Project Title: Simulation of Link State Routing Algorithm

Subject: CS-542 Computer Network I: Fundamentals

Name: Rohan Suresh Jain

CWID: A20378201

Seat Number: 39

Table of Contents

Introduction

Link State Routing Algorithm: is used in packet switching for network communication. Examples of Link State routing algorithm includes open shortest path first (OSPF) and intermediate system to intermediate system(IS-IS). Link state routing algorithm is performed by switching every router in the network. In Link State routing algorithm, the only information passed between the routers is connectivity related. In this algorithm every router creates a map of connectivity between the routers by representing which routers is connected to the other routers. Each router will calculate the best path from itself to all other routers in the network. Routing table is obtained from collection of best path. Shortest path is calculated using Dijkstra's Algorithm.

Dijkstra's Algorithm: is used to find the shortest path between the router in the network. It is also called as single sources shortest path problem. It is used to solve the single-source shortest path problem in weighted graph. Link State routing protocol applies Dijkstra's algorithm to find the best path route. It uses calculated cost along each path, from source to destination to find the total cost of the path.

Pseudo Code of Dijkstra's Algorithm:

```
dist[s] $\leftarrow 0$                                   (distance to source vertex is zero)
for all $v \in V - \{s\}$
        do dist[v] $\leftarrow \infty$                   (set all other distances to infinity)
$S \leftarrow \varnothing$                               (S, the set of visited vertices is initially empty)
$Q \leftarrow V$                                         (Q, the queue initially contains all vertices)
while $Q \neq \varnothing$                               (while the queue is not empty)
do  u $\leftarrow$ mindistance(Q,dist)                   (select the element of Q with the min. distance)
        $S \leftarrow S \cup \{u\}$                      (add u to list of visited vertices)
        for all $v \in$ neighbors[u]
            do if dist[v] > dist[u] + w(u, v)            (if new shortest path found)
                 then d[v] $\leftarrow$d[u] + w(u, v)    (set new value of shortest path)

                                                         (if desired, add traceback code)

return dist
```

Design and Work Flow

Project Description: This project is to simulate link-state routing algorithm. Program requires input file containing network information from user and print the connection table of inputted source router. Dijkstra' algorithm is applied to print the shortest path along with cost between the source and destination. Program also allows user to put down/remove an inputted router and print the new shortest path between source and destination along with the cost. It also helps to find the best router which has shortest path to all other routers in the network.

Platform of project: **Ubuntu OS**
Language used: **C Language**
Compiler: **Gcc Compiler**

Implemented **C code** is based on **user input** and it is **menu driven** program.
Functions of the code:

1. **Create Network topology**
2. **Build a connection table**
3. **Shortest path to destination**
4. **Modify a topology**
5. **Best router for broadcast**
6. **Exit**

User is first required to compile the code using **Gcc compiler** and then run it. Based on user selection, code will execute the corresponding function.

1. **Create Network Topology:** Here user is required to input the network topology matrix data file as input and code will execute **matrix_func()** function to accept the input file and print the matrix on the screen.

2. **Build a connection table:** If user calls this function without inputting the matrix file and user will be prompted to input the matrix file using option '1'. Here user is required to input the source router and **dij(startnode,1)** function will be called to print the router's connection table.

3. **Shortest path to destination:** Here user is required to enter the destination router. Source router is used from option '1'. Function **path_calc(startnode,destnode)** will be executed to find the shortest path from source to destination along with its cost.

4. **Modify a topology:** Here user is asked to input which router to be removed/made down in the inputted matrix i.e. (rows and column of that router will be changed to -1) and functions **dij(startnode,1) and path_calc(startnode,destnode)** are called again to find the shortest path from source to destination along with the cost.

5. **Best router for broadcast:** This option will display the router which has shortest path to all other routers in the network along with their cost by calling **router_shortest_path()** function.

6. **Exit:** This options help user to quit the program.

User Manual

Project consists of following files:

1. **Project_Presentation.ppt** - This file contains the presentation of the project.
2. **Project_Manual**.**pdf -** This file contains the project operation manual.
3. **linkstate.c -** This file contains the source code of the project.
4. **linkstate -** This is the executable file of the **linkstate.c**
5. **6routers.txt, 7routers.txt, 10routers.txt** – These file are sample test input files which can be used as input for the program.

Code Execution Steps:

- Gcc compiler must be installed on the system on which code is going to be executed as code is written in C language.
- Source code(**linkstate.c**) and user input file(**6routers.txt/7routers.txt/10routers.txt**) must be in the **same source directory** while running the code.
- User can provide its own input file but it should be in **.txt** format and in same directory as of source code.
- User should invoke terminal (Linux OS) or command prompt (Windows) for execution of the code.
- To compile the code user should execute -> **gcc – o linkstate linkstate.c**.
- After compiling the code, user will get an executable file named as **linkstate**.
- To run the code user need to execute -> **./linkstate**
- The code will start running and user is required to enter the options for corresponding function to execute.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//globally defined variables which will be used in all the functions
int col=1;
int row=0,i,j;
int matrix[100][100];
int path[20],dist[20];

//function to accept the matrix from user and print it
void matrix_func()
{


FILE *files;
int ch;
char ff[30];

printf("\nInput: ");
scanf("%s",ff);
files=fopen(ff,"r");

do
{
ch = fgetc(files);
if(ch=='\n')
row++;
if (ch==' ' && row==0)
 {
    col++;
 }
} while(ch != EOF);
ch = fgetc(files);

printf("\n");
fclose(files);

files=fopen(ff,"r");
//scanning of user matrix
for(i=0;i<row;i++)
    {
              for(j=0;j<col;j++)
```

```
                        {
                        fscanf(files, "%d", &matrix[i][j]);
                        }
        }
    //printing of matrix
     for(i=0;i<row;i++)
        {
           for(j=0;j<col;j++)
           {
              printf("%d \t",matrix[i][j]);
           }
           printf("\n");
        }

    printf("\n");
    }

    //function to calculate shortest path from source to destination
    void path_calc(int start_node,int dest_node)
    {
        int right_path[col],count=1;
        int final_node = dest_node;
        right_path[0]=dest_node;
        //initializing previous node as same node for each router and saving previous node of
        each visited router
        while(path[dest_node]!=dest_node)
        {
                right_path[count]=path[dest_node];
                count++;
                int temp = path[dest_node];
                dest_node=temp;
        }
        //printing shortest path from source node to destination node.
        printf("\n\nPath between %d and %d is %d",start_node,final_node,start_node);
        for(i=count-1;i>=0;i--)
                printf("-%d",right_path[i]);
        printf(" with cost %d\n\n",dist[final_node]);

