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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Алгоритмы и структуры данных»

Тема: Рекурсия

Студент гр. 9381 _____ Аухадиев А.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить основные понятия рекурсии и получить опыт работы с рекурсивными функциями на языке программирования C++, разработать программу, использующую рекурсию.

Задание.

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.

Основные теоретические положения.

Рекурсия - определение части функции через саму себя, то есть это функция, которая вызывает саму себя непосредственно (в своём теле) или косвенно (через другую функцию).

Всё решение сводится к решению основного или, как ещё его называют, базового случая. Существует такое понятие как шаг рекурсии или рекурсивный вызов. В случае, когда рекурсивная функция вызывается для решения сложной задачи (не базового случая) выполняется некоторое количество рекурсивных вызовов или шагов с целью сведения задачи к более простой. И так до тех пор пока не получится базовое решение.

Описание алгоритма.

В экземпляр класса Graph охраняются города (вершины) и дороги (рёбра) в виде квадратной симметричной относительной главной диагонали матрицы смежности с размером равным количеству вершин в квадрате. При наличии рёбер между вершинами ячейка на пересечении заполняется единицей, все остальные ячейки - нулями.

В рекурсивном методе bool findRoad(int a, int b, int tab) происходит поиск

пути из вершины a в вершину b. Алгоритм "проходит" по строчке a матрицы смежности и при нахождении единицы отмечает выбранный путь уже пройденным (в соответствующей ячейке устанавливается значение -1) и переходит на строчку по номеру, соответствующему найденному столбцу.

Проход по строчке происходит в цикле, а переход на нужную строку происходит при повторном вызове метода findRoad, пока значение a не станет равным b, после чего в локальную переменную bool res с помощью оператора |= сохраняется на всех уровнях рекурсии значение true, после чего происходит возврат значения из функции.

Сложность алгоритма оценивается как O(n), так как напрямую зависит от количества вершин графа.

Описание структур и функций.

Главной структурой программы является класс Graph:

1)Приватные поля:

1. int line - количество строк матрицы смежности

2. int** matrix - матрица смежности

2)Методы:

1. Graph() - Конструктор, инициализация графа с одной ячейкой

2. ~Graph() - Деструктор, высвобождение памяти, выделенной под матрицу

3. void push(int a, int b) - добавление в матрицу пути из a в b. Если выделенной под матрицу памяти для добавления недостаточно, память, высвобождается и выделяется новая, в которую копируются старые данные и добавляются новые.

4. bool findRoad(int a, int b, int tab) - поиск дороги из a в b, возвращает true, если дорога найдена и false в обратном случае. tab нужен для отладочного вывода.

5. bool readFile(string file_name, int& a, int& b) - считывание информации из файла с именем file_name, в a и b сохраняются дороги, путь между которыми нужно найти. Возвращает true при удачном считывании файла и false в обратном

случае.

Вывод.

Были изучены основные понятия рекурсии и получен опыт работы с рекурсивными функциями на языке программирования C++, разработана программа, использующая рекурсию.

№	Входные данные	Результат
1.	4 1 2 3 2 2 4 5 6 Найдём путь из 1 в 6	from 1 to 2 from 2 to 3 Путь через 2 и 3 проверен from 2 to 4 Путь через 2 и 4 проверен Путь через 1 и 2 проверен Путь не найден
2.	6 1 2 3 4 8 9 7 4 2 3 10 5 Найдём путь из 3 в 7	from 3 to 2 from 2 to 1 Путь через 2 и 1 проверен Путь через 3 и 2 проверен from 3 to 4 from 4 to 7 Путь через 4 и 7 проверен Путь через 3 и 4 проверен Путь найден
3.	4 90 100 40 50 50 100 90 40 Найдём путь из 40 в 100	from 40 to 50 from 50 to 100 Путь через 50 и 100 проверен Путь через 40 и 50 проверен Путь найден
4.	12 4 7 4 8 6 7 7 8 8 6 7 9 2 1 4 1 3 4 6 5 3 5 6 3 1 6 Найдём путь из 1 в 6	Найдём путь из 1 в 6 from 1 to 2 Путь через 1 и 2 проверен from 1 to 4 from 4 to 3 from 3 to 5 from 5 to 6 Путь через 5 и 6 проверен Путь через 3 и 5 проверен Путь через 4 и 3 проверен Путь через 1 и 4 проверен Путь найден
5.	4 1 2 10 2 3 2 10 4 Найдём путь из 2 в 4	from 2 to 1 Путь через 2 и 1 проверен from 2 to 3 Путь через 2 и 3 проверен from 2 to 10 from 10 to 4 Путь через 10 и 4 проверен Путь через 2 и 10 проверен

