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 _____ Тян Е.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ

Цель работы: исследовать и реализовать задачу квадрирования квадрата.

Формулировка задачи: необходимо разработать программу, которая решает задачу разбиения квадрата на минимальное число квадратов размером от 1 до *N*-1. При заполнении не должно оставаться пустот, обрезки не должны выходить за пределы изначального квадрата и не должны перекрываться.

Входные данные: целое число, которое задает размер квадрата, который нужно разбить на более мелкие. Вывод должен включать число, задающее минимальное количество обрезков, из которых можно построить квадрат заданного размера. Далее должны идти строки, содержащие три целых числа, задающих координаты левого верхнего угла и длину стороны соответствующего обрезка.

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

В данной работе используются главная функция (int main( )), дополнительные функции(point free_cell( ), bool check_edge(point p, int side), int mx_free_square(point p), void fill_sqr(int used, int squares), void even(int n), int is_prime(int n)).

Параметры, передаваемые в функцию bool check_edge(point p, int side):

- p – координаты точки;
- side – длина стороны квадрата.

Параметры, передаваемые в функцию int max_free_square(point p):

- p – координаты точки.

Параметры, передаваемые в функцию void fill_sqr(int x, int y, int side, int val):

- x – значение координаты x;
- y – значение координаты y;
- side – длины стороны квадрата;
- val – значение, которым нужно заполнить квадрат.

Параметры, передаваемые в функцию void func(int used, int squares):

- used – количество закрашенных клеточек;
- squares – количество квадратов.

Параметры, передаваемые в функцию void even(int n):

- n – длина стороны изначального квадрата.

Параметры, передаваемые в функцию void is_prime(int n):

- n – длина стороны изначального квадрата.

В функции main() считывается значение длины стороны квадрата, который требуется разрезать. Проходит проверка: если сторона квадрата четной длины, то вызывается функция void even(int n), в противном случае от квадрата отрезаются один квадрат равный N*(p+1)/(2*p), где p – минимальный простой делитель числа N, равного длине стороны квадрата, и два квадрата меньшей длины такой, что эти квадраты можно поставить рядом с квадратом с длиной стороны N*(p+1)/(2*p). Чтобы отрезать квадраты был

создан булевый массив, где хранятся нули, если место свободно, в противном случае хранятся единицы. Для отрезания трех квадратов вызывается функция void fill_sqr(int x, int y, int side, int val). Также для запоминания значений координат и длины стороны данного квадрата был создан вектор vector <square> buffer. Далее вызывается функция void func(int used, int squares), где происходит деление оставшейся части квадрата на более мелкие квадраты с использованием алгоритма поиска с возвратом.

В функции void func(int used, int squares) происходит проверка количества закрашенных клеток, т.е. обращается внимание на количество отрезанных клеток, если данное количество равно количеству клеток в изначальном квадрате, значит, что все клетки закрашены и нет смысла пытаться разбить квадрат дальше. Проверяется, является ли количество квадратов, на которое разбит изначальный квадрат меньше, чем лучший результат разбиения квадрата, если это так, то данный результат запоминается как лучший, в противном случае просто происходит возврат. Есть проверка на то, является ли нынешнее количество квадратов больше чем лучший результат, если это так, то нет смысла дальше разбивать квадрат и происходит возврат. Изначально ищется первая пустая клетка, далее определяется максимальная доступная длина для квадрата, который необходимо отрезать. Данный квадрат отрезается, булевый массив заполняется единицами, координаты и длина стороны квадрата запоминаются, происходит рекурсивный вызов функции void func(int used, int squares), где в качестве параметров used и squares передаются значения увеличенные на количество занятых клеток и единицу соответственно.

Когда был найден наилучший случай разбиения квадрата, все координаты и длины сторон выводятся на экран.

3. ТЕСТИРОВАНИЕ

Программа была собрана в компиляторе G++ в macOS 10.14. Программа может быть скомпилирована с помощью команды:

g++ <имя файла>.cpp

Тестовые случаи представлены в Приложении А.

Исходя из тестовых случаев можно увидеть, что тестовые случаи не выявили некорректной работы программы, что говорит о том, что по результатам тестирования было показано, что поставленная задача была выполнена.

4. ИССЛЕДОВАНИЕ

Компиляция была произведена в компиляторе в g++ Version 4.2.1. В других ОС и компиляторах тестирование не проводилось.

Для проведения исследования устанавливалась сложность алгоритма по количеству осуществления операции обрезания квадрата.

Результаты, полученные в следствии исследования сложности алгоритма, приведены в табл. 1, где можно заметить, что сложность алгоритма экспоненциальная.

