Practical 6 – Computer Networks Lab

Name: Neeraj Belsare

Roll No.: 79

Batch: A4

PRN: 202101040133

Title:

Routing Algorithm

Aim:

Write a program to implement Distance Vector Routing / Link State Routing

Theory:

Distance Vector Routing

As the name implies, each node maintains a vector (table) of least cost (minimum distances) to every node. The table also shows the next-hop.

Historically, it is also known as the old ARPANET routing algorithm or Bellman-Ford algorithm.

The distance vector calculation is based on minimizing the cost to each node.

Steps:

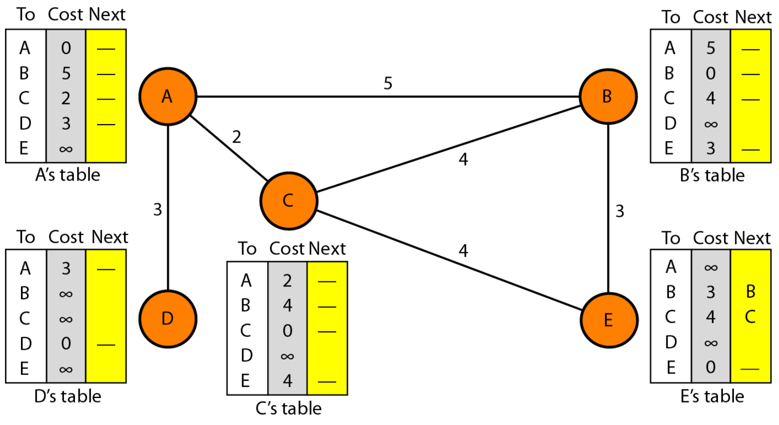
1. Initialization
2. Sharing
3. Updating
4. Problems of distance vector routing:
    1) Two node loop instability
    2) Three node instability

5. Routing Information Protocol RIP.

1. <u>Initialization</u>

At the beginning, each node knows the distance between itself and the <u>immediate neighbours</u>. The distance for others is marked as infinite.

Initialization of tables in distance vector routing



A's table

To	Cost	Next
A	0	—
B	5	—
C	2	—
D	3	—
E	∞	

B's table

To	Cost	Next
A	5	—
B	0	—
C	4	—
D	∞	
E	3	—

C's table

To	Cost	Next
A	2	—
B	4	—
C	0	—
D	∞	
E	4	—

D's table

To	Cost	Next
A	3	—
B	∞	
C	∞	
D	0	—
E	∞	

E's table

To	Cost	Next
A	∞	
B	3	B
C	4	C
D	∞	
E	0	—

2. <u>Sharing</u>

Each node shares its routing table with its immediate neighbours (i) <u>periodically</u> and (ii) <u>when there is a change in the table.</u> The third column (next hop) is not useful for the neighbour. So <u>only the first 2 columns</u> are shared.

3. <u>Updating</u>

When a node receives a two-column table from a neighbour, it needs to update its routing table. It takes <u>three steps</u>:

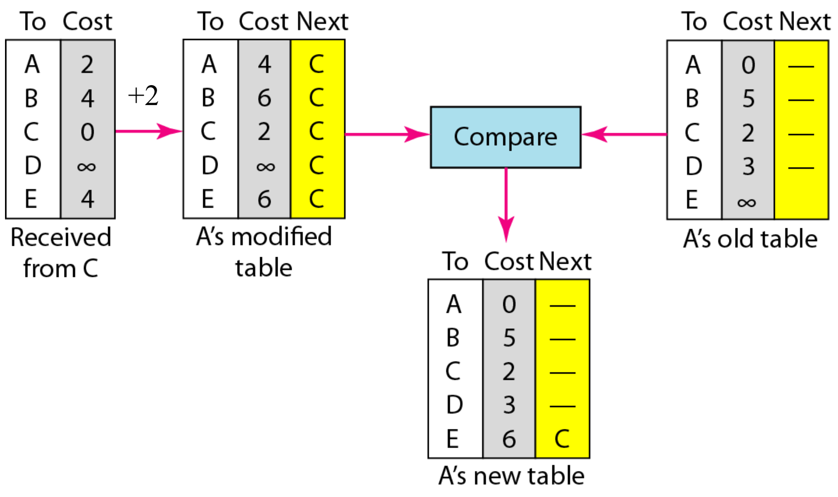
1. The receiving node adds the cost between itself and the sending node to each value in the second column.
2. The receiving node adds the name of the sending node to each row as the third column. The sending node is the next node in the route.
3. The receiving node compares each row of its old table with that of the modified version of the received table.

a. If the next-node entry is different, the receiving node chooses the row with the smaller cost. If there is a tie in the cost, the old one is kept.

b. If the next-node entry is the same, the receiving node chooses the new row.

Note: if a route from one node to another does not exist, the distance between them is written as infinite (∞)

Updating in distance vector routing



4. Problems of distance vector routing:
   i. Two node loop instability
   ii. Three node instability

5. Routing Information Protocol RIP: Routing Information Protocol (RIP) is an intradomain routing protocol. It is based on distance vector routing.

Advantages of Distance Vector Routing:

- It is simple to configure and maintain than Link State Routing.

Disadvantages of Distance Vector Routing:

- There is risk of count-to-infinity problem.
- For larger network, distance vector routing results in larger routing tables because each router must know about all other routers. This also leads to congestion.

