DISTANCE VECTOR ROUTING ALGORITHM

OBJECTIVE

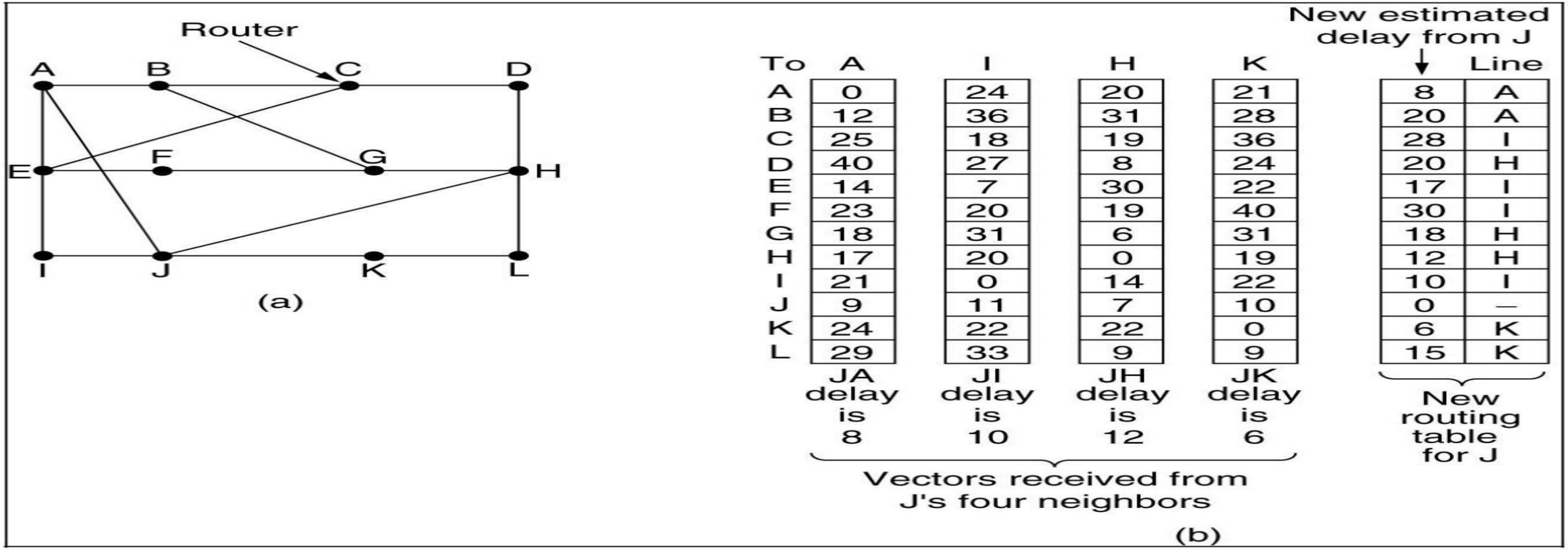
- To implement Distance vector algorithm to find suitable path for transmission

DISTANCE VECTOR ROUTING

- Adaptive/Dynamic Algorithm,
- Route decisions will change dynamically in msec.
- Each Router maintains a table called “Vector”
- Vector contains no.of. Hops & delays
- Table has the best known distance for each router
- Tables are updated by exchanging information with neighbours
- The columns of table represent the directly attached neighbors whereas the rows represent all destinations in the network

DISTANCE VECTOR ROUTING

- Each router knows the best distance to reach another router
- Also known as Bellman Ford Algo
- Each router's table has one entry for one router
- Each entry has two parts
  - Preferred outgoing line for each router
  - Estimated distance to destination router
- Distance is basically considered by no.of hops



(a)

To	A	I	H	K	New estimated delay from J	Line
A	0	24	20	21	8	A
B	12	36	31	28	20	A
C	25	18	19	36	28	I
D	40	27	8	24	20	H
E	14	7	30	22	17	I
F	23	20	19	40	30	I
G	18	31	6	31	18	H
H	17	20	0	19	12	H
I	21	0	14	22	10	I
J	9	11	7	10	0	–
K	24	22	22	0	6	K
L	29	33	9	9	15	K
	JA delay is 8	JI delay is 10	JH delay is 12	JK delay is 6		

