# Experiment – 7

**AIM**- Write a program to implement link state routing algorithm..

**Description -** Link-state routing protocols are one of the two main classes of routing protocols used in packet switching networks for computer communications, the other being distance vector routing protocols. Examples of link-state routing protocols include Open Shortest Path First (OSPF) and intermediate system to intermediate system (IS-IS).

The link-state protocol is performed by every switching node in the network (i.e., nodes that are prepared to forward packets; in the Internet, these are called routers). The basic concept of link-state routing is that every node constructs a map of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes. Each node then independently calculates the next best logical path from it to every possible destination in the network. Each collection of best paths will then form each node's routing table.

**Algorithms:**
Each node independently runs an algorithm over the map to determine the shortest path from itself to every other node in the network; generally some variant of Dijkstra's algorithm is used. This is based around a link cost across each path which includes available bandwidth among other things.

A node maintains two data structures: a tree containing nodes which are "done", and a list of candidates. The algorithm starts with both structures empty; it then adds to the first one the node itself.

The variant of a Greedy Algorithm then repetitively does the following:

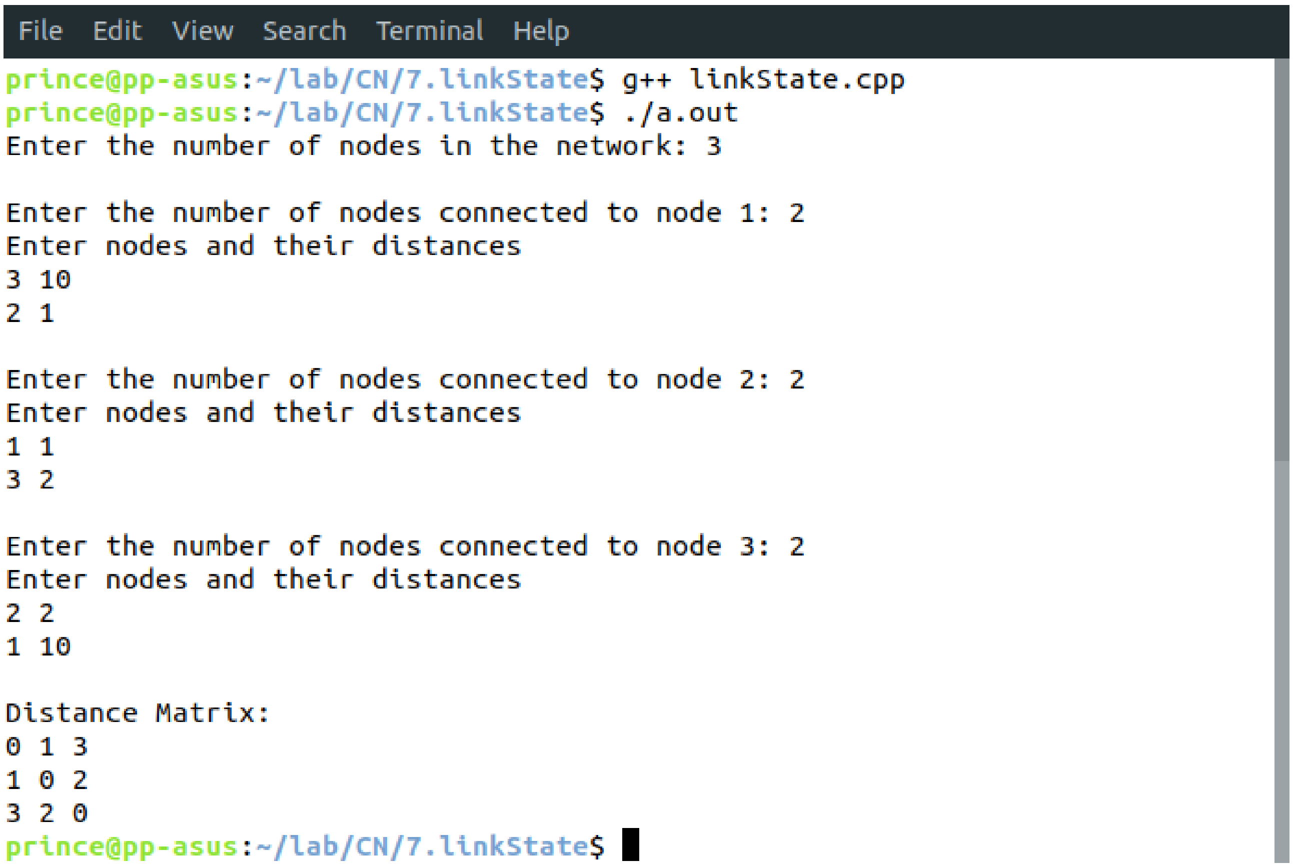
1. All neighbor nodes which are directly connected to the node are just added to the tree (excepting any nodes which are already in either the tree or the candidate list). The rest are added to the second (candidate) list.