Link State Routing

1. Each node has a list of nodes and links, & the state of its links. [type, condition (up or down) and cost (metric)]

2. The node uses Dijkstra's algorithm to build a routing table.

3. Open Shortest Path First or OSPF protocol is an intradomain routing protocol based on link state routing.

Building Routing Tables:

The routing table shows the least-cost to every other node.

Four actions are required to build routing table of each node:

1. Creation of the link state packet (LSP) by each node. LSP contains states of the links of that node.

2. Dissemination of LSPs to every other router, called flooding.

3. Formation of a shortest path tree for each node using Dijkstra's algorithm.

4. Building a routing table based on the shortest path tree.

Flooding of LSPs:

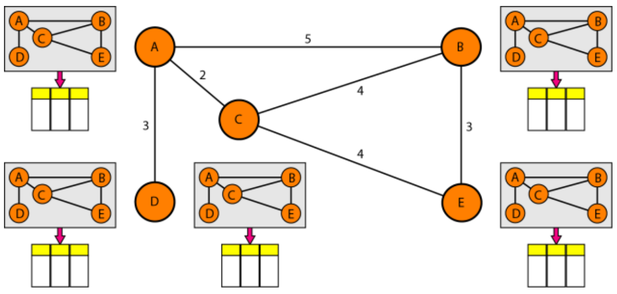
After a node has prepared an LSP, it is disseminated to all other nodes, not only to its neighbors. This process is called flooding. Thus, each node will have a copy of the whole topology.

Formation of Shortest Path Tree: Dijkstra Algorithm

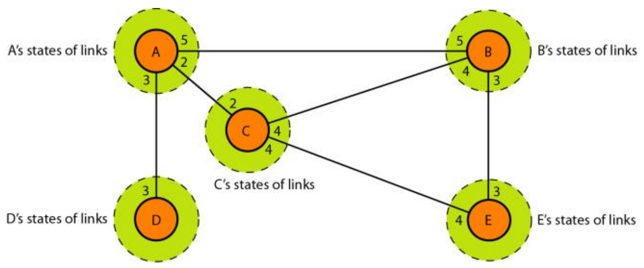
After receiving all LSPs, each node will have a copy of the whole topology. Each node then creates a shortest path tree using Dijkstra algorithm. It divides the nodes into two sets: tentative and permanent.

It finds the neighbors of the current node, and makes them tentative. Among the tentative list nodes, one with the least cost (shortest distance) is made permanent.

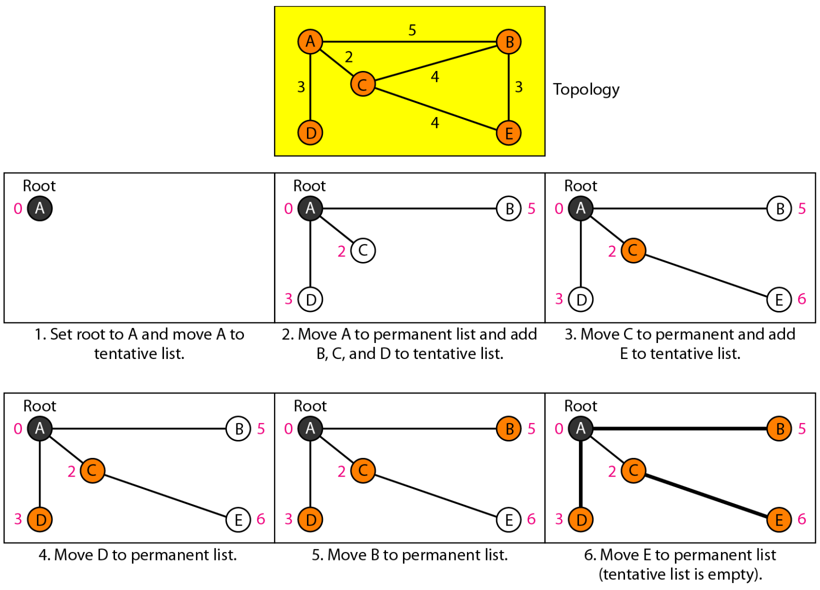
Concept of link state routing



Link state knowledge



Example of formation of shortest path tree



1. Set root to A and move A to tentative list.
2. Move A to permanent list and add B, C, and D to tentative list.
3. Move C to permanent and add E to tentative list.
4. Move D to permanent list.
5. Move B to permanent list.
6. Move E to permanent list (tentative list is empty).

Routing table for node A

Node	*Cost*	*Next Router*
A	0	—
B	5	—
C	2	—
D	3	—
E	6	C

Disadvantage of link state routing:

- Heavy traffic is created in link state routing due to flooding. Flooding can also cause infinite looping.

Procedure/Code:

a) Distance Vector Routing

```
#include<iostream>
#include<conio.h>

using namespace std;

int main() {
        int graph[50][50];
        int i,j,k,t;
        int nn;

        cout<<"Enter Number of Nodes: ";
        cin>>nn;

        for (i=0; i<nn; i++) {
                for(j=0; j<nn; j++) {
                        graph[i][j]=-1;
                }
        }
        char ch[7]= {'A','B','C','D','E','F','G'};
```

```
	for (i=0; i<nn; i++) {
		for(j=0; j<nn; j++) {
			if(i==j) {
				graph[i][j]=0;
			}

			if(graph[i][j]==-1) {
				cout<<"\n Enter Distance between "<<ch[i]<<" -
"<<ch[j]<<" : ";
				cin>>t;
				graph[i][j]=graph[j][i]=t;
			}
		}
	}

	int via[50][50];
	for (i=0; i<nn; i++) {
		for(j=0; j<nn; j++) {
			via[i][j]=-1;
		}
	}

	cout<<"\n After Initialization";
	for (i=0; i<nn; i++) {
		cout<<"\n"<<ch[i]<<" Table";
		cout<<"\nNode\tDist\tVia";
		for(j=0; j<nn; j++) {
			cout<<"\n"<<ch[j]<<"\t"<<graph[i][j]<<"\t"<<via[i][j];
```

```
          }
}

int sh[50][50][50];

for(i=0; i<nn; i++) {
          for(j=0; j<nn; j++) {
                    for (k=0; k<nn; k++) {
                              if((graph[i][j]>-1)&&(graph[j][k]>-1)) {
                                        sh[i][j][k]=graph[j][k]+graph[i][j];
                              } else {
                                        sh[i][j][k]=-1;
                              }
                    }
          }
}

for(i=0; i<nn; i++) {
          cout<<"\n\n For "<<ch[i];
          for (j=0; j<nn; j++) {
                    cout<<"\n From "<<ch[j];
                    for(k=0; k<nn; k++) {
                              cout<<"\n "<<ch[k]<<" "<<sh[i][j][k];
                    }
          }
}

int final[50][50];
```

```
        for(i=0; i<nn; i++) {
                for(j=0; j<nn; j++) {
                        final[i][j]=graph[i][j];
                        via[i][j]=i;
                        for(k=0; k<nn; k++) {
                                if((final[i][j]>sh[i][k][j]) || (final[i][j] == -1)) {
                                        if(sh[i][k][j]>-1)     {
                                                final[i][j]=sh[i][k][j];
                                                via[i][j]=k;
                                        }
                                }
                        }
                        if(final[i][j]==-1) {
                                for(k=0; k<nn; k++) {
                                        if((final[i][k]!=-1)&&(final[k][j]!=-1)) {
                                                if((final[i][j]==-1) || ((final[i][j]!=-1)
    &&(final[i][j]>final[i][k]+final[k][j]))) {
                                                        if(final[i][k]+final[k][j]>-1) {

        final[i][j]=final[i][k]+final[k][j];
                                                                via[i][j]=k;
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }
        cout<<"\n After Update :";

        for (i=0; i<nn; i++) {
                cout<<"\n"<<ch[i]<<" Table";
                cout<<"\nNode\tDist\tVia";
                for(j=0; j<nn; j++) {
                        cout<<"\n"<<ch[j]<<"\t"<<final[i][j]<<"\t";
                        if(i==via[i][j])
                                cout<<"-";
                        else
                                cout<<ch[via[i][j]];

                }
        }

        getch();
}
```

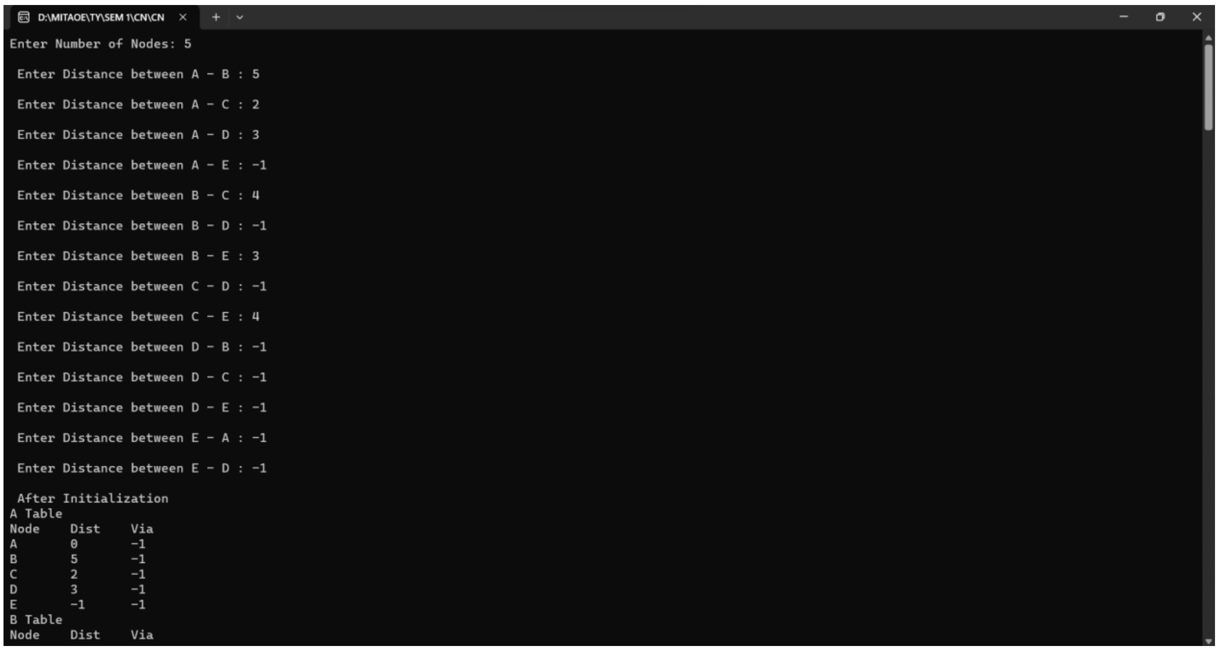
b) Link State Routing:

```
#include <stdio.h>
#include <string.h>
int main() {
        int count,src_router,i,j,k,w,v,min;
        int cost_matrix[100][100],dist[100],last[100];
        int flag[100];
        printf("\n Enter the no of routers");
```

