EXPERIMENT 8

OBJECTIVE:

Implement Distance Vector Routing Algorithm

Distance Vector Routing (DVR) Protocol is a method used by routers to find the best path for data to travel across a network. Each router keeps a table that shows the shortest distance to every other router, based on the number of hops (or steps) needed to reach them. Routers share this information with their neighbors, allowing them to update their tables and find the most efficient routes. This protocol helps ensure that data moves quickly and smoothly through the network.

A router transmits its distance vector to each of its neighbors in a routing packet.

Each router receives and saves the most recently received distance vector from each of its neighbors.

A router recalculates its distance vector when:

It receives a distance vector from a neighbor containing different information than before.

It discovers that a link to a neighbor has gone down.

The DV calculation is based on minimizing the cost to each destination

Dx(y) = Estimate of least cost from x to y
C(x,v) = Node x knows cost to each neighbor v
Dx   = [Dx(y): y ? N] = Node x maintains distance vector
Node x also maintains its neighbors' distance vectors
– For each neighbor v, x maintains Dv = [Dv(y): y ? N]

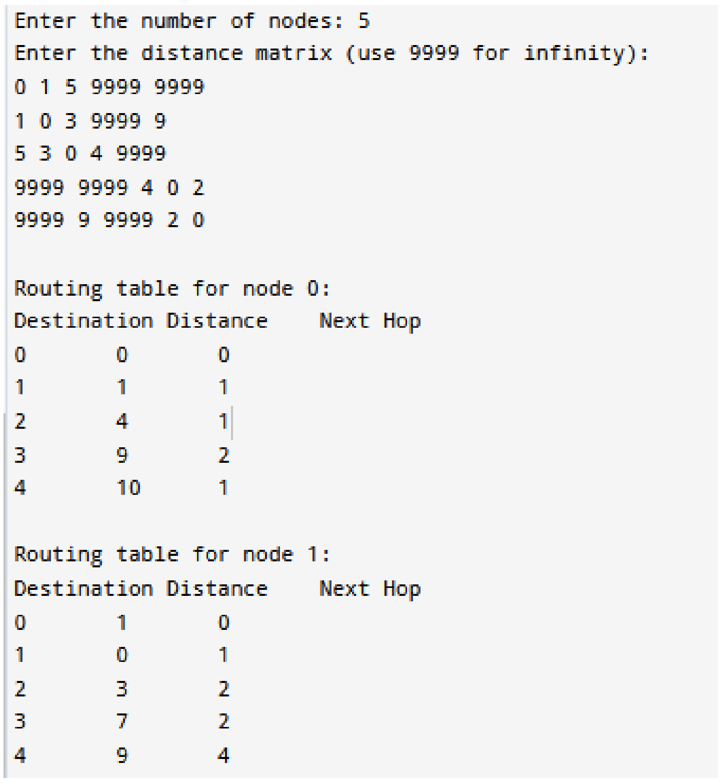
```
#include<stdio.h>
#define MAX_NODES 10
#define INF 9999
struct Node {
int distance[MAX_NODES];
int via[MAX_NODES];
} routingTable[MAX_NODES];

int main() {
intdistanceMatrix[MAX_NODES][MAX_NODES], nodes, i, j, k;

printf("Enter the number of nodes: ");
scanf("%d", &nodes);

printf("Enter the distance matrix (use %d for infinity):\n", INF);
for(i = 0; i < nodes; i++) {
for(j = 0; j < nodes; j++) {
scanf("%d", &distanceMatrix[i][j]);
distanceMatrix[i][i] = 0;  // Distance from a node to itself is 0
routingTable[i].distance[j] = distanceMatrix[i][j];
routingTable[i].via[j] = j;  // Initially, the next hop is the destination itself
        }
    }
    // Distance Vector Algorithm
for(k = 0; k < nodes; k++) {
for(i = 0; i < nodes; i++) {
for(j = 0; j < nodes; j++) {
                // If the distance from i to j via k is less than the direct distance from i to j
if(routingTable[i].distance[j] >distanceMatrix[i][k] + routingTable[k].distance[j]) {
routingTable[i].distance[j] = distanceMatrix[i][k] + routingTable[k].distance[j];
routingTable[i].via[j] = k;  // Update the next hop to the intermediary node k
                }
            }
        }
    }
    // Displaying the routing table for each node
for(i = 0; i < nodes; i++) {
printf("\nRouting table for node %d:\n", i);
printf("Destination\tDistance\tNext Hop\n");
for(j = 0; j < nodes; j++) {
printf("%d\t\t%d\t\t%d\n", j, routingTable[i].distance[j], routingTable[i].via[j]);
        }
    }
return 0;
}
```

OUTPUT



```
Enter the number of nodes: 5
Enter the distance matrix (use 9999 for infinity):
0 1 5 9999 9999
1 0 3 9999 9
5 3 0 4 9999
9999 9999 4 0 2
9999 9 9999 2 0

Routing table for node 0:
Destination Distance    Next Hop
0       0       0
1       1       1
2       4       1
3       9       2
4       10      1

Routing table for node 1:
Destination Distance    Next Hop
0       1       0
1       0       1
2       3       2
3       7       2
4       9       4
```