**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: Программирование рекурсивных алгоритмов**

Студент гр. 7383 _____ Васильев А.И.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

**Цель работы**

Познакомиться с основными понятиями и приемами рекурсивного программирования, получить навыки программирования рекурсивных функций на языке программирования C++

**Постановка задачи**

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 задается в виде последовательности пар чисел *i* и *j* ($i<j$ и $i, j \in 1..n$), указывающих, что *i*-й и *j*-й города соединены дорогами.

**Реализация задачи**

**Описание переменных:**

- `double mx` – переменная, в которую последовательно записываются данные с файла, используется для поиска максимального номера города, так же для проверки корректности входных данных;
- `int n` – хранит число городов;
- `int k` – счетчик количества поступивших чисел на вход, если переменная равна нулю либо нечетная, то программа завершается, выдавая сообщение о не корректных входных данных;
- `int mtrx[n][n]` – массив, в котором хранится матрица смежности, показывающая между какими городами существует путь;
- `double start` – хранит номер города, от которого ведется поиск пути;
- `double finish` – хранит номер города, к которому ищется путь;
- `int old_front[n]` – массив, хранящий номера городов, от которых ведется поиск путей на данном шаге;
- `int new_front[n]` – массив, хранящий номера городов, к которым найден путь на данном шаге;
- `int marker[n]` – массив, в котором записаны метки для каждого города, номер ячейки массива соответствует номеру города, уменьшенному на единицу; если значение в ячейке 0 – это означает, что к городу, соответствующему этой ячейке уже найден путь и искать новый путь к нему не требуется;
- `int question` – принимает число с консоли, соответствующее выбранному варианту ответа на вопрос «Найти новый путь между городами?»
- `int count` – счетчик, который показывает количество городов, занесенных в массив `new_front`;
- `bool way` – переменная, показывающая существует ли искомый путь из одного города (true) или нет (false).

**Описание функций:**

**Входной файловый поток `ifstream input`:**

Передает программе данные с входного файла;

**Функция `main()`:**

В головной функции реализовано открытие файла входных данных (с проверкой открытия этого файла, в случае, если файл не открыт, программа завершается и выводит сообщение об ошибке на консоль), затем в переменную `mx` считывается по одному числу из входного файла, выполняется проверка на принадлежность этого числа к целочисленному типу данных (если поступило число с плавающей точкой, либо символ, то программа завершается и выводит на консоль сообщение о не корректных входных данных), сравнивается с `n`, для поиска города, с максимальным номером, при этом с каждой итерацией цикла (то есть с каждым последующим считанным числом из файла) счетчик `k` увеличивается на единицу. После всех считанных чисел с файла выполняется проверка на количество этих чисел в файле, которое хранится в переменной `k`. Эта переменная должна быть четным натуральным числом, если это не так, программа завершается и выводит на консоль сообщение о не корректных входных данных. Далее каретка перемещается в начало файла с помощью функций `input.clear()` и `input.seekg (0)`. На следующем этапе выделяется память под динамический массив размера n на n под матрицу смежности. На основе пар городов, представленных в файле заполняется матрица смежности (массив `mtrx`). Единица в матрице смежности говорит о том, что путь между городами существует, нуль – о том, что прямого пути между городами нет. Затем файл входных данных закрывается, на экран выводится информация о количестве городов. Вызывается функция `preparation(mtrx, n)` и после выполнения этой функции освобождается память, выделенная под массив `mtrx[n][n]`.

**Функция `void preparation(int** mtrx, int n)`:**

Данная функция выполняет подготовку к использованию волнового алгоритма Ли.

В функции динамически выделяется память под 3 массива: `old_front[n]`, `new_front[n]`, `marker[n]`, которая в конце функции освобождается. Далее следует цикл `while` в котором реализовано меню программы, и функции, считывающие с консоли номера городов, между которыми ищется путь, при этом если номера заданы не корректно (нуль, один и тот же город, число с плавающей точкой, символ), то на консоль выводится

сообщение об этом и на вход снова принимается пара чисел (эта проверка на корректность так же реализована с помощью вложенного цикла `while`).

После получения всех данных происходит заполнение масивов, массив меток заполняется значениями 9999, а массивы `old_front[n]` и `new_front[n]` заполняются значениями –1. Затем первому элементу массива `old_front` присваивается число-номер стартового города, а ячейке массива `marker,` соответствующей номеру города (то есть номер города, уменьшенный на единицу) присваивается значение 0, эта метка означает, что город с этим номером просмотрен и искать путь к нему не следует. Далее переменной булевого типа присваивается значение, возвращаемое функцией `wave(mtrx, n, start, finish, old_front, new_front, marker)`, если функция вернула значение `true`, то выводится сообщение о том, что запрашиваемый путь существует, если же вернулось значение `false`, то выводится сообщение, говорящее о том, что искомого пути не нашлось.