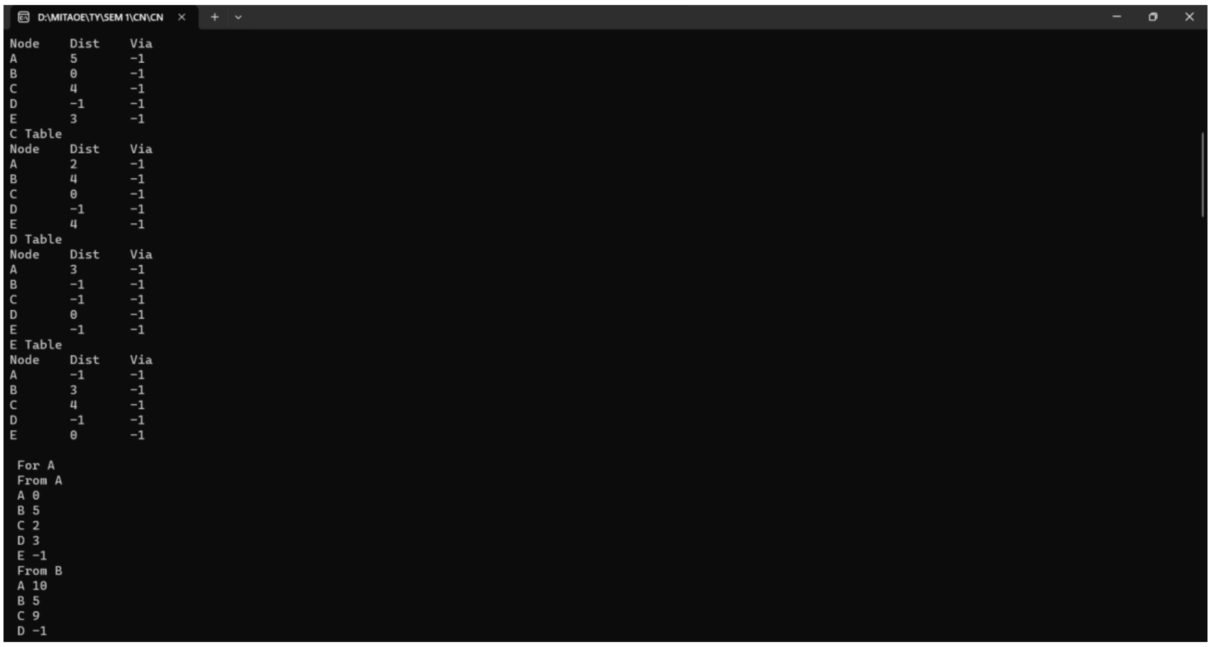
```
scanf("%d",&count);
printf("\n Enter the cost matrix values:");
for(i=0; i<count; i++) {
        for(j=0; j<count; j++) {
                printf("\n%d->%d:",i,j);
                scanf("%d",&cost_matrix[i][j]);
                if(cost_matrix[i][j]<0)cost_matrix[i][j]=1000;
        }
}
printf("\n Enter the source router:");
scanf("%d",&src_router);
for(v=0; v<count; v++) {
        flag[v]=0;
        last[v]=src_router;
        dist[v]=cost_matrix[src_router][v];
}
flag[src_router]=1;
for(i=0; i<count; i++) {
        min=1000;
        for(w=0; w<count; w++) {
                if(!flag[w])
                        if(dist[w]<min) {
                                v=w;
                                min=dist[w];
                        }
        }
        flag[v]=1;
        for(w=0; w<count; w++) {
                if(!flag[w])
                        if(min+cost_matrix[v][w]<dist[w]) {
                                dist[w]=min+cost_matrix[v][w];
                                last[w]=v;
                        }
        }
    }
    for(i=0; i<count; i++) {
        printf("\n%d==>%d:Path taken:%d",src_router,i,i);
        w=i;
        while(w!=src_router) {
                printf("\n<--%d",last[w]);
                w=last[w];
        }
        printf("\n Shortest path cost:%d",dist[i]);
    }
}
```

Output:

