Experiment-10

**Date**-June 10, 2021.

**AIM-** Implementation of Link State routing algorithm.

**Link State Routing Algorithm**

Link state routing is the second family of routing protocols. While distance vector routers use a distributed algorithm to compute their routing tables, link-state routing uses link-state routers to exchange messages that allow each router to learn the entire network topology. Based on this learned topology, each router is then able to compute its routing table by using a shortest path computation. It is a dynamic routing algorithm in which each router shares knowledge of its neighbours with every other router in the network. A router sends its information about its neighbours only to all the routers through flooding. Information sharing takes place only whenever there is a change. It makes use of **Dijkstra’s Algorithm** for making routing tables. **Problems are h**eavy traffic due to flooding of packets and flooding can result in infinite looping which can be solved by using **Time to live (TTL)** field.

**Features of link state routing protocols:**

* **Link state packet –** A small packet that contains routing information.
* **Link state database –** Collection information gathered from link state packet.
* **Shortest path first algorithm (Dijkstra algorithm) –** A calculation performed on the database results into shortest path
* **Routing table –** A list of known paths and interfaces.

**Calculation of shortest path:**To find shortest path, each node need to run the famous **Dijkstra algorithm**. This famous algorithm uses the following steps:

1. The node is taken and chosen as a root node of the tree, this creates the tree with a single node, and now set the total cost of each node to some value based on the information in Link State Database.
2. Now the node selects one node, among all the nodes not in the tree like structure, which is nearest to the root, and adds this to the tree. The shape of the tree gets changed.
3. After this node is added to the tree, the cost of all the nodes not in the tree needs to be updated because the paths may have been changed.
4. The node repeats the Step 2 and Step 3 until all the nodes are added in the tree.

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:

1. Reliable Flooding

**Initial state:** Each node knows the cost of its neighbors.

**Final state:** Each node knows the entire graph.

1. 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.

**Link State protocols in comparison to Distance Vector protocols have:**

* It requires large amount of memory.
* Shortest path computations require many CPU circles.
* If network use the little bandwidth ; it quickly reacts to topology changes
* All items in the database must be sent to neighbors to form link state packets.
* All neighbors must be trusted in the topology.
* Authentication mechanisms can be used to avoid undesired adjacency and problems.
* No split horizon techniques are possible in the link state routing.

| **BASIS FOR COMPARISON** | **DISTANCE VECTOR ROUTING** | **LINK STATE ROUTING** |
| --- | --- | --- |
| Algorithm | Bellman ford | Dijsktra |
| Network view | Topology information from the neighbour point of view | Complete information on the network topology |
| Best path calculation | Based on the least number of hops | Based on the cost |
| Updates | Full routing table | Link state updates |
| Updates frequency | Periodic updates | Triggered updates |
| CPU and memory | Low utilisation | Intensive |
| Simplicity | High simplicity | Requires a trained network administrator |
| Convergence time | Moderate | Fast |
| Updates | On broadcast | On multicast |
| Hierarchical structure | No | Yes |
| Intermediate Nodes | No | Yes |

**PROGRAM (C++)-**

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

#define INF INT\_MAX

void print\_route(vector < int >const & prev, int i) {

if (i < 0)

return;

print\_route(prev, prev[i]);

cout << i << " ";

}

int find\_vertex(int v, int \*distance, bool \*visited){

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

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

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

min\_vertex = i;

min = distance[i];

}

return min\_vertex;

}

void dijkistra(int v, int source, vector<vector<int>> vec, int destination){

bool \*visited = new bool[v];

int \*distance = new int[v];

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

visited[i] = false;

distance[i] = INF;

}

vector<int> last;

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

last.push\_back(-1);

}

distance[source] = 0;

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

int vertex = find\_vertex(v, distance, visited);

visited[vertex] = true;

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

if(visited[j] != true && vec[vertex][j] != INF){

int num = distance[vertex] + vec[vertex][j];

if(num < distance[j]){

distance[j] = num;

last[j] = vertex;

}

}

}

if(destination == -1){

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

cout << "Path (" << source << " —> " << i << "): Minimum cost = "<< distance[i] << ", Route = [ ";

print\_route(last, i);

cout << "]" << endl;

}

cout<<endl<<endl;

}

else{

cout << "Path (" << source << " —> " << destination << "): Minimum cost = "<< distance[destination] << ", Route = [ ";

print\_route(last, destination);

cout << "]" << endl;

}

}

int main() {

cout<<"Enter vertices \n";

int v; cin>>v;

cout<<"Enter edges \n";

int e; cin>>e;

vector<vector<int>> vec( v , vector<int> (v));

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

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

vec[i][j] = INF;

cout<<"Enter edges' source, destination and weight \n";

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

int x,y,d; cin>>x>>y>>d;

vec[x][y] = d;

vec[y][x] = d;

}

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

dijkistra(v, i, vec, -1);

int source, destination;

cout<<"Enter source and destination \n";

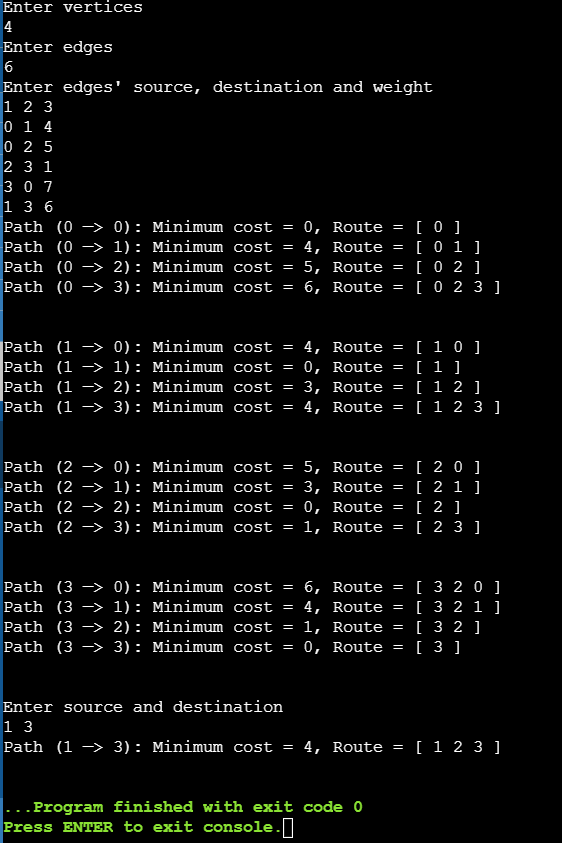
cin>>source>>destination;

dijkistra(v, source, vec, destination);

return 0;

}

**OUTPUT-**

****