МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе № 1

по дисциплине «Программирование»

Тема: Рекурсия

Студент гр. 7383 Бахеров Д.В.

Преподаватель Фирсов М.А.

Санкт-Петербург

2018

1. Цель работы.

Ознакомление с основными понятиями и приёмами рекурсивного программирования, получение навыков программирования рекурсивных функций.

2. Постановка задачи.

Вариант 3.

Имеется n городов, пронумерованных от 1 до n. Некоторые пары городов соединены дорогами. Определить, можно ли попасть по этим дорогам из одного заданного города в другой заданный город. Входная информация о дорогах задаётся в виде последовательности пар чисел i и j (i < j и i, j=1..n), указывающих, что i-й и j-й города соединены дорогами.

3. Основные теоретические положения.

Функция называется рекурсивной, если во время ее обработки возникает ее повторный вызов, либо непосредственно, либо косвенно, путем цепочки вызовов других функций.

Прямой (непосредственной) рекурсией является вызов функции внутри тела этой функции.

Косвенной рекурсией является рекурсия, осуществляющая рекурсивный вызов функции посредством цепочки вызова других функций. Все функции, входящие в цепочку, тоже считаются рекурсивными.

Полученное задание можно перефразировать так:

Задан неориентированный граф (задан ребрами). Требуется узнать, находятся ли 2 введенные вершины в одной компоненте связности.

Граф — абстрактный математический объект, представляющий собой множество вершин графа и набор рёбер, то есть соединений между парами вершин.

Матрица смежности графа G с конечным числом вершин n (пронумерованных числами от 1 до n) — это квадратная матрица A размера n, в которой значение элемента a_{ij} равно числу рёбер из i-й вершины графа в j-ю вершину.

Компонента связности графа G — максимальный (по включению) связный подграф графа G.

Таким образом, задачу можно разбить на 2 части:

1) Создание матрицы смежности.

Матрица смежности есть двумерный массив. Заполнение: попарно вводим номера вершин графа u и w и записываем ребро, соединяющее их в матрицу (элементы a[u][w] = true и a[w][u] = true)

2) Нахождение компонент связности и принадлежности вершин к компоненте.

Для решения потребуется завести массив visited, в котором будет храниться информация о посещении рекурсивным алгоритмом вершины; целочисленный массив components, в котором будет указан номер компоненты для каждой вершины. Алгоритм: проходим по вершинам в цикле, если вершина не посещена (!visited[i]), то

2.1) Запускаем функцию search_in_depth

2.2) Увеличиваем число компонент связности (num_components++).

Функция search_in_depth – рекурсивный поиск в глубину. Идея алгоритма: перебираем все исходящие из рассматриваемой вершины рёбра. Если ребро ведёт в вершину, которая не была рассмотрена ранее, то запускаем алгоритм от этой нерассмотренной вершины, а после возвращаемся и продолжаем перебирать рёбра. В моем случае, алгоритм (на вход подается вершина v):

2.1.1) components[v] = num_components;

2.1.2) В visited отмечаем, что вершина пройдена (visited[v]=true)

2.1.3) Запускаем цикл по строкам матрицы смежности

2.1.3.1) Если у вершины v есть путь в j и при этом вершина j не пройдена, то запускаем search_in_depth по вершине j.

В результате мы получим массив components, заполненный номерами компонент связности для каждой вершины (вершины в роли индексов массива). Для решения задачи осталось узнать, совпадают ли значения элементов массива components с заданными индексами (components[v] =? components[w])

4. Реализация.

Функции:

1) bool find_way(int n, int town_i, int town_j, bool** road);

Функция, которая определяет, есть ли путь между двумя вершинами. входные данные. Возвращает 1 или 0.

n - число городов

town_i - начальный город

town_j - конечный город

road - указатель на двумерный массив - матрица смежности

2) void search_in_depth (int n, int current_top, int * components, int & num_components, bool* visited, bool** road);

Рекурсивная функция "Поиск в глубину"

n - число городов

current_top - текущая вершина

components - указатель на массив номеров компонент связности для каждой вершины

num_components - количество компонент связности вершины

visited - указатель на массив, в котром хранится информация о посещении вершин

road - указатель на двумерный массив - матрица смежности

3) Int input_roads(int n, bool** road, int num_ways, std::istream& in);

Функция ввода городов, между которыми существует путь, возвращает возможную ошибку при некорректном вводе n.

n - число городов

road - указатель на двумерный массив - матрица смежности

num_ways - общее кол-во существующих путей

Тестирование.

