

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

**ОТЧЕТ**  
**по лабораторной работе №1**  
**по дисциплине «Построение и анализ алгоритмов»**  
**Тема: Поиск с возвратом**

Студентка гр. 7383

\_\_\_\_\_

Прокопенко Н.

Преподаватель

\_\_\_\_\_

Жангиров Т. Р.

Санкт-Петербург

2019

## Цель работы

Ознакомиться с алгоритмом поиска с возвратом, написать программу для квадрирования квадрата с заданной стороной с использованием поиска с возвратом.

## Реализация задачи

В данной работе для решения поставленной цели был написан класс `Mass` и несколько методов, содержащихся в данном классе.

Конструктор класса создает двумерный массив целых чисел, заполняет его нулями. Так же был написан конструктор копирования.

Метод `void squares(int l)` заполняет квадрат тремя стандартными квадратами и, если сторона четной длины, заполняет четвертым квадратом. Если длина стороны четная, первый квадрат имеет сторону в  $1/2$  от заданной, если кратна 3, то квадрат имеет сторону  $2/3$  от заданной, если кратна 5 – то  $3/5$  от исходной длины. Остальные два квадрата ставятся в соседние углы.

Метод `int size_of_squar(int x, int y)` считает длину стороны вставленного квадрата и возвращает ее.

Метод `bool find_s(int &x, int &y)` возвращает `true` если находит пустое место, иначе – `false`.

Метод `bool insert(int x, int y, int size)` возвращает `true` если можно и вставил квадрат заданной длины, иначе – `false`.

Метод `void delet(int x, int y, int size)` удаляет квадрат с заданной длиной.

Метод `void backtracking(Mass &best, int size, int x, int y, int &best_numbers)` основная рекурсивная функция, которая проверяет какое минимальное количество квадратов поместится в данный квадрат при помощи поиска с возвратом.

Метод `void print()` выводит ответ на экран.

Исходный код программы представлен в Приложении А.

## Тестирование

Программа собрана в операционной системе Ubuntu 17.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

Так же было проведено исследование алгоритма. Было изучено необходимое количество итераций при некоторых длинах квадрата. Результаты исследования представлены в табл. 1. В исследовании были использованы простые числа.

Таблица 1 – Исследование алгоритма

Длина стороны квадрата	Кол-во итераций
3	3
5	14
7	60
11	950
13	2370
17	16578
19	65461
23	250838
29	2046067

Из результатов исследования алгоритмов видно, что сложность алгоритма не превышает  $2^n$ , где  $n$  – сторона квадрата. Память алгоритм выделяет только для двух двумерных массивов, поэтому по памяти сложность константная.

## Выводы

В ходе выполнения лабораторной работы был изучен метод поиска с возвратом, была написана программа для квадрирования квадрата заданной длины и исследован алгоритм.

## ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ

### lab1.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <math.h>
#include <iomanip>
using namespace std;

class Mass{
private:
    int **Mas;
    int d;
    int sum;
public:
    Mass(int i): d(i), sum(3){
        Mas = new int *[d];
        for(int i = 0; i < d; i++){
            Mas[i] = new int[d];
            for(int i = 0; i < d; i++){
                for(int j = 0; j < d; j++){
                    Mas[i][j] = 0;
                }
            }
        }
    }
    Mass(const Mass &N){
        d = N.d;
        sum = N.sum;
        Mas = new int*[d];
        for (int i = 0; i < d; i++){
            Mas[i] = new int[d];
            for (int i = 0; i < d; i++){
                for (int j = 0; j < d; j++){
                    Mas[i][j] = N.Mas[i][j];
                }
            }
        }
    }
    Mass &operator = (Mass const &origin){
        if(this != &origin){
            Mass temp(origin);
            d = temp.d;
            sum = temp.sum;
            for(int i = 0; i < d; i++){
                for(int j = 0; j < d; j++){
                    Mas[i][j] = origin.Mas[i][j];
                }
            }
        }
        return *this;
    }
    ~Mass(){
        for (int i = 0; i < d; i++)
```

