AM.SC.P2CSC19034

SRUTHY P R

**COMPITATIVE PROGRAMING**

DFS:

#include <iostream>

#include <ctime>

#include <malloc.h>

using namespace std;

struct nod

{

int info;

struct nod \*next;

};

class stak

{

struct nod \*top;

public:

stak();

void push(int);

int pop();

bool isEmpty();

void display();

};

stak::stak()

{

top = NULL;

}

void stak::push(int data)

{

nod \*p;

if((p=(nod\*)malloc(sizeof(nod)))==NULL){

cout<<"Memory Exhausted";

exit(0);

}

p = new nod;

p->info = data;

p->next = NULL;

if(top!=NULL)

{

p->next = top;

}

top = p;

}

int stak::pop()

{

struct nod \*temp;

int value;

if(top==NULL){

cout<<"\nThe stak is Empty"<<endl;

}

else

{

temp = top;

top = top->next;

value = temp->info;

delete temp;

}

return value;

}

bool stak::isEmpty()

{

return (top == NULL);

}

void stak::display()

{

struct nod \*p = top;

if(top==NULL){

cout<<"\nNothing to Display\n";

}

else

{

cout<<"\n The contents of stak\n";

while(p!=NULL){

cout<<p->info<<endl;

p = p->next;

}

}

}

class Graph

{

private:

int n;

int \*\*A;

public:

Graph(int siz = 2);

~Graph();

bool isConnected(int, int);

void addEdge(int x, int y);

void DFS(int , int);

};

Graph::Graph(int siz)

{

int i, j;

if (siz < 2) n = 2;

else n = siz;

A = new int\*[n];

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

A[i] = new int[n];

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

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

A[i][j] = 0;

}

Graph::~Graph()

{

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

delete [] A[i];

delete [] A;

}

bool Graph::isConnected(int x, int y)

{

return (A[x-1][y-1] == 1);

}

void Graph::addEdge(int x, int y)

{

A[x-1][y-1] = A[y-1][x-1] = 1;

}

void Graph::DFS(int x, int required)

{

stak s;

bool \*visited = new bool[n+1];

int i;

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

visited[i] = false;

s.push(x);

visited[x] = true;

if(x == required) return;

cout << " Depth first Search starting from vertex ";

cout << x << " : " << endl;

while(!s.isEmpty())

{

int k = s.pop();

if(k == required) break;

cout<<k<<" ";

for (i = n; i >= 0 ; --i)

if (isConnected(k, i) && !visited[i])

{

s.push(i);

visited[i] = true;

}

}

cout<<endl;

delete [] visited;

}

int main()

{

Graph g(8);

g.addEdge(1, 2);

g.addEdge(1, 3);

g.addEdge(1, 4);

g.addEdge(2, 5);

g.addEdge(2, 6);

g.addEdge(4, 7);

g.addEdge(4, 8);

g.DFS(1, 4);

return 0;

}

Output:

