#include <iostream>

#define MAX 20 //pre-processor directives

using namespace std;

class Graph

{

//friend class stack;

int n;

int adj[MAX][MAX];

int visited[MAX];

public:

Graph();

void create();

void dfs();

void bfs();

void initialize\_visited();

};

class stack

{

int item[MAX];

int top;

public:

stack()

{

top=-1;

}

int pop();

int isEmpty();

int isFull();

void push(int);

};//Stack declaration

class queue

{

int item[MAX];

int front,rear;

public:

queue()

{

front = rear = -1;

}

int pop();

int isEmpty();

int isFull();

void push(int);

};//Queue declaration

/---- Stack definitions starts here ----/

int stack::pop()

{

int x;

if(isEmpty())

{

cout<<"\n Stack is Empty!";

}

else

{

x = item[top];

top--;

}

return x;

}//pop

int stack::isEmpty()

{

if(top==-1)

{

return 1;

}

else

{

return 0;

}

}//isEmpty

int stack::isFull()

{

if(top==MAX)

return 1;

else

return 0;

}

void stack::push(int x)

{

if(isFull())

cout<<"\nStack is full";

else

{

top++;

item[top] = x;

}

}

/---- Stack definitions ends here ----/

/---- Queue definitions starts here ----/

int queue::dequeue()

{

int x;

if(isEmpty())

{

cout<<"\n queue is empty";

}

else

{

x = item[front];

front++;

}

return x;

}

int queue::isEmpty()

{

if((front==-1)||(front>rear))

return 1;

else

return 0;

}

int queue::isFull()

{

if(front==MAX)

return 1;

else

return 0;

}

void queue::enqueue(int x)

{

if(isFull())

cout<<"\nQueue is full";

else

{

if(front==-1)

front++;

item[++rear] = x;

}

}

/---- Queue definitions ends here ----/

Graph::Graph()

{

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

{

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

{

adj[i][j]=0;

}

}

for (int i = 0; i < MAX; i

{

visited[i]=0;

}

}

void Graph :: create()

{

int userChoice;

cout<<"No. of Vertices in a graph :: ";

cin>>noOfNodes;

for (int i = 0; i < noOfNodes-1; i++)//Vertices/Nodes

{

for(int j = i+1; j < noOfNodes; j++)//Adjacency nodes

{

cout<<"Does edge is present between "<< i <<" and " <<j<<"?(1/0)- ";

cin >> userChoice;

if(userChoice==1) adj[i][j] = adj[j][i] = 1;

}//inner for -j

}//outer for - i

}//create

void Graph :: dfs()

{

stack s;

int i;

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

cout<<"Enter starting vertex :: ";

cin >> i;

s.push(i);//push starting vertex into stack

while(!s.isEmpty()) //Till stack is not empty

{

i = s.pop(); //Pop the vertex

if(visited[i]==1)

{

continue;

}

//ignore if the vertex is visited

cout<<" "<<i; //print the vertex

visited[i] = 1; //set it as visited

for (int k = 0; k < n; k++) //for accessing adjacent elements of i

{

if(adj[i][k] == 1 && visited[k] == 0) //if edge is present and new vertex is not visited

s.push(k);

}//for

}//while

}//dfs

void Graph :: initialize\_visited()

{

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

{

visited[i]=0;

}

}

void Graph :: bfs()

{

queue q;

int i;

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

cout<<"Enter starting vertex :: ";

cin >> i;

q.enqueue(i); //push starting vertex into queue

while(!q.isEmpty()) //Till queue is not empty

{

i = q.dequeue(); //Pop the vertex from queue

if(visited[i]==1) //ignore if the vertex is visited

continue;

cout<<" "<<i; //print the vertex

visited[i] = 1; //set it as visited

for (int k = 0; k < n; k++) //for accessing adjacent elements of i

{

if(adj[i][k] == 1 && visited[k] == 0) //if edge is present and new vertex is not visited

q.enqueue(k);

}

}//while

}//bfs

int main()

{

Graph g;

g.create();

g.initialize\_visited();

g.dfs();

g.initialize\_visited();

g.bfs();

return 0;

}

Step 1: Start.

Step 2: Input the value of N nodes of the graph

Step 3: Create a graph of N nodes using adjacency matrix representation.

Step 4: Print the nodes reachable from the starting node using BFS.

Step 5: Check whether graph is connected or not using DFS.

Step 6: Stop.

The DFS algorithm works as follows:

1.Start by putting any one of the graph's vertices on top of a stack.

2.Take the top item of the stack and add it to the visited list.

3.Create a list of that vertex's adjacent nodes. Add the ones which aren't in the

visited list to the top of the stack.

4.Keep repeating steps 2 and 3 until the stack is empty.

The BFS algorithm works as follows:

1.Start by putting any one of the graph's vertices at the back of a queue.

2.Take the front item of the queue and add it to the visited list.

3.Create a list of that vertex's adjacent nodes. Add the ones which aren't in the

visited list to the back of the queue.

4.Keep repeating steps 2 and 3 until the queue is empty.