

Link State Routing Protocol Using Dijkstra Algorithm

Submitted by:

Team Members Name	CWID	Section
Akash Agarwal	A20298520	Live
Nickhil Revu	A20299434	Live

CONTENT:

- Abstract
- Introduction
- Detailed description of link state routing Algorithm
- Design
- Description of Dijkstra Algorithm
 - Pseudo Code
- Shortest path calculation
 - Pseudo Code
- Test Report
- Instances

Abstract:

In this project we simulate a Link State Routing Algorithm for a given network. Using the simulation we measure the total cost associated with a path from source to destination router. The simulator determines cost by calculating the optimal path based on our Link State Routing Algorithm.

Introduction:

There are two popular classes of routing protocols in use for communication between networks, one is the Link State Routing Protocol and the other is the Distance – Vector Routing Protocol. The advantage of using LSRP over DVRP is that the configuration and algorithm of LSRP is much easier.

Instead of having each node share its routing table with its neighbors which DVRP does, in LSRP the only information passed between nodes is connectivity related. To calculate the cost and determine the best path Link State Routing Protocol considers the path with lowest cost. The algorithm calculates cost related to each router and then it is stored in the router's routing table and then these costs are used by the algorithm to calculate the best path from source and destination.

Link State Knowledge:

The topology must be dynamic, representing the latest state of each node and each link.

In Link State Routing, four set of actions are required:-

- Creation of Link State Packet (LSP)
- Flooding of LSPs in an efficient and reliable way.
- Formation of shortest path tree for every node.
- Calculation of shortest path tree based on the shortest path tree.

A LSP can carry large amount of information:

- Node identity: are used to make the topology.
- List of links: are used to make the topology with the help of Node Identity.
- Sequence number: facilitates the flooding of LSPs.
- Age: prevents LSPs to remain in the domain for long time.

LSPs are generated on only two occasions:

- If there is a change in the topology of the domain: Updated information of a node in the domain is primary requirement of LSRP and generation of LSPs is the quickest way to update the topology.
- On a periodic basis: this is done to ensure that the old information is removed from the domain. This is not a mandatory dissemination.

Design Report:

The implementation of Dijkstra Algorithm is listed below:

Variables used:

weight[][]: is a multidimensional array used by network configuration and it reads the network.txt file.

cost[]: is an array that stores distance between two intermediate nodes and is initially set to 65534.

pred[]: is an array that sets the node found after calculation of the shortest path to the first current node in that path. This array is also initialized to 65534.

visited[]: array that sets the visited nodes found after the calculation of the shortest path to flag.

path[]: it is an array that stores the shortest path between any two intermediate nodes.

curr_dist[]: current distance for the current node.

new_dist: it keeps the track of the distance of the node being traversed.

small_dist: it records the smallest distance from the node being traversed.

Methods defined:

read(): reads the input file and builds a 2D array for the routing table.

display_result(): this methods calculate the shortest path between the source and destination based on precede array.

dk(): implements the dijkstra shortest path algorithm and calculates the minimum cost between two routers.

Description for dijkstra algorithm:

In the dk function, the flags for array visit is initialized to zero and the arrays cost and precd are set to 65534. The flag visit for current node which is source node at the beginning is set to 1.

The variable curr_dist[] is set to distance of current node and for current node a loop will run where it will compare distances to other nodes and will traverse the loop and after it has found the next node, it will set the next node as the current node and the precede of this next node would be set to the current node and the same procedure would be repeated again until the current node is the destination node.

Psuedo Code:

```

cost[s]=0;
current=s;
visited[current]=1;
while(current!=d)
{
    currr_dist=cost[current];
    small_dist=65534;
    for(i=1;i<=n;i++)
    {
        if(visited[i]==0)
        {
            new_dist=currr_dist+weight[current][i];
            if(new_dist<cost[i])
            {
                cost[i]=new_dist;
                pred[i]=current;
            }
            if(cost[i]<small_dist)
            {
                small_dist=cost[i];
                k=i;
            }
        }
    }
    current=k;
    visited[current]=1;
}
}

```

Shortest path calculation:

The path to the destination router is traced by starting from source routing table and traversing through all the intermediate routing tables. display_result() prints the shortest path and also routing table depending on user choice .

