Assignment No.6

Title:

Write a program to implement link state /Distance vector routing protocol to find suitable path for transmission.

Theory:

Link State Routing

Link state routing is a technique in which each router shares the knowledge of its neighborhood with *every other router in the internetwork.*

The three keys to understand the Link State Routing algorithm:

- **Knowledge about the neighborhood:** Instead of sending its routing table, a router sends the information about its neighborhood only. A router broadcast its identities and cost of the directly attached links to other routers.
- **Flooding:** Each router sends the information to every other router on the internetwork except its neighbors. This process is known as Flooding. Every router that receives the packet sends the copies to all its neighbors. Finally, each and every router receives a copy of the same information.
- **Information sharing:** A router sends the information to every other router only when the change occurs in the information.

Link State Routing has two phases:

Reliable Flooding

- **Initial state:** Each node knows the cost of its neighbors.
- **Final state:** Each node knows the entire graph.

Route Calculation

Each node uses Dijkstra's algorithm on the graph to calculate the optimal routes to all nodes.

- The Link state routing algorithm is also known as Dijkstra's algorithm which is used to find the shortest path from one node to every other node in the network.

- The Dijkstra's algorithm is an iterative, and it has the property that after kth iteration of the algorithm, the least cost paths are well known for k destination nodes.

Let's describe some notations:

- **c(i , j):** Link cost from node i to node j. If i and j nodes are not directly linked, then $c(i , j) = \infty$.
- **D(v):** It defines the cost of the path from source code to destination v that has the least cost currently.
- **P(v):** It defines the previous node (neighbor of v) along with current least cost path from source to v.
- **N:** It is the total number of nodes available in the network.

Algorithm

```
Initialization
N = {A}     // A is a root node.
for all nodes v
if v adjacent to A
then D(v) = c(A,v)
else D(v) = infinity
loop
find w not in N such that D(w) is a minimum.
Add w to N
Update D(v) for all v adjacent to w and not in N:
D(v) = min(D(v) , D(w) + c(w,v))
Until all nodes in N
```

In the above algorithm, an initialization step is followed by the loop. The number of times the loop is executed is equal to the total number of nodes available in the network.

Disadvantage:

Heavy traffic is created in Line state routing due to Flooding. Flooding can cause an infinite looping, this problem can be solved by using Time-to-leave field

Distance Vector Routing (DVR) Protocol

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 (or known as Bellman-Ford algorithm).

Bellman Ford Basics – Each router maintains a Distance Vector table containing the distance between itself and ALL possible destination nodes. Distances,based on a chosen metric, are computed using information from the neighbors' distance vectors.

Information kept by DV router -

- Each router has an ID
- Associated with each link connected to a router,
- there is a link cost (static or dynamic).
- Intermediate hops

Distance Vector Table Initialization -

- Distance to itself = 0
- Distance to ALL other routers = infinity number.

Distance Vector Algorithm –

1. A router transmits its distance vector to each of its neighbors in a routing packet.
2. Each router receives and saves the most recently received distance vector from each of its neighbors.
3. A router recalculates its distance vector when:
   - It receives a distance vector from a neighbor containing different information than before.
   - It discovers that a link to a neighbor has gone down.

The DV calculation is based on minimizing the cost to each destination

Dx(y) = Estimate of least cost from x to y

C(x,v) = Node x knows cost to each neighbor v

Dx = [Dx(y): $y \in N$] = Node x maintains distance vector

Node x also maintains its neighbors' distance vectors

– For each neighbor v, x maintains Dv = [Dv(y): $y \in N$]

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. Hop count updates take place on a periodic basis, even if there are no changes in the network topology, so bandwidth-wasting broadcasts still occur.
- For larger networks, distance vector routing results in larger routing tables than link state since each router must know about all other routers. This can also lead to congestion on WAN links.

Conclusion:

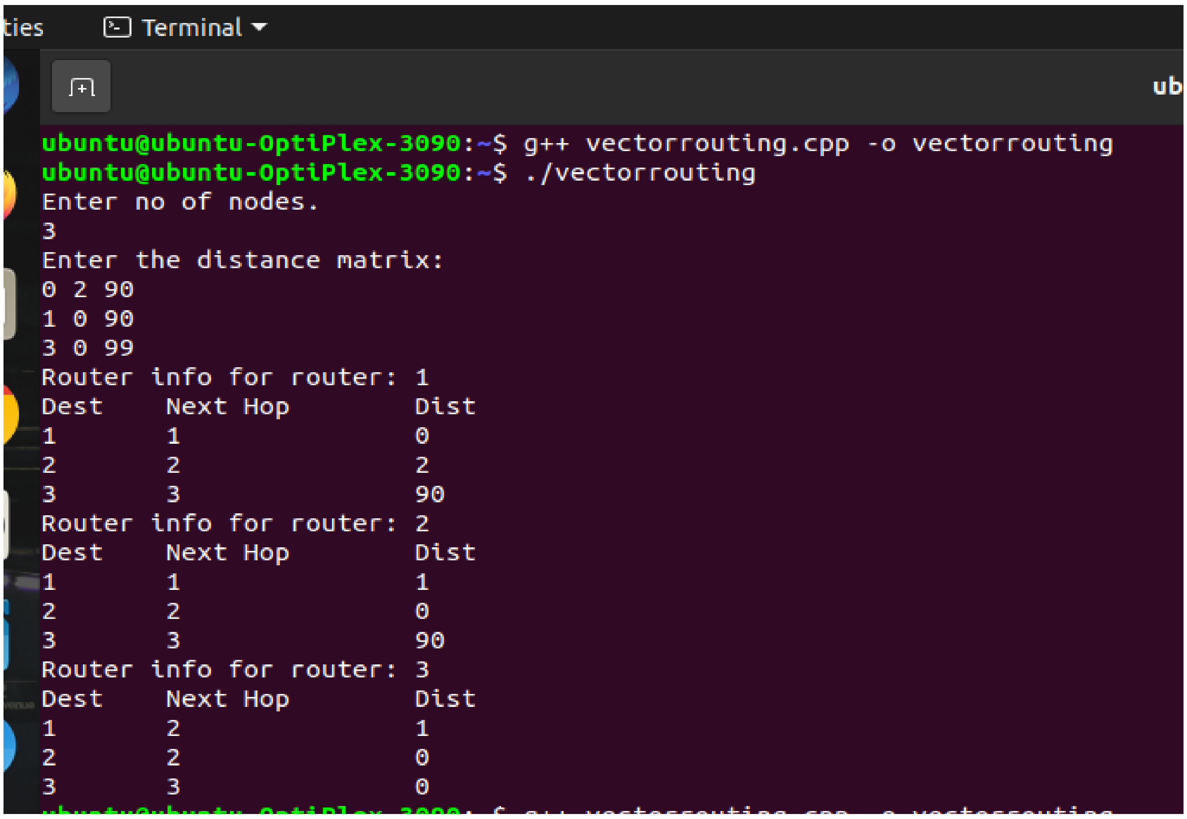
We have successfully implemented a program to implement link state /Distance vector routing protocol to find suitable path for transmission

```
#include <iostream>
using namespace std;
struct node
{
   int dist[20];
   int from[20];
} route[10];

int main()
{
   int dm[20][20], no;
   cout << "Enter no of nodes." << endl;
   cin >> no;
   cout << "Enter the distance matrix:" << endl;
   for (int i = 0; i < no; i++)
   {
      for (int j = 0; j < no; j++)
      {
         cin >> dm[i][j];
         /*  Set distance from i to i as 0 */
         dm[i][i] = 0;
         route[i].dist[j] = dm[i][j];
         route[i].from[j] = j;
      }
```

```
}
int flag;
do
{
    flag = 0;
    for (int i = 0; i < no; i++)
    {
        for (int j = 0; j < no; j++)
        {
            for (int k = 0; k < no; k++)
            {
                if ((route[i].dist[j]) > (route[i].dist[k] + route[k].dist[j]))
                {
                    route[i].dist[j] = route[i].dist[k] + route[k].dist[j];
                    route[i].from[j] = k;
                    flag = 1;
                }
            }
        }
    }
} while (flag);
for (int i = 0; i < no; i++)
{
    cout << "Router info for router: " << i + 1 << endl;
```

```
        cout << "Dest\tNext Hop\tDist" << endl;

        for (int j = 0; j < no; j++)

            printf("%d\t%d\t\t%d\n", j + 1, route[i].from[j] + 1,
route[i].dist[j]);

    }

    return 0;

}
```

OUTPUT:



```
ubuntu@ubuntu-OptiPlex-3090:~$ g++ vectorrouting.cpp -o vectorrouting
ubuntu@ubuntu-OptiPlex-3090:~$ ./vectorrouting
Enter no of nodes.
3
Enter the distance matrix:
0 2 90
1 0 90
3 0 99
Router info for router: 1
Dest    Next Hop        Dist
1       1               0
2       2               2
3       3               90
Router info for router: 2
Dest    Next Hop        Dist
1       1               1
2       2               0
3       3               90
Router info for router: 3
Dest    Next Hop        Dist
1       2               1
2       2               0
3       3               0
```