Experiment-9

**Date**-June 03, 2021.

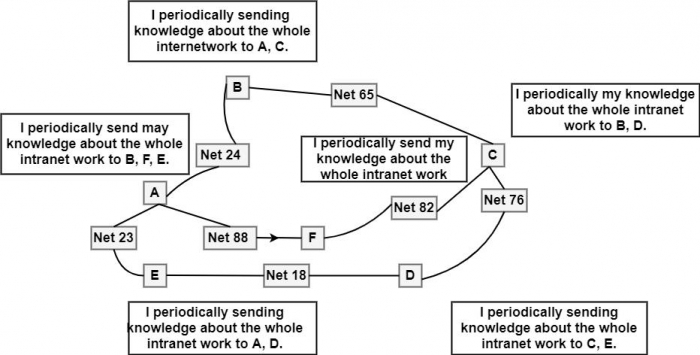
**AIM-** Implementation of distance vector routing algorithm.

**Distance Vector Routing Algorithm**

A distance-vector routing (DVR) protocol requires that a router inform its neighbors of topology changes periodically. Historically known as the old ARPANET routing algorithm. A distance-vector routing protocol in data networks determines the best route for data packets based on distance. Distance-vector routing protocols measure the distance by the number of routers a packet has to pass, one router counts as one hop. Some distance-vector protocols also take into account network latency and other factors that influence traffic on a given route. To determine the best route across a network, routers, on which a distance-vector protocol is implemented, exchange information with one another, usually routing tables plus hop counts for destination networks and possibly other traffic information. Distance-vector routing protocols also require that a router informs its neighbors of network topology changes periodically. Distance-vector routing protocols use the Bellman–Ford algorithm to calculate the best route. Another way of calculating the best route across a network is based on link cost, and is implemented through link-state routing protocols. The Distance vector algorithm is a dynamic algorithm. It is mainly used in ARPANET, and RIP. Each router maintains a distance table known as Vector.

The Distance vector algorithm is iterative, asynchronous and distributed.

* **Distributed**: It is distributed in that each node receives information from one or more of its directly attached neighbors, performs calculation and then distributes the result back to its neighbors.
* **Iterative**: It is iterative in that its process continues until no more information is available to be exchanged between neighbors.
* **Asynchronous**: It does not require that all of its nodes operate in the lock step with each other.



**Advantages of Distance Vector routing –**

* It is simpler to configure and maintain than link state routing.

**Disadvantages of Distance Vector routing –**

* It is slower to converge than link state.
* It is at risk from the count-to-infinity problem.

It creates more traffic than link state since a hop count change must be propagated to all routers and processed on each router.

**PROGRAM (JAVA)-**

import java.io.\*;

import java.util.\*;

public class A {

static int graph[][];

static int via[][];

static int rt[][];

static int v;

static int e;

public static void main(String args[]) {

Scanner scn = new Scanner(System.in);

v = 4;

e = 5;

graph = new int[v][v];

via = new int[v][v];

rt = new int[v][v];

for (int i = 0; i < v; i++)

for (int j = 0; j < v; j++)

if (i == j)

graph[i][j] = 0;

else

graph[i][j] = 9999;

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

System.out.println("\nPlease enter data for Edge " + (i + 1) + ":");

System.out.print("Source: ");

int s = scn.nextInt();

System.out.print("Destination: ");

int d = scn.nextInt();

System.out.print("Cost: ");

int c = scn.nextInt();

graph[s - 1][d - 1] = c;

graph[d - 1][s - 1] = c;

}

System.out.print("\nEnter Source Node : ");

int src = scn.nextInt();

System.out.print("Enter destination node : ");

int dest = scn.nextInt();

dvr\_calc\_disp("\nThe initial min distance b/w src and dest : ", src - 1, dest - 1);

System.out.print("\nPlease enter the Source Node for the edge whose cost has changed: ");

int s = scn.nextInt();

System.out.print("Please enter the Destination Node for the edge whose cost has changed: ");

int d = scn.nextInt();

System.out.print("Please enter the new cost: ");

int c = scn.nextInt();

graph[s - 1][d - 1] = c;

graph[d - 1][s - 1] = c;

dvr\_calc\_disp("\nThe final min distance b/w src and dest : ", src - 1, dest - 1);

}

static void dvr\_calc\_disp(String message, int src, int dest) {

init\_tables();

update\_tables();

System.out.print(message);

print\_tables(src, dest);

}

static void update\_table(int source) {

for (int i = 0; i < v; i++)

if (graph[source][i] != 9999) {

int dist = graph[source][i];

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

int inter\_dist = rt[i][j];

if (via[i][j] == source)

inter\_dist = 9999;

if (dist + inter\_dist < rt[source][j]) {

rt[source][j] = dist + inter\_dist;

via[source][j] = i;

}

}

}

}

static void update\_tables() {

int k = 0;

for (int i = 0; i < 4 \* v; i++) {

update\_table(k);

k++;

if (k == v) k = 0;

}

}

static void init\_tables() {

for (int i = 0; i < v; i++)

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

if (i == j) {

rt[i][j] = 0;

via[i][j] = i;

} else {

rt[i][j] = 9999;

via[i][j] = 100;

}

}

}

static void print\_tables(int src, int dest) {

for (int i = 0; i < v; i++)

for (int j = 0; j < v; j++)

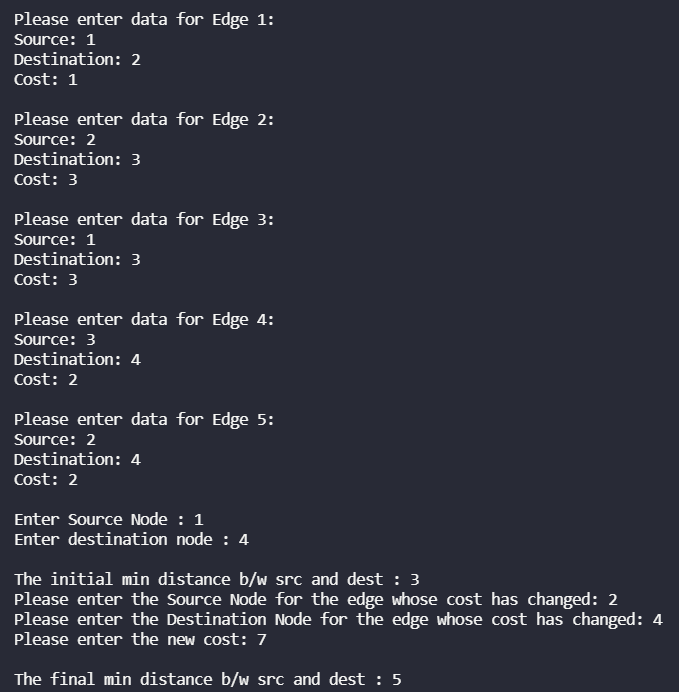
if (i == src && j == dest)

System.out.print(rt[i][j]);

}

}

**OUTPUT-**

****