Министерство образования Республики Беларусь

Учреждение образования

“Брестский Государственный технический университет”

Кафедра ИИТ

Лабораторная работа №1

По дисциплине «Математические основы И С»

Тема: Нахождение компонент связности неориентированного графа

Выполнил:

Студент 2 курса

Группы ИИ-23

Макаревич Н. Р.

Проверил:

Козинский А. А.

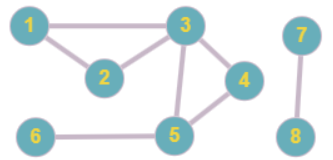
Брест 2023

Вариант 3

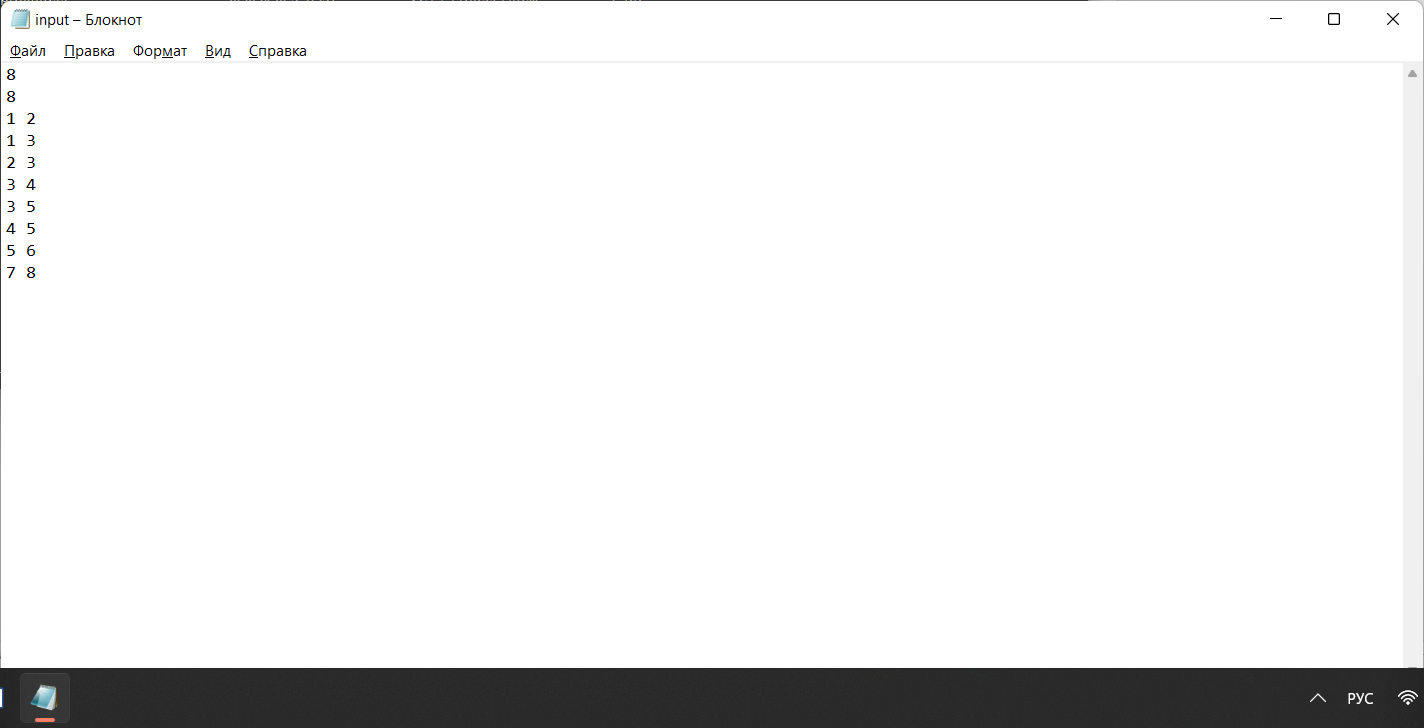
***Задание.***

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.
3. Варианты заданий указаны в таблице 1.
4. В таблице граф задан списком ребер.

|  |  |  |  |
| --- | --- | --- | --- |
| ***№*** | ***Кол.***  ***вершин*** | ***Кол.***  ***ребер*** | ***Задание***  ***графа*** |
| 3. | 8 | 8 | (1,2),(1,3),(2,3),(3,4),(3,5)(4,5),  (5,6)(7,8) |



Файл с входными данными input.txt:



Код программы:

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <queue>

using namespace std;

class Graph {

public:

string name;

int\*\* matrix;

int numOfVertex, numOfEdge;

struct vertex {

int num;

bool visited = false;

vector <vertex\*> next;

};

vertex\* arr;

Graph(string name) {

this->name = name;

ifstream fin;

fin.open(name);

fin >> numOfVertex >> numOfEdge;

arr = new vertex[numOfVertex];

}

void createAdjacencyMatrix() {

ifstream fin;

fin.open(name);

matrix = new int\* [numOfVertex];

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

matrix[i] = new int[numOfVertex];

for (int i = 0; i < numOfVertex; i++) {

for (int j = 0; j < numOfVertex; j++) {

matrix[i][j] = 0;

}

}

int trash;

fin >> trash>>trash;

for (int i = 0; i < numOfEdge; i++) {

int from, to;

fin >> from >> to;

matrix[from-1][to-1] = 1;

}

ofstream fout;

fout.open("Adjacency.txt");

fout << "\t";

for (int i = 1; i <= numOfVertex; i++)

fout << i << "\t";

fout << endl;

for (int i = 0; i < numOfVertex; i++) {

fout << i + 1 << "\t";

for (int j = 0; j < numOfVertex; j++) {

fout << matrix[i][j] << "\t";

}

fout << endl;

}

fout.close();

fin.close();

cout << "complited adj" << endl;

}

void createIncidenceMatrix() {

ifstream fin;

fin.open(name);

int trash;

fin >> trash >> trash;

matrix = new int\* [numOfVertex];

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

matrix[i] = new int[numOfEdge];

for (int i = 0; i < numOfVertex; i++) {

for (int j = 0; j < numOfEdge; j++) {

matrix[i][j] = 0;

}

}

for (int i = 0; i < numOfEdge; i++) {

int from, to;

fin >> from >> to;

matrix[from - 1][i] = 1;

matrix[to - 1][i] = -1;

}

ofstream fout;

fout.open("Incidence.txt");

fout << "\t";

for (int i = 1; i <= numOfEdge; i++)

fout << i << "\t";

fout << endl;

for (int i = 0; i < numOfVertex; i++) {

fout << i + 1 << "\t";

for (int j = 0; j < numOfEdge; j++) {

fout << matrix[i][j] << "\t";

}

fout << endl;

}

fin.close();

fout.close();

cout << "complited inc" << endl;

}

void createGraph() {

ifstream fin;

fin.open(name);

int trash;

fin >> trash >> trash;

for (int i = 0; i < numOfVertex; i++) {

int from, to;

vertex\* temp;

fin >> from >> to;

temp = &arr[to - 1];

arr[from - 1].next.push\_back(temp);

arr[i].num = i;

}

}

int compDFS() {

int res = 0;

for (int i = 0; i < numOfVertex; i++) {

if (!arr[i].visited) {

res++;

dfs(i);

}

}

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

arr[i].visited = false;

return res;

}

int compBFS() {

int res = 0;

for (int i = 0; i < numOfVertex; i++) {

if (!arr[i].visited) {

res++;

bfs(i);

}

}

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

arr[i].visited = false;

return res;

}

private:

void dfs(int v) {

arr[v].visited = true;

for (int i = 0; i < arr[v].next.size(); i++) {

if (!arr[v].next[i]->visited) {

dfs(arr[v].next[i]->num);

}

}

}

void bfs(int v) {

arr[v].visited = true;

queue <vertex\*> q;

q.push(&arr[v]);

while (!q.empty()) {

int v = q.front()->num;

arr[v].visited = true;

q.pop();

for (int i = 0; i < arr[v].next.size(); i++) {

if (!arr[v].next[i]->visited) {

q.push(arr[v].next[i]);

}

}

}

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

string name;

cout << "Введите название текстового файла с расширением: ";

cin >> name;

Graph graph(name);

graph.createAdjacencyMatrix();

graph.createIncidenceMatrix();

graph.createGraph();

