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 полученииепрактических навыков их использования.

2. Задание

8	3	Матрица инцидентности

Задача 3.

Задача Эйлеровых циклов. Дан граф, требуется определить, возможно ли пройти из указанной вершины по всем ребрам графа, и, если возможно, указать данный путь.

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;
    }
};
```

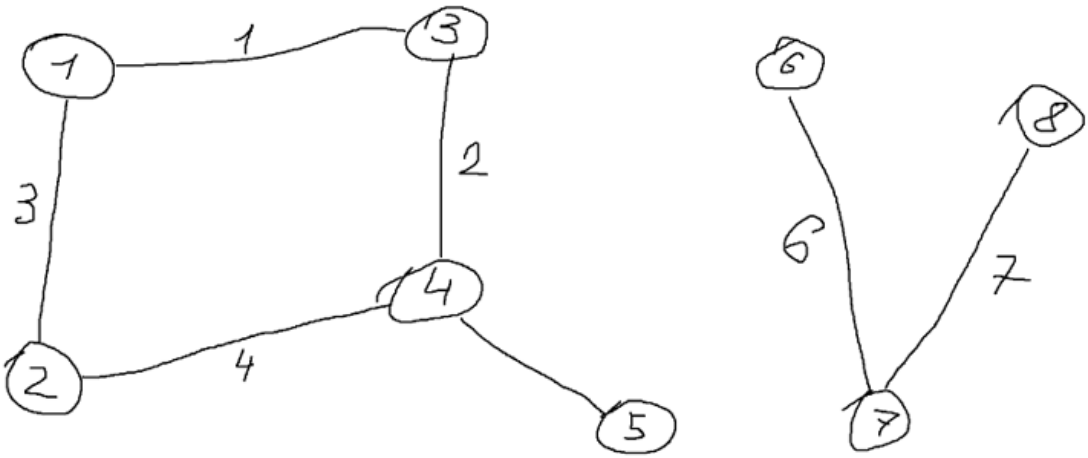
main.cpp

```
// #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)
```

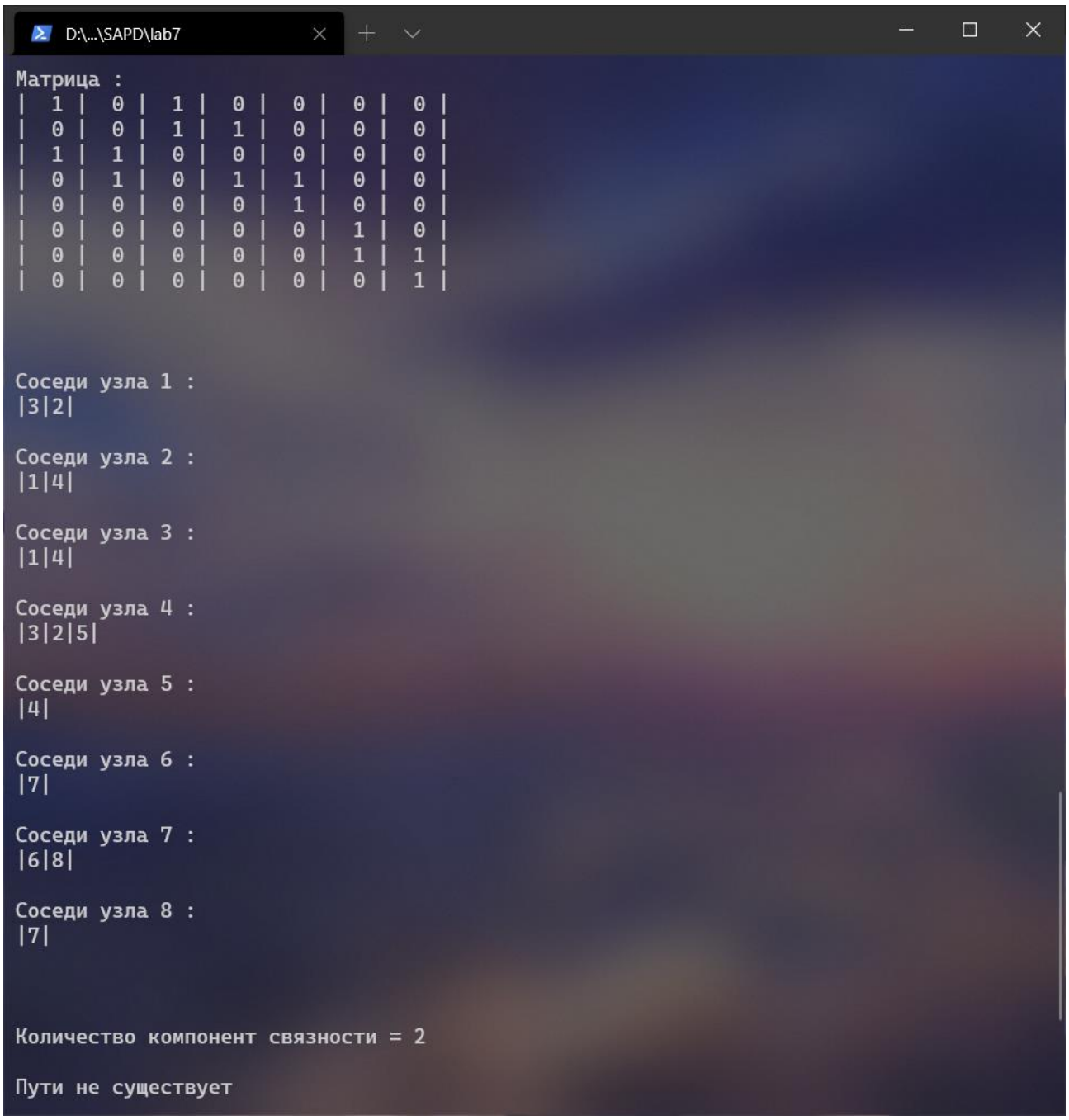
```
    {
        wcout << endl << L"Пути не существует" << endl << endl;
    }
    else
    {
        for (auto i : p)
        {
            wcout << L"->" << i+1;
        }
    }

