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. 8304 _____ Мельникова О.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2020

Цель работы.

С помощью алгоритма поиска с возвратом реализовать заполнение квадрата N×N минимальным количеством квадратов.

Вариант 1и.

Выполнение работы.

Для хранения частичных решений был использован массив типа **square** (класс, включающий координаты левого верхнего угла и длину его стороны). Массив заполняется по мере нахождения минимальных комбинаций. Изначально минимальным количеством квадратов считается площадь квадрата. Затем с помощью методов pushSquare, GetMaximumSquare и getNextEmptyPoint свободные клетки случайно заполняются максимально большими квадратами, поиск же решения заключается в последовательном удалении минимальных квадратов (каждый раз ширина становится на 1 больше) и перебора всех вариантов расположения. Если новое число минимальных квадратов становится больше текущего, то этот вариант рассматривать не следует. Если же находится такое разбиение, что количество квадратом меньше текущего, оно запоминается, и массив сохраняется в minSquares. Изначально алгоритм заканчивался, когда самый первый, самый большой квадрат разбивался и в нем рассматривались варианты. Однако алгоритм был оптимизирован составлением сразу нескольких квадратов. Квадраты кратные двум заполняются четырьмя равными квадратами, кратные трем — шестью, кратные пяти — восемью, в остальные квадраты сразу можно вписать три квадрата, а остальные заполнить оптимальным образом. Примеры представлены на рисунке 1.

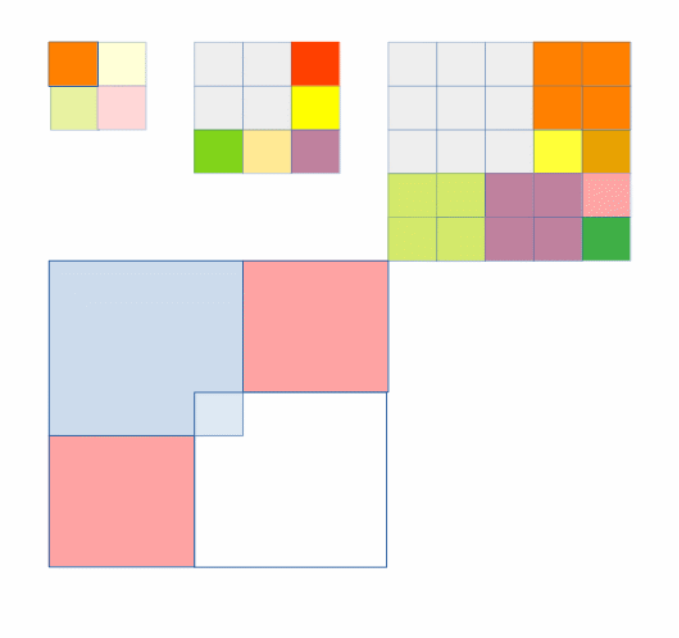

Рисунок 1 — Оптимизация алгоритма

Сложность алгоритма можно проанализировать по посчитанному времени поиска решения. Результат работы программы записан в таблице 1.

Таблица 1 — Время и число квадратов.

N	t
2	38
3	18
4	42
5	36
7	115
9	53
11	923
13	771
15	145
16	86
17	10356
19	30060
20	69
21	322
22	66
23	82902
24	77
25	4333
26	88
27	785
28	105
29	653180
30	138
31	2.36*10^(6)
34	115
35	8414
36	176
37	8,8*10^(6)
38	171
39	2588
40	240
41	59.972*10^(6)

Графики отображающие соотношение число элементов — время представлены на рисунке 2. Для анализа маленьких чисел большие результаты были исключены из списка.