		Путь найден
6.	3 1 2 2 3 3 4 Найдём путь из 1в 5	from 1 to 2 from 2 to 3 from 3 to 4 Путь через 3 и 4 проверен Путь через 2 и 3 проверен Путь через 1 и 2 проверен Путь не найден
7.	6 1 2 3 4 8 9 7 4 2 3 10 5 Найдём путь из 7 в 3	from 7 to 4 from 4 to 3 Путь через 4 и 3 проверен Путь через 7 и 4 проверен Путь найден
8.	10 1 5 5 2 5 3 5 1 8 10 10 3 4 7 7 3 4 2 10 11 Найдём путь из 1 в 11	from 1 to 5 from 5 to 2 from 2 to 4 from 4 to 7 from 7 to 3 from 3 to 5 Путь через 3 и 5 проверен from 3 to 10 from 10 to 8 Путь через 10 и 8 проверен from 10 to 11 Путь через 10 и 11 проверен Путь через 3 и 10 проверен Путь через 7 и 3 проверен Путь через 4 и 7 проверен Путь через 2 и 4 проверен Путь через 5 и 2 проверен Путь через 1 и 5 проверен Путь найден
9.	6 2 3 4 2 6 2 3 6	from 2 to 3 from 3 to 6 from 6 to 2 from 2 to 4 Путь через 2 и 4 проверен

	7 3 6 8 Найдём путь из 2 в 8	Путь через 6 и 2 проверен from 6 to 8 Путь через 6 и 8 проверен Путь через 3 и 6 проверен Путь через 2 и 3 проверен Путь найден

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

class Graph{
private:
    int line;
    int** matrix;
public:
    Graph(){
        line = 1;
        matrix = new int*[1];
        matrix[0] = new int[1];
        matrix[0][0] = 0;
    }

    ~Graph(){
        for(int i = 0; i<line; i++){
            delete [] matrix[i];
        }
        delete []matrix;
    }

    void push(int a, int b){
        int old_line = line;
        int size_check = 0;
        if(a >= b && line < a) {  //проверка, достаточно
ли в матрице смежности
            line = a;           //строк и столбцов
            size_check++;
        }
        else if(b >= a && line < b) {
            line = b;
            size_check++;
        }
        if(size_check){       //увеличение матрицы до
необходимых размеров
            int **new_matrix = new int *[line];
            for (int i = 0; i < line; i++) {
```

```
                new_matrix[i] = new int[line];
            }
            for(int i = 0; i<line; i++)
                for(int j = 0; j<line; j++)
                    new_matrix[i][j] = 0;

            for (int i = 0; i < old_line; i++)
                for (int j = 0; j < old_line; j++)
                    new_matrix[i][j] = matrix[i][j];
            for (int i = 0; i < old_line; i++)
                delete [] matrix[i];
            delete [] matrix;
            matrix = new_matrix;
        }
        matrix[a-1][b-1] = 1; //заполнение соответствующих
элементов матрицы единицами,
        matrix[b-1][a-1] = 1; //обозначающими, что путь
между элементами существует
    }

    bool findRoad(int a, int b, int tab){
        if(a == b){      //Если искомый путь равен
конечному
            return true; //возвращается true
        }
        bool res = false;
        for(int j = 0; j<line; j++) {
            if (matrix[a][j] == 1) {
                matrix[a][j] = -1; //дорога перестаёт быть
проходимой
                matrix[j][a] = -1;//после первого прохода
по ней
                for (int i = 0; i < tab; i++) //Вывод
табуляции для наглядности
                    cout << "\t";       //глубины рекурсии
                cout << "from " << a + 1 << " to " << j +
1 << '\n';
                res |= findRoad(j, b, tab + 1);
//рекурсивный вызов функции
                for (int i = 0; i < tab; i++)
                    cout << "\t";
                cout << "Путь через " << a+1 << " и " <<
j+1 << " проверен\n";
                if (res)
```

```
                    return res;
                }
            }
            return res;
        }

        bool readFile(string file_name, int& a, int& b){
            ifstream input_file(file_name);
            if (!input_file.is_open()) {
                cerr << "Не удалось открыть файл\n" << endl;
                return false;
            }
            int n = 0, x1 = 0, x2 = 0;
            input_file >> n;
            for(int i = 0; i < n; i++) {
                input_file >> x1 >> x2;
                push(x1, x2);
            }
            input_file >> a >> b;
            cout << "Найдём путь из " << a << " в " << b << '\
n';
            input_file.close();
            return true;
        }
};


int main(){
    int choise = 0, n = 0;
    int a = 0, b = 0;
    Graph *graph = new Graph();
    std::cout << "Откуда будет производиться ввод? (0 -
консоль, 1 - файл)\n";
    std::cin >> choise;
    if(choise != 0 && choise != 1){
        std::cout << "Введите 0 для выбора консоли или 1
для выбора файла\n";
        delete graph;
        return 0;
    }
    if(choise == 0) {
        std::cout << "Сколько все дорог, соединяющих
города?\n";
        std::cin >> n;
```

```
        for (int i = 0; i < n; i++) {
            std::cin >> a >> b;
            graph->push(a, b);
        }
        cout << "Найдём путь из ";
        cin >> a;
        cout << " в ";
        cin >> b;
    }else if(choise == 1){
        std::cout << "Введите имя файла\n";
        string file_name;
        cin >> file_name;
        if(!graph->readFile(file_name, a, b))
            return 0;
    }
    int tab = 0;
    if(graph->findRoad(a-1, b-1, tab))
        std::cout << "Путь найден\n";
    else
        std::cout << "Путь не найден\n";
    delete graph;
    return 0;
}
```