**ASSIGNMENT NO.3.**

**Aim :-**

There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight takes to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Justify the storage representations used .

**Program Code:-**

#include <iostream>

#include<iomanip>

using namespace std;

const int MAX=30;

class Queue //Queue for BFS TRAVERSAL

{

int front,rear;

string data[MAX];

public:

Queue()

{

front=-1;

rear=-1;

}

bool empty()

{

if(rear==-1)

return 1;

else

return 0;

}

bool inqueue(string str)

{

if(front==-1 && rear==-1)

{

front=rear=0;

data[rear]=str;

return true;

}

else

{

rear=rear+1;

data[rear]=str;

return true;

}

}

string dequeue()

{

string p;

if(front==rear)

{

p=data[front];

front=-1;

rear=-1;

}

else

{

p=data[front];

front=front+1;

}

return p;

}

};

class node //node class for each airport

{

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 //Contains total graph of airports

{

node \*head[MAX];

int n;

int visited[MAX];

public:

graph(int num)

{

n=num;

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

head[i]=NULL;

}

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

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

void readdata(int gType);

int getindex(string s1);

void outFlights();

void inFlights();

void DFS(string str);

void BFS();

void dfsTraversal();

};

void graph::BFS()

{

string str=head[0]->city;

int j;

//node \*p;

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

visited[i]=0;

Queue queue;

queue.inqueue(str);

int i=getindex(str);

cout<<"BFS Traversal: \n";

cout<<" "<<str<<" ";

visited[i]=1;

while(!queue.empty())

{

string p=queue.dequeue();

i=getindex(p);

//visited[i]=1;

for(node \*q=head[i];q!=NULL;q=q->next)

{

i=getindex(q->city);

str=q->city;

if(!visited[i])

{

queue.inqueue(q->city);

visited[i]=1;

cout<<" "<<str<<" ";

}

}

}

cout<<"\n";

}

void graph::dfsTraversal()

{

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

visited[i]=0;

cout<<"\nDFS TRAVERSAL: \n";

DFS(head[0]->city);

cout<<"\n";

}

void graph::DFS(string str)

{

node \*p;

int i=getindex(str);

cout<<" "<<str<<" ";

p=head[i];

visited[i]=1;

while(p!=NULL)

{

str=p->city;

i=getindex(str);

if(!visited[i])

DFS(str);

p=p->next;

}

}

void graph::inFlights()

{

int count[n];

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

count[i]=0;

cout<<"====== In degree =========\n";

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

{

cout<<"\n"<<setw(8)<<"Source"<<setw(8)<<"Destin."<<setw(8)<<"Time";

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

{

node \*p=head[j]->next;

while(p!=NULL)

{

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

{

count[i]=count[i]+1;

cout<<"\n"<<setw(8)<<head[j]->city<<setw(8)<<head[i]->city<<setw(8)<<p->timeCost;

}

p=p->next;

}

}

cout<<"\nFlights to "<<head[i]->city<<" = "<<count[i]<<endl;

cout<<"-------------------------------------\n";

}

}

void graph::outFlights()

{

int count;

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

{

node \*p=head[i]->next;

count=0;

cout<<"\n"<<setw(8)<<"Source"<<setw(8)<<"Destin."<<setw(8)<<"Time";

if(p==NULL)

{

cout<<"\nNo Flights from "<<head[i]->city;

}

else

{

while(p!=NULL)

{

cout<<"\n"<<setw(8)<<head[i]->city<<setw(8)<<p->city<<setw(8)<<p->timeCost;

count++;

p=p->next;

}

}

cout<<"\nNo. of flights: "<<count<<endl;;

cout<<"-------------------------------------\n";

}

}

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

if(head[ind]==NULL)

head[ind]=dest;

else

{

source=head[ind];

while(source->next!=NULL)

source=source->next;

source->next=dest;

}

}

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

{

node \*source;

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

node \*dest2=new node(city1,time); //for second flight insertion

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

int ind2=getindex(city2);

/\* if(head[ind]==NULL && head[ind2]==NULL) //when no flights in graph

{

head[ind]=dest;

head[ind2]=dest2;

}

else if(head[ind]==NULL && head[ind2]!=NULL) //no flight in first list but flight in second list

