LAB EXPERIMENT -4

AIM: Write a C program to Implement distance vector routing algorithm for obtaining routing tables at each node.

Distance Vector Routing:

- **Nodes and Distance Vectors**: The program initializes a set of nodes and their distance vectors.
- **Bellman-Ford Algorithm**: This is used to update the distance vectors until they stabilize.
- **Routing Table Construction**: Each node maintains a routing table with the next hop and the distance to each destination.

Explanation:

1. **Initialization**:
   - The `initialize` function sets up the distance vectors and routing tables for each node.
   - If there's a direct link between two nodes, the initial distance is set to the cost of that link, and the next hop is set to the destination node itself. Otherwise, the distance is set to `INF` (representing no direct link), and the next hop is set to `-1`.
2. **Distance Vector Routing Algorithm**:
   - The `distanceVectorRouting` function uses the Bellman-Ford algorithm to update the distance vectors and next hops.
   - The algorithm iteratively updates the distance vectors until no further updates are necessary, indicating convergence.
3. **Printing the Routing Tables**:
   - The `printRoutingTable` function displays the routing table for each node, showing the destination, next hop, and distance.
4. **Main Function**:
   - The `main` function reads the number of nodes and the cost matrix from the user, initializes the data structures, runs the distance vector routing algorithm, and prints the routing tables.

CODE:

```
#include <stdio.h>

#include <stdlib.h>



#define INF 9999

#define MAX_NODES 10
```

```
// Function to initialize distance vector and routing table
void initialize(int numNodes, int costMatrix[MAX_NODES][MAX_NODES], int distVector[MAX_NODES][MAX_NODES], int nextHop[MAX_NODES][MAX_NODES]) {
    for (int i = 0; i < numNodes; i++) {
        for (int j = 0; j < numNodes; j++) {
            distVector[i][j] = costMatrix[i][j];
            if (costMatrix[i][j] != INF && i != j) {
                nextHop[i][j] = j;
            } else {
                nextHop[i][j] = -1;
            }
        }
    }
}

// Function to print routing table for each node
void printRoutingTable(int numNodes, int distVector[MAX_NODES][MAX_NODES], int nextHop[MAX_NODES][MAX_NODES]) {
    for (int i = 0; i < numNodes; i++) {
        printf("Routing table for node %d:\n", i);
        printf("Destination\tNext Hop\tDistance\n");
        for (int j = 0; j < numNodes; j++) {
            if (distVector[i][j] == INF) {
                printf("%d\t\t-\t\tINF\n", j);
            } else {
                printf("%d\t\t%d\t\t%d\n", j, nextHop[i][j], distVector[i][j]);
            }
        }
```

```
        printf("\n");
    }
}

// Function to implement Distance Vector Routing algorithm
void distanceVectorRouting(int numNodes, int costMatrix[MAX_NODES][MAX_NODES], int distVector[MAX_NODES][MAX_NODES], int nextHop[MAX_NODES][MAX_NODES]) {
    int updated;

    do {
        updated = 0;
        for (int i = 0; i < numNodes; i++) {
            for (int j = 0; j < numNodes; j++) {
                for (int k = 0; k < numNodes; k++) {
                    if (distVector[i][k] + distVector[k][j] < distVector[i][j]) {
                        distVector[i][j] = distVector[i][k] + distVector[k][j];
                        nextHop[i][j] = nextHop[i][k];
                        updated = 1;
                    }
                }
            }
        }
    } while (updated);
}

int main() {
    int numNodes, costMatrix[MAX_NODES][MAX_NODES];
    int distVector[MAX_NODES][MAX_NODES];
```

```
    int nextHop[MAX_NODES][MAX_NODES];

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

    printf("Enter the cost matrix (use %d for INF):\n", INF);
    for (int i = 0; i < numNodes; i++) {
        for (int j = 0; j < numNodes; j++) {
            scanf("%d", &costMatrix[i][j]);
        }
    }

    initialize(numNodes, costMatrix, distVector, nextHop);
    distanceVectorRouting(numNodes, costMatrix, distVector, nextHop);
    printRoutingTable(numNodes, distVector, nextHop);

    return 0;
}
```

INPUT:

```
Enter the number of nodes: 4
Enter the cost matrix (use 9999 for INF):
0       12      9       4
12      0       1       9999
9       1       0       2
4       9999    2       0
```

CHECK THE OUTPUT FOR THE ABOVE INPUT.