    return 0;
}
```

4. Пример выполнения программы

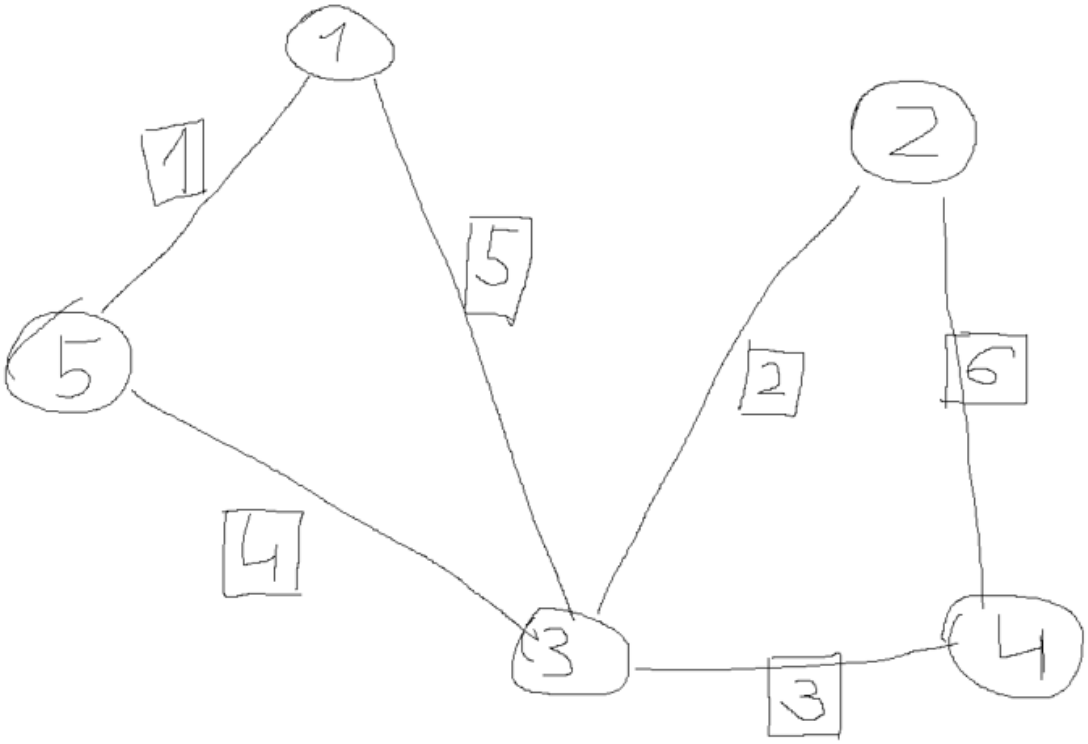
Пример графа:



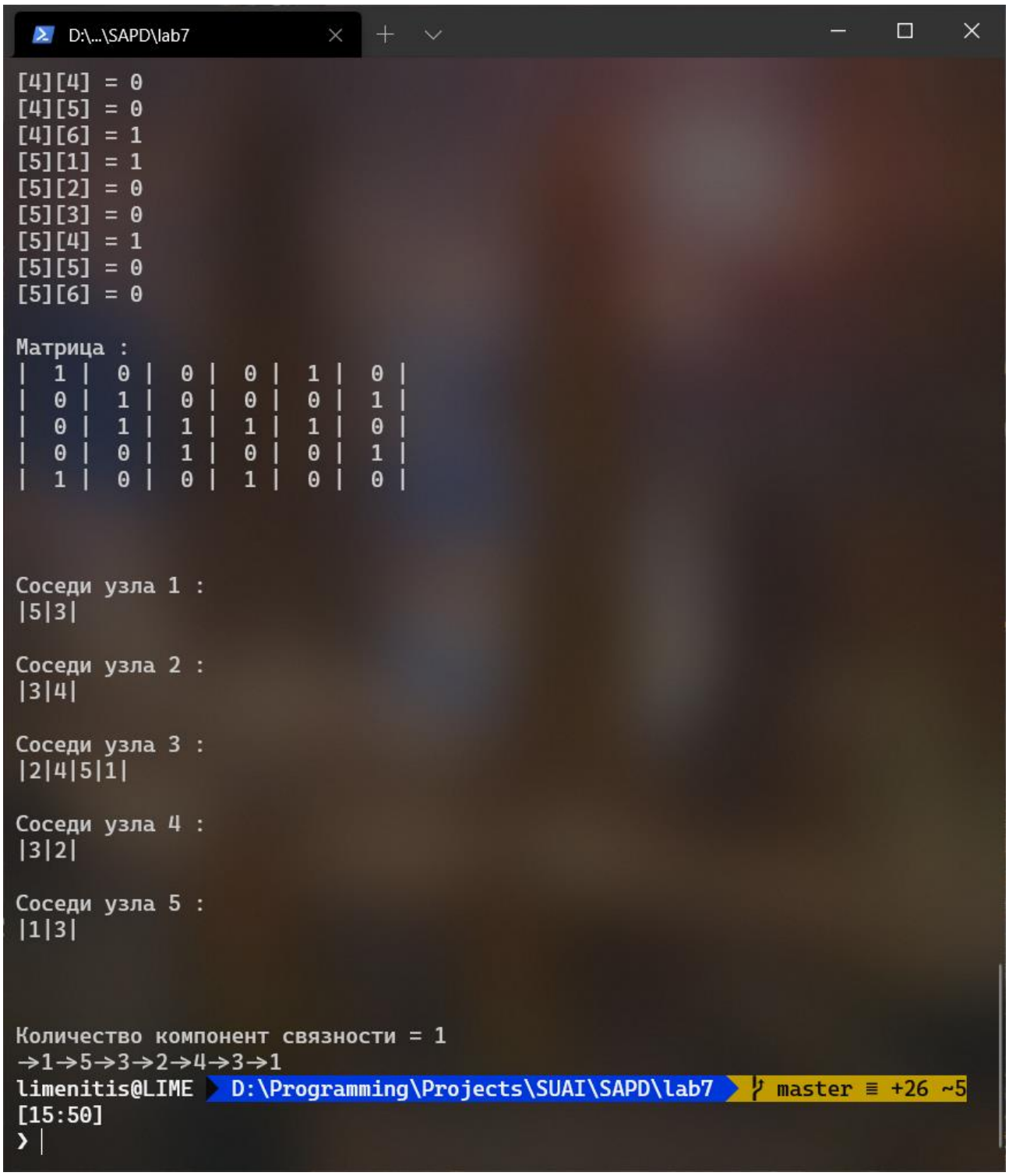
Попытка найти Эйлеров путь:



Второй пример графа:



Решение:



5. Выводы

- Алгоритмы на графах сложны в реализации, но эффективны