{

head[ind]=dest; //inserted first flight

source=head[ind2];

while(source->next!=NULL)

source=source->next;

source->next=dest2;

}

else if(head[ind]!=NULL && head[ind2]==NULL)

{

head[ind2]=dest2; //inserted first flight

source=head[ind];

while(source->next!=NULL)

source=source->next;

source->next=dest;

}

else

{\*/

source=head[ind];

while(source->next!=NULL)

source=source->next;

source->next=dest;

source=head[ind2];

while(source->next!=NULL)

source=source->next;

source->next=dest2;

//}

}

void graph::readdata(int gType)

{

string city1,city2,tmpcity;

int fcost;

int flight;

cout<<"\nENter City Details:\n ";

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

{

head[i]=new node;

cout<<"Enter City "<<i+1<<" ";

cin>>tmpcity;

head[i]->city=tmpcity;

}

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

cin>>flight;

if(gType==1)

{

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);

}

}

else

{

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

{

cout<<"\nEnter Source:";

cin>>city1;

cout<<"Enter Destination:";

cin>>city2;

cout<<"Enter Time:";

cin>>fcost;

insertUndirected(city1,city2,fcost);

//cout<<"\ninserted"<<i+1;

}

}

}

int main() {

int number,choice;

int graphype;

cout<<"-------INDIA AIRLINES PVT LTD--------"

<<"\n0. Undirected\n1.Directed\nEnter Flight data Insertion Type:";

cin>>graphype;

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

cin>>number;

graph g1(number);

g1.readdata(graphype);

do

{

cout<<"-------- Menu --------"

<<"\n1.Incoming Flights(In degree)"

<<"\n2.Outgoing Flights(Out degree)"

<<"\n3.DFS"

<<"\n4.BFS"

<<"\n5.Exit"

<<"\nEnter your choice: ";

cin>>choice;

switch(choice)

{

case 1:

cout <<"" << endl;

g1.inFlights();

break;

case 2:

g1.outFlights();

break;

case 3:

g1.dfsTraversal();

break;

case 4:

g1.BFS();

break;

default:

cout<<"\nWrong Choice";