    }
    //function to remove the router and change the matrix accordingly
    void change_topology(int rem_node)
    {
        for(i=0;i<col;i++)
        {
                for(j=0;j<col;j++)
```

```
            {
                    //Setting rows and columns value as -1 for the router to be removed
                    if(i==rem_node || j==rem_node)
                            matrix[i][j]=-1;
            }
    }
}

//function which performs dijsktra's alorithm
void dij(int start_node, int print_flag)
{
    int v_node[col-1],temp_cost=0;
    //Initializing visited nodes, distance array and path
    for(i=0;i<col;i++)
    {
            v_node[i]=0;
            dist[i]=-1;
            path[i]=i;
    }
    //Setting the distance(cost) for source node as 0
    dist[start_node]=0;
    //Copying the values for routers that have direct link to source node from matrix array.
    for(i=0;i<col;i++)
    {
            dist[i]=matrix[start_node][i];
    }

    int next_node=0;
    //If start node is 0 then the next should be 1
    if(start_node==0)
            next_node=1;
    //Find the node with the least cost to start_node
    for(j=0;j<col;j++)
    {
            for(i=0;i<col;i++)
            {
                    //If node is start_node or a visited node then skip the iteration.
                    if(i==start_node || v_node[i]==1)
                    {
                            //printf("nothing %d",i);
                    }
                    else
                    {
                            if(dist[i]!=-1)
                            {
```

```
				if(dist[next_node]>dist[i] || dist[next_node]==-1 ||
v_node[next_node]==1)
				{
					next_node = i;

				}
			}
		}
	}
	v_node[next_node]=1;


	//Update the dist array based on comaprison with cost from next_node
	for(i=0;i<col;i++)
	{
		if(matrix[next_node][i]!=-1)
		{
			if(i==start_node || v_node[i]==1)
			{
			}
			else
			{
				temp_cost = dist[next_node]+matrix[next_node][i];
				if(dist[i]>temp_cost || dist[i]==-1)
				{
					dist[i]=temp_cost;
					//Setting previous node as the next node if cost is
less through next_node
					path[i]=next_node;

				}
		}
	}
}
}
if(print_flag==1)
{
//printing destination node and shortest distance from source node.
	printf("\nNode\tInterface");
	for(i=0;i<col;i++)
	{
		printf("\n%d",i);
		if(i==start_node)
			printf("\t-");
		else
```

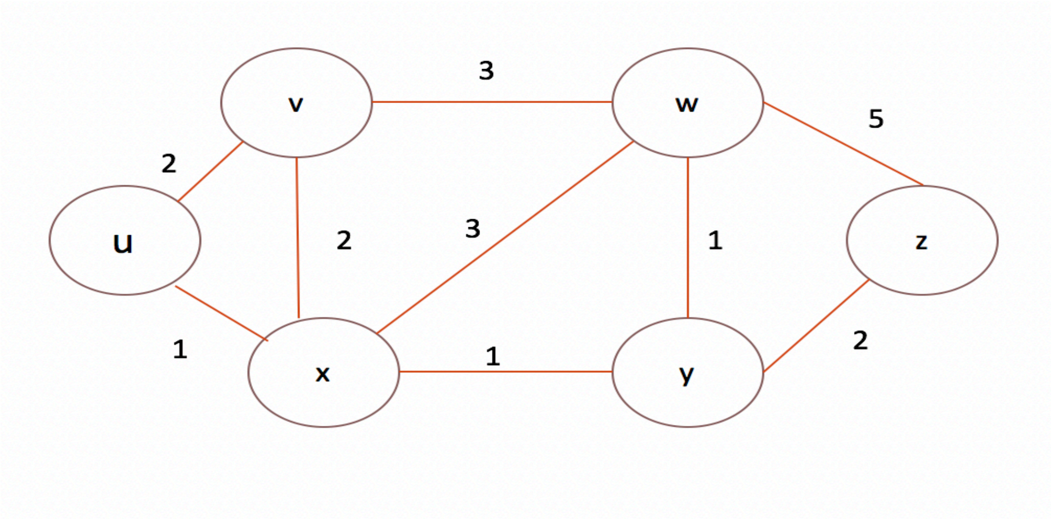
```
                                printf("\t%d",dist[i]);
                }
        }
}
//function to find shortes path from source to destination
void router_shortest_path()
{
    int temp_cost=0, final_cost[20];
    int l=0,m=0;
    for(l=0;l<col;l++)
    {
            dij(l,0);
            //calculating shortest path from one router to all other router
            for(m=0;m<col;m++)
                    {temp_cost=temp_cost+dist[m];}
            final_cost[l]=temp_cost;
            temp_cost=0;
    }
    int least_cost_router=0;
    for(l=0;l<col;l++)
    {
            if(final_cost[least_cost_router]>=final_cost[l])
                    least_cost_router=l;
    }
    printf("The router with shortest path to all other routers is %d with total cost
    %d\n\n",least_cost_router,final_cost[least_cost_router]);
}

int main()
{
int k, matrix_flag=0;
int startnode,destnode,remnode;
do
{
printf("\n");
printf("\nCS542- Simulator for Link State Routing Algorithm:\n");
printf("\n1. Create Network Topology\n2. Build a connection table\n3. Find shortest
    path\n4. Modify a topology\n5. Best router for broadcast\n6. EXIT\n");
printf("Enter your option: ");
scanf("%d",&k);
switch(k)
{
case 1:
    matrix_func();
    matrix_flag=1;
    break;
case 2:
    if(matrix_flag==0)
    {
            printf("\nPlease select option 1 and set the network topology");
            break;
    }
    printf("\nEnter a source node: ");
    scanf("%d", &startnode);
    //calling dijkstra’s algorithm
    dij(startnode,1);
    break;
case 3:
    printf("\nSource router is %d",startnode);
    printf("\nEnter a destination node: ");
    scanf("%d", &destnode);
    //calculating shortest path from source to destination
    path_calc(startnode,destnode);
    break;
case 4:
    printf("\nSelect the router to be removed: ");
    scanf("%d", &remnode);
    //calling function to remove the entered router
    change_topology(remnode);
    //caling dijkstra’s algorithm again
    dij(startnode,1);
    //calcuating shortest past
    path_calc(startnode,destnode);
    break;
case 5:
    //calculating shortest path from one router all other router
    router_shortest_path();
    break;
case 6:
    printf("EXITING -> GOOD BYE\n\n");
    //exiting
    return(0);
    break;
default :
printf("BYE BYE\n\n");
}
}while(k!=0);
return 0; }
```

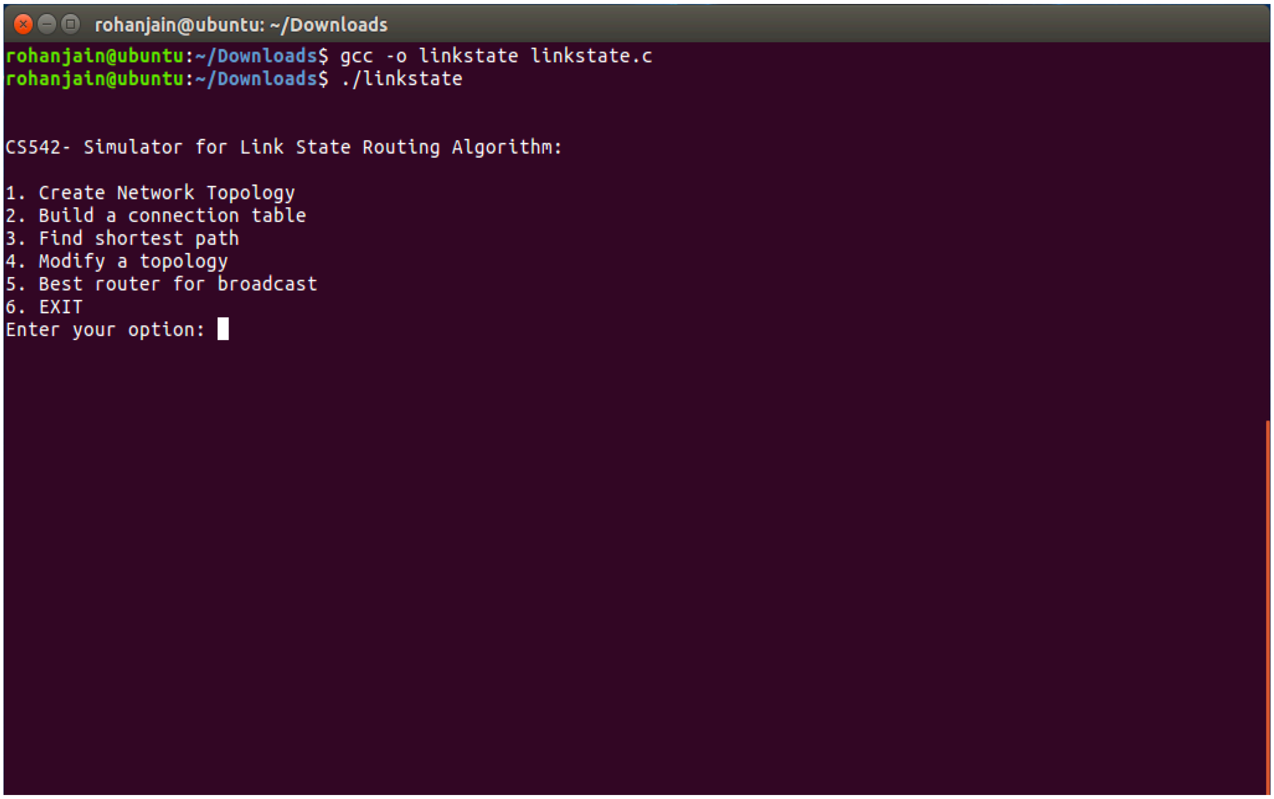
Screenshots of Results

Below are screenshots of code execution.

Following code is executed considering the given below example.

For 6 Routers:

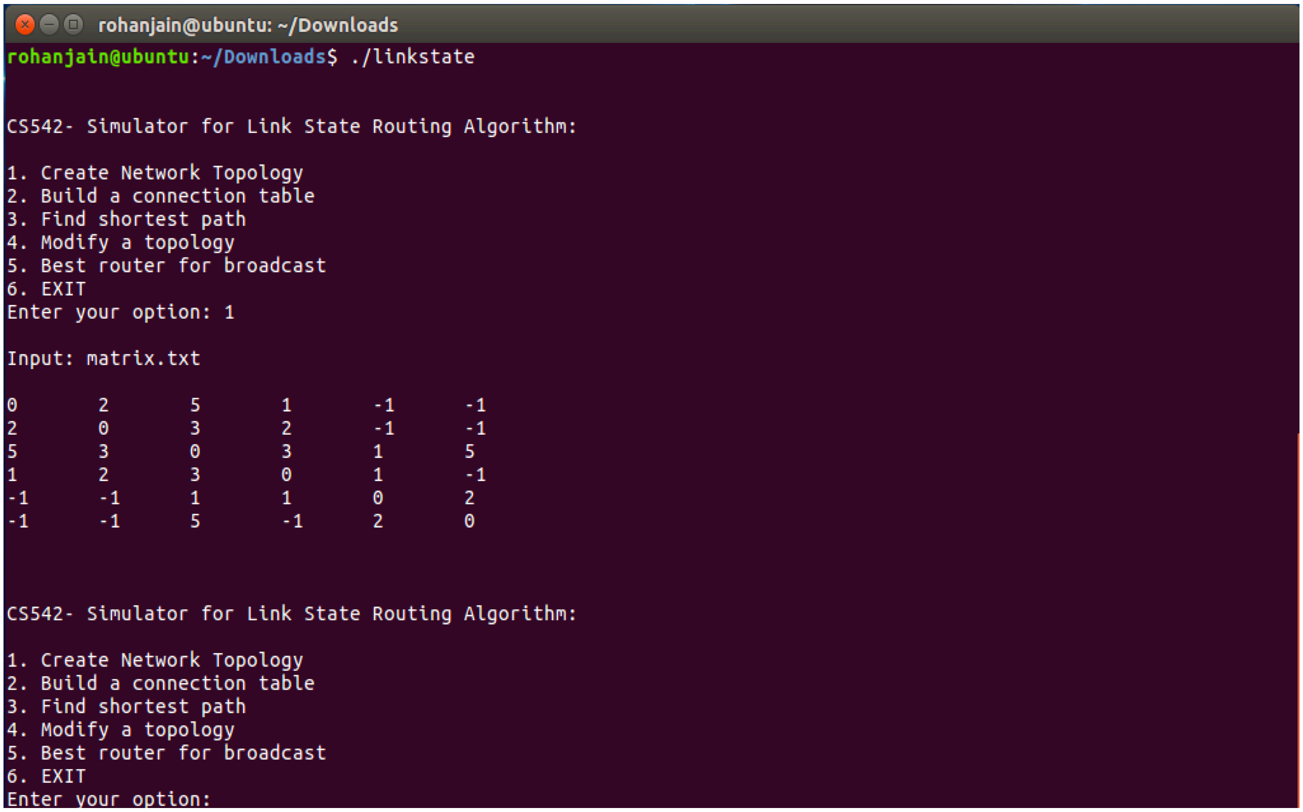




rohanjain@ubuntu: ~/Downloads

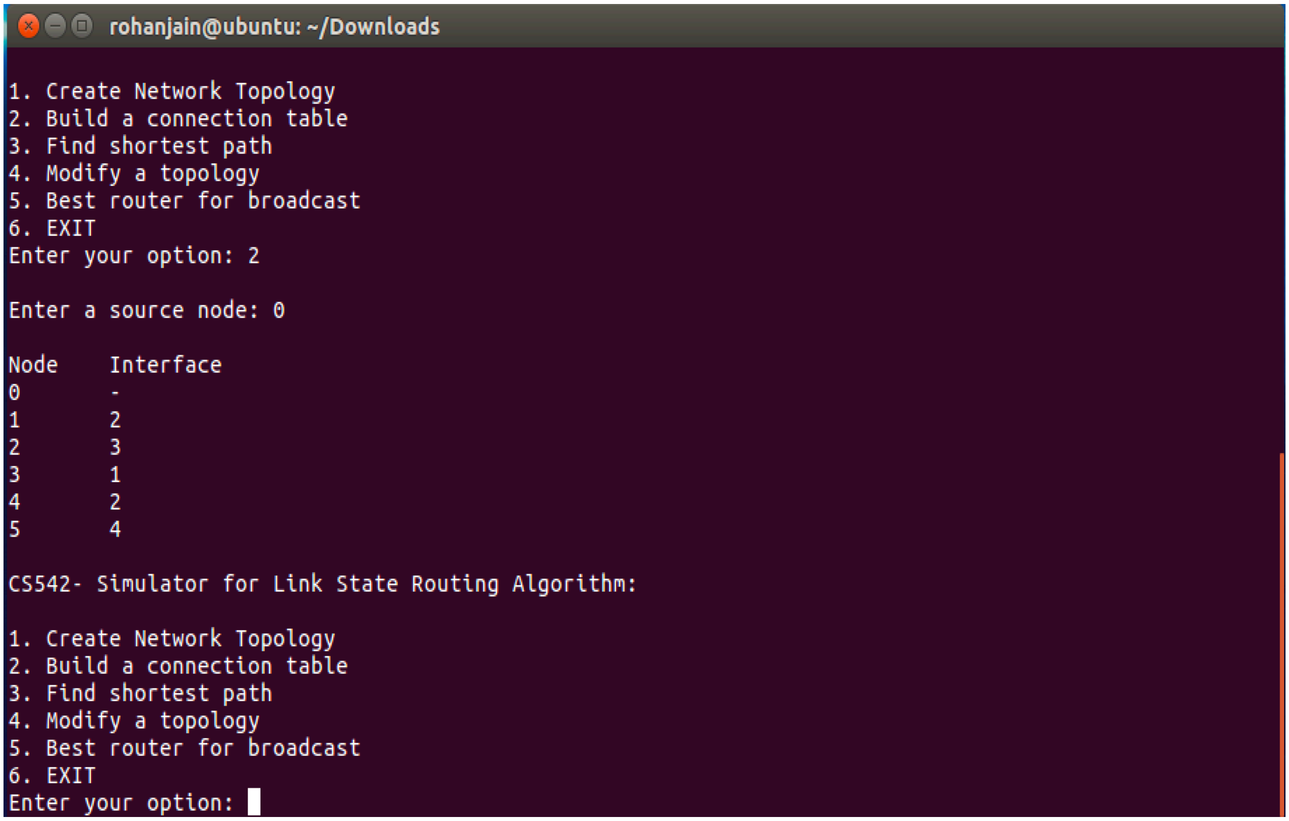
```
rohanjain@ubuntu:~/Downloads$ gcc -o linkstate linkstate.c
rohanjain@ubuntu:~/Downloads$ ./linkstate