2. Each node in the candidate list is compared to each of the nodes already in the tree. The candidate node which is closest to any of the nodes already in the tree is itself moved into the tree and attached to the appropriate neighbor node. When a node is moved from the candidate list into the tree, it is removed from the candidate list and is not considered in subsequent iterations of the algorithm.

**Code:**

```
#include<bits/stdc++.h>
using namespace std;

class compare{
    public:
        bool operator()(pair<int, int> child, pair<int, int> parent){
            return child.first > parent.first;
        }
};

vector<int> dijkstra(vector<vector<int> > &dMat, int node){
    vector<int> dist(dMat.size(), 10000);
    priority_queue<pair<int, int>, vector<pair<int, int>>, compare> q;
    q.push(make_pair(0, node));

    while(!q.empty()){
        int n = q.top().second;
        int d = q.top().first;
        q.pop();
        for(int i=1; i<dMat.size(); i++){
            if(dist[i] > d + dMat[n][i]){
                dist[i] = d + dMat[n][i];
                q.push(make_pair(dist[i], i));
            }
        }
    }
    return dist;
}

int main(){
    int n;
    cout<<"Enter the number of nodes in the network: ";
    cin>>n;
    vector<vector<int>> dMat(n+1, vector<int>(n+1, 10000));

    for(int i=1; i<=n; i++){
        int childs;
        cout<<endl;
        cout<<"Enter the number of nodes connected to node "<<i<<": ";
        cin>>childs;
        dMat[i][i] = 0;
        if(childs > 0){
            cout<<"Enter nodes and their distances "<<endl;
        }
        for (int j=0; j<childs; j++){
            int sibling;
            int distance;
            cin >> sibling >> distance;
            dMat[i][sibling] = distance;
            dMat[sibling][i] = distance;
        }
    }

    vector<vector<int> > dMatTemp = dMat;

    for (int i=1; i<=n; i++){
        vector<int> d = dijkstra(dMatTemp, i);
        dMat[i] = d;
    }

    cout<<endl;
    cout<<"Distance Matrix: "<<endl;
    for (int i=1; i<=n; i++){
        for (int j=1; j <=n;  j++){
            if(dMat[i][j] >= 10000)
                dMat[i][j] = -1;
            cout<<dMat[i][j]<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```

**Output:**



```
prince@pp-asus:~/lab/CN/7.linkState$ g++ linkState.cpp
prince@pp-asus:~/lab/CN/7.linkState$ ./a.out
Enter the number of nodes in the network: 3

Enter the number of nodes connected to node 1: 2
Enter nodes and their distances
3 10
2 1

Enter the number of nodes connected to node 2: 2
Enter nodes and their distances
1 1
3 2

Enter the number of nodes connected to node 3: 2
Enter nodes and their distances
2 2
1 10

Distance Matrix:
0 1 3
1 0 2
3 2 0
prince@pp-asus:~/lab/CN/7.linkState$ 
```

**Learnings –**

1. It is difficult to configure and administer.
2. This algorithm uses the shortest path to get a common view of the entire networktopology.
3. It is not as susceptible count-to-infinity problem.
4. It also uses flooding of LSR Packets for exchange of information.