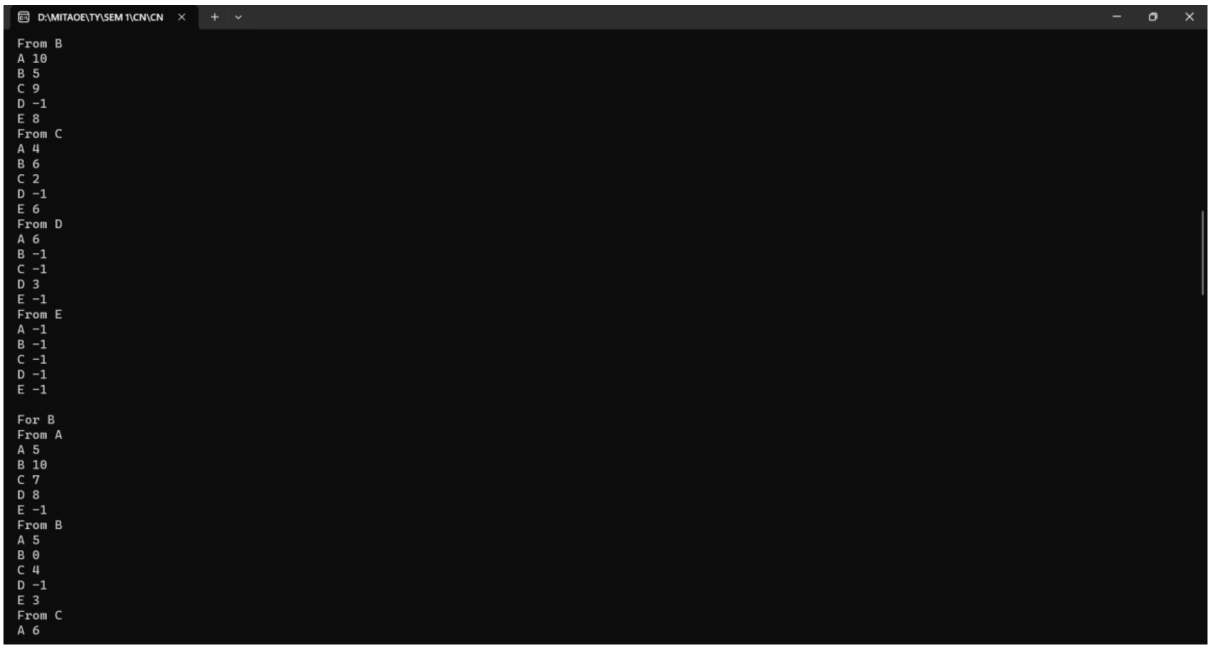
a) Distance Vector Routing



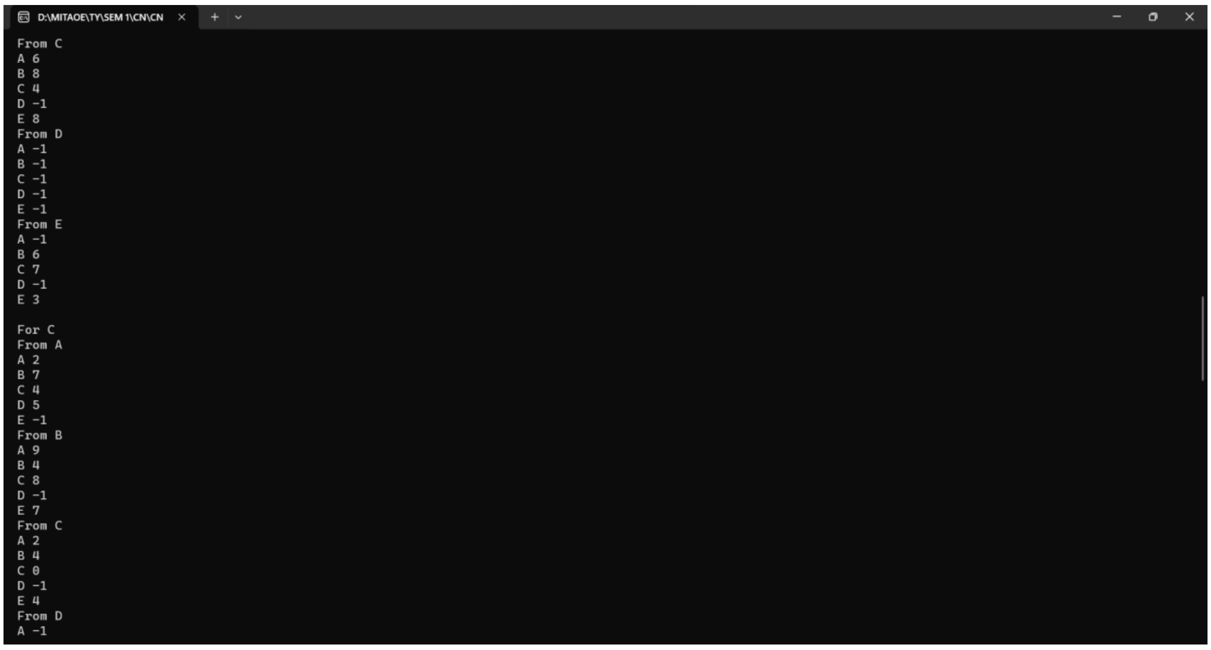
```
Enter Number of Nodes: 5

 Enter Distance between A - B : 5

 Enter Distance between A - C : 2

 Enter Distance between A - D : 3

 Enter Distance between A - E : -1

 Enter Distance between B - C : 4

 Enter Distance between B - D : -1

 Enter Distance between B - E : 3

 Enter Distance between C - D : -1

 Enter Distance between C - E : 4

 Enter Distance between D - B : -1

 Enter Distance between D - C : -1

 Enter Distance between D - E : -1

 Enter Distance between E - A : -1

 Enter Distance between E - D : -1

 After Initialization
A Table
Node    Dist    Via
A       0       -1
B       5       -1
C       2       -1
D       3       -1
E       -1      -1
B Table
Node    Dist    Via
```