CS542- Simulator for Link State Routing Algorithm:

1. Create Network Topology
2. Build a connection table
3. Find shortest path
4. Modify a topology
5. Best router for broadcast
6. EXIT
Enter your option: 
```

1. **Compiling and running of code(linkstate.c) using gcc compiler.**



```
rohanjain@ubuntu:~/Downloads$ ./linkstate


CS542- Simulator for Link State Routing Algorithm:

1. Create Network Topology
2. Build a connection table
3. Find shortest path
4. Modify a topology
5. Best router for broadcast
6. EXIT
Enter your option: 1

Input: matrix.txt

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



CS542- Simulator for Link State Routing Algorithm:

1. Create Network Topology
2. Build a connection table
3. Find shortest path
4. Modify a topology
5. Best router for broadcast
6. EXIT
Enter your option:
```

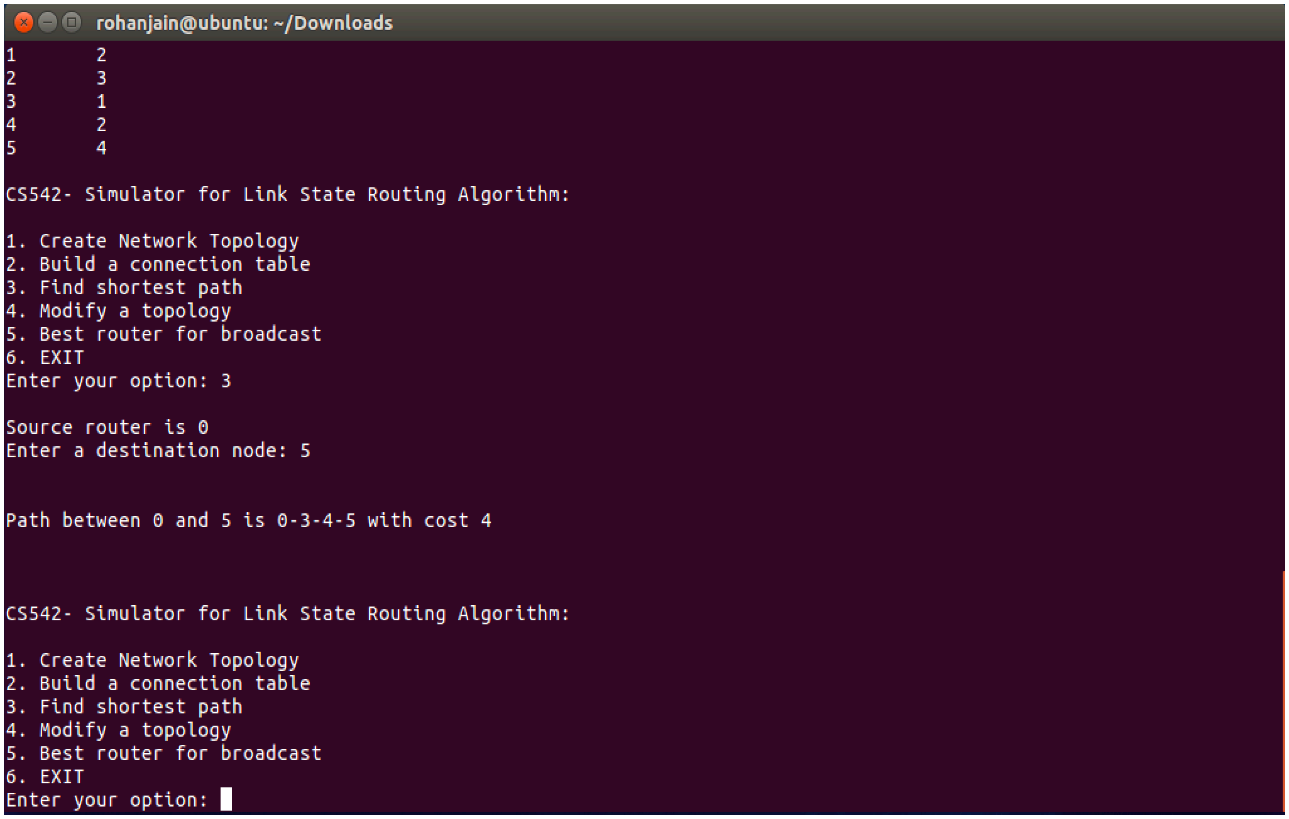
2. User have selected option '1' and have entered 'matrix.txt' as input and program is displaying the matrix which is in inputted file.



rohanjain@ubuntu: ~/Downloads

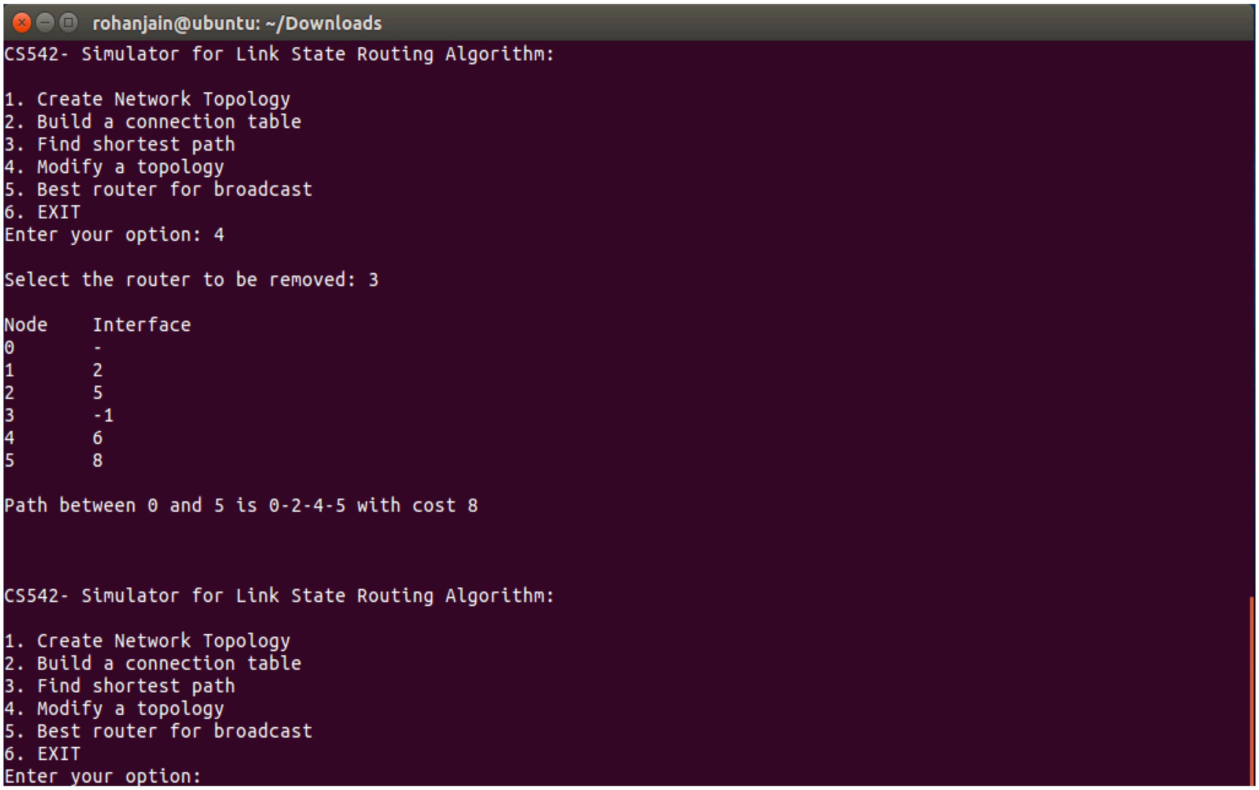
```
1. Create Network Topology
2. Build a connection table
3. Find shortest path
4. Modify a topology
5. Best router for broadcast
6. EXIT
Enter your option: 2