**Функция `bool wave (int** mtrx, int n, int start, int finish, int *old_front, int* new_front, int* marker, bool way=false):`**

В данной рекурсивной функции реализован волновой алгоритм Ли поиска путей в графе.

Начинается функция с того, что просматривается массив `old_front`, и если значение элемента массива не равно –1, то есть там записан номер города от которого следует найти все возможные пути, то выполняется поиск этих путей, причем города, которые уже просмотрены (то есть метка на этих городах равна 0) не участвуют в этом поиске. Проверка существования пути осуществляется с помощью просмотра строки матрицы смежности, номер которой соответствует номеру города из которого ведется поиск пути, если найден какой-нибудь путь в город m, то метка (`marker[m-1] = 0`), соответствующая этому городу изменяется на нуль, номер этого города записывается в массив `new_front`, и производится проверка на соответствие этого номера числу, записанного в переменной `finish`, то есть номеру города, в который ведется поиск пути. Если такое соответствие установлено, то функция возвращает значение `true` и завершает свою работу.

Если же такое соответствие не установлено, то подсчитывается количество элементов в массиве `new_front`, не равных —1. Если таких

элементов 0, то искомого пути не существует и функция завершает свою работу, возвращая значение `false`. Если же такие элементы присутствуют, то массив `new_front` копируется в массив `old_front`, а сам заполняется значениями –1. Далее идет рекурсивный вызов функции `wave.`

**Описание алгоритма**

В данной работе реализован упрощенный волновой алгоритм Ли поиска путей в графе.

На вход алгоритму поступает непустой граф, представленный в данной работе матрицей смежности. Требуется найти путь между вершинами start и finish графа (start не совпадает с finish). Алгоритм Ли позволяет найти кратчайший путь, но, так как в поставленной задаче этого не требуется, то будем использовать упрощенный алгоритм Ли.

Алгоритм заключается в следующем:

1. Каждой вершине $v_i$ приписывается целое число marker($v_i$) - волновая метка (начальное значение marker($v_i$)= 9999);
2. Заводятся два списка old_front и new_front (старый и новый «фронт волны»);
3. old_front:={start}; new_front:={}; marker(start):=0;
4. Для каждой из вершин, входящих в old_front, просматриваются инцидентные (смежные) ей вершины $u_j$, и если ($u_j$) = 9999, то marker($u_j$):=0, new_front:=new_front + {$u_j$};
5. Если new_front = {}, то выход (нет пути);
6. Если finish = new_front{$u_j$} (т.е. одна из вершин $u_j$ совпадает с finish), то найден путь между start и finish; выход («путь найден»);
7. old_front:=new_front; new_front:={}.

**Тестирование программы**

**Процесс тестирования:**

Программа собрана в операционной системе Ubuntu 18.04, с использованием компилятора g++. В других операционных системах и компиляторах тестирование не проводилось.

**Результаты тестирования:**

Тестовые случаи и выявленные в процессе тестирования ошибки представлены в приложении.

По результатам тестирования было показано, что поставленная задача выполнена.

**Выводы**

В ходе выполнения лабораторной работыб были получены навыки программирования рекурсивных функций на языке программирования C++. Так же был изучен волновой алгоритм Ли поиска путей в графе и его рекурсивная реализация.

# ПРИЛОЖЕНИЕ А

## Тестовые случаи

| Файл входных данных | Пара городов, между которыми ищется путь | Результат | Правильность результата |
|---|---|---|---|
| 1 2<br>1 3<br>2 3<br>2 4<br>3 4<br>4 5<br>4 6<br>5 6<br>7 8<br>8 9<br>2 10 | 1 6 | Между городами существует путь! | Верно |
| | 9 4 | Между городами пути нет! | Верно |
| | 0 4 | Не корректно указаны номера город. Пожалуйста, введите пару натуральных чисел, не превосходящих 10: | Верно |
| | 8 8 | Из города в тот же самый город искать путь необходимости нет. Введите новую пару чисел: | Верно |
| 1 2<br>2 3<br>t 5 | - | На вход поступили не корректные данные. Пожалуйста, исправьте файл входных данных! | Верно |
| Пустой файл | - | На вход поступили не корректные данные. Пожалуйста, исправьте файл входных данных! | Верно |

## ПРИЛОЖЕНИЕ Б

### Изменения кода

При тестировании программы была выявлена следующая ошибка: если в графе имеется больше трёх вершин, и хотя бы 2 инцидентны друг другу, а вводится пара городов, между которыми пути нет, то программа зацикливалась, переходя из одной вершины в другую и наоборот. Например для входных данных:1 2, 2 3, 4 5, граф представлен на рисунке 1:

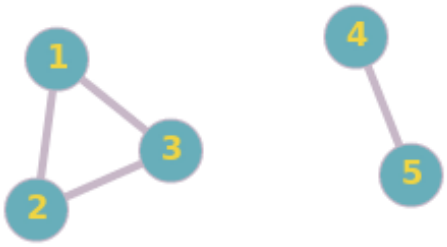


Рисунок 1

при поиске пути из вершины 1 в вершину 4 программа “прыгала” по вершинам 1, 2, 3.