Pseudo Code:

```

while(pred[i]!=s)
{
    j=pred[i];
    i=j;
    path[fin]=i;
    fin++;
}

path[fin]=s;
if(check==0)
{
    for(i=fin;i>0;i--)
        d1=weight[path[i]][path[i-1]];
    if(d1<65000)
    {
        System.out.println("\n Shortest Path:\n");
        for(i=fin;i>0;i--)
            System.out.print(path[i]+" ----> ");
        System.out.print(path[i]+"\n\n");
        for(i=fin;i>0;i--)
            System.out.println( path[i]+"->" +path[i-1]+"\t\t with cost =" +weight[path[i]][path[i-1]]);
        System.out.println("\nFor total cost = "+cost[d]);
    }
    else
        System.out.println("\n\n There is no path between the given Source and Destination\n");
}
else
{
    for(i=fin;i>0;i--)
        d1=weight[path[i]][path[i-1]];
    System.out.print(" |");
    if(d1>64999)
    {
        if (d<10)
            System.out.print(" ");
        System.out.println("    "+d+" -->  -\t |");
    }
    else
    {if (d<10)
        System.out.print(" ");
        System.out.println("    "+d+" -->  "+path[fin-1]+"\t |");
    }
}
}
}

```

Test Report:**TestCase#1:**

Initiation: our program shall compile successfully.

Input: javac CS_542.java

Java CS_542

Expected result: our routing console is displayed successfully.

Result: Pass

Screen Shot:

```

C:\> Command Prompt - java CS_542
E:\CN1\Project>javac CS_542.java
E:\CN1\Project>java CS_542

LINK STATE ROUTING PROTOCOL USING DIJKSTRA ALGORITHM

-----Submitted by:-----
Akash Agarwal A20298520
Nickhil Revu A20299434
-----

-----Routing Console-----
1 - Load File
2 - Build Routing Table For Router
3 - Compute Optimal Path and Minimum Cost
4 - Exit
-----
Enter Your Choice :

```

TestCase#2:

Initiation: network.txt file shall load successfully.

Input: Choose 1; input the file path.

Expected result: the program shall display the network.txt file and prompt the user.

Result: Pass

Screen Shot:

```

C:\> Command Prompt - java CS_542
Enter Your Choice :
1
Please load original routing table data file:
C:\Users\Akash\Desktop\network.txt
Original routing table is as follows:
0 2 5 1 -1
2 0 8 1 9
5 8 0 -1 4
1 7 -1 0 10
-1 9 4 10 0

File Loaded
Number of Routers=5

-----Routing Console-----
1 - Load File
2 - Build Routing Table For Router
3 - Compute Optimal Path and Minimum Cost
4 - Exit
-----

```

TestCase#3:

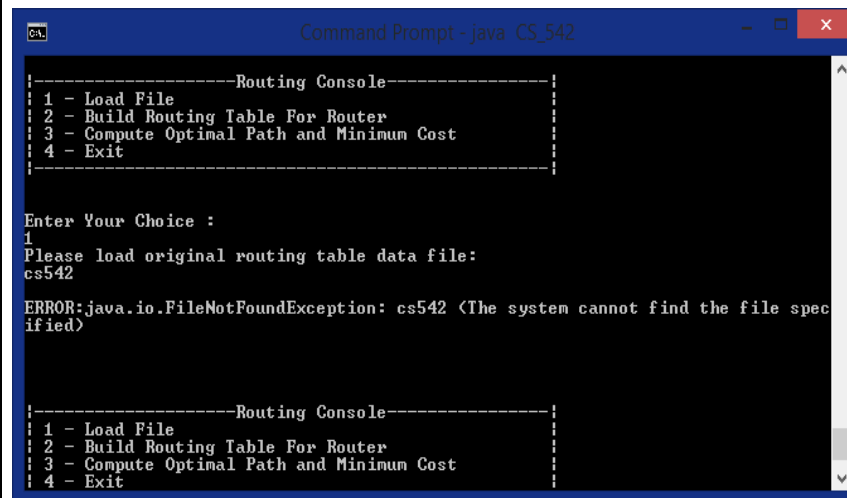
Initiation: the input file will not be loaded.

Input: Choose 1; input invalid file name like a character.

Expected result: the program shall raise an Exception.

