***Министерство образования Республики Беларусь***

***Учреждение образования***

***«Брестский государственный технический университет»***

***Кафедра ИИТ***

**Лабораторная работа №5**

**По дисциплине МОИС за III семестр**

**Тема: «Определение компонент сильной связности в орграфе.**

**Определение компонент двусвязности для неориентированного графа»**

**Выполнил:**

Студент группы ИИ-15 (1)

2-го курса

Волк И. А.

**Проверил:**

Шуть В. Н.

Брест 2018

Цель: научиться определять компоненты сильной связности в орграфе, определять компоненты двусвязности в неориентированном графе.

***Задание.***

1. Написать программу нахождения компонент сильной связности орграфа **. Варианты заданий указаны в таблице 1. Орграф задан списком дуг. Изобразить граф, пометив его компоненты сильной связности.
2. Написать программу нахождения компонент двусвязности для неориентированного графа**. Варианты заданий указаны в таблице 1. Граф задан списком ребер. Изобразить граф, пометив точки сочленения и мосты, и его компоненты двусвзности

|  |  |  |
| --- | --- | --- |
| № | Орграф | Неориентированный граф |
| 1. | (a,b),(a,d),(b,e),(c,a),  (c,b),(c,d),(e,c) | (a,b),(a,c),(b,d),(c,d),  (d,e),(d,f),(e,f) |

Код программы:

//main.cpp

#include "pch.h"

#include <iostream>

#include "Graphs.h"

#include "scc.h"

#include "lab5\_2.h"

#define a 1

#define b 2

#define c 3

#define d 4

#define e 5

#define f 6

//(a, b), (a, d), (b, e), (c, a),

//(c, b), (c, d), (e, c)

int main()

{

Graph gr(GraphType::DIRECTED);

gr.addVer(5);

gr.addE(a, b);

gr.addE(a, d);

gr.addE(b, e);

gr.addE(c, a);

gr.addE(c, b);

gr.addE(c, d);

gr.addE(e, c);

SCC scc(gr);

auto sccs = scc.getSCCs();

for (int i = 0; i < sccs.size(); i++)

{

for (int j = 0; j < sccs[i].size(); j++)

std::cout << (char)(sccs[i][j] + 96) << " ";

std::cout << std::endl;

}

Graph gr2;

gr2.addVer(6);

gr2.addE(a, b);

gr2.addE(a, c);

gr2.addE(b, d);

gr2.addE(c, d);

gr2.addE(d, e);

gr2.addE(d, f);

gr2.addE(e, f);

std::vector<Graph> biconnComp = Lab5::findBiconnComp(gr2);

for (Graph& gr : biconnComp)

{

show(gr, ALPHA);

std::cout << std::endl;

}

std::cin.get();

}

//scc.h

#pragma once

#include "graph.h"

#include "Path.h"

class SCC

{

List<List<GraphVertex>> SCCs;

void algo(GraphVertex, Graph&, bool[]);

public:

SCC();

SCC(Graph&);

void find(Graph);

List<List<GraphVertex>>& getSCCs();

};

//scc.cpp

#include "stdafx.h"

#include <stack>

#include "scc.h"

#include "grtrav.h"

SCC::SCC()

{}

SCC::SCC(Graph& graph)

{

find(graph);

}

void SCC::find(Graph graph)

{

if (graph.type() == UNDIRECTED)

{

std::cout << "Graph is undirected!" << std::endl;

return;

}

DepthFirst df(graph);

std::stack<GraphVertex> order;

df.getOrder(order);

transpose(graph);

bool \* visited = new bool[graph.getNumOfVer()];

for (unsigned int i = 0; i < graph.getNumOfVer(); i++)

visited[i] = false;

for (unsigned int i = 1; i <= graph.getNumOfVer(); i++)

if (!visited[i - 1])

{

List<GraphVertex> connComp;

SCCs.push\_back(connComp);

algo(i, graph, visited);

}

}

void SCC::algo(GraphVertex vertex, Graph& graph, bool visited[])

{

visited[vertex - 1] = true;

SCCs.getAccess(SCCs.size() - 1).push\_back(vertex);

for (unsigned int i = 0; i < graph.getNumOfE(); i++)

{

if (graph[i].fVertex == vertex)

{

if (!visited[graph[i].secVertex - 1])

algo(graph[i].secVertex, graph, visited);

}

}

}

List<List<GraphVertex>>& SCC::getSCCs()

{

return SCCs;

}

//lab5\_2.h

#pragma once

#include "graph.h"

#include <vector>

namespace Lab5

{

std::vector<bool> findCutPoint(const Graph& gr);

std::vector<Graph> findBiconnComp(const Graph& graph);

}

//lab5\_3.h

#include "stdafx.h"

#include <exception>

#include "lab5\_2.h"

namespace Lab5

{

template<typename T>

T min(std::initializer\_list<T> list)

{

if (list.size() == 0)

return T();

T min = \*list.begin();

for (T el : list)

if (el < min)

min = el;

return min;

}

static class DFS

{

public:

const Graph& graph;

std::vector<bool>& cutPoints;

std::vector<int> tin;

std::vector<int> up;

std::vector<bool> visited;

DFS(const Graph& graph, std::vector<bool>& cutPoints)

: graph(graph), cutPoints(cutPoints),

tin(graph.getNumOfVer()),

up(graph.getNumOfVer()),

visited(graph.getNumOfVer())

{

for (int i = 0; i < this->cutPoints.size(); i++)

this->cutPoints[i] = false;

for (int i = 0; i < visited.size(); i++)

visited[i] = false;

}

void operator()(int vertex, int parent, int time = 0)

{

up[vertex - 1] = tin[vertex - 1] = time;

visited[vertex - 1] = true;

int count = 0;

for (int u = 1; u <= graph.getNumOfVer(); u++)

{

if (graph.edgeId(vertex, u) == -1)

continue;

if (u == parent)

continue;

if (visited[u - 1])

up[vertex - 1] = min({ up[vertex - 1], tin[u - 1] });

else

{

this->operator()(u, vertex, time + 1);

count++;

up[vertex - 1] = min({ up[vertex - 1], up[u - 1] });

if (parent != -1 && up[u - 1] >= tin[vertex - 1])

cutPoints[vertex - 1] = true;

}

}

if (parent == -1 && count >= 2)

cutPoints[vertex - 1] = true;

}

bool isVisited(int vertex) const

{

return visited[vertex - 1];

}

};

std::vector<bool> findCutPoint(const Graph& graph)

{

std::vector<bool> cutPoints(graph.getNumOfVer());

DFS dfs(graph, cutPoints);

for (int i = 1; i <= graph.getNumOfVer(); i++)

if (!dfs.isVisited(i))

dfs(i, -1);

return cutPoints;

}

static class BiconnComp

{

const Graph& graph;

std::vector<bool> visited;

std::vector<Graph> result;

void find()

{

std::vector<bool> cutPoints(graph.getNumOfVer());

DFS dfs(graph, cutPoints);

for (int i = 1; i <= graph.getNumOfVer(); i++)

if (!dfs.isVisited(i))

dfs(i, -1);

for (int i = 1; i <= graph.getNumOfVer(); i++)

if (!visited[i - 1])

{

result.push\_back(Graph());

result[result.size() - 1].addVer(graph.getNumOfVer());

algo(i, -1, result.size() - 1, dfs);

}

}

void algo(int vertex, int parent, int curComp, DFS& dfs)

{

visited[vertex - 1] = true;

for (int u = 1; u <= graph.getNumOfVer(); u++)

{

if (graph.edgeId(vertex, u) == -1)

continue;

if (u == parent)

continue;

if (!visited[u - 1])

{

if (dfs.up[u - 1] >= dfs.tin[vertex - 1])

{

result.push\_back(Graph());

result[result.size() - 1].addVer(graph.getNumOfVer());

result[result.size() - 1].addE(vertex, u);

algo(u, vertex, result.size() - 1, dfs);

}

else

{

result[curComp].addE(vertex, u);

algo(u, vertex, curComp, dfs);

}

}

else if (dfs.tin[u - 1] < dfs.tin[vertex - 1])

result[curComp].addE(vertex, u);

}

}

public:

BiconnComp(const Graph& graph)

: graph(graph), visited(graph.getNumOfVer())

{

for (int i = 0; i < visited.size(); i++)

visited[i] = false;

find();

}

std::vector<Graph>& get()

{

return result;

}

};

std::vector<Graph> findBiconnComp(const Graph& graph)

{

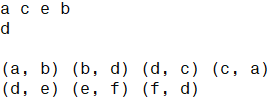
BiconnComp bc(graph);

return bc.get();

}

} // Lab5

Результаты выполнения программы:



Вывод: научиться определять компоненты сильной связности в орграфе, определять компоненты двусвязности в неориентированном графе.