```

        delete[] Mas[i];
        delete[] Mas;
    }
void squares(int l){
    for(int i = 0; i < l; i++){
        for(int j = 0; j < l; j++){
            Mas[i][j] = 1;
        }
        for(int i = 0; i < d - 1; i++){
            for(int j = 1; j < d; j++){
                Mas[i][j] = 2;
            }
        }
        for(int i = 1; i < d; i++){
            for(int j = 0; j < d - 1; j++){
                Mas[i][j] = 3;
            }
        }
        if(d % 2 == 0){
            for(int i = 1; i < d; i++){
                for(int j = 1; j < d; j++){
                    Mas[i][j] = 4;
                }
            }
            sum++;
        }
    }
}
bool find_s(int &x, int &y){
    for(int i = d - 1; i >= d / 2; i--){
        for(int j = d - 1; j >= d / 2; j--){
            if(Mas[i][j] == 0){
                x = j;
                y = i;
                return true;
            }
        }
    }
    return false;
}
void delet(int x, int y, int size){
    for(int i = y; i > (y - size) && (Mas[i][x] == sum); i--){
        for(int j = x; j > (x - size) && (Mas[i][j] == sum); j--){
            Mas[i][j] = 0;
        }
        sum--;
    }
}
bool insert(int x, int y, int size){
    bool empty_space = true;
    if((x - size) < 0 || (y - size) < 0)
        return false;
    for(int i = y; i > y - size; i--){
        for(int j = x; j > x - size; j--){
            if(Mas[i][j] != 0)
                return empty_space = false;
        }
    }
}

```

```

    }

    sum++;
    for(int i = y; i > y - size; i--)
        for(int j = x; j > x - size ; j--)
            Mas[i][j] = sum;
    return empty_space;
}
int size_of_squar(int x, int y){
    int count = 0;
    int check = Mas[y][x];
    for(int j = x; j < d; j++){
        if(Mas[y][j] != check)
            return count;
        else
            count++;
    }
    return count;
}

void backtracking(Mass &best, int size, int x, int y, int
&best_numbers ){
    if(sum >= best_numbers)
        return;
    for(int i = size; i > 0; i--){
        if(insert(x, y, i)){
            int x1 = x;
            int y1 = y;
            if(find_s(x1, y1)){
                backtracking(best, size, x1, y1, best_numbers);
            }
            else{
                if(sum < best_numbers){
                    best_numbers = sum;
                    best = *this;
                }
                delet(x, y, size);
                return;
            }
            delet(x, y, size);
        }
    }
}

}
void print(){
    vector<int> check;
    int count = 0;
    for(int i = 0; i < d; i++){
        for(int j = 0; j < d; j++){
            if(find(check.begin(), check.end(), Mas[i][j]) ==
check.end()){
                check.push_back(Mas[i][j]);

```

```

        cout <<setw(2)<<right<< i + 1 <<setw(5)<<right<< j +
1 <<setw(5)<<right<< size_of_squar(j, i) << endl;
        count = 0;
    }
}
};
int main(){
    int size, k;
    cout <<" Введите длину квадрата: ";
    cin >> size;
    int best_numbers = pow(size, 2);
    Mass mas(size);
    Mass best(size);
    if(size % 2 == 0){
        k = size / 2;
        mas.squares(k);
        best = mas;
        best_numbers = 4;
    }
    else{
        if(size % 3 == 0)
            k = size * 2 / 3;
        else if(size % 5 == 0)
            k = size * 3 / 5;
        else
            k = (size + 1) / 2;
        mas.squares(k);
        mas.backtracking(best, size - k, size - 1, size - 1,
best_numbers);
    }
    cout <<" Минимальное число квадратов: ";
    cout << best_numbers << endl;
    best.print();
    cout << endl;
    return 0;
}

```

## ПРИЛОЖЕНИЕ Б.

### ТЕСТОВЫЕ СЛУЧАИ

Результаты тестов представлены на рис. 1-3.

```
Введите длину квадрата: 4
Минимальное число квадратов: 4
1      1      2
1      3      2
3      1      2
3      3      2
```

Рисунок 1 – Тест №1

```
Введите длину квадрата: 37
Минимальное число квадратов: 15
1      1      19
1      20     18
19     20      2
19     22      5
19     27      4
19     31      7
20     1      18
20     19      1
21     19      3
23     27      1
23     28      3
24     19      7
24     26      2
26     26     12
31     19      7
```

Рисунок 2 – Тест №2

```
Введите длину квадрата: 15
Минимальное число квадратов: 6
1      1      10
1      11      5
6      11      5
11     1       5
11     6       5
11     11      5
```

Рисунок 3 – Тест №3