Vectors received from J's four neighbors

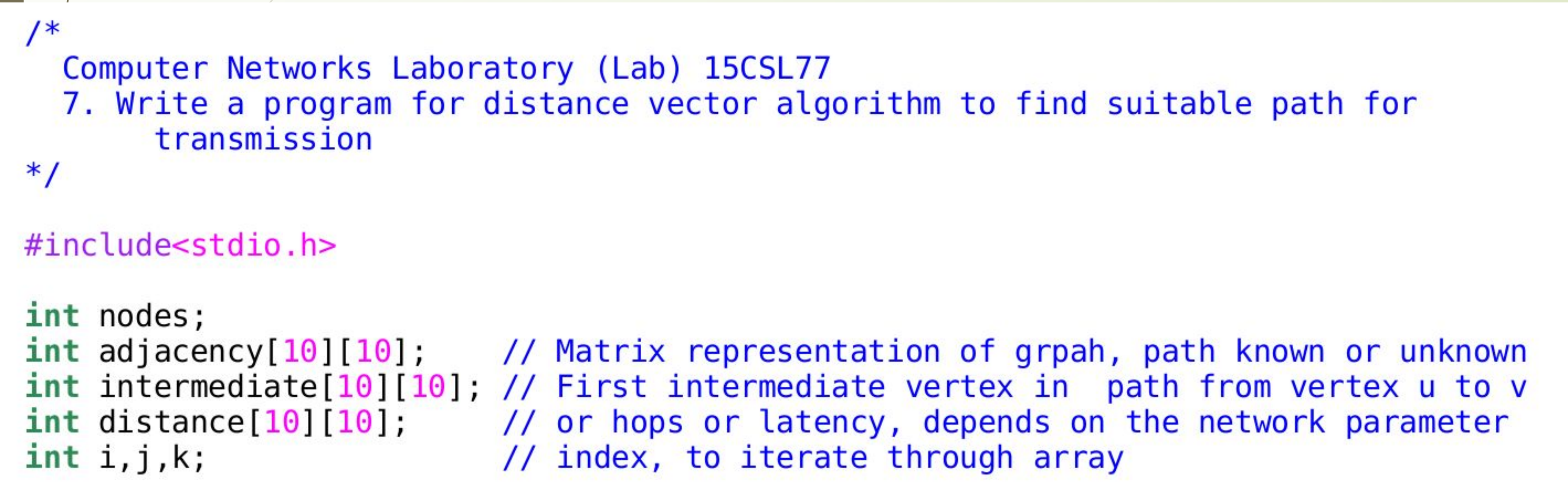
New routing table for J

(b)

ALGORITHM

```
1. send my routing table to all my neighbors whenever my link table changes
2. when I get a routing table from a neighbor on port P with link metric M:
    a. add L to each of the neighbor's metrics
    b. for each entry (D, P', M') in the updated neighbor's table
        i) if I do not have an entry for D, add (D, P, M') to my routing table
        ii) if I have an entry for D with metric M", add (D, P, M') to my routing table if M' < M“
3. if my routing table has changed, send all the new entries to all my neighbors.
```

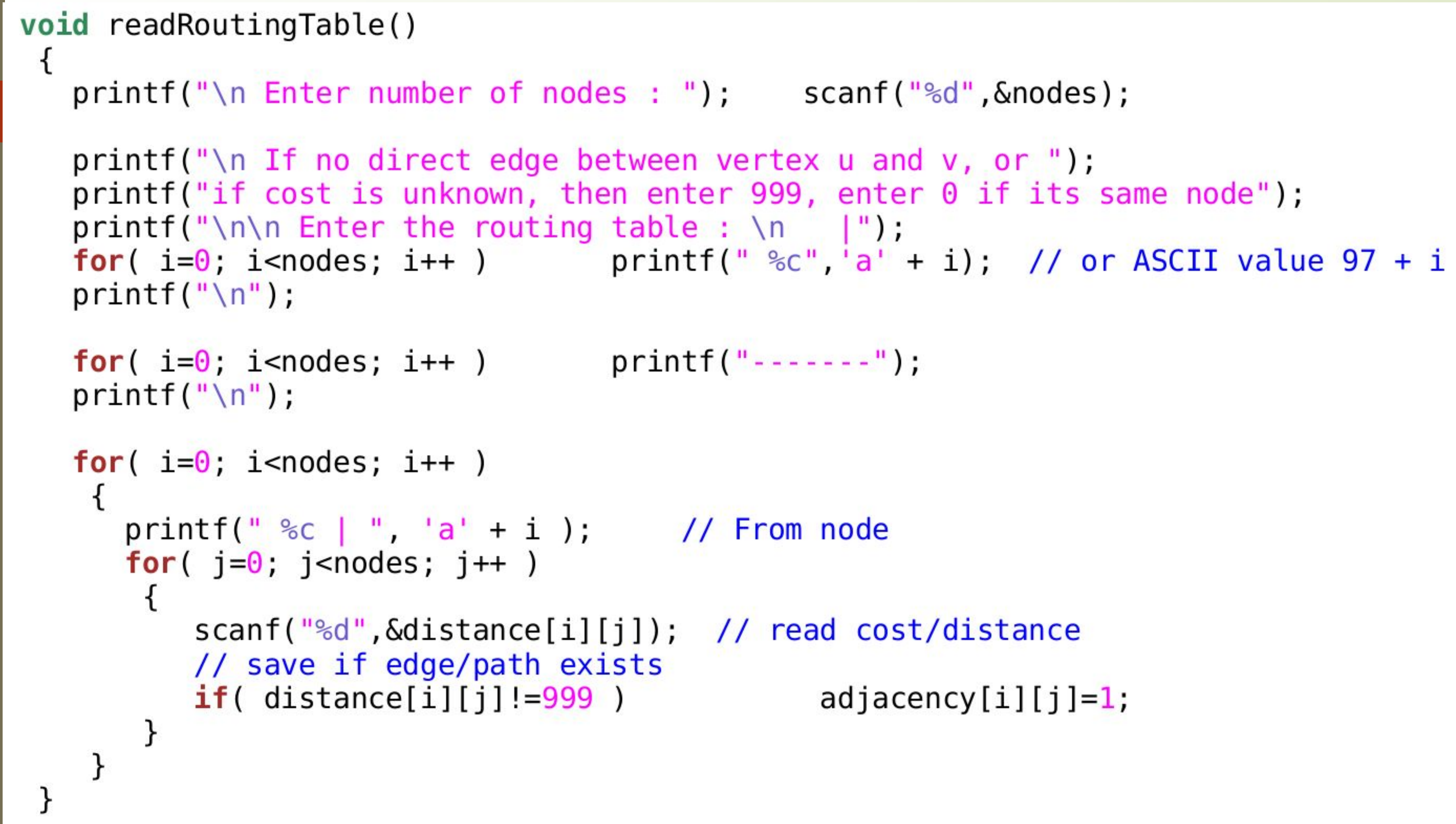
SOURCE CODE



```
/*
  Computer Networks Laboratory (Lab) 15CSL77
  7. Write a program for distance vector algorithm to find suitable path for
       transmission
*/

#include<stdio.h>

int nodes;
int adjacency[10][10];    // Matrix representation of grpah, path known or unknown
int intermediate[10][10]; // First intermediate vertex in  path from vertex u to v
int distance[10][10];     // or hops or latency, depends on the network parameter
int i,j,k;                // index, to iterate through array
```