Были написаны 5 тестов для данной программы, а также скрипт для тестирования и компиляции.

Результаты тестирования представлены на рисунках 1-5.

```
Введите количество городов:
9
Введите количество путей: 5
Введите номера городов, между которыми существует дорога:
1 6
6 7
5 9
7 8
5 6
Введите города, между которыми нужно узнать существование пути:
1 8

Матрица смежности:
0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 1
1 0 0 0 1 0 1 0 0
0 0 0 0 0 1 0 1 0
0 0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0 0
Глубина рекурсии: 0
       Глубина рекурсии: 1
              Глубина рекурсии: 2
                     Глубина рекурсии: 3
              Глубина рекурсии: 2
                     Глубина рекурсии: 3
Глубина рекурсии: 0
Глубина рекурсии: 0
Глубина рекурсии: 0

Путь между городами 1 и 8 существует.
```

Рисунок 1

```
Введите количество городов:
24
Введите количество путей: 12
Введите номера городов, между которыми существует дорога:
1 6
4 9
9 15
2 19
14 21
21 24
4 17
10 15
4 6
6 9
9 10
15 22
Введите города, между которыми нужно узнать существование пути:
4 22

Матрица смежности:
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
Глубина рекурсии: 0
        Глубина рекурсии: 1
                Глубина рекурсии: 2
                        Глубина рекурсии: 3
                                Глубина рекурсии: 4
```

```
                                        Глубина рекурсии: 5
                                                Глубина рекурсии: 6
                        Глубина рекурсии: 3
Глубина рекурсии: 0
        Глубина рекурсии: 1
Глубина рекурсии: 0
Глубина рекурсии: 0
Глубина рекурсии: 0
Глубина рекурсии: 0
Глубина рекурсии: 0
Глубина рекурсии: 0
Глубина рекурсии: 0
Глубина рекурсии: 0
        Глубина рекурсии: 1
                Глубина рекурсии: 2
Глубина рекурсии: 0
Глубина рекурсии: 0
Глубина рекурсии: 0
Глубина рекурсии: 0

Путь между городами 4 и 22 существует.
```

Рисунок 2

```
Введите количество городов:
16
Введите количество путей: 8
Введите номера городов, между которыми существует дорога:
4 6
4 8
5 8
7 9
3 14
10 12
12 16
6 7
Введите города, между которыми нужно узнать существование пути:
3 15

Матрица смежности:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
Глубина рекурсии: 0
Глубина рекурсии: 0
Глубина рекурсии: 0
        Глубина рекурсии: 1
Глубина рекурсии: 0
        Глубина рекурсии: 1
                Глубина рекурсии: 2
                        Глубина рекурсии: 3
        Глубина рекурсии: 1
                Глубина рекурсии: 2
Глубина рекурсии: 0
        Глубина рекурсии: 1
                Глубина рекурсии: 2
Глубина рекурсии: 0
Глубина рекурсии: 0
Глубина рекурсии: 0

Путь между городами 3 и 15 не существует.
```

Рисунок 3

```
Введите количество городов:
14
Введите количество путей: 9
Введите номера городов, между которыми существует дорога:
1 6
2 8
8 12
12 13
13 14
5 7
7 9
5 11
6 13
Введите города, между которыми нужно узнать существование пути:
3  11

Матрица смежности:
0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0 1 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 1 0
0 0 0 0 0 1 0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0 0 0 0 0 1 0
Глубина рекурсии: 0
        Глубина рекурсии: 1
                Глубина рекурсии: 2
                        Глубина рекурсии: 3
                                Глубина рекурсии: 4
                                        Глубина рекурсии: 5
                        Глубина рекурсии: 3
Глубина рекурсии: 0
Глубина рекурсии: 0
Глубина рекурсии: 0
        Глубина рекурсии: 1
                Глубина рекурсии: 2
        Глубина рекурсии: 1
Глубина рекурсии: 0

Путь между городами 3 и 11 не существует.
```

