1. Shortest-Path Routing

#include<iostream>

#include<vector>

#include<set>

#include<map>

#include<algorithm>

using namespace std;

class RoutingTable{

    private:

        int \*dist;

        int numberOfNodes;

        vector<vector<int> >\*path;

    public:

        RoutingTable(){

        dist = NULL;

        numberOfNodes=0;

        }

        RoutingTable(int numberOfNodes) :numberOfNodes(numberOfNodes){

            dist = new int[numberOfNodes];

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

                dist[i] = INT32\_MAX;

            path = new vector<vector<int> >(numberOfNodes);

        }

        vector<int>& getPath(int node){

            return (\*path)[node];

        }

        int getDistance(int node){

            return dist[node];

        }

        void setDistance(int node,int distance){

            dist[node] = distance;

        }

        void print(){

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

                cout<<"Distance to "<<i<<" is "<<(dist[i] == INT32\_MAX ? "INFINITY" : to\_string(dist[i]))<<" and path is ";

                for(int j: (\*path)[i])

                    cout<<j<<" ";

                cout<<endl;

            }

        }

        void setPath(int node,vector<int>& newPath){

            (\*path)[node] = newPath;

        }

        ~RoutingTable(){

            delete dist;

            delete path;

        }

};

class Network{

    private:

        int numberOfNodes;

        map<int,vector<pair<int,int> > >adjacencyList;

        bool \*working;

        RoutingTable \*\*routingTables;

        public:

        void ShortestDistance(int source,int destination){

            vector<int>parent(numberOfNodes);

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

                parent[i] = -1;

            set<pair<int,int> >distance;

            map<int,int>dist;

            for(int node=0;node<numberOfNodes;node++)

                dist[node] = INT32\_MAX;

            dist[source] = 0;

            distance.insert({0,source});

            while(!distance.empty()){

                int currentNode = (\*distance.begin()).second;

                for(pair<int,int> adjacentNodeDetails: adjacencyList[currentNode]){

                    int adjNode = adjacentNodeDetails.first;

                    int adjNodeDist = adjacentNodeDetails.second;

                    if((dist[currentNode] + adjNodeDist < dist[adjNode])){

                        if(distance.find({dist[adjNode],adjNode}) != distance.end())

                            distance.erase(distance.find({dist[adjNode],adjNode}));

                        dist[adjNode] = dist[currentNode] + adjNodeDist;

                        distance.insert({dist[adjNode],adjNode});

                        parent[adjNode] = currentNode;

                    }

                }

                distance.erase(distance.begin());

            }

            vector<int> path;

            if(parent[destination] == -1)

                return;

            int aux = destination;

            while(destination!=source){

                path.push\_back(destination);

                destination = parent[destination];

            }

            path.push\_back(source);

            reverse(path.begin(),path.end());

            routingTables[source]->setPath(aux,path);

            routingTables[source]->setDistance(aux,dist[aux]);

        }

    public:

        Network(int numberOfNodes): numberOfNodes(numberOfNodes){

                working = new bool[numberOfNodes];

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

                    working[i] = true;

                routingTables = new RoutingTable\*[numberOfNodes];

                for(int node =0 ; node< numberOfNodes; node++)

                    routingTables[node] = new RoutingTable(numberOfNodes);

        }

        void addEdge(int first,int second,int distance){

            adjacencyList[first].push\_back({second,distance});

            adjacencyList[second].push\_back({first,distance});

        }

        void alterWorkingNode(int node){

            working[node] = !working[node];

        }

        void updateRoutingTable(){

            for(int src=0;src<numberOfNodes;src++){

                for(int dest=0;dest<numberOfNodes;dest++){

                    ShortestDistance(src,dest);

                }

            }

        }

        void displayRoutingTable(){

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

                cout<<"Routing Table for "<<i<<endl;

                routingTables[i]->print();

            }

        }

};

int main(){

    int nodes;

    int edges;

    cin>>nodes>>edges;

    Network network(nodes);

    while(edges--){

        int a,b,c;

        cin>>a>>b>>c;

        network.addEdge(a,b,c);

    }

    network.updateRoutingTable();

    network.displayRoutingTable();

}

1. FLOODING PACKETS

#include <bits/stdc++.h>

#include<algorithm>

using namespace std;

const int MAX\_NODES = 30;

int n, e, v1, v2, s, d, pkt\_cnt, pkt\_sz;

vector<int> g[MAX\_NODES];

struct PacketInfo{

    int curr, count\_val, time\_elapsed;

};

struct PacketParentInfo{

    int curr, count\_val, time\_elapsed, parent;

};

int bfs\_all\_lines(int src, int dest, int cnt){

    queue<PacketInfo> q;

    q.push({src, cnt, 0});

    while(!q.empty()){

        PacketInfo p = q.front();

        q.pop();

        if(p.curr==dest) return p.time\_elapsed;

        if(p.count\_val>1){

            for(int i: g[p.curr]){

                q.push({i, p.count\_val-1, p.time\_elapsed+1});

            }

        }

    }

    return -1;

}

int bfs\_except\_incoming(int src, int dest, int cnt){

    queue<PacketParentInfo> q;

    q.push({src, cnt, 0, -1});

    while(!q.empty()){

        PacketParentInfo p = q.front();

        q.pop();

        if(p.curr==dest) return p.time\_elapsed;

        if(p.count\_val>1){

            for(int i: g[p.curr]){

                if(i!=p.parent) q.push({i, p.count\_val-1, p.time\_elapsed+1, p.curr});

            }

        }

    }

    return -1;

}

int main(){

    cout<<"Enter the number of nodes: ";

    cin>>n;

    cout<<"Enter the number of edges: ";

    cin>>e;

    for(int i=0; i<e; i++){

        cout<<"Enter the two vertices of edge "<<i+1<<": ";

        cin>>v1>>v2;

        v1--;

        v2--;

        g[v1].push\_back(v2);

        g[v2].push\_back(v1);

    }

    cout<<"Enter source: ";

    cin>>s;

    cout<<"Enter destination: ";

    cin>>d;

    s--, d--;

    cout<<"Enter the packet counter initial value: ";

    cin>>pkt\_cnt;

    cout<<"Enter packet size: ";

    cin>>pkt\_sz;

    int time\_elapsed;

    double bw;

    cout<<"---------------Flooding all lines---------------\n";

    time\_elapsed = bfs\_all\_lines(s, d, pkt\_cnt);

    if(time\_elapsed!=-1){

        cout<<"One packet sent from source to destination\n";

        cout<<"Time elapsed: "<<time\_elapsed<<"\n";

        bw = (double)pkt\_sz / (double)time\_elapsed;

        cout<<"Bandwidth: "<<bw<<"\n";

    }

    else cout<<"The packet would never reach the destination\n";

    cout<<"--------------Flooding all line except incoming--------------\n";

    time\_elapsed = bfs\_all\_lines(s, d, pkt\_cnt);

    if(time\_elapsed!=-1){

        cout<<"One packet sent from source to destination\n";

        cout<<"Time elapsed: "<<time\_elapsed<<"\n";

        bw = (double)pkt\_sz / (double)time\_elapsed;

        cout<<"Bandwidth: "<<bw<<"\n";

    }

    else cout<<"The packet would never reach the destination\n";

    return 0;

}