Enter a source node: 0

Node    Interface
0       -
1       2
2       3
3       1
4       2
5       4

CS542- Simulator for Link State Routing Algorithm:

1. Create Network Topology
2. Build a connection table
3. Find shortest path
4. Modify a topology
5. Best router for broadcast
6. EXIT
Enter your option: 
```

3. User have selected option ‘2’ and have entered ‘0’ as source router and router ‘0’s connection table is printed.



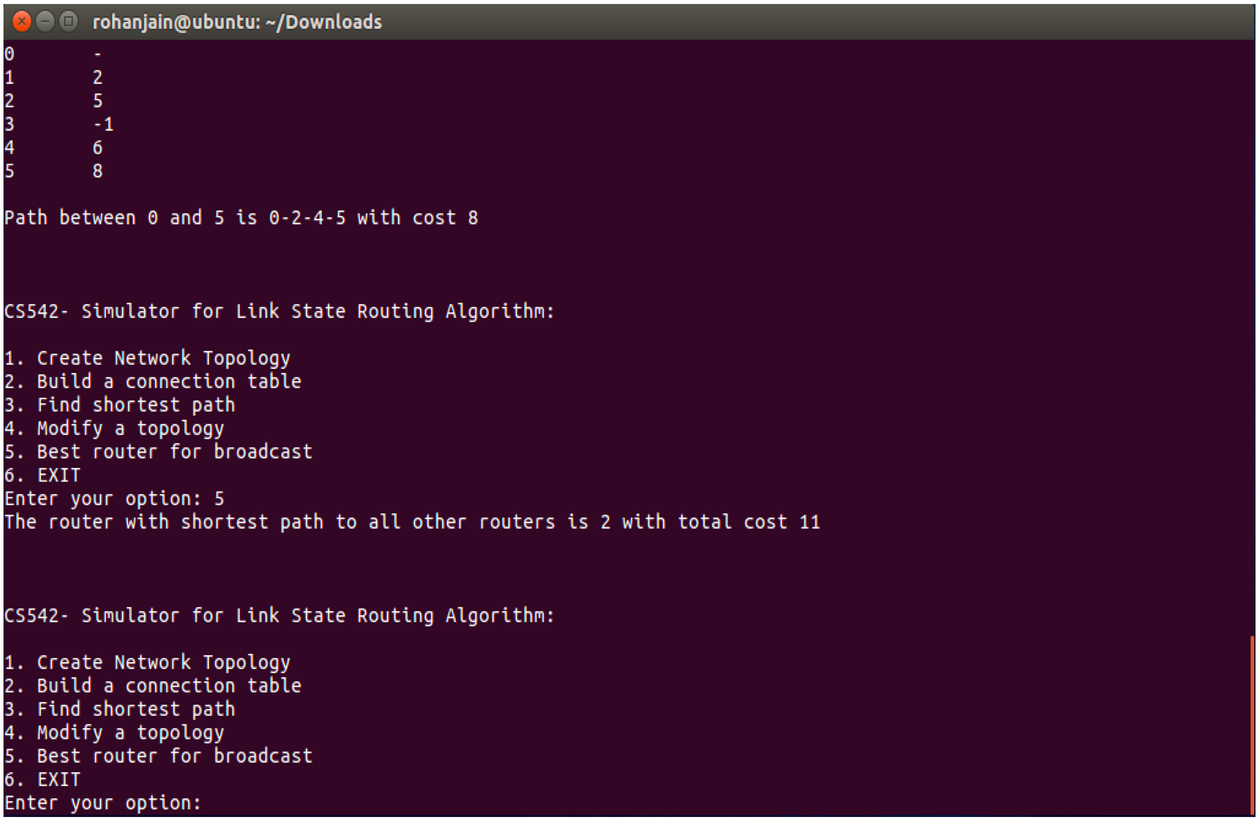
```
1       2
2       3
3       1
4       2
5       4

CS542- Simulator for Link State Routing Algorithm:

1. Create Network Topology
2. Build a connection table
3. Find shortest path
4. Modify a topology
5. Best router for broadcast
6. EXIT
Enter your option: 3

Source router is 0
Enter a destination node: 5


Path between 0 and 5 is 0-3-4-5 with cost 4



CS542- Simulator for Link State Routing Algorithm:

1. Create Network Topology
2. Build a connection table
3. Find shortest path
4. Modify a topology
5. Best router for broadcast
6. EXIT
Enter your option: 
```

4. **User have selected option '3' and shortest path is displayed from source (router '0') to destination (router '5') along with cost ('4').**



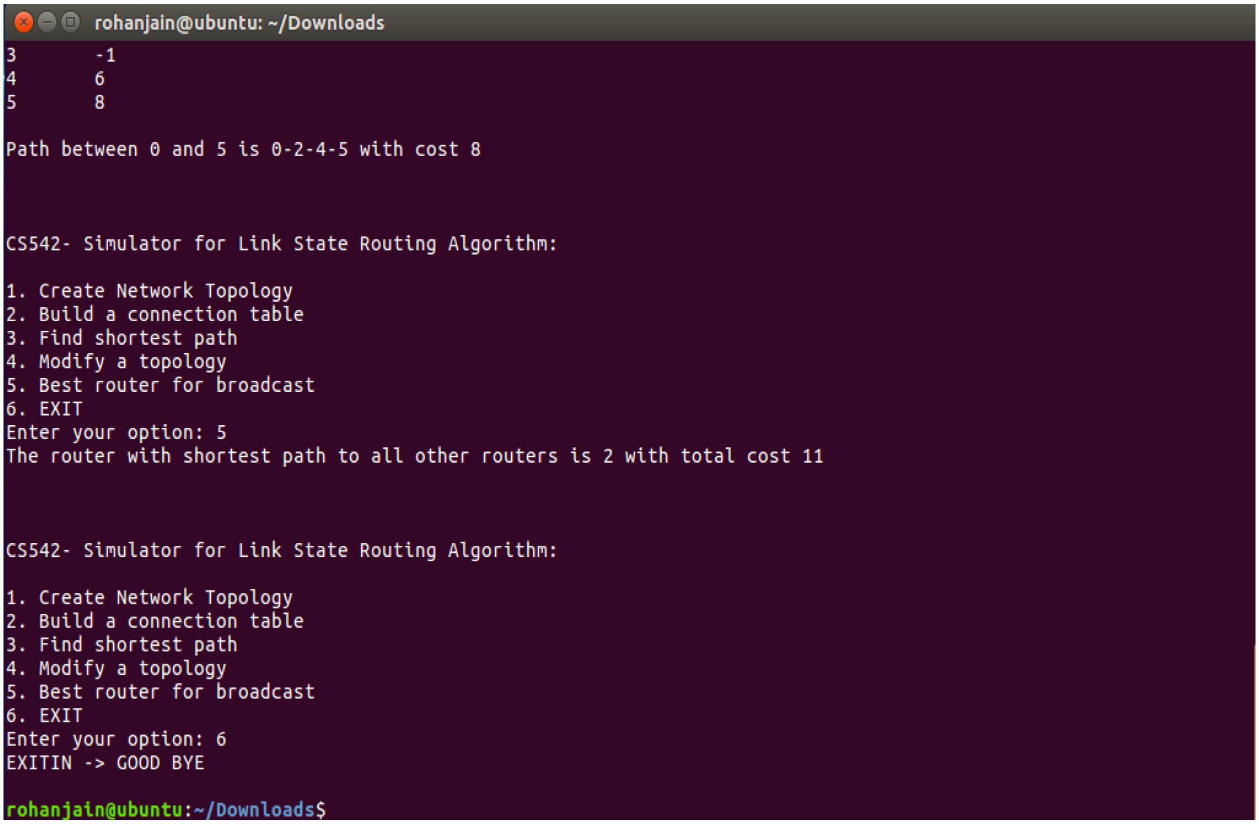
```
CS542- Simulator for Link State Routing Algorithm:

1. Create Network Topology
2. Build a connection table
3. Find shortest path
4. Modify a topology
5. Best router for broadcast
6. EXIT
Enter your option: 4

Select the router to be removed: 3

Node    Interface
0       -
1       2
2       5
3       -1
4       6
5       8

Path between 0 and 5 is 0-2-4-5 with cost 8



CS542- Simulator for Link State Routing Algorithm:

1. Create Network Topology
2. Build a connection table
3. Find shortest path
4. Modify a topology
5. Best router for broadcast
6. EXIT
Enter your option:
```

5. **User have selected option ‘4’ and have selected router (‘3’) to be removed and code have displayed router(‘0’) new connection table and new shortest path between source router(‘0’) and destination router(‘5’) along with its cost.**



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

Path between 0 and 5 is 0-2-4-5 with cost 8



CS542- Simulator for Link State Routing Algorithm:

1. Create Network Topology
2. Build a connection table
3. Find shortest path
4. Modify a topology
5. Best router for broadcast
6. EXIT
Enter your option: 5
The router with shortest path to all other routers is 2 with total cost 11



CS542- Simulator for Link State Routing Algorithm:

1. Create Network Topology
2. Build a connection table
3. Find shortest path
4. Modify a topology
5. Best router for broadcast
6. EXIT
Enter your option:
```

6. **User have selected option ‘5’ which displays the router(‘2’) that have shortest path to all other router along with its cost (‘11’).**



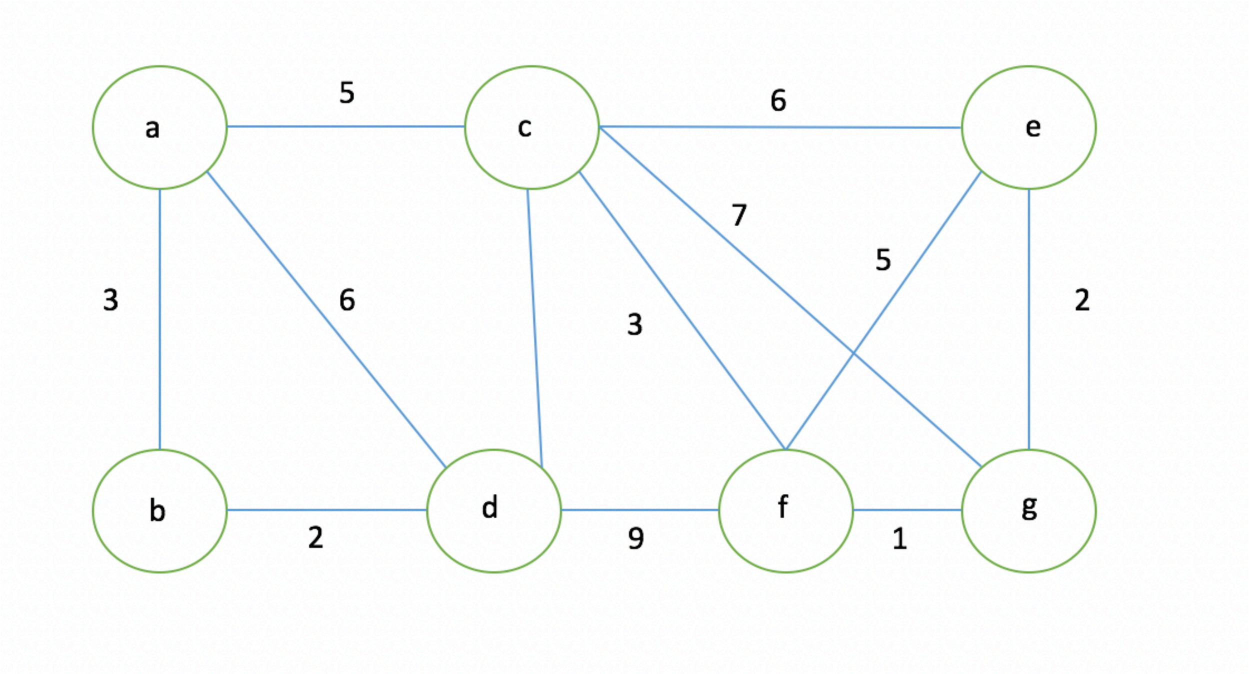
```
3       -1
4       6
5       8

Path between 0 and 5 is 0-2-4-5 with cost 8



CS542- Simulator for Link State Routing Algorithm:

1. Create Network Topology
2. Build a connection table
3. Find shortest path
4. Modify a topology
5. Best router for broadcast
6. EXIT
Enter your option: 5
The router with shortest path to all other routers is 2 with total cost 11



CS542- Simulator for Link State Routing Algorithm:

1. Create Network Topology
2. Build a connection table
3. Find shortest path
4. Modify a topology
5. Best router for broadcast
6. EXIT
Enter your option: 6
EXITIN -> GOOD BYE

rohanjain@ubuntu:~/Downloads$
```

