Experiment No.9
Implementation and evaluation of Routing Algorithms

AIM: To implement and evaluate the routing algorithms

DESCRIPTION: The main function of network layer is routing. When a device has multiple paths to reach a destination, it always selects one path by preferring it over others. This selection process is termed as Routing. Routing is done by special network devices called routers or it can be done by means of software processes. The software based routers have limited functionality and limited scope.

A router is always configured with some default route. A default route tells the router where to forward a packet if there is no route found for specific destination. In case there are multiple paths to reach the same destination, router can make decision based on the following information like *Hop Count, Bandwidth, Metric, Prefix-length, Delay etc.* Routers actually do two things, one is routing and the other is packet forwarding as shown in Figure 9.1. They look into their routing table and find the outgoing line (I/F) and forward the packet over that line. The routing tables are built by the routing protocols that are driven by the rouging algorithms.

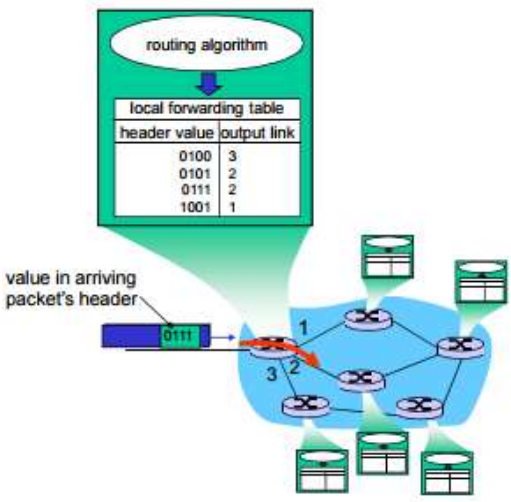


Figure 9.1: Interplay of routing and forwarding

Routes can be *statically configured* or *dynamically learnt*. One route can be configured to be preferred over others. Routing can be classified into the following categories of:

- Unicast
- Broadcast
- Multicast or
- Any cast

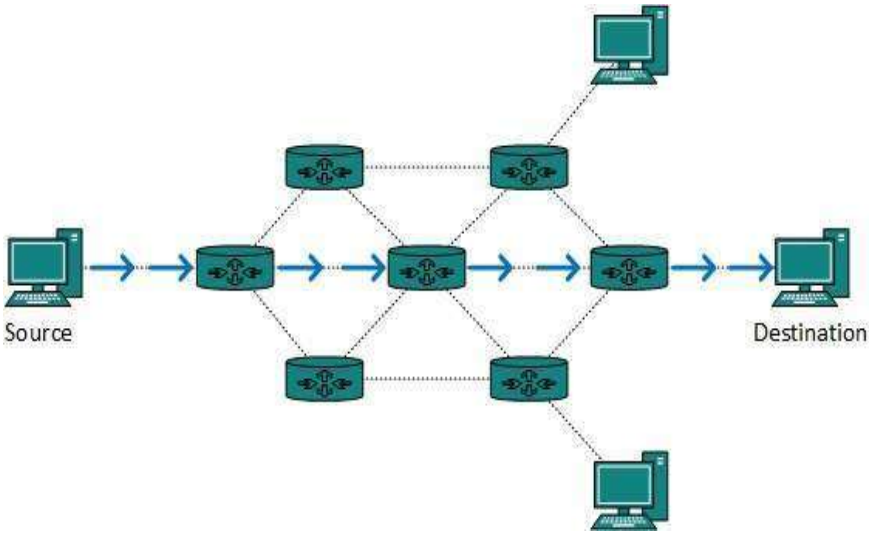


Figure 9.2.: Example for Unicast routing

Unicast routing: Most of the traffic on the internet and intranets known as unicast data or unicast traffic is sent with specified destination. Routing unicast data over the internet is called *unicast routing*. It is the simplest form of routing because the destination is already known. Hence the router just has to look up the routing table and forward the packet to next hop as shown in Figure 9.2.

