**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.**
   1. Java Code :

import java.util.Arrays;

import java.util.Scanner;

public class DistanceVectorRouting {

private int vertices;

private int[][] distanceTable;

private int[] distance;

public DistanceVectorRouting(int vertices) {

this.vertices = vertices;

distanceTable = new int[vertices][vertices];

distance = new int[vertices];

}

public void initializeGraph(int[][] graph) {

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

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

distanceTable[i][j] = graph[i][j];

}

}

}

public void bellmanFord(int src) {

Arrays.fill(distance, Integer.MAX\_VALUE);

distance[src] = 0;

for (int i = 1; i < vertices; i++) {

for (int u = 0; u < vertices; u++) {

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

if (distanceTable[u][v] != Integer.MAX\_VALUE) {

if (distance[u] != Integer.MAX\_VALUE &&

distance[u] + distanceTable[u][v] < distance[v]) {

distance[v] = distance[u] + distanceTable[u][v];

}

}

}

}

}

for (int u = 0; u < vertices; u++) {

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

if (distanceTable[u][v] != Integer.MAX\_VALUE) {

if (distance[u] != Integer.MAX\_VALUE &&

distance[u] + distanceTable[u][v] < distance[v]) {

System.out.println("Graph contains negative weight cycle!");

return;

}

}

}

}

}

public void printDistanceTable() {

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

System.out.println("Distance vector for router " + i + ":");

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

if (distanceTable[i][j] == Integer.MAX\_VALUE) {

System.out.print("INF ");

} else {

System.out.print(distanceTable[i][j] + " ");

}

}

System.out.println();

}

}

public void printResult(int src) {

System.out.println("Router " + src + " distance table:");

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

if (distance[i] == Integer.MAX\_VALUE) {

System.out.println("Destination " + i + ": INF");

} else {

System.out.println("Destination " + i + ": " + distance[i]);

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of routers: ");

int vertices = sc.nextInt();

DistanceVectorRouting dvr = new DistanceVectorRouting(vertices);

int[][] graph = new int[vertices][vertices];

System.out.println("Enter the cost matrix (use 9999 for no direct link): ");

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

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

graph[i][j] = sc.nextInt();

if (graph[i][j] == 9999) {

graph[i][j] = Integer.MAX\_VALUE;

}

}

}

dvr.initializeGraph(graph);

dvr.printDistanceTable();

System.out.print("Enter the source router: ");

int src = sc.nextInt();

dvr.bellmanFord(src);

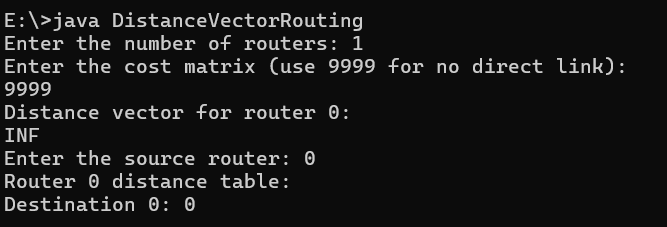
dvr.printResult(src);

sc.close();

}

}

* **Output :-**

****

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

import java.util.Scanner;

public class LinkStateRouting {

private int vertices;

private int[][] graph;

private boolean[] visited;

private int[] distance;

public LinkStateRouting(int vertices) {

this.vertices = vertices;

graph = new int[vertices][vertices];

visited = new boolean[vertices];

distance = new int[vertices];

}

public void initializeGraph(int[][] graphInput) {

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

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

graph[i][j] = graphInput[i][j];

}

}

}

public void dijkstra(int src) {

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

distance[i] = Integer.MAX\_VALUE;

visited[i] = false;

}

distance[src] = 0;

for (int i = 0; i < vertices - 1; i++) {

int u = getMinimumDistanceNode();

visited[u] = true;

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

if (!visited[v] && graph[u][v] != 0 &&

distance[u] != Integer.MAX\_VALUE &&

distance[u] + graph[u][v] < distance[v]) {

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

}

}

}

}

private int getMinimumDistanceNode() {

int min = Integer.MAX\_VALUE, minIndex = -1;

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

if (!visited[i] && distance[i] < min) {

min = distance[i];

minIndex = i;

}

}

return minIndex;

}

public void printResult(int src) {

System.out.println("Router " + src + " shortest path to other routers:");

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

if (distance[i] == Integer.MAX\_VALUE) {

System.out.println("Destination " + i + ": INF");

} else {

System.out.println("Destination " + i + ": " + distance[i]);

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of routers: ");

int vertices = sc.nextInt();

LinkStateRouting lsr = new LinkStateRouting(vertices);

int[][] graph = new int[vertices][vertices];

System.out.println("Enter the cost matrix (use 0 for no direct link): ");

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

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

graph[i][j] = sc.nextInt();

}

}

lsr.initializeGraph(graph);

System.out.print("Enter the source router: ");

int src = sc.nextInt();

lsr.dijkstra(src);

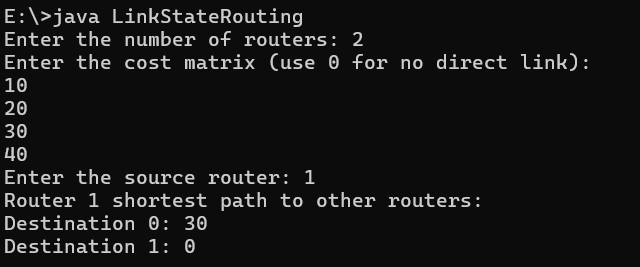
lsr.printResult(src);

sc.close();

}

}

* **Output :-**

****