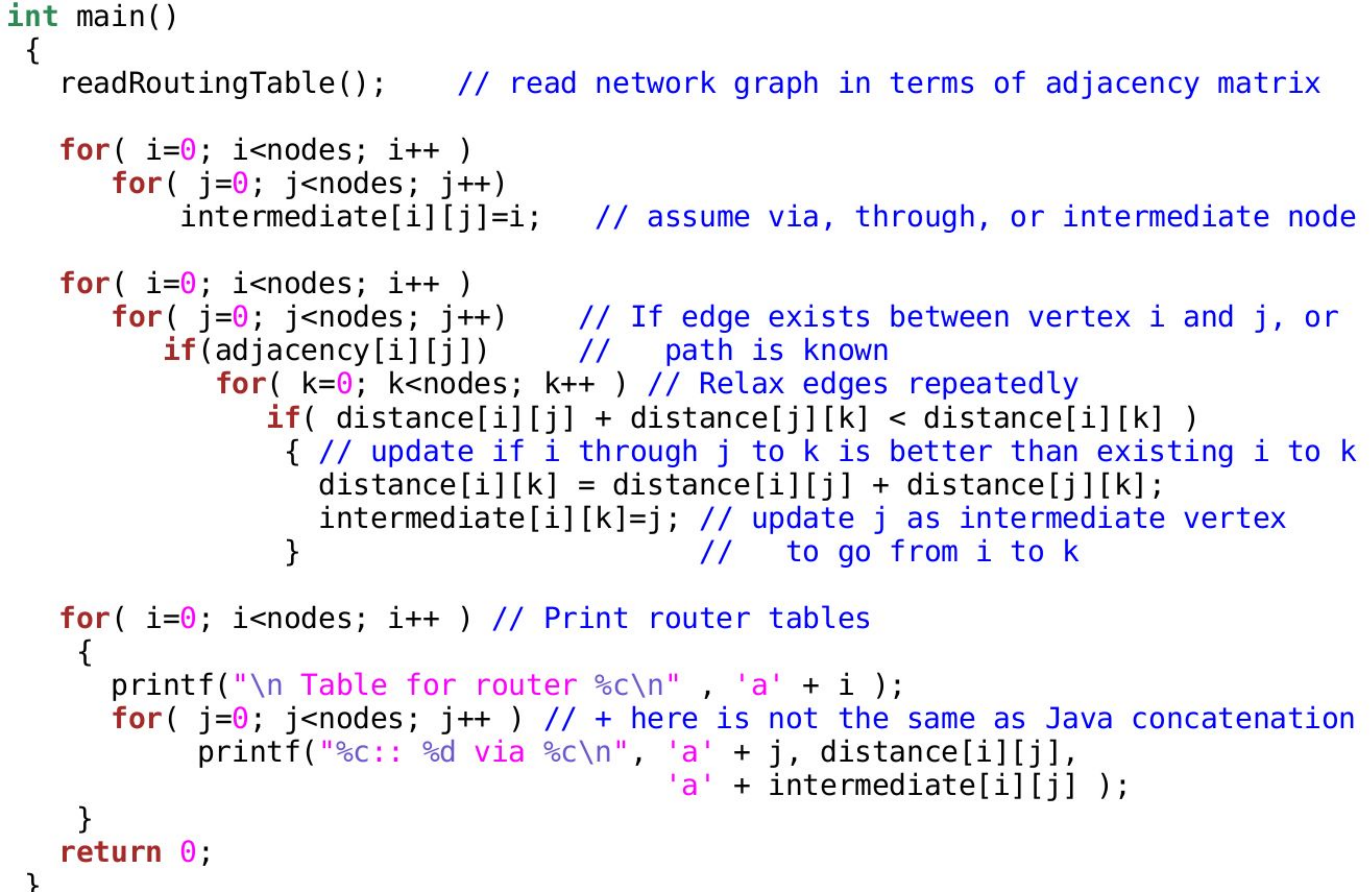


```c
void readRoutingTable()
 {
   printf("\n Enter number of nodes : ");    scanf("%d",&nodes);

   printf("\n If no direct edge between vertex u and v, or ");
   printf("if cost is unknown, then enter 999, enter 0 if its same node");
   printf("\n\n Enter the routing table : \n   |");
   for( i=0; i<nodes; i++ )       printf(" %c",'a' + i);  // or ASCII value 97 + i
   printf("\n");

   for( i=0; i<nodes; i++ )       printf("-------");
   printf("\n");

   for( i=0; i<nodes; i++ )
    {
      printf(" %c | ", 'a' + i );     // From node
      for( j=0; j<nodes; j++ )
       {
          scanf("%d",&distance[i][j]);  // read cost/distance
          // save if edge/path exists
          if( distance[i][j]!=999 )           adjacency[i][j]=1;
       }
    }
 }
```



```
int main()
 {
   readRoutingTable();    // read network graph in terms of adjacency matrix

   for( i=0; i<nodes; i++ )
      for( j=0; j<nodes; j++)
          intermediate[i][j]=i;   // assume via, through, or intermediate node

   for( i=0; i<nodes; i++ )
      for( j=0; j<nodes; j++)    // If edge exists between vertex i and j, or
         if(adjacency[i][j])     //   path is known
            for( k=0; k<nodes; k++ ) // Relax edges repeatedly
               if( distance[i][j] + distance[j][k] < distance[i][k] )
                { // update if i through j to k is better than existing i to k
                  distance[i][k] = distance[i][j] + distance[j][k];
                  intermediate[i][k]=j; // update j as intermediate vertex
                }                       //   to go from i to k

   for( i=0; i<nodes; i++ ) // Print router tables
    {
      printf("\n Table for router %c\n" , 'a' + i );
      for( j=0; j<nodes; j++ ) // + here is not the same as Java concatenation
           printf("%c:: %d via %c\n", 'a' + j, distance[i][j],
                                       'a' + intermediate[i][j] );
    }
   return 0;
 }
```