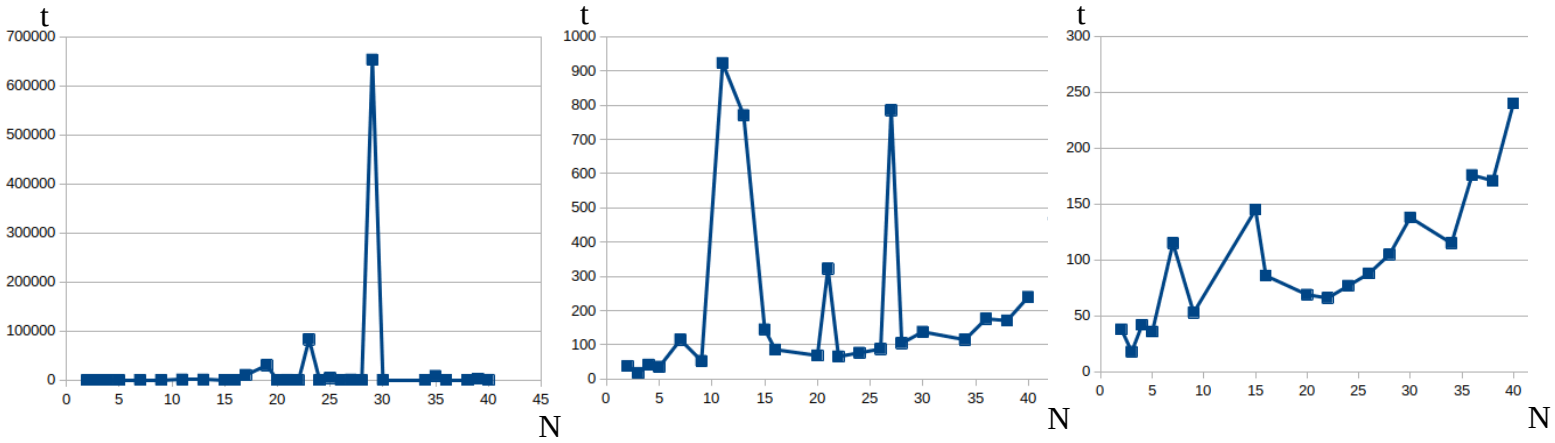

Рисунок 2 — Зависимость времени выполнения программы от N

Из графиков можно сделать выводы, что в среднем случае зависимость квадратичная.

Были найдены результаты для чисел не кратных 2,3,5. Точки и функция аппроксимации отмечены на графике с помощью сайта *http://mathhelpplanet.com/static.php?p=onlayn-mnk-i-regressionniy-analiz*.

Можно сделать вывод, что для таких чисел $o(1.4^N)$. Результат на рисунке 3.

Рисунок 3 — зависимость не кратных 5,3,2 N от t.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	7	Минимальное количество квадратов: 9 4 4 4 5 1 3 1 5 3 1 1 2 1 3 2 3 1 2 3 3 1 3 4 1 4 3 1 Поиск занял 0.000109 секунд
2.	2	Минимальное количество квадратов: 4 2 2 1 2 1 1 1 2 1 1 1 1 Поиск занял 2.1e-05 секунд
3.	3	Минимальное количество квадратов: 6 1 1 2 3 1 1 1 3 1 2 3 1 3 2 1 3 3 1 Поиск занял 1.2e-05 секунд
4.	25	Минимальное количество квадратов: 8 1 1 15 16 1 5 1 16 5

		1 21 5 6 16 10 16 6 10 16 16 10 21 1 5 Поиск занял 0.004635 секунд

Выводы.

Был изучен поиск с возвратом, разработана программа, выполняющая считывание с клавиатуры исходных данных и находящая минимальное количество квадратов, которыми можно заполнить квадрат N×N, а также было посчитано время выполнения программы.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <cmath>
#include <ctime>

using namespace std;

class Point {
public:
    int x, y;
    Point(int x, int y) {
        this->x = x;
        this->y = y;
    }
};

class square {
public:
    Point* point;
    int width;
    square(Point* point, int width) {
        this->point = point;
        this->width = width;
    }
    ~square() {
        delete (this->point);
    }

    square* copy() {
          return new square(new Point(this->point->x, this->point->y), this->width);
    }
};

class table {
private:
    int** _array;                       //для 0 и 1
    int _emptySpace;                    //кол-во 0
```

```
    int _squaresCount;                  //кол-во
    square** _squares;                  //

    void clearTable() {
        for (int x = 0; x < this->width; x++)
            for (int y = 0; y < this->width; y++)
                _array[x][y] = 0;
        _squaresCount = 0;
        _emptySpace = width*width;
    }

public:
    int width;

    square** minSquares;                //минимальный массив квадратов

    table (int width) {
        this->width = width;
        _array = new int* [width];
        for (int i = 0; i < width; i++)
            _array[i] = new int[width];
        this->clearTable();               //заполнение 0
        _emptySpace = width*width;
        _squares = new square*[_emptySpace];   //стек квадратов
        _squaresCount = 0;
    }
    ~table() {
        for (int i = 0; i < width; i++)
            delete(_array[i]);
        delete(_array);
        delete(_squares);
    }

    int pushSquare(square* square) {
        if (this->isFull()) return 0;
         for (int x = square->point->x; x < square->point->x + square->width;
x++)      //по координатам заполняем квадрат _array единицами
                for (int y = square->point->y; y < square->point->y + square-
>width; y++)
                 _array[x][y] = 1;
        delete(_squares[_squaresCount]);    //??????
        _squares[_squaresCount++] = square;  //добавляем в стек
```

```
            _emptySpace -= square->width * square->width;   //уменьшаем число
свободных
        return 1;
    }
    square* popSquare() {
        if (_squaresCount == 0) return NULL;
        square* lastSquare = _squares[_squaresCount-1];
            for (int x = lastSquare->point->x; x < lastSquare->point->x +
lastSquare->width; x++)  //по коорд. послед. в стеке заполняем нулями _array
               for (int y = lastSquare->point->y; y < lastSquare->point->y +
lastSquare->width; y++)
                _array[x][y] = 0;
        _squaresCount--;
        _emptySpace += lastSquare->width * lastSquare->width;
        return lastSquare;
    }