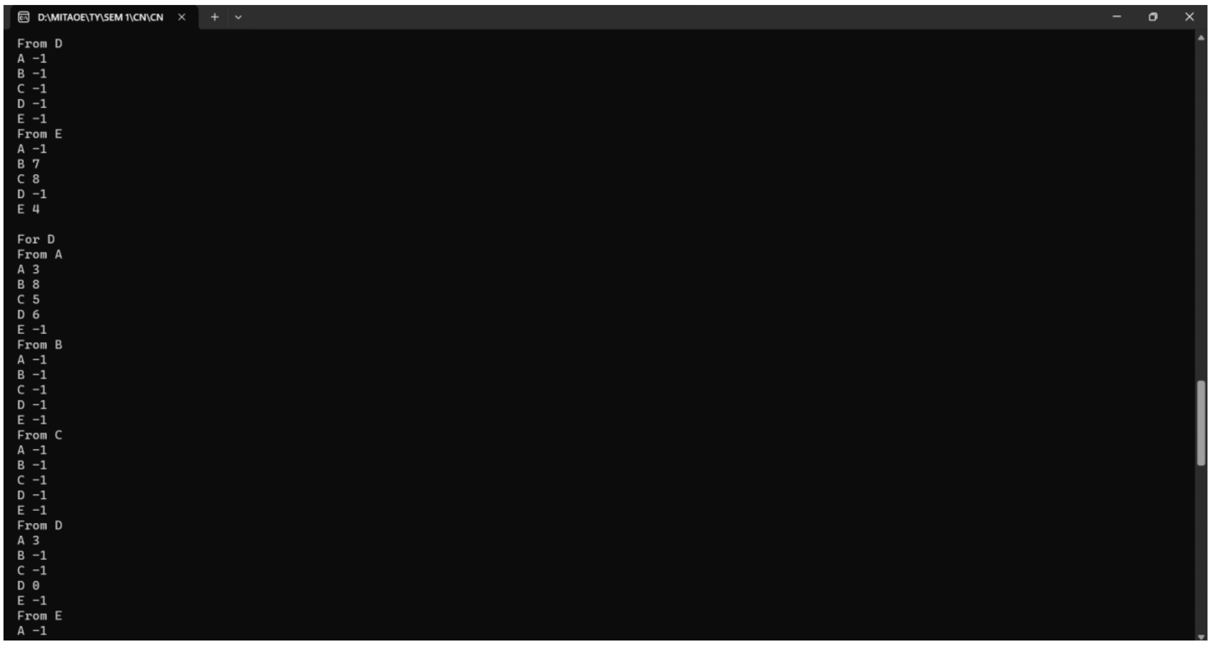
```
Node    Dist    Via
A       5       -1
B       0       -1
C       4       -1
D       -1      -1
E       3       -1
C Table
Node    Dist    Via
A       2       -1
B       4       -1
C       0       -1
D       -1      -1
E       4       -1
D Table
Node    Dist    Via
A       3       -1
B       -1      -1
C       -1      -1
D       0       -1
E       -1      -1
E Table
Node    Dist    Via
A       -1      -1
B       3       -1
C       4       -1
D       -1      -1
E       0       -1

 For A
 From A
 A 0
 B 5
 C 2
 D 3
 E -1
 From B
 A 10
 B 5
 C 9
 D -1
```



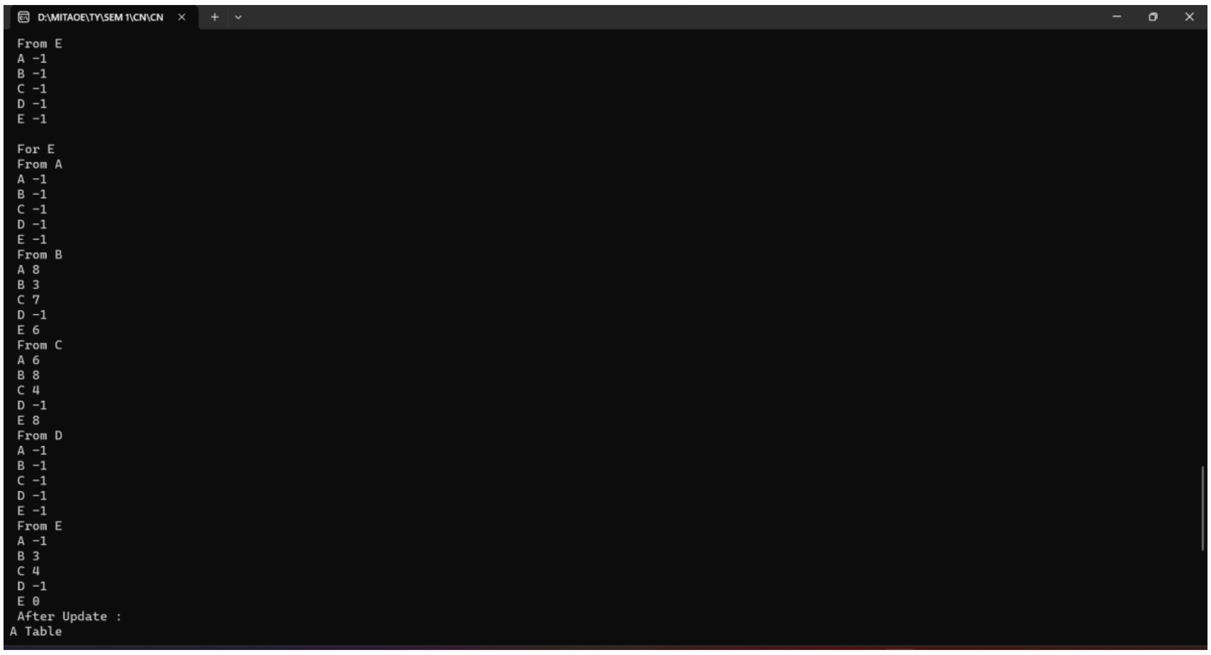
```
From B
A 10
B 5
C 9
D -1
E 8
From C
A 4
B 6
C 2
D -1
E 6
From D
A 6
B -1
C -1
D 3
E -1
From E
A -1
B -1
C -1
D -1
E -1

For B
From A
A 5
B 10
C 7
D 8
E -1
From B
A 5
B 0
C 4
D -1
E 3
From C
A 6
```