EXPECTED OUTPUT CASE:1

```
enter the value of no. of nodes
4
Enter the routing table :
   |a b c d
----------------------------------
a  |0 5 1 4
b  |5 0 6 2
c  |1 6 0 3
d  |4 2 3 0
```

EXPECTED OUTPUT

```
table for router a
a:: 0 via a
b:: 5 via a
c:: 1 via a
d:: 4 via a
table for router b
a:: 5 via b
b:: 0 via b
c:: 5 via d
d:: 2 via b
```

EXPECTED OUTPUT

```
table for router c
a:: 1 via c
b:: 5 via d
c:: 0 via c
d:: 3 via c
table for router d
a:: 4 via d
b:: 2 via d
c:: 3 via d
d:: 0 via d
```

EXPECTED OUTPUT

```
do you want to change the cost(1/0)
1
enter the vertices which you want to change the cost
1 3
enter the cost
2
table for router a
a:: 0 via a
b:: 5 via a
c:: 2 via a
d:: 4 via a
table for router b
a:: 5 via b
b:: 0 via b
c:: 5 via d
d:: 2 via b
```

EXPECTED OUTPUT

```
table for router c
a:: 2 via c
b:: 5 via d
c:: 0 via c
d:: 3 via c
table for router d
a:: 4 via d
b:: 2 via b
c:: 3 via d
d:: 0 via d
do you want to change the cost(1/0)
0
```

EXPECTED OUTCOME

Students will be able to implement Distance vector Routing Algorithm.