Рисунок 4

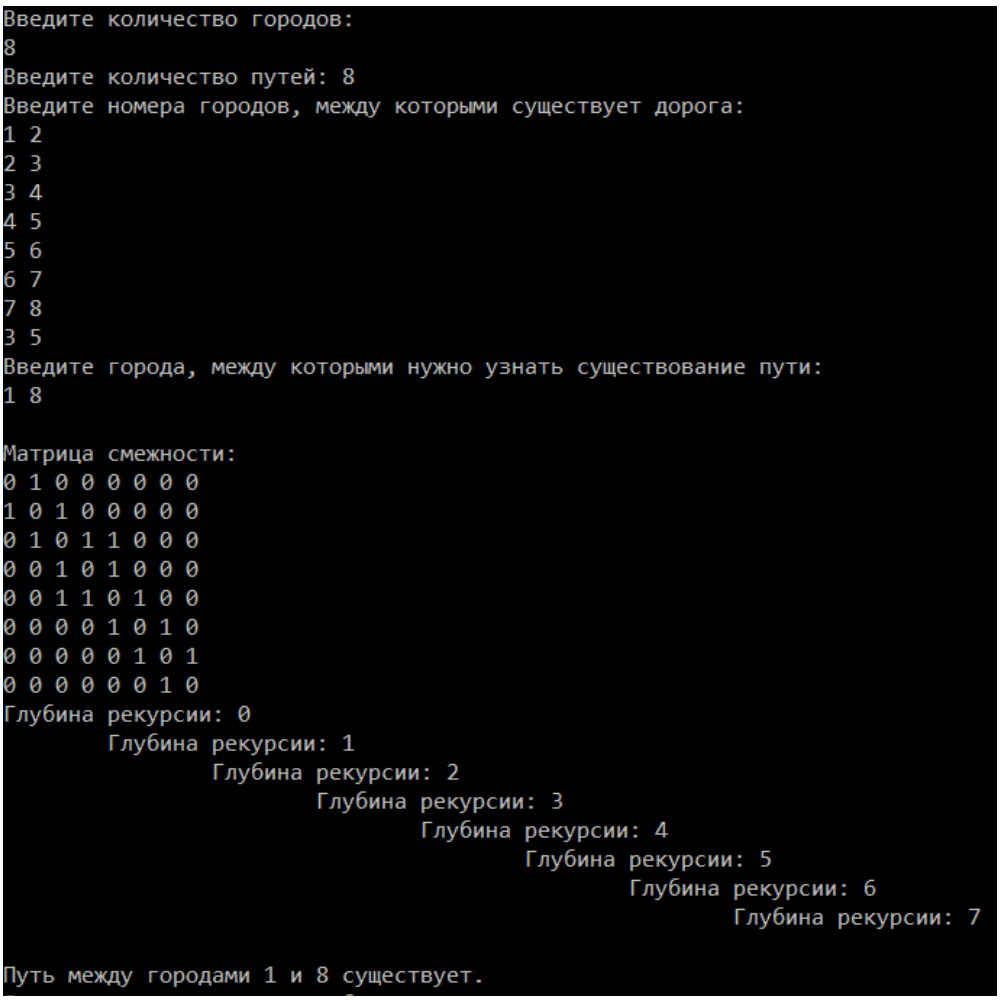

Рисунок 5

Выводы.

В ходе выполнения лабораторной работы были получены навыки реализации рекурсивных функций на языке программирования C++.

Так же научились работать с неориентированными графами, а именно создавать матрицу смежности, зная информацию о ребрах графа; определять компоненты связности графа и находить путь между двумя вершинами с помощью рекурсивного алгоритма поиска в глубину. К недостаткам решения данной задачи требуется отнести, что при довольно большом количестве вершин требуется не рекурсивное решение задачи, что будет более оптимальным. Стоит заметить, что данный алгоритм так же может найти кратчайший путь, но только в невзвешенном графе. Для поиска кратчайшего пути в взвешенном графе рекомендуется использовать алгоритм Дейкстры.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```cpp
#include <iostream>
#include <fstream>
#include <cmath>

using namespace std;

/*  Функция, которая определяет: существует ли путь между двумя вершинами.
    Для этого создаем массив в котором храним номер компоненты связности для каждой вершины. Затем достаточно узнать,
    в одной ли компоненте связности находятся заданные вершины
*/
bool find_way(int n, int town_i, int town_j, bool** road);

/*  Рекурсивная функция "Поиск в глубину"
    Идея: для каждой непройденной вершины находим все непройденные смежные вершины и повторяем поиск для них
*/

void search_in_depth(int n, int current_top, int* components, int& num_components, bool* visited, bool** road);

/*  Функция ввода городов, между которыми существует путь,
    возвращает возможную ошибку при некорректном вводе n
*/

int input_roads(int n, bool** road, int num_ways, std::istream& in);

int main()
{
    setlocale(LC_ALL, "Russian");
    int n = 0;              // количество городов
    bool **road;            // матрица смежности
    int town_i = 0;         // начальный город
    int town_j = 0;         // конечный город
    int num_ways = 0;       // общее количество существующих путей

    cout << "Введите количество городов: " << endl;
    cin >> n;

    cout << "Введите количество путей: ";
    cin >> num_ways;

    road = new bool*[n];
    for (int i = 0; i < n; i++)
        road[i] = new bool[n];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            road[i][j] = 0;

    cout << "Введите номера городов, между которыми существует дорога: " << endl;
    int res = input_roads(n, road, num_ways, cin);

    if (res == 1)
    {
        cout << "Неверно введены данные о городах (выход за пределы границ 1<=town<=n)" << endl;
        return -1;
    }
    if (res == 2)
    {
        cout << "Ошибка! По условию i < j" << endl;
        return -1;
    }

    cout << "Введите города, между которыми нужно узнать существование пути: " << endl;
    cin >> town_i >> town_j;
    town_i--;
    town_j--;
    cout << endl;
```

```
    cout << "Матрица смежности: \n";
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            cout << road[i][j] << " ";
        cout << endl;
    }

    if (find_way(n, town_i, town_j, road))
        cout << "\nПуть между городами " << town_i + 1 << " и " << town_j + 1 << " существует." << endl;
    else
        cout << "\nПуть между городами " << town_i + 1 << " и " << town_j + 1 << " не существует." << endl;
    for (int i = 0; i < n; i++)
        delete[] road[i];
    delete[] road;
    road = NULL;
    system("pause");
    return 0;
}

int input_roads(int n, bool** road, int num_ways, std::istream& in)
{
    int town_i, town_j;
    for(int i = 0; i < num_ways; i++)
    {
        in >> town_i;
        in >> town_j;
        if (town_i >= town_j)
            return 2;
        if ((town_i < 1) || (town_i > n) || (town_j < 1) || (town_j > n))
            return 1;
        road[town_i - 1][town_j - 1] = true;
        road[town_j - 1][town_i - 1] = true;
    }
    return;
}

bool find_way(int n, int town_i, int town_j, bool** road)
{
    bool *visited;              //  указатель на массив, в котром хранится информация о посещении вершин
    int *components;            //  указатель на массив номеров компонент связности для каждой вершины
    int num_components = 0;     //  количество компонент связности вершины
    components = new int[n];
    visited = new bool[n];
    for (int i = 0; i < n; i++)
    {
        visited[i] = false;
        components[i] = 0;
    }
    for (int i = 0; i < n; i++)
    {
        if (!visited[i])
        {
            search_in_depth(n, i, components, num_components, visited, road, 0);
            num_components++;
        }
    }
    if (components[town_i] == components[town_j])
        return true;
    else return false;
}

void search_in_depth(int n, int current_top, int* components, int& num_components, bool* visited, bool** road, int count)
{
    components[current_top] = num_components;
    visited[current_top] = true;
    cout << "Глубина рекурсии: " << count << endl;
    for (int j = 0; j < n; j++)
    {
        if (road[current_top][j] != 0)
            if (!visited[j])
                search_in_depth(n, j, components, num_components, visited, road, ++count);
    }
}
```