ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | С. А. Рогачев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №7 |
| АЛГОРИТМЫ НА ГРАФАХ |
| по курсу: Структуры и алгоритмы обработки данных |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4932 |  |  |  | Н.С. Иванов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

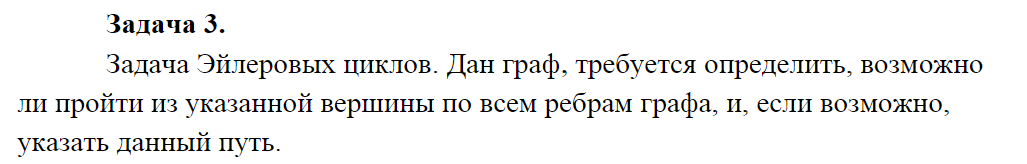
Санкт-Петербург 2020

*1. Цель работы*

Целью работы является изучение графов и получениепрактических навыков их использования.

*2. Задание*





*3. Текст программы*

## Lab7.h

#include <iostream>

#include <iomanip>

#include <vector>

#include <queue>

#include <stack>

using namespace std;

#define col count\_node

#define row count\_edge

class Graph

{

private:

public:

int \*\*matrix = nullptr;

int count\_node; // col

int count\_edge; // row

int count\_odd\_vertex = 0;

int odd\_vertex = 0;

Graph()

{

wcout << L"Введите количество вершин : ";

wcin >> count\_node;

wcout << L"Введите количество ребер : ";

wcin >> count\_edge;

// init

matrix = new int\* [row];

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

{

matrix[i] = new int[col];

}

// fill

wcout << L'\n';

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

{

for (int j = 0; j < row; j++) // заполняем столбец

{

wcout << L"[" << i + 1 << L"][" << j + 1 << L"] = "; // were change i <-> j

wcin >> matrix[j][i];

// wcout << L'\n';

}

}

}

bool check\_for\_euler\_path()

{

int pow\_vertex = 0;

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

{

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

{

if (matrix[j][i] > 0) { pow\_vertex++; }

}

if (pow\_vertex % 2 == 1){ count\_odd\_vertex++; odd\_vertex = i; }

}

if (count\_odd\_vertex > 2) { return false; }

if(find\_count\_components() > 1) { return false; }

return true;

}

vector <int> find\_euler\_path()

{

vector<int> res;

if(check\_for\_euler\_path() == false)

{

return res;

}

int start;

if(count\_odd\_vertex != 0)

{

start = odd\_vertex;

}

else

{

start = 0;

}

vector<int> neigh;

stack<int> s;

s.push(start);

int u;

while (!s.empty())

{

int v = s.top();

neigh = find\_neigh(v);

if(!neigh.empty())

{

u = neigh[neigh.size()-1];

neigh.pop\_back();

s.push(u);

delete\_edge(edge\_between\_vert(v, u)); // удалить связь между вершинами

}

else

{

s.pop();

res.push\_back(v);

}

}

return res;

}

void delete\_edge(int num\_v) // return int - num edge

{

int i = num\_v;

for (int k = 0; k < col; k++) // пройти этот слолбец

{

if (matrix[i][k] == 1){ matrix[i][k] = 0; } // и добавить соседей

}

}

int edge\_between\_vert(int v, int u)

{

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

{

if (matrix[i][v] == 1 && matrix[i][u] == 1) { return i; }

}

return 0;

}

int find\_count\_components()

{

vector<bool> vizited;

vector<int> components;

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

{

components.push\_back(0);

vizited.push\_back(0);

}

int component\_count = 0;

int start;

int tmp;

queue <int> q;

vector <int> neigh;

while (find\_zero\_val(components) != col)

{

start = find\_zero\_val(components);

component\_count += 1;

q.push(start); // push start vertex

while (!q.empty())

{

tmp = q.front(); // достать вершину

components[tmp] = component\_count; // обработка вершины

neigh = find\_neigh(tmp); // найти соседей

vizited[tmp] = 1;

q.pop(); // удалить обработанную вершину

for (int i : neigh)

{

if (vizited[i] != 1)

{

q.push(i); // добавить соседей в очередь

}

}

neigh.clear(); // отчистить список соседей

}

}

return component\_count;

}

vector <int> find\_neigh(int num\_v) // neighboring

{

vector<int> neigh;

int i = num\_v;

for (int j = 0; j < row; j++) // пройти по строке

{

if (matrix[j][i] == 1) // если найдена 1

{

for (int k = 0; k < col; k++) // пройти этот слолбец

{

if (matrix[j][k] == 1 && k!=i){ neigh.push\_back(k); } // и добавить соседей

}

}

}

return neigh;