- **Unicast routing protocols: RIP** (Distance Vector Routing Algorithm based) and **OSPF, ISIS** (Link state routing Algorithm based)**.**

Broadcast routing: By default, the broadcast packets are not routed and forwarded by the routers on any network. Routers create broadcast domains. But it can be configured to forward broadcasts in some special cases. A broadcast message is destined to all network devices.

Broadcast routing can be done in two ways (algorithm):

1. A router creates a data packet and then sends it to each host one by one. In this case, the router creates multiple copies of single data packet with different destination addresses. All packets are sent as unicast but because they are sent to all, it simulates as if router is broadcasting.
    - This method consumes lots of bandwidth and router must know the destination address of each node.
2. Secondly, when router receives a packet that is to be broadcasted, it simply floods those packets out of all interfaces. All routers are configured in the same way as shown in the Figure 9.3.

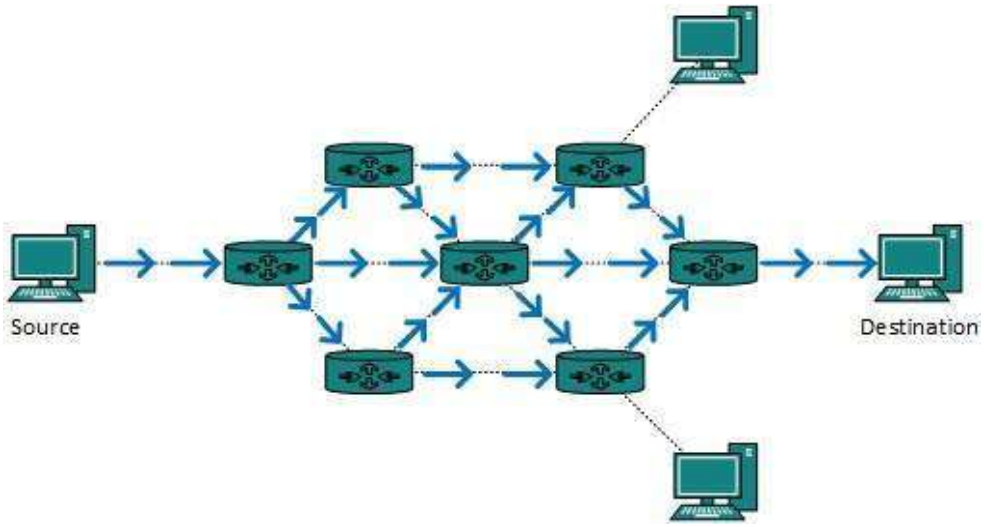


Figure 9.3: Example for broadcast routing

- This method is easy on router's CPU but may cause the problem of duplicate packets received from peer routers.

- **Reverse path forwarding** is a technique, in which router knows in advance about its predecessor from where it should receive broadcast. This technique is used to detect and discard duplicates.

Multicast Routing: Multicast routing is special case of broadcast routing with significance difference and challenges. In broadcast routing, packets are sent to all nodes even if they do not want it. But in Multicast routing, the data is sent to only nodes which wants to receive the packets as shown in Figure 9.4.

