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

Студент гр. 7383 _____ Рудоман В.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Содержание

Цель работы

Ознакомиться с алгоритмами поиска с возвратом, создать программу, которая квадрирует квадрат с заданной стороной, использующую метод поиска с возвратом.

Формулировка задачи

Разбить квадрат со стороной N на минимально возможное число квадратов со сторонами от 1 до N-1. Внутри квадрата не должно быть пустот, квадраты не должны перекрывать друг друга и выходить за пределы основного квадрата. Программа должна вывести количество квадратов, а также координаты левого верхнего угла и размеры стороны каждого квадрата.

Реализация задачи

Для реализации задачи были созданы два двумерных массива целых чисел и инициализированы нулями.

Метод int space(int** Sqr, int N, int* x, int *y)

Функция поиска свободного места в квадрате.

Метод int length(int** Sqr, int N, int x, int y)

Функция поиска длины стороны вставляемого квадрата.

Метод void full(int** Sqr, int L, int x, int y, int color)

Функция вставки квадрата.

Метод int find_full(int** Sqr, int N, int* x, int *y)

Функция поиска вставляемого квадрата.

Метод void rec(int** Sqr, int** minSqr, int N, int color, int* min)

Рекурсивная функция (алгоритм поиска с возвратом)

Исходный код программы представлен в приложении Б.

Исследование алгоритма

Было принято исследовать сложность алгоритма по количеству вызовов функции void rec. Длина стороны поля – простое число. Количество итераций для некоторых простых чисел приведено в таблице ниже.

Таблица 1 – Количество итераций алгоритма поиска с возвратом.

Размер стороны квадрата	Количество итераций
3	4
5	17
7	71
11	1197
13	3025
17	21467
19	64352

Из таблицы видно, что сложность алгоритма не превышает 2^N , где N – длина стороны квадрата.

Тестирование

1. Процесс тестирования

Программа собрана в операционной системе macOS Sierra компилятором g++. В других ОС и компиляторах тестирование не проводилось.

2. Результаты тестирования

В результате тестирования не было обнаружено ошибок, приводящих к некорректным результатам на некоторых исходных данных. Тестовые случаи представлены в приложении А.

Вывод

В результате выполнения данной работы был изучен алгоритм поиска с возвратом. Была написана программа, применяющая данный метод для поиска разбиения квадрата на минимальное возможное число меньших квадратов. Также была исследована сложность алгоритма.

ПРИЛОЖЕНИЕ А. ТЕСТОВЫЕ СЛУЧАИ

Результаты тестирования приведены в таблице 2.

Таблица 2 – Результаты тестирования.

Размер стороны квадрата	Результат
5	8 1 1 3 4 1 2 4 3 1 5 3 1 1 4 2 3 4 2 5 4 1 5 5 1
7	9 1 1 4 5 1 3 5 4 1 6 4 2 1 5 3 4 5 2 6 6 2 4 7 1 5 7 1
13	11 1 1 7 8 1 6 8 7 1 9 7 3 12 7 2 1 8 6 7 8 2 12 9 2 7 10 4 11 10 1 11 11 3
27	10 1 1 15 16 1 12 16 13 3 19 13 9 1 16 12 13 16 6 13 22 6 19 22 6 25 22 3 25 25 3