7. User have selected option ‘6’ which exits the program.

Test Cases

The implemented link state routing algorithm in this project works under various test cases and is proved to be successful. This project is tested for various number of router like 6 routers, 7 routers, 10 routers, etc. Following are the few test cases:

Case 1 – For 7 Routers:



Option 1: 7routers.txt

Input Matrix:
0 3 5 6 -1 -1 -1
3 0 -1 2 -1 -1 -1
5 -1 0 2 6 3 7
6 2 2 0 -1 9 -1
-1 -1 6 -1 0 5 2
-1 -1 3 9 5 0 1
-1 -1 7 -1 2 1 0

Option 2:

```
Source Node: 0
Router 0's connection table:

Node   Interface
0      -
1      3
2      5
3      5
4      11
5      8
6      9
```

Option 3:

```
Source router is 0
Enter a destination node: 6
Path between 0 and 6 is 0-2-5-6 with cost 9
```

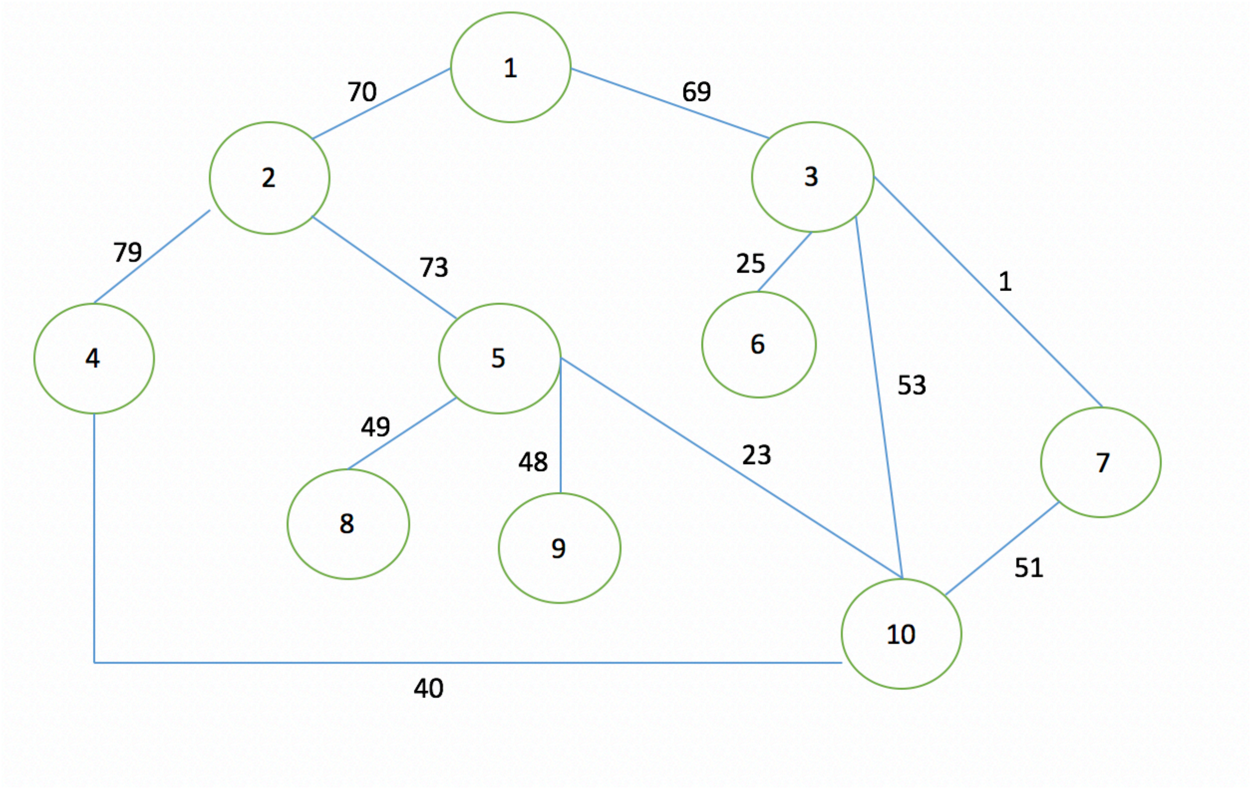
Option 4:

```
Select the router to be removed: 5

Node   Interface
0      -
1      3
2      5
3      5
4      11
5      -1
6      12

Path between 0 and 6 is 0-2-6 with cost 12
```

Option 5:

```
The router with shortest path to all other routers is 2 with total cost 23
```

Case 2 – For 10 Routers:



Option 1: 10routers.txt

Input Matrix:

```
0	70	69	-1	-1	-1	-1	-1	-1	-1
70	0	-1	79	73	-1	-1	-1	-1	-1
69	-1	0	-1	-1	25	1	-1	-1	53
-1	79	-1	0	-1	-1	-1	-1	-1	40
-1	73	-1	-1	0	-1	-1	49	48	23
-1	-1	25	-1	-1	0	-1	-1	-1	-1
-1	-1	1	-1	-1	-1	0	-1	-1	51
-1	-1	-1	-1	49	-1	-1	0	-1	-1
-1	-1	-1	-1	48	-1	-1	-1	0	29
-1	-1	53	40	23	-1	51	-1	29	0
```

Option 2:

Source Node: 0

Router 0’s connection table:

Node	Interface
0	-
1	70
2	69
3	149
4	143
5	94
6	70
7	192
8	150
9	121

Option 3:

Source router is 0

Enter a destination node: 9

Path between 0 and 9 is 0-2-6-9 with cost 121

Option 4:

Select the router to be removed: 6

Node	Interface
0	-
1	70
2	69
3	149
4	143
5	94
6	-1
7	192
8	151
9	122

Path between 0 and 9 is 0-2-9 with cost 122

Option 5:

The router with shortest path to all other routers is 9 with total cost 512

Conclusion

- The implemented code for Link State routing algorithm works successfully for any size of network matrix.
- Shortest path from source router to destination along with the cost is printed successfully.
- Additional functionalities of project include removal of inputted router from the network and printing new shortest path from source router to destination router. It also displays the best router which has shortest path to all other routers.
- The project is test under various test case and is proved to be successful for each of them.