    Point* getNextEmptyPoint() {  //нахождение пустой клетки
        for (int x = 0; x < this->width; x++)
            for (int y = 0; y < this->width; y++)
                if (_array[x][y] == 0)
                    return new Point(x, y);
        return NULL;
    }

    bool isFull() {   //весь ли квадрат заполнен
        return !_emptySpace;
    }

      square* getMaximumSquare(Point* point) {     //заполнение максимальным
квадратом
        int width = 1;
        bool flag = true;
        while (flag) {
            width++;
             flag = (point->x + width - 1 < this->width) &&  //не вышли ли за
пределы
                    (point->y + width - 1 < this->width);
            if (flag) {
                for (int i = 0; i < width; i++) {  //ищем макс. размер
                    if ((_array[point->x + width - 1][point->y + i] == 1) ||
```

```
                        (_array[point->x + i][point->y + width - 1] == 1)) {
                        flag = false;
                        break;
                    }for (int i = 0; i < width; i++) {  //ищем макс. размер
                         if ((_array[point->x + width - 1][point->y + i] == 1)
||
                                (_array[point->x + i][point->y + width - 1] ==
1)) {
                            flag = false;
                            break;
                        }
                    }
                }
            }
        }
        return new square(point, width - 1);
    }

    int getMinimumCountOfSquares() {
        int minimumCount = this->_emptySpace;  //изначально максимум
        minSquares = new square*[this->width * this->width];

        //оптимизация
        if ((width%2) == 0){
             pushSquare(new square(new Point(this->width / 2.0, this->width /
2.0), ceil(this->width / 2.0)));
                pushSquare(new square(new Point(ceil(this->width / 2.0), 0),
this->width / 2));
                pushSquare(new square(new Point(0, ceil(this->width / 2.0)),
this->width / 2));
        }
        else if ((width%3) == 0){
            pushSquare(new square(new Point(0, 0), this-> width * 2 / 3.0));
             pushSquare(new square(new Point(this-> width * 2 / 3.0, 0), this-
>width / 3.0));
             pushSquare(new square(new Point(0, this-> width * 2 / 3.0), this-
>width / 3.0));
        }
        else if ((width%5) == 0){
            pushSquare(new square(new Point(0, 0), this-> width * 3 / 5.0));
```

```
            pushSquare(new square(new Point(this-> width * 3 / 5.0, 0), this-
>width / 5.0));
            pushSquare(new square(new Point(0, this-> width * 3 / 5.0), this-
>width / 5.0));
        }
        else{
             pushSquare(new square(new Point(this->width / 2.0, this->width /
2.0), ceil(this->width / 2.0)));
                pushSquare(new square(new Point(ceil(this->width / 2.0), 0),
this->width / 2));
                pushSquare(new square(new Point(0, ceil(this->width / 2.0)),
this->width / 2));
        }

        //поиск наилучшего варианта
        while (!this->isFull()) {
            pushSquare(getMaximumSquare(getNextEmptyPoint()));
        }

        minimumCount = _squaresCount;

        for (int i = 0; i < _squaresCount; i++) {
            minSquares[i] = _squares[i]->copy();
        }

        if ( width==2) return minimumCount;

        int resizedSquareIndex;
        while ( 1 ) {

               resizedSquareIndex = _squaresCount - 1;   //удалили единичные
квадраты
            while (_squares[resizedSquareIndex--]->width == 1)
                popSquare();
            resizedSquareIndex++;
            if (resizedSquareIndex < 3) break;

                square* poppedSquare = popSquare();   //замена след. мин. на
квадрат с шириной-1
                    pushSquare(new  square(new  Point(poppedSquare->point->x,
poppedSquare->point->y), poppedSquare->width - 1));
```

```
                while (!this->isFull() && (_squaresCount < minimumCount)) {
//заполнение макс.
                    pushSquare(getMaximumSquare(getNextEmptyPoint()));
                }

                if (_squaresCount < minimumCount) {   //если нашли минимум - запоминаем
                    minimumCount = _squaresCount;
                    for (int j = 0; j < _squaresCount; j++) {
                        minSquares[j] = _squares[j]->copy();
                    }
                }
            }
            return minimumCount;
        }
};

int main(){
    int tableWidth;
    cout<<"Введите N: "; cin >> tableWidth;
    table table(tableWidth);
    unsigned int start_time = clock();
    int minimumCountOfSquares = table.getMinimumCountOfSquares();
    unsigned int end_time = clock();
    unsigned int time = end_time-start_time;
     cout<< "Минимальное количество квадратов: " << minimumCountOfSquares << endl;
    for (int i = 0; i < minimumCountOfSquares; i++)
                 cout  <<  table.minSquares[i]->point->x  +  1  <<  "  "  <<  table.minSquares[i]->point->y  +  1  <<  "  "  <<  table.minSquares[i]->width  <<  endl;
      cout  <<  "Поиск  занял  "<<  ((float)time)  /  CLOCKS_PER_SEC<<"  секунд"<<  endl;
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Если результаты тестирования велики (больше 1 страницы), то их выносят в приложение.

Процесс тестирования можно представить в виде таблицы, например:

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
4.			
5.			
6.			
...			

Обратите внимание, что в нумерации таблицы в приложении обязательно должен быть в качестве префикса номер самого приложения: А.