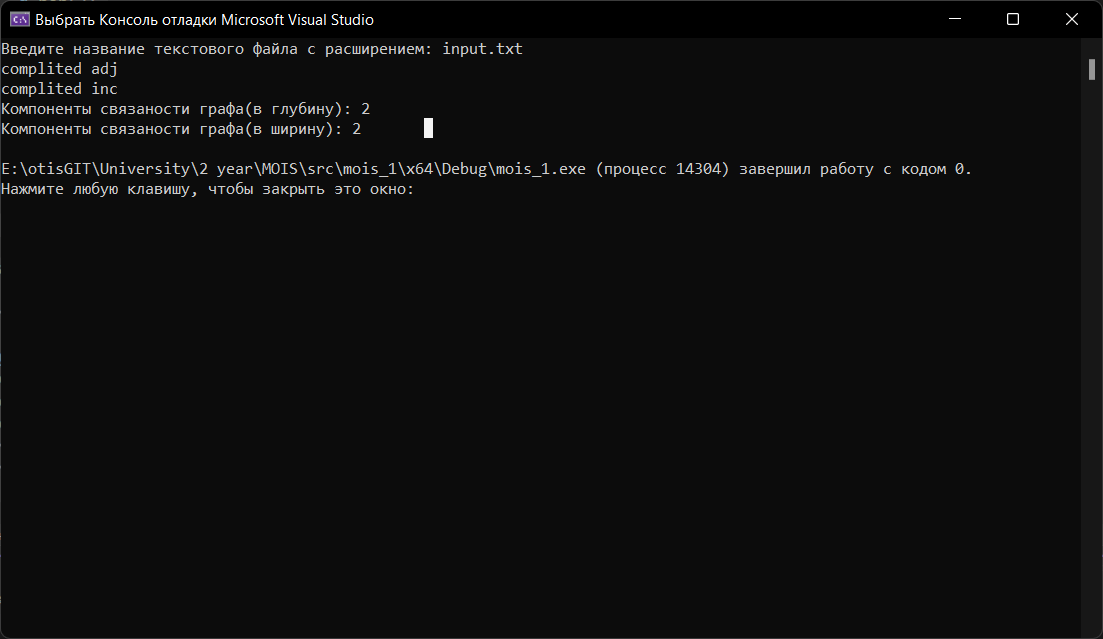
cout << "Компоненты связаности графа(в глубину): " << graph.compDFS() << endl;

cout << "Компоненты связаности графа(в ширину): " << graph.compBFS() << endl;

