Joseph Garcia

4/28/2020

CS 413

Project 2 Report

#include <bits/stdc++.h>

using namespace std;

// Class for an undirected graph

class Und\_Graph

{

int Nodes; // Number of nodes in a graph

list<int>\* graph;

bool hasCycle(int curr\_node, bool visited[], int parent);

public:

Und\_Graph(int nodes);

void addEdge(int a, int b);

bool Cyclic();

};

// List for holding nodes of a graph

Und\_Graph::Und\_Graph(int nodes)

{

this->Nodes = nodes;

graph = new list<int>[nodes];

}

// Function for adding edges to a graph

void Und\_Graph::addEdge(int a, int b)

{

graph[a].push\_back(b);

graph[b].push\_back(a);

}

// Function for detecting cycles in a graph

bool Und\_Graph::hasCycle(int curr\_Node, bool visited[], int parent)

{

visited[curr\_Node] = true; // the node has been visited

list<int>::iterator iter;

for (iter = graph[curr\_Node].begin(); iter != graph[curr\_Node].end(); iter++)

{

// The adjacent node has not been visited

if (!visited[\*iter])

{

if (hasCycle(\*iter, visited, curr\_Node))

return true;

}

// An adjacent node has been visited and is not a parent

else if (\*iter != parent)

return true;

}

return false;

}

// If the graph contains a cycle, return true, otherwise return false.

bool Und\_Graph::Cyclic()

{

bool\* visited = new bool[Nodes];

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

visited[i] = false;

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

if (!visited[j])

if (hasCycle(j, visited, -1))

return true;

return false;

}

// Main method

int main()

{

cout << "Does each graph contain a cycle?\n";

Und\_Graph g1(5);

g1.addEdge(1, 2);

g1.addEdge(2, 3);

g1.addEdge(3, 4);

g1.addEdge(4, 2);

g1.addEdge(2, 3);

g1.Cyclic() ? cout << "Graph g1: true\n" :

cout << "Graph g1: false\n";

Und\_Graph g2(7);

g2.addEdge(1, 4);

g2.addEdge(6, 4);

g2.addEdge(5, 4);

g2.addEdge(3, 4);

g2.addEdge(2, 3);

g2.Cyclic() ? cout << "Graph g2: true\n" :

cout << "Graph g2: false\n";

return 0;

}