Experiment 4: Distance Vector Routing

Aim: To generate routing tables for a network of routers using Distance Vector Routing

Objective: After carrying out this experiment, students will be able to:

- Generate routing tables for a given network using Distance Vector Routing
- Analyze the reasons why Distance Vector Routing is adaptive in nature

Problem statement: You are required to write a program that can generate routing tables for a network of routers. Take the number of nodes and the adjacency matrix as input from user. Your program should use this adjacency matrix and create routing tables for all the nodes in the network. The routing table should consist of one entry per destination. This entry should contain the total cost and the outgoing line to reach that destination.

Analysis: While analyzing your program, you are required to address the following points:

- Why is Distance Vector Routing classified as an adaptive routing algorithm?
- Limitations of Distance Vector Routing

MARKS DISTRIBUTION

Component	Maximum Marks	Marks Obtained
Preparation of Document	7	
Results	7	
Viva	6	
Total	**20**	

Submitted by: Subhendu Maji

Register No: 18ETCS002121

1. Algorithm/Flowchart

STEP 1: Start

STEP 2: declare variables

STEP 3: totalNodes, adjMat, source ← from user

STEP 4: for i from 0 to totalNodes, do

    4.1: distances[i] ← ∞

STEP 5: distances[source] ← 0

STEP 6: for i from 0 to totalNodes, do

    6.1: for j from 0 to totalNodes, do

        6.1.1: if j is source or j equal to i or adjMat[i][j] is ∞, continue

        6.1.2: else, relax node by checking if distances[j] greater than distance[i] + adjMat[i][j]. If yes, distance[j] ← distance[i] + adjMat[i][j]

STEP 7: display results

STEP 8: Stop

2. Program

```c
#include <stdio.h>

// struct containing information about routers
typedef struct node
{
    unsigned dist[20];
    unsigned via[20];
} Router;

int main()
{
    // declaring variables
    Router rt[20];
    int adjMat[20][20];
    int n, i, j, k, count = 0;

    // getting information from user
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    printf("Enter the cost matrix:\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            scanf("%d", &adjMat[i][j]);
            adjMat[i][i] = 0;
            rt[i].dist[j] = adjMat[i][j];
            rt[i].via[j] = j;
        }
    }

    // applying bellman ford algorithm
    do
    {
        count = 0;
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                for (k = 0; k < n; k++)
                    // relax edge if a better edge exists
                    if (rt[i].dist[j] > adjMat[i][k] + rt[k].dist[j])
                    {
                        rt[i].dist[j] = adjMat[i][k] + rt[k].dist[j];
                        rt[i].via[j] = k;
                        count++;
                    }

    } while (count != 0);

    // printing results
    for (i = 0; i < n; i++)
    {
        printf("\n\nState value for router %d : \n", i + 1);

        for (j = 0; j < n; j++)
        {
            printf("node %d via %d - Distance: %d\n", j + 1, rt[i].via[j] + 1, rt[i].dist[j]);
        }
    }
    printf("\n");
}

```

Figure 1 Source Code with comments

3. Results

makefile

```
all:
    @gcc lab4.c

run: all
    @./a.out

clean:
    @rm -r *.out

.PHONY: clean
```

Figure 2 makefile

```
subhendu@LAPTOP-C8GE1OC8:/mnt/d/RUAS-sem-05/CN/lab04$ make run
Enter the number of nodes: 4
Enter the cost matrix:
0 3 5 99
3 0 99 1
5 4 0 2
99 1 2 0


State value for router 1 :
node 1 via 1 - Distance: 0
node 2 via 2 - Distance: 3
node 3 via 3 - Distance: 5
node 4 via 2 - Distance: 4


State value for router 2 :
node 1 via 1 - Distance: 3
node 2 via 2 - Distance: 0
node 3 via 4 - Distance: 3
node 4 via 4 - Distance: 1


State value for router 3 :
node 1 via 1 - Distance: 5
node 2 via 4 - Distance: 3
node 3 via 3 - Distance: 0
node 4 via 4 - Distance: 2


State value for router 4 :
node 1 via 2 - Distance: 4
node 2 via 2 - Distance: 1
node 3 via 3 - Distance: 2
node 4 via 4 - Distance: 0

subhendu@LAPTOP-C8GE1OC8:/mnt/d/RUAS-sem-05/CN/lab04$
```

Figure 3 execution

4. Analysis and Discussions

Bellman Ford is one of the algorithms used to calculate the least cost to travel from vertex to vertex. The uniqueness that Bellman Ford bring to the table is that it works with negative edge weights as well. The adjacency matrix consisting of the edge weights of the graph and the number of are given by the user. Using this matrix, “number of nodes minus one” operations for Bellman Ford algorithm were executed. Every vertex was compared with other vertices if there exists an edge between them and if the weight already present is lesser or greater than the new weight. If the new weight is lesser than the old weight, it is exchanged from the new weight. Bellman Ford protocol requires that a router informs its neighbours of topology changes periodically. That is why the Bellman Ford algorithm is called adaptive routing algorithm. The algorithm implementation always checks for N-1 conditions even after these are no more changes to the graph. This is inefficient and extra work for the program.

5. Conclusions

The Distance Vector Routing method using Bellman Ford algorithm was learned and implemented in C.