Министерство образования Республики Беларусь

Учреждения образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №6

По дисциплине МОИС за 1 семестр

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

***Определение компонент двусвязности для неориентированного графа***»

Выполнил

Студент группы ИИ-21

Корпач Д.Р.

Проверил

Козинский А.А.

Брест 2022

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

1.

#include <iostream>

#include "../graph\_LIB.hh"

int main(){

    convert c;

    std::string file\_path = "inf1.txt";

    std::vector<int> nodes = c.reading\_file(file\_path);

    int max\_node = c.count\_of\_nodes(nodes);

    std::vector<std::vector<int>> oriented\_adjacencyMatrix = c.oriented\_adjancy(nodes, max\_node);

    int n = nodes.size()/2;

    for(int i=0;i<oriented\_adjacencyMatrix.size();i++){

        for(int j=0;j<oriented\_adjacencyMatrix.size();j++){

            std::cout<<oriented\_adjacencyMatrix[i][j]<<" ";

        }std::cout<<std::endl;

    }

    alg calculation;

    std::cout<<std::endl<<"algorithm kosaraju"<<std::endl;

    std::vector<std::vector<int>> component = calculation.kosaraju(oriented\_adjacencyMatrix);

    return 0;

}

2.

#include<iostream>

#include<vector>

#include "../graph\_LIB.hh"

using namespace std;

int main(){

    convert c;

    std::string file\_path = "inf2.txt";

    std::vector<int> nodes = c.reading\_file(file\_path);

    int max\_node = c.count\_of\_nodes(nodes);

    std::vector<std::vector<int>> adjance\_matrix\_ambiguity = c.adjancy(nodes, max\_node);

    int n = nodes.size()/2;

    alg calculation;

    std::cout<<std::endl<<"points of articulation: ";

    std::vector<int> articulation\_points = calculation.points\_of\_articulation(adjance\_matrix\_ambiguity, max\_node);

    for(int i=0;i<articulation\_points.size();i++){

        std::cout<<articulation\_points[i]<<" ";

    }std::cout<<std::endl;

    std::cout<<std::endl<<"bridges: "<<std::endl;

    int m = 2;

    std::vector<std::vector<int>> array\_edge(n, std::vector<int>(m));

    for (int i = 0, j = 0; i < n, j < 2 \* n; i++, j+=2){

            array\_edge[i][0] = nodes[j];

        }

    for (int i = 0, j = 1; i < n, j < 2 \* n; i++, j+=2){

            array\_edge[i][1] = nodes[j];

    }

    std::vector<std::vector<int>> Graph\_edges(max\_node + 1, std::vector<int>());  // your Graph

    for (int i = 0; i < n; ++i) {  // read edges with weigths

        Graph\_edges[array\_edge[i][0]].push\_back(array\_edge[i][1]);

        Graph\_edges[array\_edge[i][1]].push\_back(array\_edge[i][0]);

    }

    calculation.DFS\_for\_bridges(max\_node, Graph\_edges);

    std::cout<<"components double connected: "<<std::endl;

    int size\_articulation\_points = articulation\_points.size();

    std::vector<std::vector<int>> number\_subgraphs\_in\_graph=adjance\_matrix\_ambiguity;

    int double\_connected\_components = calculation.component\_double\_conected(adjance\_matrix\_ambiguity, articulation\_points,max\_node);

    std::cout<<double\_connected\_components<<std::endl;

    return 0;

}