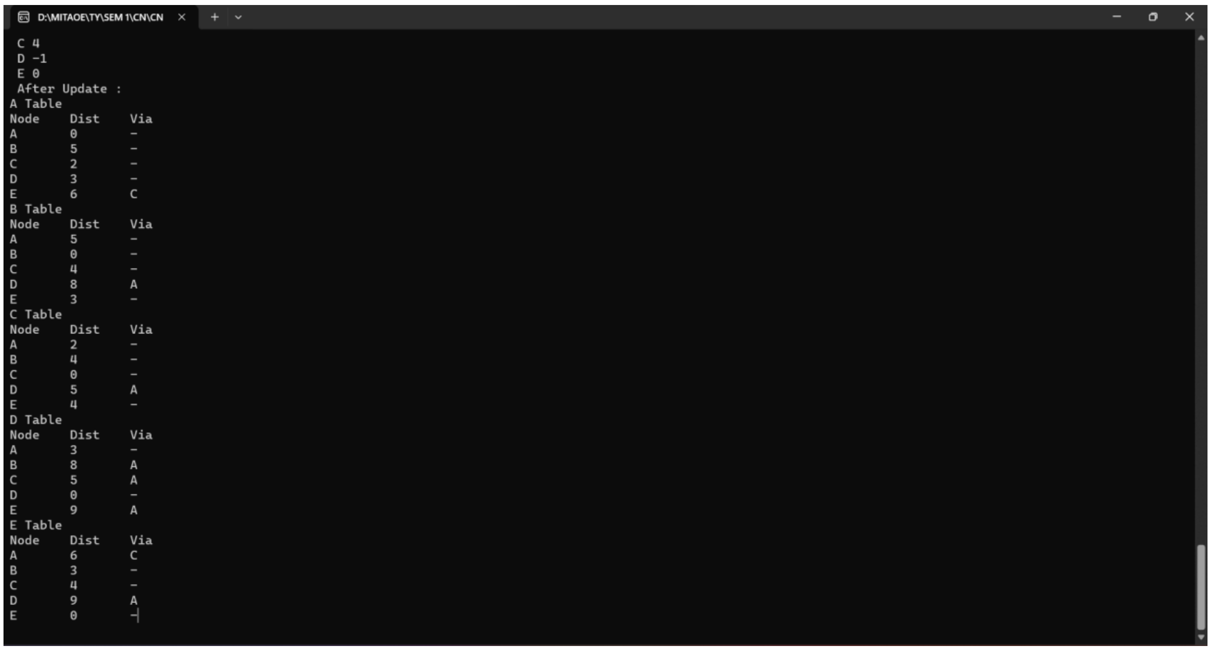
```
From C
A 6
B 8
C 4
D -1
E 8
From D
A -1
B -1
C -1
D -1
E -1
From E
A -1
B 6
C 7
D -1
E 3

For C
From A
A 2
B 7
C 4
D 5
E -1
From B
A 9
B 4
C 8
D -1
E 7
From C
A 2
B 4
C 0
D -1
E 4
From D
A -1
```



```
From D
A -1
B -1
C -1
D -1
E -1
From E
A -1
B 7
C 8
D -1
E 4

For D
From A
A 3
B 8
C 5
D 6
E -1
From B
A -1
B -1
C -1
D -1
E -1
From C
A -1
B -1
C -1
D -1
E -1
From D
A 3
B -1
C -1
D 0
E -1
From E
A -1
```

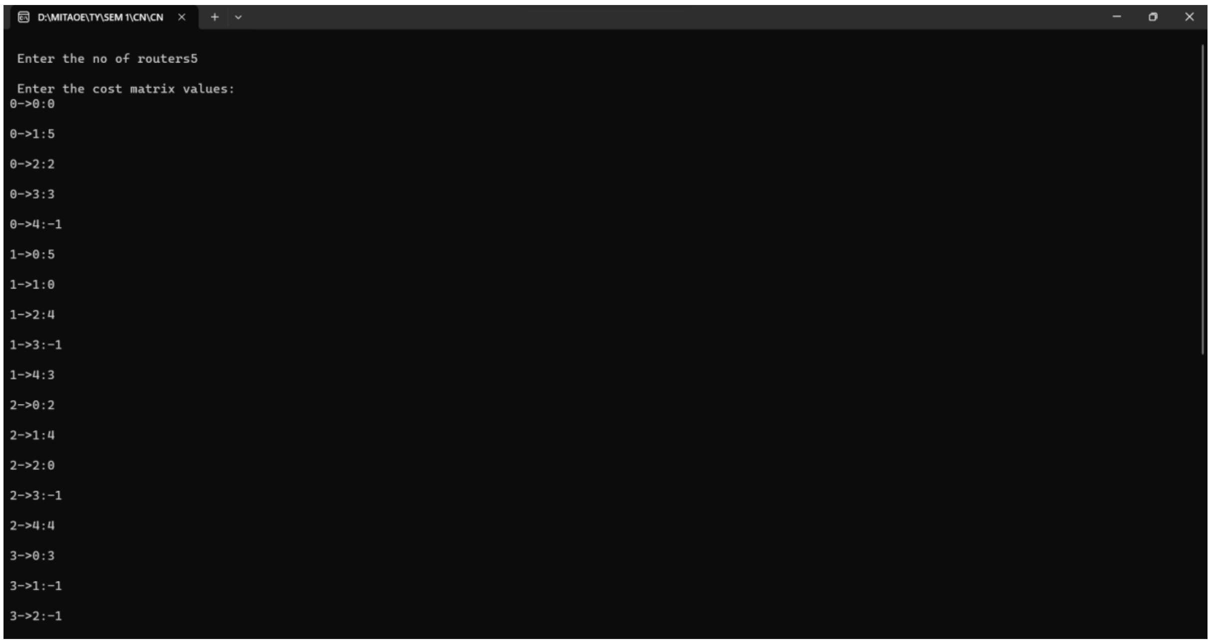


```
 From E
 A -1
 B -1
 C -1
 D -1
 E -1

 For E
 From A
 A -1
 B -1
 C -1
 D -1
 E -1
 From B
 A 8
 B 3
 C 7
 D -1
 E 6
 From C
 A 6
 B 8
 C 4
 D -1
 E 8
 From D
 A -1
 B -1
 C -1
 D -1
 E -1
 From E
 A -1
 B 3
 C 4
 D -1
 E 0
 After Update :
A Table
```

