Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №1**

По дисциплине «Математические основы интеллектуальных систем»

Тема: «Нахождение компонент связности неориентированного графа»

**Выполнил:**

Студент 2 курса

Группы ИИ-21

Карагодин Д.Л.

**Проверил:**

Козинский А.А.

Брест 2022

**Цель работы:** Изучение и решение заданий по построению таблицы смежности и инцидентности, нахождения компоненты связности неориентированного графа через поиск в ширину и глубину.

**Ход работы:**

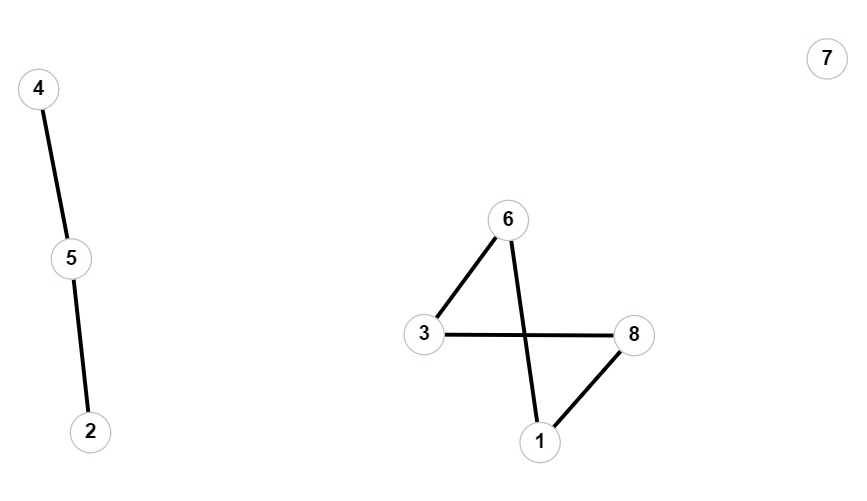
1. Построить матрицу смежности и инцидентности для заданного графа. Изобразить граф.

2. Используя поиск в глубину и поиск в ширину написать программу, определяющую число компонент связности графа. Методы представляются в виде функций.

**Вариант 4**

|  |  |  |  |
| --- | --- | --- | --- |
| ***№*** | ***Кол.***  ***вершин*** | ***Кол.***  ***ребер*** | ***Задание***  ***графа*** |
| 4. | 8 | 6 | (1,6),(1,8),(2,5),(3,6),(3,8),(4,5) |

**Изображение графа:**



**Код программы:**

#include <fstream>

#include <iostream>

#include <vector>

#include <string>

#include <stack>

using namespace std;

class Graphs

{

public:

struct graph\_struct { short first; short second; };

vector<graph\_struct> graph;

Graphs() {

}

Graphs(string fileName) {

ifstream ifs(fileName);

size\_t i = 0;

while (!ifs.eof()) {

short first, second;

ifs >> first >> second;

graph.push\_back(graph\_struct(first, second));

i++;

}

ifs.close();

};

~Graphs() { cout << "No Graphs :(" << endl; }

void Out() {

for (auto i : graph)

cout << i.first << "->" << i.second<<endl;

}

};

short count\_v(Graphs& gh) {

short max = 0;

for (short i = 0; i < gh.graph.size(); i++) {

if (gh.graph.at(i).first > max) {

max = gh.graph.at(i).first;

continue;

}

if (gh.graph.at(i).second > max) {

max = gh.graph.at(i).second;

continue;

}

}

return max;

}

vector<vector<bool>> matrix\_adj(Graphs& gh, bool output = false) {

short max = count\_v(gh);

vector<vector<bool>> adj(max, vector<bool>(max, false));

for (size\_t i = 0; i < max; i++) {

for (size\_t j = 0; j < max; j++) {

for (size\_t k = 0; k < gh.graph.size(); k++){

if ((gh.graph.at(k).first - 1 == i && gh.graph.at(k).second - 1 == j)

|| (gh.graph.at(k).first - 1 == j && gh.graph.at(k).second - 1 == i)) {

adj[i][j] = true;

break;}}}}

if (output) {

cout << " ";

for (size\_t i = 0; i < max; i++) cout << " " << i + 1;

for (size\_t i = 0; i < max; i++){

cout << endl << i + 1 << "|";

for (size\_t j = 0; j < max; j++) cout << adj[i][j] << " ";}

cout << endl;}

return adj;

}

void matrix\_incid(Graphs& gh) {

short max = count\_v(gh);

vector<vector<bool>> inc(gh.graph.size(), vector<bool>(max, false));

for (size\_t i = 0; i < gh.graph.size(); i++) {

for (size\_t j = 0; j < max; j++) {

if (gh.graph.at(i).first - 1 == j || gh.graph.at(i).second - 1 == j) {

inc[i][j] = true;}}}

cout << " ";

for (size\_t i = 0; i < max; i++){cout << " " << i + 1;}

for (size\_t i = 0; i < gh.graph.size(); i++)

{

cout << endl << gh.graph.at(i).first << gh.graph.at(i).second << "|";

for (size\_t j = 0; j < max; j++){cout << inc[i][j] << " ";}

}

cout << endl;}

void find\_in\_deep(Graphs& gh) {

short global\_component = 0,max = count\_v(gh),i = 0;

bool all\_visited = false;

vector<bool> visited(max, false);

vector<vector<bool>> mat\_adj = matrix\_adj(gh);

stack<short> stack;

for (short j = 0; j < max; j++) {

if (visited[j]) continue;

else {

stack.push(j);

all\_visited = false;

i = j;

visited[i] = true;}

while (!all\_visited) {

size\_t st = stack.size();

for (short k = 0; k < max; k++) {

if (mat\_adj[i][k] && !visited[k]) {

stack.push(k);visited[k] = true;

break;}}

if (stack.size() == st) stack.pop();

if (!stack.empty()) i = stack.top();

all\_visited = stack.empty();}

++global\_component;}

cout << "Global\_Copmonent: " << global\_component << endl;