Решением проблемы послужило добавление массива меток. К каждой вершине соответствовала метка, в начале работы алгоритмы равная 9999. После просмотра вершины метка приравнивалась к нулю, тем самым программа не возвращалась к просмотренным вершинам.

# ПРИЛОЖЕНИЕ В
## Исходный код программы

```
#include <iostream>
#include <fstream>
#include <cstdio>

using namespace std;

bool wave (int** mtrx, int n, int start, int finish, int *old_front,
           int* new_front, int* marker, bool way=false) {

    int count = 0;

    for (int i = 0; i < n; i++) {
        if (old_front[i] != -1) {
            cout << "Поиск путей из города " << old_front[i] << endl;
            for (int j = 0; j < n; j++) {
                if (mtrx[old_front[i]-1][j] != 0 && marker[j] != 0)
                {
                    marker[j] = 0;
                    new_front[j] = j+1;
                    cout << "Найден путь в город " << new_front[j] << endl;
                    if (new_front[j] == finish)
                        return true;
                }
            }
        }
    }

    for (int i = 0; i < n; i++) {
        if (new_front[i] != -1)
            count++;
    }

    if (count == 0) {
        return way;
    }

    for (int y = 0; y < n; y++) {
        old_front[y] = new_front[y];
        if (new_front[y]!=-1) {

        }
        new_front[y] = -1;
    }

        if(!way)
            way = wave(mtrx, n, start, finish, old_front, new_front, marker,
way);

}

void preparation(int** mtrx, int n) {

    double start = 0;
    double finish = 0;
    int *old_front = new int[n];
    int *new_front = new int[n];
    int *marker = new int[n];
```

```
    int question = 1;

    while (question == 1) {
        cout << "Введите пару городов, между которыми требуется установить
наличие пути:";
        cin >> start >> finish;
        cout << endl;

        while ((start > n) || (finish > n) || (start == 0) || (finish == 0) ||
(start == finish) ||
               ((start!=int(start))) || ((finish!=int(finish)))) {
            if (start == finish) {
                cout << "Из города в тот же самый город искать путь
необходимости нет. Введите новую пару чисел:" << endl;
                cin >> start >> finish;
                cout << endl;
            }
            else {
                cout << "Не корректно указаны номера городов. Пожалуйста введите
пару натуральных чисел, не"
                        "превосходящих " << n << ":";
                cin >> start >> finish;
                cout << endl;
            }
        }

        for (int i = 0; i < n; i++) {
            marker[i] = 9999;
            new_front[i] = -1;
            old_front [i] = -1;
        }

        old_front[0] = start;
        marker[int(start)-1] = 0;

       bool ch = wave(mtrx, n, start, finish, old_front, new_front, marker);

       if (ch) {
           cout << "Между городами существует путь!" << endl << endl;
       }
       else {
           cout << "Между городами пути нет!" << endl << endl;
       }
       cout << "Найти новый путь между городами?" << endl << "1. Да" << endl <<
"2. Нет" << endl;
       cin >> question;
       cout << endl;
    }

    delete [] old_front;
    delete [] new_front;
    delete [] marker;
}

int main()
{
    setlocale(LC_ALL, "rus");

    int i = 0,j = 0;
    double mx = 0;
    int n = 0;
```

```
    int k = 0;

    ifstream input("input.txt", ios_base::in);

        if (!input.is_open()) // если файл не открыт
        {
            cerr << "Ошибка открытия файла!\n";
            exit(1);

        }

        while (input >> mx) {
            k++;
            if ((mx!=int(mx))) {      //проверка на целочисленный тип данных
                cerr << "На вход поступили не корректные данные. Пожалуйста,
исправьте файл входных данных!" << endl;
                exit(1);
            }
            if (mx > n)
                n = mx;
        }
        if (k == 0) {
            cerr << "На вход поступили не корректные данные. Пожалуйста,
исправьте файл входных данных!" << endl;
            exit(1);
        }

        if (k%2 == 1) {
            cerr << "На вход поступили не корректные данные. Пожалуйста,
исправьте файл входных данных!" << endl;
            exit(1);
        }

        input.clear();
        input.seekg (0); //перемещение каретки в начало файла

        int **mtrx = new int*[n];
        for (int count = 0; count < n; count++) {
            mtrx[count] = new int[n];
        }

        for (int k = 0; k<n; k++)
            for (int m = 0; m<n; m++)
                mtrx[k][m] = 0;

        while (input >> i && input >> j) {
            mtrx[i-1][j-1] = 1;
            mtrx[j-1][i-1] = 1;
        }

        input.close();

        cout << "Количество городов: " << n << endl;

        preparation (mtrx, n);

        for (int count = 0; count < n; count++)
                delete [] mtrx[count];
        delete [] mtrx;
```

```
    return 0;
}
```