AODV

Code:

#include<bits/stdc++.h>

using namespace std;

class Node{

    int id;

    unordered\_map<int,pair<int,int>> table;

    public:

    Node(int \_id){

        id=\_id;

    }

    int get\_id(){

        return id;

    }

    void setNexthop(int dest,int \_id){

        table[dest].first=\_id;

    }

    void setPrevhop(int src,int \_id){

        table[src].second=\_id;

    }

    bool hasEntry(int dest){

        if(table.find(dest)==table.end())

        return false;

        return true;

    }

    int next\_hop(int dest){

        return table[dest].first;

    }

    int prev\_hop(int dest){

        return table[dest].second;

    }

    void printTable(){

        cout<<"Dest    | Previous Hop | Next Hop\n";

        for(auto x:table){

            cout<<x.first<<"        "<<x.second.second<<"               "<<x.second.first<<"\n";

        }

    }

};

class AODV{

    unordered\_map<int,vector<int>> g;

    unordered\_map<int,Node\*> id\_node;

    public:

    void create\_node(int \_id){

        Node\* current=new Node(\_id);

        g[\_id]={};

    }

    void add\_edge(int src\_id,int dest\_id){

        if(g.find(src\_id)==g.end()){

            Node\* new\_node=new Node(src\_id);

            id\_node[src\_id]=new\_node;

            g[src\_id]={};

        }

        if(g.find(dest\_id)==g.end()){

            Node\* new\_node=new Node(dest\_id);

            id\_node[dest\_id]=new\_node;

            g[dest\_id]={};

        }

        (g[src\_id]).push\_back(dest\_id);

        (g[dest\_id]).push\_back(src\_id);

    }

    bool BFS(int src, int dest,unordered\_map<int,int> &pred){

        queue<int> q;

        unordered\_map<int,bool> visited;

        visited[src] = true;

        q.push(src);

        while (!q.empty()) {

            int u = q.front();

            q.pop();

            for (int i = 0; i < g[u].size(); i++) {

                if (visited.find(g[u][i]) == visited.end()) {

                    visited[g[u][i]] = true;

                    pred[g[u][i]] = u;

                    q.push(g[u][i]);

                    if (g[u][i] == dest)

                        return true;

                }

            }

        }

        return false;

}

    void RREQ(int src,int dest){

          unordered\_map<int,int> pred;

          //RREQ

          BFS(src,dest,pred);

          //updating table

          Node\* towords\_dest=id\_node[dest];

          Node\* towords\_src=id\_node[dest];

          int temp=dest;

          int next=-1;

          /\* RREP\*/

          bool found=false;

          while(temp!=src){

            towords\_dest->setNexthop(dest,next);

            towords\_dest->setPrevhop(dest,pred[temp]);

            towords\_src->setNexthop(src,pred[temp]);

            towords\_src->setPrevhop(src,next);

            towords\_dest=towords\_src=id\_node[pred[temp]];

            next=temp;

            cout<<temp<<" ";

            temp=pred[temp];

          }

           cout<<temp<<endl;

            towords\_dest->setNexthop(dest,next);

            towords\_dest->setPrevhop(dest,-1);

            towords\_src->setNexthop(src,-1);

            towords\_src->setPrevhop(src,next);

    }

    void get\_path(int src\_id,int dest\_id){

         if(g.find(src\_id)==g.end()||g.find(dest\_id)==g.end()){

            cout<<"node is not part of network\n";

            return;

         }

         Node \*src=id\_node[src\_id];

         if(!(src->hasEntry(dest\_id))){

            RREQ(src\_id,dest\_id);

         }

         while(src->next\_hop(dest\_id)!=dest\_id){

               cout<<src->get\_id()<<"->";

               src=id\_node[src->next\_hop(dest\_id)];

         }

         cout<<dest\_id<<"\n";

    }

    void get\_route\_table(int id){

         if(id\_node.find(id)==id\_node.end()){

            cout<<"Not part of network";

            return;

         }

        Node\* node\_=id\_node[id];

        node\_->printTable();

    }

};

int main(){

    char ch='8';

    AODV network;

    while(ch!='4'){

        cout<<"1.Add adges\n2.Find path\n3.See route route table\n4.Exit\n";

        cin>>ch;

        if(ch=='1'){

            cout<<"Enter Number of edge to add:";

            int n;

            cin>>n;

            cout<<"Enter source id and dest id:\n";

            while(n--){

            int src,dest;

            cin>>src>>dest;

            network.add\_edge(src,dest);

            }

        }

        if(ch=='2'){

            cout<<"Enter source id and dest id:";

            int src,dest;

            cin>>src>>dest;

            network.get\_path(src,dest);

        }if(ch=='3'){

            cout<<"Enter router id:";

            int id;

            cin>>id;

            network.get\_route\_table(id);

        }

    }

}

OUTPUT:

