// ConsoleApplication11.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

#include <stack>

using namespace std;

// Abstraction of a graph

class Graph

{

private:

int n; // Number of vertices

int\*\* adj; // Adjacency matrix

public:

Graph(int size); // Constructor

~Graph(); // Destructor

void addEdge(int x, int y); // Add an edge the the graph

bool isConnected(int x, int y); // Check if two vertices are connected

void DFS(int start, int end); // Depth first search

};

// Constructor

Graph::Graph(int size)

{

if (size < 2) // The graph must have at least 2 vertices

{

n = 2;

}

else

{

n = size;

}

adj = new int\*[n]; // Allocate memory for adjacency matrix

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

{

adj[i] = new int[n]; // Allocate memory for adjacency matrix

}

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

{

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

{

adj[i][j] = 0; // Initialize the vertices to 0

}

}

}

// Destructor

Graph::~Graph()

{

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

{

delete[] adj[i];

}

delete[] adj;

}

// Add an edge to the graph

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

{

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

}

// Check if two vertices are connected

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

{

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

}

// Depth first search

void Graph::DFS(int start, int end)

{

stack <int> s; // Create a stack of ints

bool\* visited = new bool[n + 1]; // Boolean array to represent which vertices we've visited

for (int i = 0; i <= n; ++i) // Initialize all elements of visited to false

{

visited[i] = false;

}

s.push(start); // Push the vertex onto the stack

visited[start] = true; // Mark this node as visited

if (start == end) // End the function if we're at our destination

{

return;

}

cout << "Depth-first search starting from point: " << start << endl;

while (!s.empty()) // While the stack is not empty

{

int k = s.top(); // Save the top element before popping it off

s.pop();

if (k == end) // Break out of the loop if we've reached our destination

{

break;

}

cout << k << " ";

for (int i = n; i >= 0; --i) // For all adjacent vertices

{

if (isConnected(k, i) && !visited[i]) // If they're connected and we haven't visited it yet

{

s.push(i); // Add it to our stack, and mark that we've visited it

visited[i] = true;

}

}

}

cout << end << endl;

delete[] visited; // De-allocate the visited array

}

// Main function

int main()

{

Graph g(8);

g.addEdge(1, 2);

g.addEdge(1, 3);

g.addEdge(1, 4);

g.addEdge(2, 3);

g.addEdge(3, 4);

g.addEdge(2, 5);

g.addEdge(2, 6);

g.addEdge(3, 6);

g.addEdge(5, 8);

g.addEdge(5, 6);

g.addEdge(6, 8);

g.addEdge(6, 7);

g.addEdge(7, 8);

g.addEdge(4, 6);

g.addEdge(4, 7);

g.DFS(1, 7);

return 0;

}