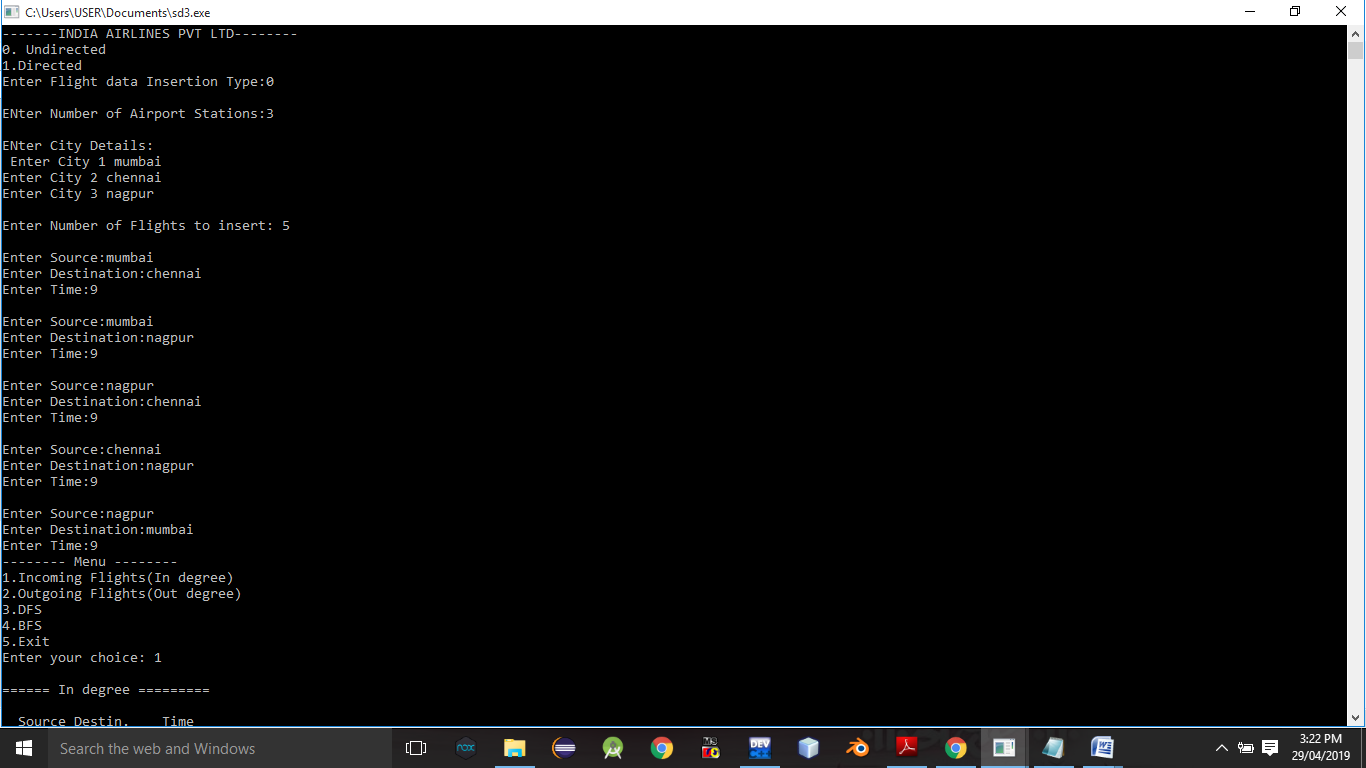
}

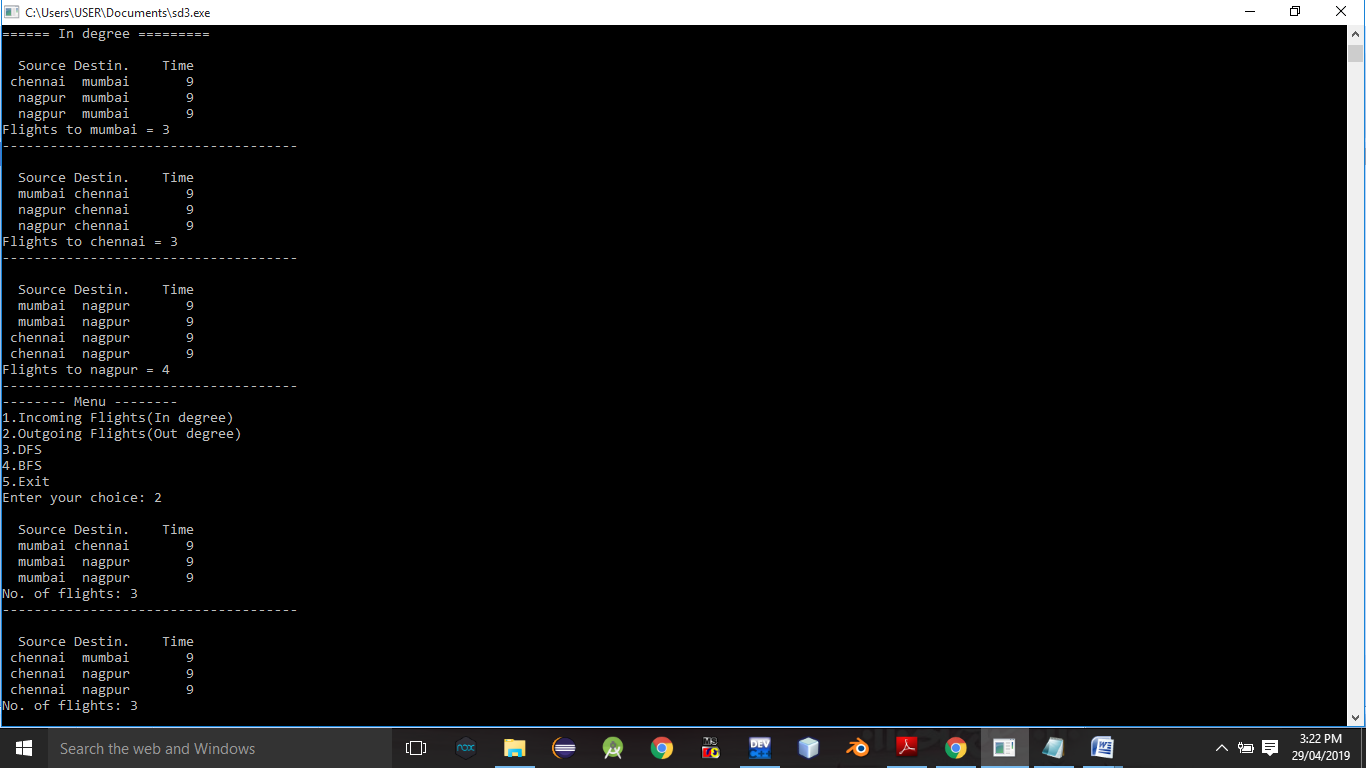
}while(choice!=5);