Result: Pass

Screen Shot:

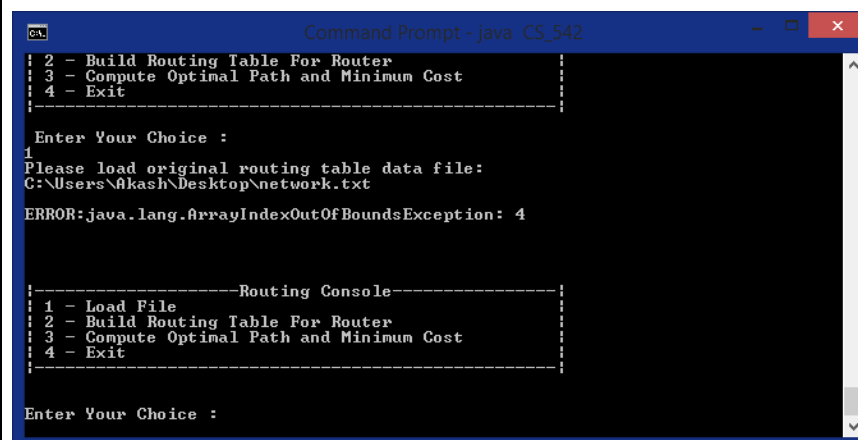
(FileNotFoundException Handler)

```
Command Prompt - java CS_542

-----Routing Console-----
1 - Load File
2 - Build Routing Table For Router
3 - Compute Optimal Path and Minimum Cost
4 - Exit

Enter Your Choice :
1
Please load original routing table data file:
cs542
ERROR:java.io.FileNotFoundException: cs542 (The system cannot find the file specified)

-----Routing Console-----
1 - Load File
2 - Build Routing Table For Router
3 - Compute Optimal Path and Minimum Cost
4 - Exit
```

(ArrayIndexOutOfBoundsException Handler)

```
Command Prompt - java CS_542

2 - Build Routing Table For Router
3 - Compute Optimal Path and Minimum Cost
4 - Exit

Enter Your Choice :
1
Please load original routing table data file:
C:\Users\Akash\Desktop\network.txt
ERROR:java.lang.ArrayIndexOutOfBoundsException: 4

-----Routing Console-----
1 - Load File
2 - Build Routing Table For Router
3 - Compute Optimal Path and Minimum Cost
4 - Exit

Enter Your Choice :
```


TestCase#4:

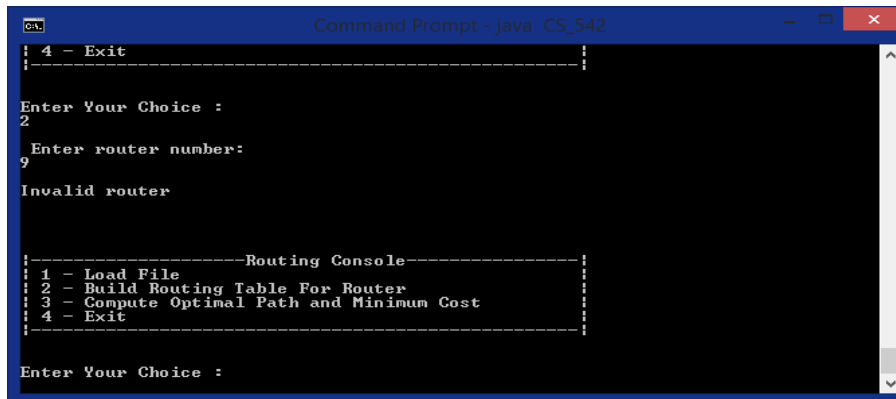
Initiation: display routing table

Input: Choose 2; enter invalid router number

Expected result: the program shall raise an Exception.