}

int find\_zero\_val(vector<int> arr)

{

int i = 0;

while (i < arr.size())

{

if (arr[i] == 0) { return i; }

i++;

}

return i;

}

void print()

{

wcout << endl << L"Матрица : " << endl;

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

{

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

{

wcout << L"|" << setw(3) << matrix[j][i] << " ";

}

wcout << L"| \n";

}

wcout << L"\n";

}

~Graph()

{

// устранение утечек памяти

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

{

delete[] matrix[i];

}

delete[] matrix;

}

};

## main.cpp

// #include "color\_out.h"

#include "lab7.h"

#include <iostream>

#include <Windows.h>

#include <fcntl.h>

#include <io.h>

#include <vector>

#include <queue>

using namespace std;

//----

// Для обнаружения утечек памяти

#define \_CRTDBG\_MAP\_ALLOC

#include <stdlib.h>

#include <crtdbg.h>

#ifdef \_DEBUG

#ifndef DBG\_NEW

#define DBG\_NEW new ( \_NORMAL\_BLOCK , \_\_FILE\_\_ , \_\_LINE\_\_ )

#define newDBG\_NEW

#endif

#endif

#define DumpMemoryLeaks \

\_CrtSetReportMode(\_CRT\_WARN, \_CRTDBG\_MODE\_FILE); \

\_CrtSetReportFile(\_CRT\_WARN, \_CRTDBG\_FILE\_STDOUT); \

\_CrtSetReportMode(\_CRT\_ERROR, \_CRTDBG\_MODE\_FILE); \

\_CrtSetReportFile(\_CRT\_ERROR, \_CRTDBG\_FILE\_STDOUT); \

\_CrtSetReportMode(\_CRT\_ASSERT, \_CRTDBG\_MODE\_FILE); \

\_CrtSetReportFile(\_CRT\_ASSERT, \_CRTDBG\_FILE\_STDOUT); \

\_CrtDumpMemoryLeaks();

int main()

{

int input\_int;

input\_int = \_setmode(\_fileno(stdout), \_O\_U16TEXT);

vector<int> v;

Graph g;

g.print();

wcout << L"\n" << endl;

for (int k = 0; k < g.count\_node; k++)

{

wcout << L"Соседи вершины " << k+1 << L" : " << endl;

v = g.find\_neigh(k);

for (auto i : v)

{

wcout << L"|" << i+1;

}

wcout << L"| \n" << endl;

}

int c = g.find\_count\_components();

wcout << L"\n\nКоличество компонент связности = " << c << endl;

auto p = g.find\_euler\_path();

if(p.size() == 0)

{

wcout << endl << L"Пути не существует" << endl << endl;

}

else

{

for (auto i : p)

{

wcout << L"->" << i+1;

}

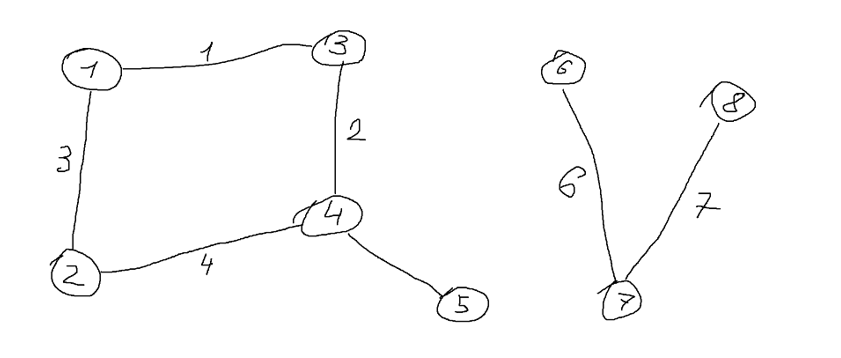
}

return 0;

