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 _____ Ласковенко Е.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Содержание

Цель работы

Ознакомиться с алгоритмом 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 размер стороны каждого квадрата.

Реализация задачи

Для реализации поставленной задачи был создан класс `Square`.

Конструктор класса инициализирует поле размера стороны квадрата, далее создает два двухмерных массива целых чисел и инициализирует их нулями.

Метод `void framing()` в зависимости от размера стороны квадрата вызывает соответствующие методы для реализации поиска с возвратом, содержащиеся в private поле.

Метод `Point find_NoClrLeft()` находит в двухмерном массиве пустой элемент, который является верхним левым углом вставляемого квадрата.

Метод `void framing_odd()` необходим для запуска алгоритма бэктрэкинга и вызова метода вывода координат.

Метод `void framing_even()` необходим для квадрирования квадратов с четной длиной стороны.

Метод `void framing_3_5(unsigned k, unsigned fl)` необходим для квадрирования квадратов с длиной стороны, кратной 3 или 5.

Метод `unsigned b_track(unsigned count, unsigned& min, unsigned c=0)` реализует поиск с возвратом, заполняет двухмерный массив с результатом.

Метод `bool is_fit(Point p, unsigned sz)` возвращает значение 1, если квадрат с заданной длиной стороны может поместиться в заданную область.

Метод `void set_main_sq()` добавляет в массив первые 3 квадрата. Если длина стороны четная, первый квадрат имеет сторону в 1/2 от исходной, если она кратна 3, то в 2/3, если кратна 5, то в 3/5 от исходной длины. В остальных случаях квадрат имеет сторону в 1/2 от исходной длины плюс 1. Второй и третий квадраты добавляются в соседние с первым квадратом углы и имеют максимальный возможный размер.

Метод `void print_coordinates(unsigned k=1)` выводит на экран координаты квадратов из массива с результатом.

Метод `void delete_fr_sq(Point st, unsigned sz)` удаляет квадрат из массива с заданной стороной и в заданной области.

Исходный код программы представлен в приложении Б.

Исследование алгоритма

Было принято исследовать сложность алгоритма по количеству вызовов функции, добавляющей квадрат на поле. 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	3
5	15
7	52
11	705
13	1603
17	9961
19	28263
23	105687
29	733266
31	1746937
37	8463487
41	28047082

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

Тестирование

1. Процесс тестирования

Программа собрана в операционной системе Linux Mint 18 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 разбиение квадрата на минимально 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
8	4 0 0 4 0 4 4 4 0 4 4 4 4
27	3 6 0 0 18 0 18 9 9 18 9 18 0 9 18 9 9 18 18 915 8 0 0 21 0 21 14 14 21 14 21 0 14 21 14 7 28 14 7 28 21 7 28 28 7
35	8 1 1 21 22 1 14

	1 22 14 15 22 14 22 15 7 29 15 7 29 22 7 29 29 71
37	8463487 15 0 0 19 0 19 18 18 19 2 18 21 5 18 26 11 19 0 18 19 18 1 20 18 3 23 18 8 29 26 3 29 29 8 31 18 6 31 24 1 31 25 1 32 24 5