Result: Pass

Screen Shot:



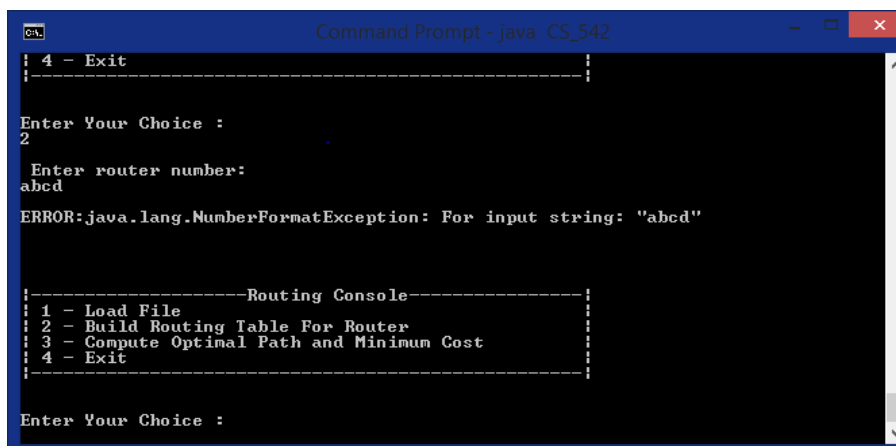
```
Command Prompt - java CS_542

| 4 - Exit
|-----|

Enter Your Choice :
2
Enter router number:
9
Invalid router

|-----Routing Console-----|
| 1 - Load File
| 2 - Build Routing Table For Router
| 3 - Compute Optimal Path and Minimum Cost
| 4 - Exit
|-----|

Enter Your Choice :
```

(NumberFormatException Handler)

```
Command Prompt - java CS_542

| 4 - Exit
|-----|

Enter Your Choice :
2
Enter router number:
abcd
ERROR:java.lang.NumberFormatException: For input string: "abcd"

|-----Routing Console-----|
| 1 - Load File
| 2 - Build Routing Table For Router
| 3 - Compute Optimal Path and Minimum Cost
| 4 - Exit
|-----|

Enter Your Choice :
```

TestCase#5:

Initiation: display routing table

Input: Choose 2; enter router number.

Expected result: the routing table for the selected router number is displayed.

Result: Pass

Screen Shot:

```

Command Prompt - java CS_542
Enter Your Choice :
2
Enter router number:
1
Routing Table for router number
1
!Destination --> Next-hop!
-----
!      2  -->  2      !
!      3  -->  3      !
!      4  -->  4      !
!      5  -->  3      !
!-----!

-----Routing Console-----
1 - Load File
2 - Build Routing Table For Router
3 - Compute Optimal Path and Minimum Cost
4 - Exit

```

TestCase#6:

Initiation: optimal path from source to destination.

Input: Choose 3; enter source and destination.

Expected Result: display the shortest path along with the cost.

Result: Pass

Screen Shot:

```

Command Prompt - java CS_542
Enter Your Choice :
3
Enter the source node 1 to 5
1
Enter the destination node 1 to 5
5
Shortest Path:
1 -----> 3 -----> 5
1->3          with cost =5
3->5          with cost =4
For total cost = 9

-----Routing Console-----
1 - Load File
2 - Build Routing Table For Router
3 - Compute Optimal Path and Minimum Cost

```

TestCase#7:

Initiation: optimal path from source to destination.

Input: Choose 3; enter invalid source and destination.

Expected Result: the program shall raise an Exception.

Result: Pass

Screen Shot:

(NumberFormatException Handler)

```

Command Prompt - java CS_542

| 4 - Exit |
|-----|

Enter Your Choice :
3

Enter the source node 1 to 5
a

ERROR:java.lang.NumberFormatException: For input string: "a"

|-----Routing Console-----|
| 1 - Load File |
| 2 - Build Routing Table For Router |
| 3 - Compute Optimal Path and Minimum Cost |
| 4 - Exit |
|-----|

Enter Your Choice :

```

(ArrayIndexOutOfBoundsException Handler)

```

Command Prompt - java CS_542

Enter Your Choice :
3

Enter the source node 1 to 5
-1

Enter the destination node 1 to 5
-1

ERROR:java.lang.ArrayIndexOutOfBoundsException: -1

|-----Routing Console-----|
| 1 - Load File |
| 2 - Build Routing Table For Router |
| 3 - Compute Optimal Path and Minimum Cost |
| 4 - Exit |
|-----|

Enter Your Choice :

```

(Invalid Router Number)

```

Command Prompt - java CS_542

Enter Your Choice :
3

Enter the source node 1 to 5
1

Enter the destination node 1 to 5
9

Invalid Source or Destination

|-----Routing Console-----|
| 1 - Load File |
| 2 - Build Routing Table For Router |
| 3 - Compute Optimal Path and Minimum Cost |
| 4 - Exit |
|-----|

Enter Your Choice :