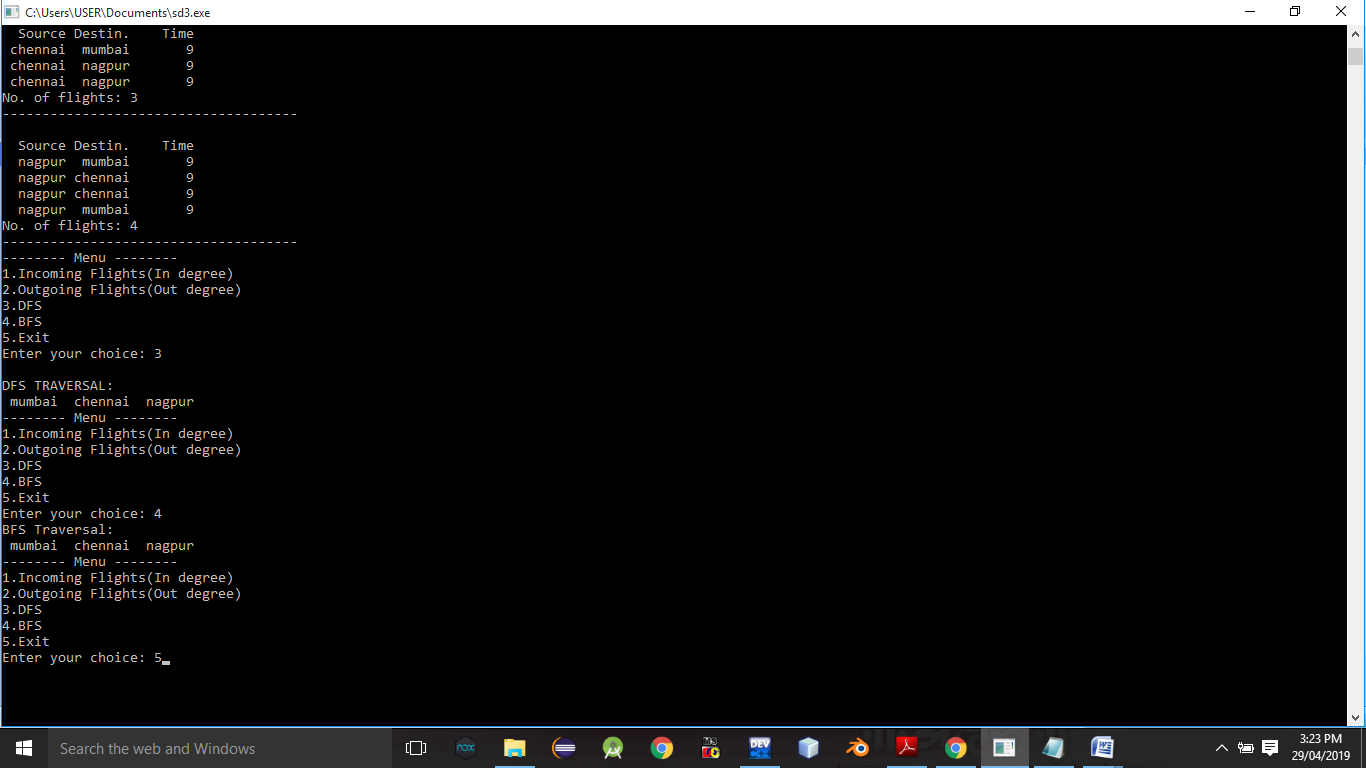
return 0;

}

**Output Screenshots:-**







**Conclusion:-** Thus,we have studied adjacency graph representation.