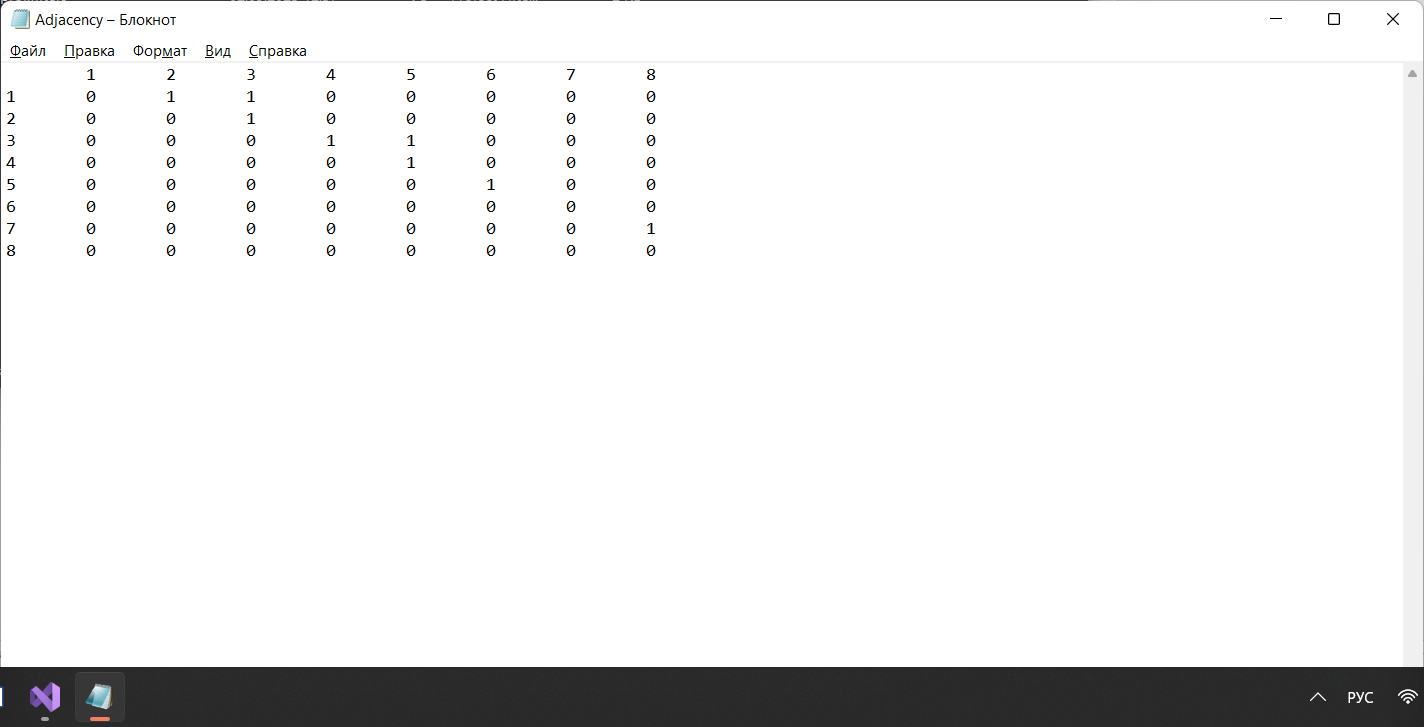
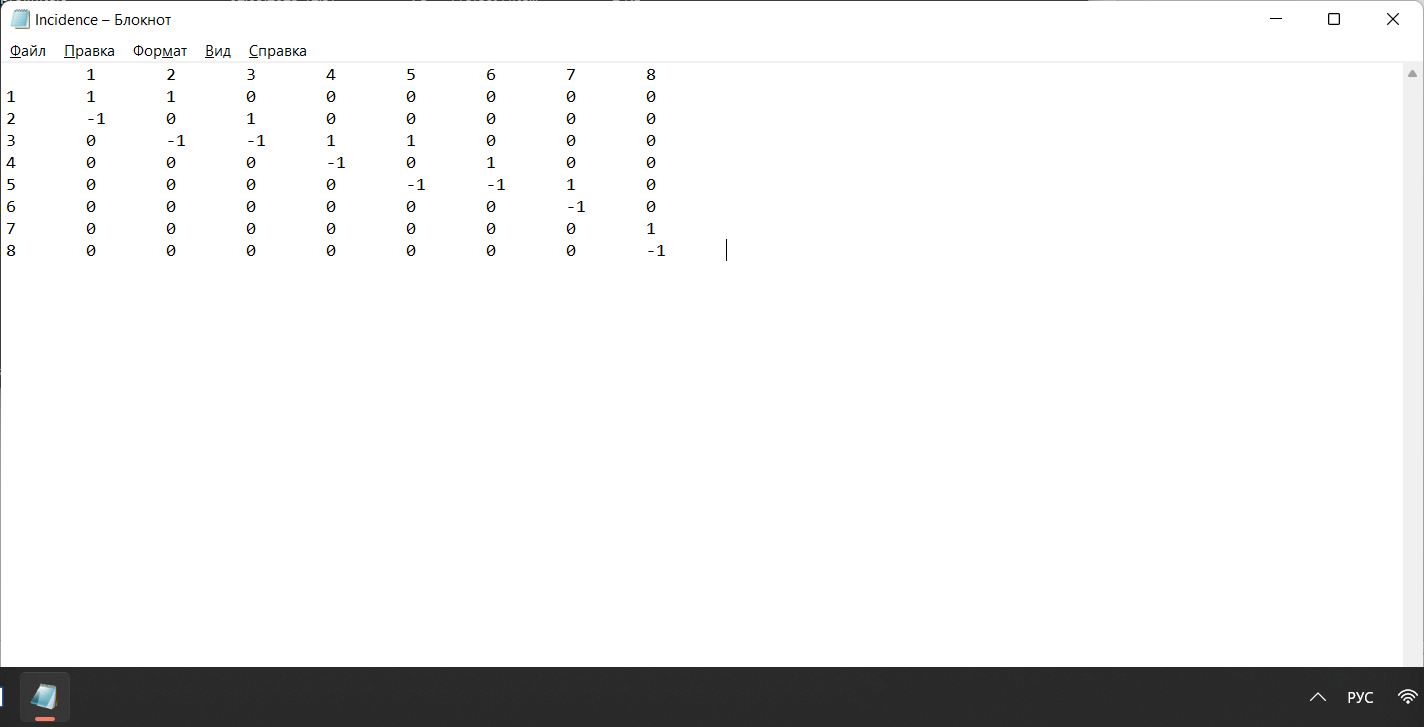
return 0;

}

Вывод программы:



Матрица смежности и инцидентности:

Вывод:

Построил матрицу смежности и инцидентности графа. Использовал поиск в глубину и в ширину для определения числа компонент связности графа.