```

TestCase#8:

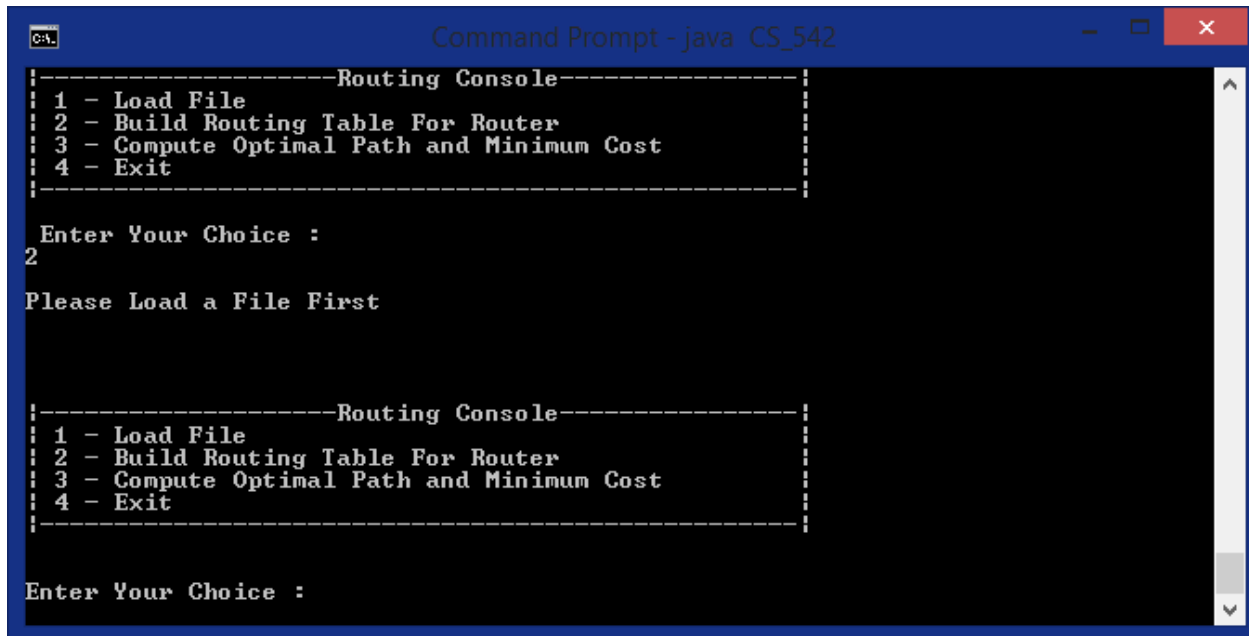
Initiation: Load file first.

Input: Choose 2 at start-up.

Expected Result: an error is raised and the program shall notify the user.

Result: Pass

Screen Shot:



```
Command Prompt - java CS_542

-----Routing Console-----
1 - Load File
2 - Build Routing Table For Router
3 - Compute Optimal Path and Minimum Cost
4 - Exit

Enter Your Choice :
2
Please Load a File First

-----Routing Console-----
1 - Load File
2 - Build Routing Table For Router
3 - Compute Optimal Path and Minimum Cost
4 - Exit

Enter Your Choice :
```

TestCase#9:

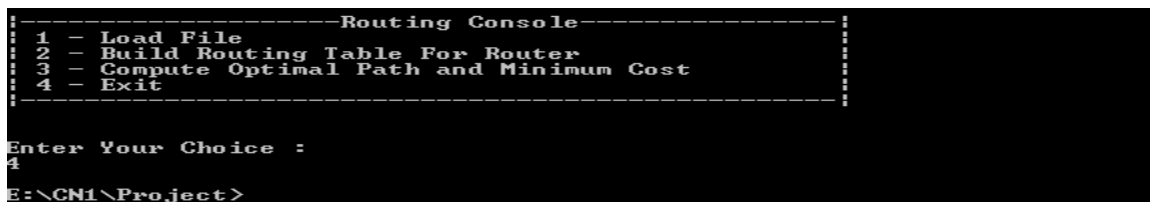
Initiation: exit our program.