Таблица 1 — Результаты исследования зависимости числа операций вставки квадрата от размера заданного квадрата

Заданный размер	Количество совершенных операций
2	4
3	6
5	15
7	40
11	383
13	835
17	4636
19	12248
23	45112
29	306212
31	695903
37	3483105

5. ВЫВОД

В ходе выполнения лабораторной работы была решена задача нахождения минимального разбиения квадрата на более мелкие квадраты на языке C++ с использованием алгоритма поиска с возвратом.

Была исследована сложность алгоритма. Исследования показали, что она экспоненциальная.

ПРИЛОЖЕНИЕ А. ТЕСТОВЫЕ СЛУЧАИ

№	Ввод	Вывод
1	4	4 1 1 2 1 3 2 3 1 2 3 3 2
2	13	11 1 1 7 1 8 6 8 1 6 7 8 2 7 10 4 8 7 1 9 7 3 11 10 1 11 11 3 12 7 2 12 9 2
3	25	8 1 1 15 1 16 10 16 1 10 11 16 10 16 11 5 21 11 5 21 16 5 21 21 5

4	31	15 1 1 16 1 17 15 17 1 15 16 17 3 16 20 6 16 26 6 17 16 1 18 16 1 19 16 4 22 20 1 22 21 1 22 22 10 23 16 6 29 16 3 29 19 3
5	37	15 1 1 19 1 20 18 20 1 18 19 20 2 19 22 5 19 27 11 20 19 1 21 19 3 24 19 8 30 27 3 30 30 8 32 19 6 32 25 1 32 26 1 33 25 5

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp:

```
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

struct point {
    int x, y;
    point(int _x, int _y) {
        x = _x; y = _y;
    }
};

struct square {
    int x, y, side;
    square(int _x, int _y, int _side) {
        x = _x; y = _y; side = _side;
    }
};

int n;
int best_result;
bool a[50][50];
vector<square> buffer, answer;

point free_cell() {
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= n; ++j)
            if (!a[i][j])
                return point(i, j);
    return point(0, 0);
}

bool check_edge(point p, int side) {
    if (p.x + side - 1 > n || p.y + side - 1 > n)
        return false;
    for (int i = 0; i <= side - 1; ++i)
        if (a[p.x + side - 1][p.y + i] || a[p.x + i][p.y + side - 1])
            return false;
    return true;
}

int max_free_square(point p) {
    int max_side = 1;
    while (check_edge(p, max_side + 1) and max_side < n - 1)
        max_side++;
```

```
    return max_side;
}

void fill_sqr(int x, int y, int side, int val) {
    for (int i = x; i < x + side; ++i)
        for (int j = y; j < y + side; ++j)
            a[i][j] = val;
}

void func(int used, int squares) {
    if (used == n * n) {
        if (squares < best_result) {
            best_result = squares;
            answer.clear();
            for (int i = 0; i < buffer.size(); ++i)
                answer.push_back(buffer[i]);
        }
        return;
    }
    if (squares + 1 >= best_result) return;
    point p = free_cell();
    int max_side = min(max_free_square(p), n - 1);
    for (int i = max_side; i >= 1; --i) {
        fill_sqr(p.x, p.y, i, 1);
        buffer.push_back(square(p.x, p.y, i));
        func(used + i*i, squares + 1);
        buffer.pop_back();
        fill_sqr(p.x, p.y, i, 0);
    }
}

void even(int n) {
    cout << 4 << endl;
    int m = n / 2;
    cout << 1 << " " << 1 << " " << m << endl;
    cout << 1 << " " << m + 1 << " " << m << endl;
    cout << m + 1 << " " << 1 << " " << m << endl;
    cout << m + 1 << " " << m + 1 << " " << m << endl;
}

int is_prime(int n) {
    int min_div=1;
    for(int i=3;i<=sqrt(n);){
        if((n % i == 0)){
            min_div=i;
            break;
        }else
            ++i;
    }
    int sqr;
```

```
    if(min_div == 1)
        sqr=(n+1)/2;
    else
        sqr=(n/min_div)*(min_div+1)/2;
    return sqr;
}

int main() {
    ios::sync_with_stdio(false);
    cin >> n;
    if (n % 2 == 0){
        even(n);
        return 0;
    }else{
        best_result = n * n;
        int m1 = is_prime(n);
        int m2 = n - m1;
        int quantity=3;
        int squares=m1 * m1 + 2 * m2 * m2;
        fill_sqr(1, 1, m1, 1); buffer.push_back(square(1, 1, m1));
        fill_sqr(1, m1 + 1, m2, 1); buffer.push_back(square(1, m1 + 1, m2));
        fill_sqr(m1 + 1, 1, m2, 1); buffer.push_back(square(m1 + 1, 1, m2));
        func(squares, quantity);
    }
    cout << answer.size() << endl;
    for (int i = 0; i < answer.size(); ++i) {
        square s = answer[i];
        cout << s.x << " " << s.y << " " << s.side << endl;
    }
    return 0;
}
```