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД

```
#include <iostream>

using namespace std;

int q = 0;

int space(int** Sqr, int N, int* x, int *y) {    //функция поиска свободного места
    for(*x = 0; *x < N; (*x)++) {
        for(*y = 0; *y < N; (*y)++) {
            if(Sqr[*x][*y] == 0)
                return 0;
        }
    }
    return 1;
}

int length(int** Sqr, int N, int x, int y) {    //функция поиска длины стороны вставляемого квадрата
    int L = 0;
    for(; ((L + x) < N) && ((L + y) < N); L++) {
        if(Sqr[x + L][y] != 0 || Sqr[x][y + L] != 0) {
            break;
        }
    }
    return L;
}

void full(int** Sqr, int L, int x, int y, int color) { //функция вставки квадрата
    for (int a = 0; a < L; a++)
        for (int b = 0; b < L; b++)
            Sqr[x+a][y+b] = color;
}

int find_full(int** Sqr, int N, int* x, int *y){ //функция поиска вставленного квадрата
    for(*y = 0; *y < N; (*y)++)  {
        for(*x = 0; *x < N; (*x)++){
            if(Sqr[*x][*y] != 0)
                return 0;
        }
    }
```

```
    return 1;
}

void rec(int** Sqr, int** minSqr, int N, int color, int* min) {
//рекурсивная функция
    q++;
    if(color > *min)               //если количество цветов в текущей
раскраске уже превышает минимальную,
        return;                 //то возвращаемся назад
    int x,y;
    int Ney1 = space(Sqr, N, &x, &y);

    if( Ney1 == 0) {
        int l = length(Sqr, N, x, y);
        if (l == N)
            l--;
        for(; l > 0; l--) {         //цикл, чтобы при разматывании
поставить квадрат меньше и найти другое решение
            full(Sqr, l, x, y, l);
            rec(Sqr, minSqr, N, color+1, min);
            full(Sqr, l, x, y, 0);
        }
    }

    else {
        if(color < *min) {  //запоминание минимальной раскраски
            for(x = 0; x < N; x++) {
                for(y =0; y < N; y++) {

                    minSqr[x][y] = Sqr[x][y];
                }
            }
            *min = color;
        }
    }

}

int main() {
    int N;
    cin >> N;
    int answer;
    int zoom = 1; //переменная для масштабирования
        int** Square;
        int** miniSquare;
```

```
    if(N % 2 != 0){ //если длина квадрата нечетная
        if(N%3 == 0 && N!= 3)
            zoom = 3;
        if(N%5 == 0 && N!= 5)
            zoom = 5;

        int new_N = N/zoom; //длина стороны квадрата с учетом масштаба
     Square = new int*[N/zoom];                //создание массивов для
хранения текущего и минимального решения
          miniSquare = new int*[N/zoom];

        for(int i = 0; i < N/zoom; i++) {
           Square[i] = new int[N/zoom];
           miniSquare[i] = new int[N/zoom];
     }

     for(int x = 0; x < N/zoom; x++) {          //заполнение массивов
нулями
          for(int y = 0; y < N/zoom; y++) {
           miniSquare[x][y] = 0;
           Square[x][y] = 0;
          }
     }

        full(Square,new_N/2+1,0,0,new_N/2+1);              //заполнение
первыми тремя квадратами для минмального решения
        full(Square,new_N/2,new_N/2+1,0,new_N/2);
        full(Square,new_N/2,0,new_N/2+1,new_N/2);

        answer = 2*N+1;                     // максимальное количество
квадратов + 1

        rec(Square, miniSquare, N/zoom, 3, &answer);
        int x = 0, y =0;
        cout << answer << endl;

     while(find_full(miniSquare,N/zoom,&x,&y) == 0){  //вывод ответа в
формате координаты и длина
        int tmp = miniSquare[x][y];
        cout << x*zoom +1<< ' ' << y*zoom +1<< ' ' << tmp*zoom <<
endl;
        full(miniSquare, tmp,x, y, 0);

     }
```

```
     for(int i = 0; i < N/zoom; i++) {
            delete [] Square[i];
            delete [] miniSquare[i];
     }
        delete [] Square;
        delete [] miniSquare;

    }

    else{                                //если длина стороны четная
       cout << 4 <<endl;
       cout << 1 << ' ' << 1 << ' ' << N/2 << endl;
       cout << N/2 +1 << ' ' << 1 << ' ' << N/2 << endl;
       cout << 1 << ' ' << N/2 +1<< ' ' << N/2 << endl;
       cout << N/2+1 << ' ' << N/2+1 << ' ' << N/2 << endl;

    }

   printf("%d\n", q );
    return 0;
}
```