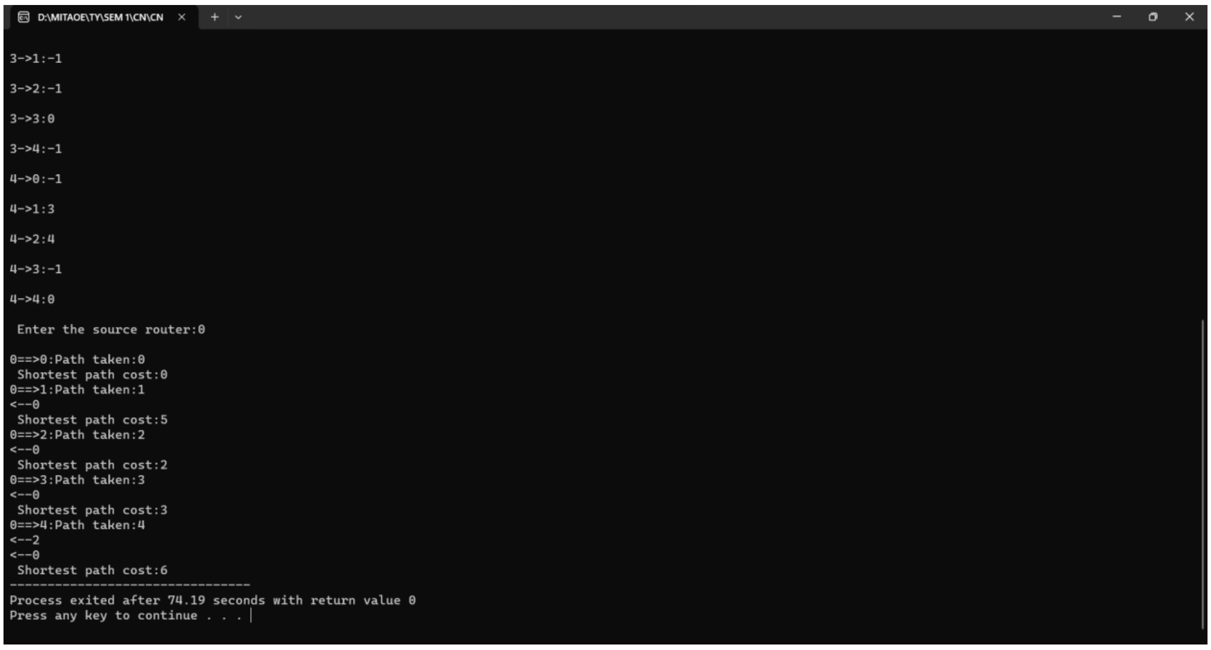


```
 C 4
 D -1
 E 0
 After Update :
A Table
Node    Dist    Via
A       0       -
B       5       -
C       2       -
D       3       -
E       6       C
B Table
Node    Dist    Via
A       5       -
B       0       -
C       4       -
D       8       A
E       3       -
C Table
Node    Dist    Via
A       2       -
B       4       -
C       0       -
D       5       A
E       4       -
D Table
Node    Dist    Via
A       3       -
B       8       A
C       5       A
D       0       -
E       9       A
E Table
Node    Dist    Via
A       6       C
B       3       -
C       4       -
D       9       A
E       0       -
```

b) Link State Routing



```

 Enter the no of routers5

 Enter the cost matrix values:
0->0:0

0->1:5

0->2:2

0->3:3

0->4:-1

1->0:5

1->1:0

1->2:4

1->3:-1

1->4:3

2->0:2

2->1:4

2->2:0

2->3:-1

2->4:4

3->0:3

3->1:-1

3->2:-1
```



```
3->1:-1

3->2:-1

3->3:0

3->4:-1

4->0:-1

4->1:3

4->2:4

4->3:-1

4->4:0

 Enter the source router:0

0==>0:Path taken:0
 Shortest path cost:0
0==>1:Path taken:1
<--0
 Shortest path cost:5
0==>2:Path taken:2
<--0
 Shortest path cost:2
0==>3:Path taken:3
<--0
 Shortest path cost:3
0==>4:Path taken:4
<--2
<--0
 Shortest path cost:6
--------------------------------
Process exited after 74.19 seconds with return value 0
Press any key to continue . . .
```