Input: Choose 4.

Expected Result: the program shall exit.

Result: Pass.

Screen Shot:



```
-----Routing Console-----
1 - Load File
2 - Build Routing Table For Router
3 - Compute Optimal Path and Minimum Cost
4 - Exit

Enter Your Choice :
4
E:\CN1\Project>
```

Instances:

1.) Input Matrix:

```

0 2 -1 -1 3 -1 -1 -1 -1 -1
2 0 5 -1 -1 -1 -1 -1 -1 -1
-1 5 0 10 -1 2 4 -1 -1 -1
-1 -1 10 0 -1 -1 -1 9 -1 -1
3 -1 -1 -1 0 1 -1 -1 7 -1
-1 -1 2 -1 1 0 6 -1 -1 -1
-1 -1 4 -1 -1 6 0 8 -1 -1
-1 -1 -1 9 -1 -1 8 0 1 -1
-1 -1 -1 -1 7 -1 -1 1 0 5
-1 -1 -1 -1 -1 -1 -1 15 0

```

Enter your choice: 2

Enter router number: 2

Routing table for router number: 2

1→1

3→3

4→3

5→1

6→1

7→3

8→1

9→1

10→1

Enter your choice: 3

Enter the source node 1 to 10: 3

Enter the destination node 1 to 10: 8

Shortest Path: 3→6→5→9→8

3→6 with cost =2; 6→5 with cost=1; 5→9 with cost=7; 9→8 with cost =1; for total cost = 11

2.) Input Matrix:

```

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

```

Enter your choice: 2

Enter router number: 1

Routing table for router number: 1

2→2

3→3

4→3

5→5

6→2

7→7

8→9

9→9

10→10

Enter your choice: 3

Enter the source node 1 to 10: 9

Enter the destination node 1 to 10: 6

Shortest Path: 9→2→6

9→2 with cost =4; 2→6 with cost=1

For total cost = 5

3.) Input Matrix:

```

0 5 4 -1 -1 -1 -1 -1 -1 8
5 0 3 10 3 -1 -1 -1 -1 9
4 3 0 -1 -1 -1 -1 -1 -1 -1
-1 10 -1 0 -1 -1 -1 -1 -1 6
-1 3 -1 -1 0 8 -1 -1 -1 -1
-1 -1 -1 -1 8 0 15 10 -1 -1
-1 -1 -1 -1 -1 15 0 -1 1 -1
-1 -1 -1 -1 -1 10 -1 0 12 -1
-1 -1 -1 -1 -1 -1 1 12 0 -1
8 9 -1 6 -1 -1 -1 -1 -1 0

```

Enter your choice: 2

Enter router number: 7

Routing table for router number: 7

1→6

2→6

3→6

4→6

5→6

6→6

8→9

9→9

10→6

Enter your choice: 3

Enter the source node 1 to 10: 7

Enter the destination node 1 to 10: 4

Shortest Path: 7→6→5→2→4

7→6 with cost =15; 6→5 with cost=8; 5→2 with cost 3; 2→4 with cost 10; for total cost = 36

4.) Input Matrix:

```

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

```

Enter your choice: 2

Enter router number: 11

Routing table for router number: 11

```

1→7
2→7
3→7
4→7
5→7
6→6
7→7
8→8
9→12
10→7
12→12
13→13
14→16
15→15
16→16

```

Enter your choice: 3

Enter the source node 1 to 16: 1

Enter the destination node 1 to 16: 14

Shortest Path: 1→16→14

1→16 with cost =2; 16→14 with cost=1

For total cost = 3

5.) Input Matrix:

```

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

```

Enter your choice: 2

Enter router number: 11

Routing table for router number: 11

1→3

2→2

3→3

4→4

5→5

6→--

7→7

8→8

9→9

10→10

11→11

12→--

13→13

14→--

15→15

16→16
17→17
18→3
19→19
20→20
22→22
23→23
24→--
25→--

Enter your choice: 3

Enter the source node 1 to 25:21

Enter the destination node 1 to 25:12

There is no path between the given Source and Destination.

Enter your choice: 3

Enter the source node 1 to 25: 25

Enter the destination node 1 to 25:2

Shortest Path: 25→2

25→2 with cost =1

For total cost = 1