}

void find\_in\_weight(Graphs& gh) {

short global\_component = 0,max = count\_v(gh),i = 0;

bool all\_visited = false;

vector<bool> visited(max, false);

vector<vector<bool>>mat\_adj = matrix\_adj(gh);

stack<short> stack;

for (short j = 0; j < max; j++) {

if (visited[j]) continue;

else {

stack.push(j);

all\_visited = false;

i = j;

visited[i] = true;

}

while (!all\_visited) {

size\_t st = stack.size();

for (short k = 0; k < max; k++) {

if (mat\_adj[i][k] && !visited[k]) {

stack.push(k);visited[k] = true;}}

if (stack.size() == st) stack.pop();

if (!stack.empty()) i = stack.top();

all\_visited = stack.empty();}

++global\_component;}

cout << "Global\_Copmonent: " << global\_component << endl;

}

int main(){

Graphs gh("graph.txt");

gh.Out();

cout << "Martix ADJ: " << endl;

matrix\_adj(gh, true);

cout << "Matrix INC: " << endl;

matrix\_incid(gh);

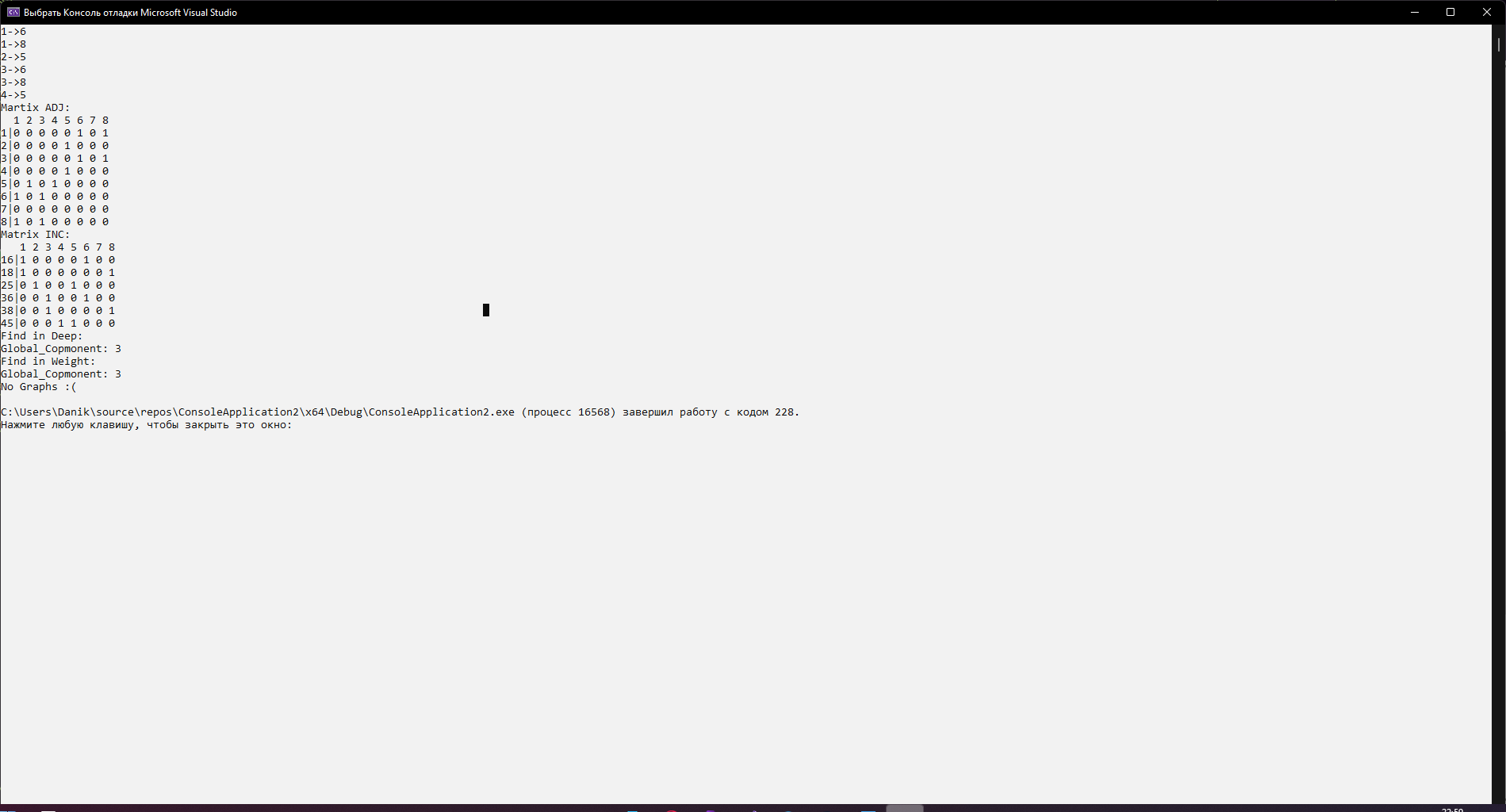
cout << "Find in Deep: "<<endl;

find\_in\_deep(gh);

cout << "Find in Weight: "<<endl;

find\_in\_weight(gh);

return 228;}

**Результат программы:**

**Вывод:** Изучил и решил задания по построению таблицы смежности и инцидентности, нахождения компоненты связности неориентированного графа через поиск в ширину и глубину.