**Lab Practical #12:**

To develop network using distance vector routing protocol and link state routing protocol.

**Practical Assignment #12:**

1. **C/Java Program: Distance Vector Routing Algorithm using Bellman Ford's Algorithm.**

import java.util.Arrays;

public class DistanceVectorRouting {

    private static final int V = 5;  // Number of vertices (or routers)

    private static final int INF = 999;  // Infinity representation

    static class Node {

        int[] distance = new int[V];

        int[] nextHop = new int[V];

    }

    public static void bellmanFord(int[][] graph) {

        Node[] routingTable = new Node[V];

        for (int i = 0; i < V; i++) {

            routingTable[i] = new Node();

            for (int j = 0; j < V; j++) {

                routingTable[i].distance[j] = graph[i][j];

                routingTable[i].nextHop[j] = j;

            }

        }

        // Relax all edges V-1 times (Bellman-Ford)

        for (int k = 0; k < V - 1; k++) {

            for (int i = 0; i < V; i++) {

                for (int j = 0; j < V; j++) {

                    for (int v = 0; v < V; v++) {

                        if (routingTable[i].distance[v] > graph[i][j] + routingTable[j].distance[v]) {

                            routingTable[i].distance[v] = graph[i][j] + routingTable[j].distance[v];

                            routingTable[i].nextHop[v] = j;

                        }

                    }

                }

            }

        }

        // Print the final routing table

        for (int i = 0; i < V; i++) {

            System.out.println("Routing table for router " + (i+1) + ":");

            System.out.println("Destination\tNext Hop\tDistance");

            for (int j = 0; j < V; j++) {

                System.out.println(j + "\t\t\t" + routingTable[i].nextHop[j] + "\t\t\t" + routingTable[i].distance[j]);

            }

            System.out.println();

        }

    }

    public static void main(String[] args) {

        int[][] graph = {

                {0, 2, INF, 1, INF},

                {2, 0, 3, INF, INF},

                {INF, 3, 0, 2, 1},

                {1, INF, 2, 0, 3},

                {INF, INF, 1, 3, 0}

        };

        bellmanFord(graph);

    }

}

1. **C/Java Program: Link state routing algorithm.**

import java.util.Arrays;

public class LinkStateRouting {

    private static final int V = 5;  // Number of vertices (or routers)

    private static final int INF = 9999;  // Infinity

    int minDistance(int[] dist, boolean[] sptSet) {

        int min = INF, min\_index = -1;

        for (int v = 0; v < V; v++) {

            if (!sptSet[v] && dist[v] <= min) {

                min = dist[v];

                min\_index = v;

            }

        }

        return min\_index;

    }

    void dijkstra(int[][] graph, int src) {

        int[] dist = new int[V];

        boolean[] sptSet = new boolean[V];

        Arrays.fill(dist, INF);

        Arrays.fill(sptSet, false);

        dist[src] = 0;

        for (int count = 0; count < V - 1; count++) {

            int u = minDistance(dist, sptSet);

            sptSet[u] = true;

            for (int v = 0; v < V; v++) {

                if (!sptSet[v] && graph[u][v] != 0 && dist[u] != INF && dist[u] + graph[u][v] < dist[v]) {

                    dist[v] = dist[u] + graph[u][v];

                }

            }

        }

        System.out.println("Vertex \t Distance from Source " + src);

        for (int i = 0; i < V; i++) {

            System.out.println(i + " \t\t " + dist[i]);

        }

    }

    public static void main(String[] args) {

        int[][] graph = {

                {0, 2, INF, 1, INF},

                {2, 0, 3, INF, INF},

                {INF, 3, 0, 2, 1},

                {1, INF, 2, 0, 3},

                {INF, INF, 1, 3, 0}

        };

        LinkStateRouting lsr = new LinkStateRouting();

        lsr.dijkstra(graph, 0);

    }

}