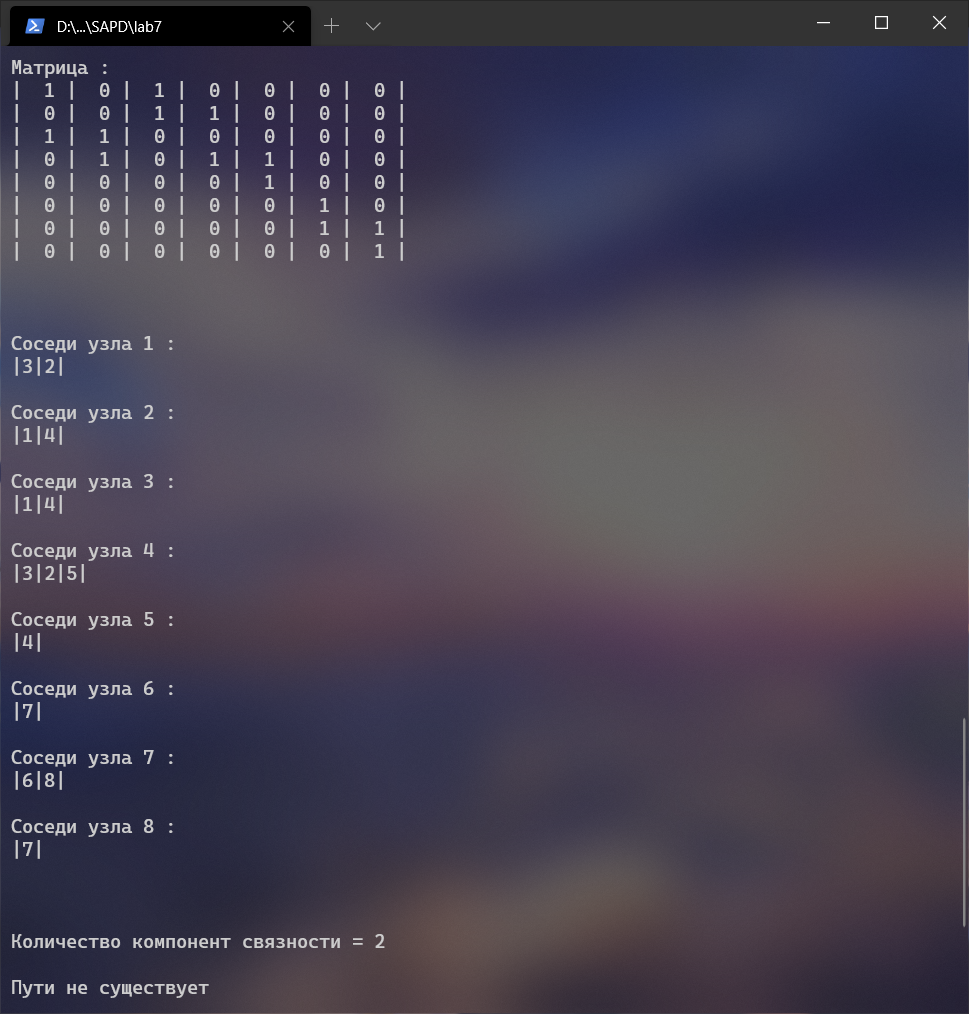
}

*4. Пример выполнения программы*

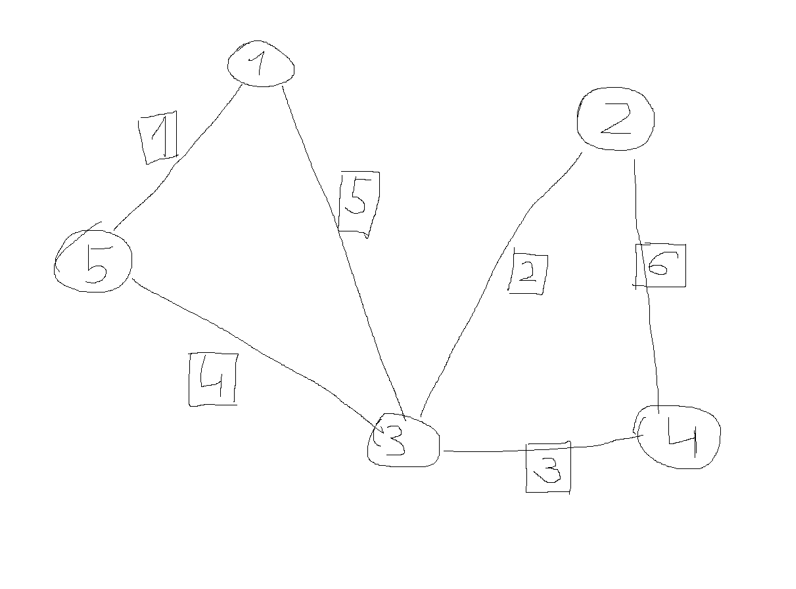
Пример графа:



Попытка найти Эйлеров путь:



Второй пример графа:



Решение:



*5. Выводы*

* Алгоритмы на графах сложны в реализации, но эффективны