
                        listOfMinEdge.add(y, nextMin);
                        listOfMinEdge.remove(y + 1);
                        listofParentNodes.add(y, nextMinvaluePosition);
                        listofParentNodes.remove(y + 1);
                }
        }
}

// finding neighbors of current node
public static void findNeighborNode(int nextMinvaluePosition) {
        int x;
        listofAdjNodes.clear();
        for (int j = 0; j < column; j++) {
                x = fileData[nextMinvaluePosition][j];
                if (nodesVisited.contains(j)) {
                        // nothing , don't add the visited node;
                } else {
                        if ((x != -1) && (x > 0)) {
                                listofAdjNodes.add(j);

                        }

                }//else

        }//for
}


// finding shorest path and cost
public static void dispShortestPathandCost() {

        // coding is based on assumption destinationRouter index
        // 0,1,2.., not 1,2,3
        destNode = destNode - 1;
        int leastCost = 0;
        int currNode = sourceNode;
        int findNode = 0;
        Stack<Integer> lifo = new Stack<Integer>();
        if (listofParentNodes.get(destNode) == currNode) {
                leastCost = leastCost + fileData[currNode][destNode];
                lifo.push(destNode);

        } else {
                if (listofParentNodes.get(destNode) == -1) {
```

```
                        // skip
                    } else {
                        int i = 0;
                        findNode = destNode;
                        int currSource;
                        while (i != 1) {

                            lifo.push(findNode);
                            currSource = listofParentNodes.get(findNode);
                            leastCost = leastCost + fileData[currSource][findNode];

                            if (listofParentNodes.get(currSource) == currNode) {
                                leastCost = leastCost + fileData[currNode][currSource];
                                lifo.push(currSource);
                                i = 1;
                            } else {
                                findNode = currSource;

                            }
                        }//while
                    }//inner-else
                }//outer-else

                lifo.push(currNode);

                System.out.print("\nThe shortest path from the router "
                        + (sourceNode + 1) + " to router " + (destNode + 1)
                        + " is : ");

                if (lifo.size() == 1) {
                    System.out.print(" NO PATH ");
                } else {

                    while (!lifo.empty()) {
                        int x = lifo.pop();
                        x++;
                        System.out.print(x);
                        if (!lifo.empty())
                            System.out.print('-');
                    }//while
                }//else
```

```
                System.out.print(" || The total cost is: " + leastCost);

        }

        public static void modifytopology(){
                scan = new Scanner(System.in);
                int option;
                System.out.println("Modifying the Topology");
                System.out.println("Please select your option of Modification");
                System.out.println("(1)Add a Node");
                System.out.println("(2)Delete a Node");
                option=scan.nextInt();
                switch(option){
                        case 1:
                                //Adding a node
                                System.out.println("Enter the place you want to add
the new node");

                                try {
                                        nodeToInsert=scan.nextInt();

                                        if(nodeToInsert > column+1 ||
nodeToInsert<1)
                                        {
                                                System.out.println("Out of Range");
                                                break;
                                        }
                                        nodeToInsert=nodeToInsert-1;
                                        inArray=new int[column+1];
                                        System.out.println("Enter the routing values
for the new node");
                                        for(int i=0;i<column+1;i++){
                                                inArray[i]=scan.nextInt();
                                        }
                                        Insertnode();

                                } catch (Exception e) {
                                        System.err.print("Out of Range\n");
                                        System.out.println();
                                }

                                break;
                        case 2:
                                //Deleting a node
```

```
                        System.out.println("Enter The Node number to be deleted");
                        try{
                        nodeToDelete=scan.nextInt();
                        if(nodeToDelete > column || nodeToDelete<1)
                        {
                                System.out.println("Wrong Router Number");
                                break;
                        }
                        nodeToDelete=nodeToDelete-1;
                        Deletenode();
                        }catch (Exception e) {
                                System.err.print("Wrong Router Number\n");
                                System.out.println();
                        }
                        break;
                default:
                        System.out.println("Enter the Right Choice");
        break;
                }
        }

        //Inserting a node
        public static void Insertnode(){

                int OldNodevalue;
                OldNodevalue=column;
                column=column+1;
                int[][]tempNodeValue=new int[column][column];

                for(int i=0;i<column;i++){
                        for(int j=0;j<column;j++){
                                tempNodeValue[i][j]=0;
                        }
                }
                for(int i=0;i<OldNodevalue;i++){
                for(int j=0;j<OldNodevalue;j++){
                                tempNodeValue[i][j]=fileData[i][j];
                        }
                }

                if(nodeToInsert>0){
                        for(int i=0;i<nodeToInsert;i++){
```

```
                        for(int j=column-1;j>nodeToInsert;j--){
                            tempNodeValue[i][j]=tempNodeValue[i][j-
1];

                            tempNodeValue[j][i]=tempNodeValue[i][j];
                        }
                    }
                }
                for(int i=column-2;i>nodeToInsert;i--){
                    for(int j=column-1;j>i;j--){
                        tempNodeValue[i][j]=tempNodeValue[i-1][j-1];
                        tempNodeValue[j][i]=tempNodeValue[i][j];
                    }
                }
                for(int i=0;i<column;i++){
                    tempNodeValue[nodeToInsert][i]=inArray[i];
                    tempNodeValue[i][nodeToInsert]=inArray[i];
                }
                tempNodeValue[nodeToInsert][nodeToInsert]=0;
                fileData=new int[column][column];

                for(int i=0;i<column;i++){
                    for(int j=0;j<column;j++){
                        fileData[i][j]=tempNodeValue[i][j];
                    }
                }
                System.out.println("The Total Number of Routers present after
addition of a node are:"+column);
                //Printing the modified
                System.out.println("The following is the Routing Table after
Addition of a node");
                dispTopology(fileData);
                System.out.print("\n");
                    System.out.print("\n");

                System.out.print("\n");
                System.out.print("\n");

            }

            //Deleting a node
            public static void Deletenode(){

                int OldNodevalue;
                OldNodevalue=column;
                int[][]tempNodeValue=new int[column][column];
```

```
                    for(int i=0;i<column;i++){
                        for(int j=0;j<column;j++)
                        {
                            tempNodeValue[i][j]=fileData[i][j];
                        }
                    }
                    column=column-1;

                    for(int i=nodeToDelete;i<OldNodevalue-1;i++){
                        tempNodeValue[i][i]=tempNodeValue[i+1][i+1];
                    }
                    if(nodeToDelete>0){
                        for(int i=0;i<nodeToDelete;i++){
                            for(int j=nodeToDelete;j<OldNodevalue-1;j++){

    tempNodeValue[i][j]=tempNodeValue[i][j+1];
                                tempNodeValue[j][i]=tempNodeValue[i][j];
                            }
                        }
                    }
                    for(int i=nodeToDelete;i<OldNodevalue-2;i++){
                        for(int j=i+1;j<OldNodevalue-1;j++){
                            tempNodeValue[i][j]=tempNodeValue[i+1][j+1];
                            tempNodeValue[j][i]=tempNodeValue[i][j];
                        }
                    }
                    for(int i=0;i<column;i++){
                        for(int j=0;j<column;j++){
                            fileData[i][j]=tempNodeValue[i][j];
                        }
                    }
                    System.out.println("The Total Number of Routers present after deletion of a node are:"+column);
                    //Printing the modified
                    System.out.println("The following is the Routing Table after Deletion of a node");
                    dispTopology(fileData);
                        System.out.print("\n");

                    System.out.print("\n");
                    System.out.print("\n");
                }
```

References

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