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД

```
#include <iostream>

using namespace std;

struct Point
{
    unsigned i;
    unsigned j;
};

class Square
{
private:
    unsigned size;
    unsigned** arr;
    unsigned** framing_arr;

private:
    void framing_odd()
    {
        this->set_main_sq();
        unsigned min = size*size - size - (size+1)/2;
        cout << this->b_track(3, min) << endl;
        cout << min << endl;
        print_coordinates();
    }

    void framing_even()
    {
        for(unsigned i=0; i<size/2; ++i)
        {
            for(unsigned j=0; j<size/2; ++j)
                framing_arr[i][j] = size/2;
        }

        for(unsigned i=size/2; i<size; ++i)
        {
            for(unsigned j=0; j<size/2; ++j)
                framing_arr[i][j] = size/2;
```

```
            }

            for(unsigned i=0; i<size/2; ++i)
            {
                for(unsigned j=size/2; j<size; ++j)
                    framing_arr[i][j] = size/2;
            }

            for(unsigned i=size/2; i<size; ++i)
            {
                for(unsigned j=size/2; j<size; ++j)
                    framing_arr[i][j] = size/2;
            }

            cout << 4 << endl;
            print_coordinates();
        }

        void framing_3_5(unsigned k, unsigned fl)
        {
            Square tmp(fl);

            tmp.set_main_sq();
                unsigned min = tmp.size*tmp.size - tmp.size -
(tmp.size+1)/2 + 3;
            cout << tmp.b_track(3, min) << endl;
            cout << min << endl;
            tmp.print_coordinates(k);
        }

             unsigned b_track(unsigned count, unsigned& min,
unsigned c=0)
        {
            if(min <= count)
                return c;

            Point tmp = this->find_NoClrLeft();
            if(tmp.i == size && tmp.j == size)
            {
                if(min > count)
                {
                    min = count;
```

```
                for(unsigned j=0; j<size; ++j)
                {
                    for(unsigned k=0; k<size; ++k)
                        framing_arr[j][k] = arr[j][k];
                }
            }

            return c;
        }

        for(unsigned i=size/2; i>0; --i)
        {
            if(is_fit(tmp, i))
            {
                count++;
                for(unsigned j=tmp.i; j<tmp.i+i; ++j)
                {
                    for(unsigned k=tmp.j; k<tmp.j+i; ++k)
                        arr[j][k] = i;
                }
                c++;

                c = b_track(count, min, c);

                count--;
                for(unsigned j=tmp.i; j<tmp.i+i; ++j)
                {
                    for(unsigned k=tmp.j; k<tmp.j+i; ++k)
                        arr[j][k] = 0;
                }
            }
        }
        return c;
    }

    bool is_fit(Point p, unsigned sz)
    {
        for(unsigned i=p.i; i<p.i+sz; ++i)
        {
            for(unsigned j=p.j; j<p.j+sz; ++j)
            {
                if(i==size || j==size || arr[i][j])
```

```
                    return false;
                }
            }

            return true;
        }

        void set_main_sq()
        {
            for(unsigned i=0; i<(size+1)/2; ++i)
            {
                for(unsigned j=0; j<(size+1)/2; ++j)
                    arr[i][j] = (size+1)/2;
            }

            for(unsigned i=size-1; i>size/2; --i)
            {
                for(unsigned j=0; j<size/2; ++j)
                    arr[i][j] = size/2;
            }

            for(unsigned i=0; i<size/2; ++i)
            {
                for(unsigned j=size-1; j>size/2; --j)
                    arr[i][j] = size/2;
            }
        }

        void print_coordinates(unsigned k=1)
        {
            for(unsigned i=0; i<size; ++i)
            {
                for(unsigned j=0; j<size; ++j)
                {
                    if(framing_arr[i][j])
                    {
                            cout << i*k << ' ' << j*k << ' ' <<
k*framing_arr[i][j] << endl;
                             delete_fr_sq({i, j}, framing_arr[i]
[j]);
                    }
                }
```

```
        }
    }

    void delete_fr_sq(Point st, unsigned sz)
    {
        for(unsigned j=st.i; j<st.i+sz; ++j)
        {
            for(unsigned k=st.j; k<st.j+sz; ++k)
                framing_arr[j][k] = 0;
        }
    }

public:
    Square(unsigned sz)
    {
        size = sz;
        arr = new unsigned*[size];
        for(unsigned i=0; i<size; ++i)
        {
            arr[i] = new unsigned[size];
            for(unsigned j=0; j<size; ++j)
                arr[i][j] = 0;
        }
        framing_arr = new unsigned*[size];
        for(unsigned i=0; i<size; ++i)
        {
            framing_arr[i] = new unsigned[size];
            for(unsigned j=0; j<size; ++j)
                framing_arr[i][j] = 0;
        }
    }

    void framing()
    {
        if(!(size%2))
            framing_even();
        else if(!(size%3))
            framing_3_5(size/3, 3);
        else if(!(size%5))
            framing_3_5(size/5, 5);
        else
            framing_odd();
```

```
    }

    Point find_NoClrLeft()
    {
        for(unsigned i=size/2; i<size; ++i)
        {
            for(unsigned j=size/2; j<size; ++j)
            {
                if(!arr[i][j])
                    return {i, j};
            }
        }

        return {size, size};
    }

    ~Square()
    {
        for(unsigned i=0; i<size; ++i)
        {
            delete arr[i];
            delete framing_arr[i];
        }
        size = 0;
    }
};


int main()
{
    unsigned N;
    cin >> N;
    Square a(N);
    a.framing();

    return 0;
}
```