Exp3

**INPUT::**

Write C++ program to implement the graph using adjacency list/adjacency matrix.

/\*Program for sdjacency matrix with BFS\*/

#include<iostream>

//#include<iomanip>

#define MAX 30

using namespace std;

//const int MAX=30;

class node

{

            node \*next;

            string city;

            int timeCost;

public:

            friend class graph;

            node()

            {

                        next=NULL;

                        city="";

                        timeCost=-1;

            }

            node(string city,int weight)

            {

                        next=NULL;

                        this->city=city;

                        timeCost=weight;

            }

};

class graph

{

            node \*head[MAX];

            int n;

public:

    static int cnt;

    string vis[MAX];

    friend class que;

            graph(int num)

{

                        n=num;

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

                                    head[i]=NULL;

}

            void insert(string city1,string city2,int time);

            void readdata();

            int getindex(string s1);

            void bfs(string src);

            int visited(/\* cnt,\*/string city);

            void display();

};

int graph::cnt=0;

class que

{

    int f,r;            //data members

   //string q[MAX];

    public:

        string q[MAX];

    que() //Default constructor

    {

     f=r=-1; //init front and rear to -1.

    }

    //To check Q is full or not

    int isQfull()

    {

        if(r==MAX-1)

            return 1;

        else

            return 0;

    }

    //To check Q is empty or not

    int isQempty()

    {

        if((f==-1)&& (r==-1))

            return 1;

        else

            return 0;

    }

    void add(string); //member functions

    int search(string);

    string del();

};

//To insert the job inside the simple queue.

void que::add(string city)

{

    if(isQfull())

        cout<<"\nSorry !! Queue is full....\n";

    else

    {

        if(f==-1)

        {

            f=r=0;

            q[r]=city;

        }

        else

        {

            r=r+1;

            q[r]=city;

        }

    }

}

//To delete job from the simple queue.

string que::del()

{

    string city;

    if(isQempty())

        cout<<"\nSorry Q is empty...\n";

    else

    {

        if(f==r)

        {

        city=q[f];

        f=r=-1;

        }

        else

        {

        city=q[f];

        f=f+1;

        }

       return city;

    }

}

int que::search(string city )

{

    if(r==-1 && f==-1)

    {

        return 0;

    }

    else

    {

        for(int i=f;i<=r;i++)

        {

            if(q[i]==city )

            return 1;

        }

        return 0;

   }

}

    int graph:: visited(/\* cnt,\*/string city)

    {

        for(int i=0;i<cnt;i++)

        {

            if(vis[i]==city)

            return 1;

        }

        return 0;

    }

void graph::bfs(string src)

{

    que qobj;

    node \*start;

    string str;

    qobj.add(src);

   while(qobj.isQempty()==0)

   {

    str=qobj.del();

    cout<<"\t"<<str;

     vis[cnt]=str;

    cnt++;

    int ind=getindex(str);

    start=head[ind];

    while(start->next!=NULL)

    {

        start=start->next;

        if((!qobj.search(start->city))&&(!visited(start->city)))

        qobj.add(start->city);

    }

    }

}

int graph::getindex(string s1)

{

            for(int i=0;i<n;i++)

            {

                        if(head[i]->city==s1)

                                    return i;

            }

            return -1;

}

void graph::insert(string city1,string city2,int time)

{

            node \*source;

            node \*dest=new node(city2,time);

            int ind=getindex(city1); //for getting head nodes index in array

            source=head[ind];

            while(source->next!=NULL)

                        source=source->next;

            source->next=dest;

}

void graph::display()

{

            node \*source;

            for(int i=0;i<n;i++)

            {

                        source=head[i];

                        while(source!=NULL)

                        {

                                    cout<<source->city<<"("<<source->timeCost<<")""->";

                                    source=source->next;

                        }

                        cout<<"NULL"<<endl;

            }

}

void graph::readdata()

{

            string city1,city2,tmpcity;

            int fcost

            ;

            int flight;

            cout<<"\nEnter City Details:\n";//number of vertices

            for(int i=0;i<n;i++)

            {

                        head[i]=new node;

                        cout<<"Enter name of city "<<i+1<<" ";

                        cin>>head[i]->city;

            }

            cout<<"\nEnter Number of Flights to insert: ";//number of edges

            cin>>flight;

            for(int i=0;i<flight;i++)

            {

                        cout<<"\nEnter Source:";

                        cin>>city1;

                        cout<<"Enter Destination:";

                        cin>>city2;

                        cout<<"Enter Time:";

                        cin>>fcost;

                        insert(city1,city2,fcost);

            }

}

int main() {

            int number,choice;

            string city1;

            cout<<"\nEnter Number of Airport Stations:";

            cin>>number;

            graph g1(number);

            que q;

            do

            {

                        cout<<"\n------Menu------"

                                                <<"\n1.Insert Flight detail"

                                                <<"\n2.Display"

                                                <<"\n3.BFS"

                                                <<"\n4.Exit"

                                                <<"\nEnter your choice: ";

                        cin>>choice;

                        switch(choice)

                        {

                        case 1:

                                    g1.readdata();

                                    break;

                        case 2:

                                    cout <<"\*\*Adjacency List\*\*"<< endl;

                                    g1.display();

                                    break;

                        case 3:

                                    cout <<"Enter Staring City"<< endl;

                                    cin>>city1;

                                    cout<<"\n BFS Traversal of given graph is:\n";

                                    g1.bfs(city1);

                                    break;

                        default:

                                    cout<<"\nWrong Choice";

                        }

            }while(choice!=4);

            return 0;

}

**Output::**

------Menu------  
1.Insert Flight detail  
2.Display  
3.BFS  
4.Exit  
Enter your choice: 1  
  
Enter City Details:  
Enter name of city 1 a  
Enter name of city 2 b  
Enter name of city 3 c  
Enter name of city 4 d  
Enter name of city 5 e  
Enter name of city 6 f  
  
Enter Number of Flights to insert: 8  
  
Enter Source:a  
Enter Destination:b  
Enter Time:2  
  
Enter Source:b  
Enter Destination:c  
Enter Time:3  
  
Enter Source:b  
Enter Destination:e  
Enter Time:4  
  
Enter Source:b  
Enter Destination:d  
Enter Time:5  
  
Enter Source:d  
Enter Destination:f  
Enter Time:6  
  
Enter Source:f  
Enter Destination:e  
Enter Time:7  
  
Enter Source:e  
Enter Destination:c  
Enter Time:8  
  
Enter Source:c  
Enter Destination:e  
Enter Time:9  
  
------Menu------  
1.Insert Flight detail  
2.Display  
3.BFS  
4.Exit  
Enter your choice: 3  
Enter Staring City  
a  
  
 BFS Traversal of given graph is:  
a b c e d f  
------Menu------