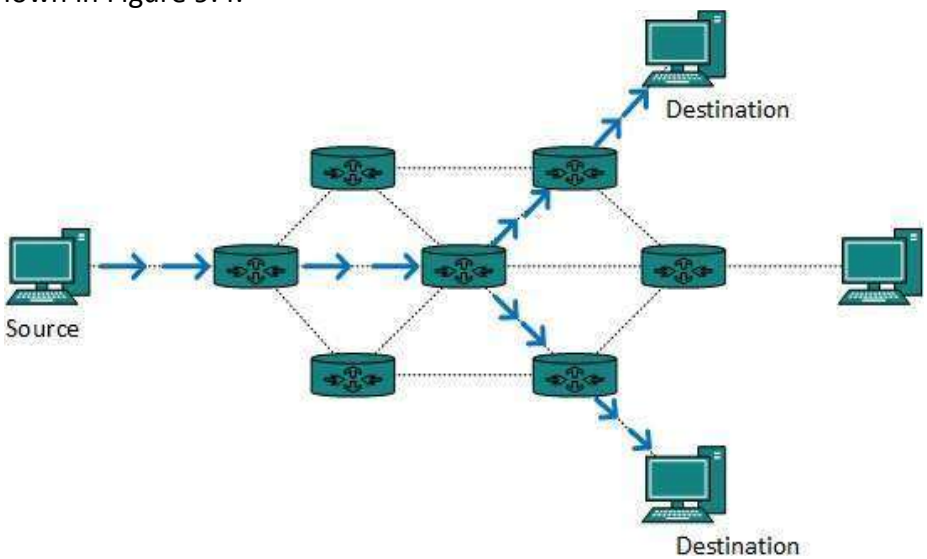


Figure 9.4: Example for Multicast Routing

- The router must know that there are nodes, which wish to receive multicast packets (or stream) then only it should forward. Multicast routing works spanning tree protocol to avoid looping.
- Multicast routing also uses reverse path Forwarding technique, to detect and discard duplicates and loops.

Multicast routing protocols: **DVMRP** (Distance Vector Multicast Routing Protocol), **MOSPF** (Multicast Open Shortest Path First), **CBT** (Core Based Tree), **PIM** (Protocol independent Multicast)

Anycast Routing: Anycast packet forwarding is a mechanism where multiple hosts can have same logical address. When a packet destined to this logical address is received, it is sent to the host which is nearest in routing topology.

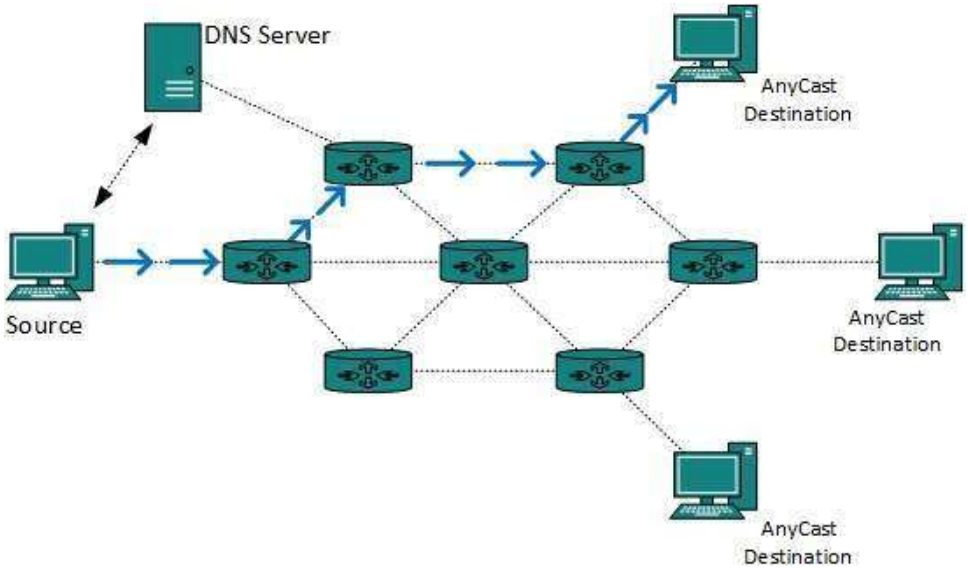


Figure 9.5: Example of Anycast Routing

Anycast routing is done with help of DNS server. Whenever an Anycast packet is received it is enquired with DNS to where to send it. DNS provides the IP address which is the nearest IP configured on it.

Routing Algorithms: The routing algorithms are used to build the routing tables. These algorithms broadly classified as

- *Adaptive* (dynamic) routing algorithms (Example: shortest path routing algo.) and
- *Non-adaptive (static) routing algorithms (Distance Vector and Link state routing algorithms)*

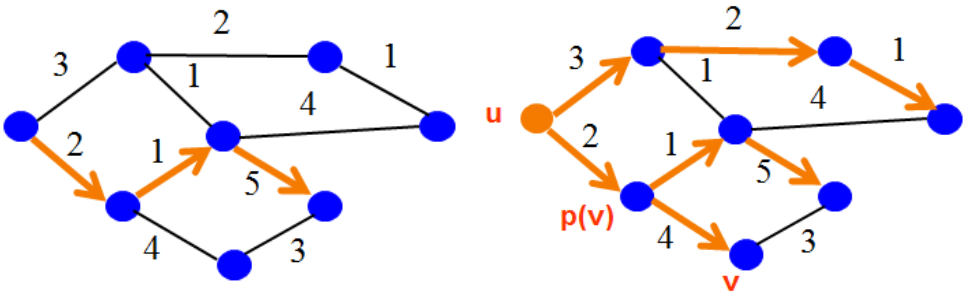
Flooding: It is simplest method packet forwarding. When a packet is received, the routers send it to all the interfaces except the one on which it was received. This creates too much burden on the network and lots of duplicate packets wandering in the network.

- Time to Live (TTL) can be used to avoid infinite looping of packets. There exists another approach for flooding, which is called Selective Flooding to reduce the overhead on the network. In this method, the router does not flood out on all the interfaces, but selective ones.

Shortest Path Routing Algorithm: Routing decision in networks, are mostly taken on the basis of cost between source and destination. Hop count plays major role here. Shortest path is a technique which uses various algorithms to decide a path with minimum number of hops.

Shortest path problem:

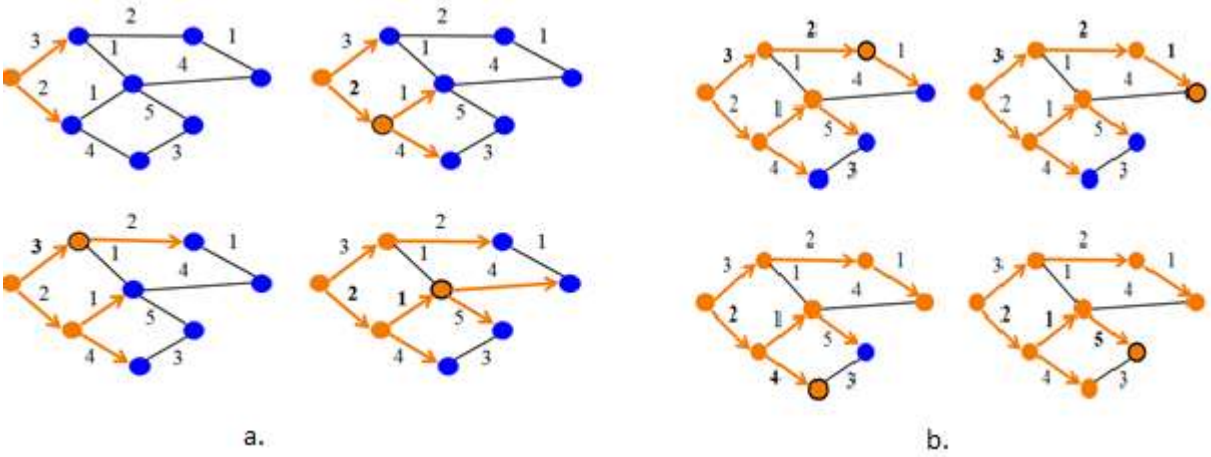
- Path-selection model
  - Destination-based
  - Load-insensitive (e.g., static link weights)
  - Minimum hop count or sum of link weights



- Given: network topology with link costs
  - **c(x,y)**: link cost from node x to node y
  - Infinity if x and y are not direct neighbors
- Compute: least-cost paths to all nodes
  - From a given source u to all other nodes
  - **p(v)**: predecessor node along path from source to v
- Iterative algorithm
  - After k iterations, know least-cost path to k nodes
- **S**: nodes whose least-cost path definitively known
  - Initially, **S = {u}** where u is the source node
  - Add one node to S in each iteration
- **D(v)**: current cost of path from source to node v
  - Initially, **D(v) = c(u,v)** for all nodes v adjacent to u
  - … and **D(v) = ∞** for all other nodes v
  - Continually update D(v) as shorter paths are learned

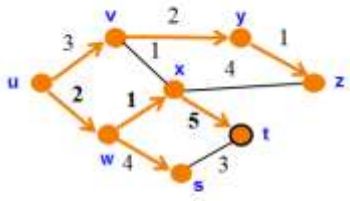
ALGORITHM:

```
1  Initialization:
2    S = {u}
3    for all nodes v
4      if v adjacent to u {
5          D(v) = c(u,v)
6      else D(v) = ∞
7
8  Loop
9    find w not in S with the smallest D(w)
10   add w to S
11   update D(v) for all v adjacent to w and not in S:
12       D(v) = min{D(v), D(w) + c(w,v)}
13 until all nodes in S
```



a. b.

• Shortest-path tree from u • Forwarding table at u



	link
v	(u,v)
w	(u,w)
x	(u,w)
y	(u,v)
z	(u,v)
s	(u,w)
t	(u,w)

Figure 1: Shortest path tree

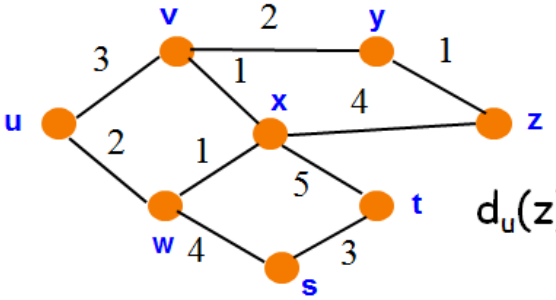
Distance Vector Routing Algorithm: Is one of the two major classes of intra domain routing protocols, the other major class being the link-state protocol. Distance-vector routing protocols use the Bellman–Ford algorithm, Ford–Fulkerson algorithm, or DUAL FSM (in the case of Cisco Systems's protocols) to calculate paths.

A distance-vector routing protocol requires that a router inform its neighbors of topology changes periodically. Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead

The term *distance vector* refers to the fact that the protocol manipulates *vectors* (arrays) of distances to other nodes in the network. The vector distance algorithm was the original ARPANET routing algorithm and was also used in the internet under the name of **RIP** (Routing Information Protocol).

Examples of distance-vector routing protocols include RIPv1 and RIPv2, IGRP and Babel.

- Define distances at each node x
  - $d_x(y)$ = cost of least-cost path from x to y
- Update distances based on neighbors
  - $d_x(y) = \min \{c(x,v) + d_v(y)\}$ over all neighbors v



$$d_u(z) = \min\{c(u,v) + d_v(z), \; c(u,w) + d_w(z)\}$$

- $c(x,v)$ = cost for direct link from x to v
  - Node x maintains costs of direct links c(x,v)
- $D_x(y)$ = estimate of least cost from x to y
  - Node x maintains distance vector $D_x = [D_x(y): y \in N]$
- Node x maintains its neighbors' distance vectors
  - For each neighbor v, x maintains $D_v = [D_v(y): y \in N]$
- Each node v periodically sends D_v to its neighbors
  - And neighbors update their own distance vectors
  - $D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\}$ for each node $y \in N$
- Over time, the distance vector D_x converges

Iterative, asynchronous:

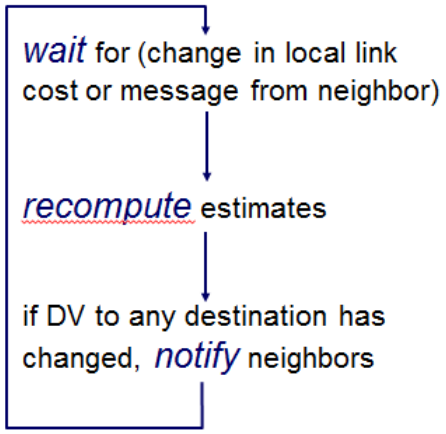
each local iteration caused by:

- Local link cost change
- Distance vector update message from neighbor

Distributed:

- Each node notifies neighbors *only* when its DV changes
- Neighbors then notify their neighbors if necessary

Each node:

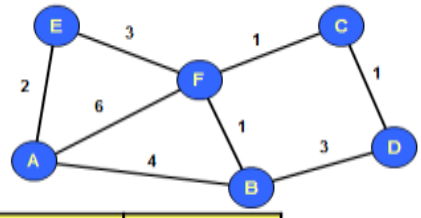


Distance Vector Example: Step 0

Optimum 1-hop paths

Table for A			Table for B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A
B	4	B	B	0	B
C	∞	–	C	∞	–
D	∞	–	D	3	D
E	2	E	E	∞	–
F	6	F	F	1	F

Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	∞	–	A	∞	–	A	2	A	A	6	A
B	∞	–	B	3	B	B	∞	–	B	1	B
C	0	C	C	1	C	C	∞	–	C	1	C
D	1	D	D	0	D	D	∞	–	D	∞	–
E	∞	–	E	∞	–	E	0	E	E	3	E
F	1	F	F	∞	–	F	3	F	F	0	F

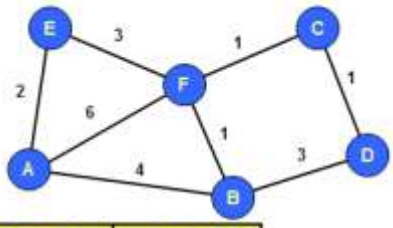


Distance Vector Example: Step 2

Optimum 2-hop paths

Table for A			Table for B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A
B	4	B	B	0	B
C	7	F	C	2	F
D	7	B	D	3	D
E	2	E	E	4	F
F	5	E	F	1	F

Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	7	F	A	7	B	A	2	A	A	5	B
B	2	F	B	3	B	B	4	F	B	1	B
C	0	C	C	1	C	C	4	F	C	1	C
D	1	D	D	0	D	D	∞	–	D	2	C
E	4	F	E	∞	–	E	0	E	E	3	E
F	1	F	F	2	C	F	3	F	F	0	F

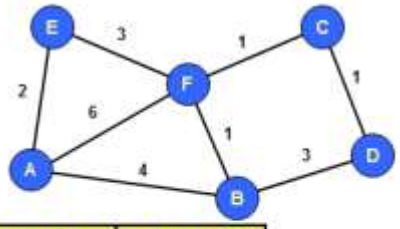


Distance Vector Example: Step 3

Optimum 3-hop paths

Table for A			Table for B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A
B	4	B	B	0	B
C	6	E	C	2	F
D	7	B	D	3	D
E	2	E	E	4	F
F	5	E	F	1	F

Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	6	F	A	7	B	A	2	A	A	5	B
B	2	F	B	3	B	B	4	F	B	1	B
C	0	C	C	1	C	C	4	F	C	1	C
D	1	D	D	0	D	D	5	F	D	2	C
E	4	F	E	5	C	E	0	E	E	3	E
F	1	F	F	2	C	F	3	F	F	0	F



Why does routing matters:

- End-to-end performance
  - Quality of the path affects user performance
  - Propagation delay, throughput, and packet loss
- Use of network resources
  - Balance of the traffic over the routers and links
  - Avoiding congestion by directing traffic to lightly-loaded links
- Transient disruptions during changes
  - Failures, maintenance, and load balancing
  - Limiting packet loss and delay during changes

```
/* Dijkstra's Algorithm for displaying the path from source node to the destination node*/
#include<stdio.h>
#include<conio.h>
#include<process.h>
#include<string.h>
#include<math.h>
#define IN 99
#define N 6
int dijkstra(int cost[][N], int source, int target);
int main()
{
    int cost[N][N],i,j,w,ch,co;
    int source, target,x,y;
    printf("\t The Shortest Path Algorithm ( DIJKSTRA'S ALGORITHM in C \n\n");
    for(i=1;i< N;i++)
    for(j=1;j< N;j++)
    cost[i][j] = IN;
    for(x=1;x< N;x++)
    {
```

```
        for(y=x+1;y< N;y++)
        {
            printf("Enter the weight of the path between nodes %d and %d: ",x,y);
            scanf("%d",&w);
            cost [x][y] = cost[y][x] = w;
        }
        printf("\n");
    }
    printf("\nEnter the source:");
    scanf("%d", &source);
    printf("\nEnter the target");
    scanf("%d", &target);
    co = dijsktra(cost,source,target);
    printf("\nThe Shortest Path: %d",co);
}
int dijsktra(int cost[][N],int source,int target)
{
    int dist[N],prev[N],selected[N]={0},i,m,min,start,d,j;
    char path[N];
    for(i=1;i< N;i++)
    {
        dist[i] = IN;
        prev[i] = -1;
    }
    start = source;
    selected[start]=1;
    dist[start] = 0;
    while(selected[target] ==0)
    {
        min = IN;
        m = 0;
        for(i=1;i< N;i++)
        {
            d = dist[start] +cost[start][i];
            if(d< dist[i]&&selected[i]==0)
            {
                dist[i] = d;
                prev[i] = start;
            }
            if(min>dist[i] && selected[i]==0)
            {
                min = dist[i];
                m = i;
            }
        }
        start = m;
        selected[start] = 1;
    }
    start = target;
    j = 0;
    while(start != -1)
    {
        path[j++] = start+65;
        start = prev[start];
    }
```

```
    path[j]='\0';
    strrev(path);
    printf("%s", path);
    return dist[target];
}
```

/* Distance Vector Routing using Bellman Ford Algorithm: */

```
#include<stdio.h>
struct node
{
    unsigned dist[20];
    unsigned from[20];
}rt[10];
int main()
{
    int costmat[20][20];
    int nodes,i,j,k,count=0;
    printf("\nEnter the number of nodes : ");
    scanf("%d",&nodes);//Enter the nodes
    printf("\nEnter the cost matrix :\n");
    for(i=0;i<nodes;i++)
    {
        for(j=0;j<nodes;j++)
        {
            scanf("%d",&costmat[i][j]);
            costmat[i][i]=0;
            rt[i].dist[j]=costmat[i][j];//initialise the distance equal to cost
matrix
            rt[i].from[j]=j;
        }
    }
        do
        {
            count=0;
            for(i=0;i<nodes;i++)//We choose arbitary vertex k and we calculate the
direct distance from the node i to k using the cost matrix
            //and add the distance from k to node j
            for(j=0;j<nodes;j++)
            for(k=0;k<nodes;k++)
                if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])
                {//We calculate the minimum distance
                    rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
                    rt[i].from[j]=k;
                    count++;
                }
        }while(count!=0);
        for(i=0;i<nodes;i++)
        {
            printf("\n\n For router %d\n",i+1);
            for(j=0;j<nodes;j++)
            {
```

```
                printf("\t\nnode %d via %d Distance %d
",j+1,rt[i].from[j]+1,rt[i].dist[j]);
            }
        }
    printf("\n\n");
    getch();
}
```

```
/*
A sample run of the program works as:-
Enter the number of nodes :
3
Enter the cost matrix :
0 2 7
2 0 1
7 1 0
For router 1
node 1 via 1 Distance 0
node 2 via 2 Distance 2
node 3 via 3 Distance 3
For router 2
node 1 via 1 Distance 2
node 2 via 2 Distance 0
node 3 via 3 Distance 1
For router 3
node 1 via 1 Distance 3
node 2 via 2 Distance 1
node 3 via 3 Distance 0
*/
```

Task: Modify the above program to display the routing tables in tabular form:
Example: ROUTING Table of router 1:

```
 ---------------------------
|Destination | Via |Distance |
 ---------------------------
| 1          | 1   | 0       |
 ---------------------------
| 2          | 3   | 12      |
 ---------------------------
. . . . . . . . . . . . . . .
```

CONCLUSIONS:

- **Run the above programs with different inputs and write your conclusions**

References:

http://www.tutorialspoint.com/data_communication_computer_network/network_layer_routing.htm
http://www.cs.ccsu.edu/~stan/classes/cs490/slides/networks4-ch4-4.pdf
www.cs.princeton.edu/courses/archive/spr06/cos461/.../15**Routing**.ppt