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

# ОТЧЕТ

**по лабораторной работе №1**

**по дисциплине «Построение и анализ алгоритмов»**

**Тема: Поиск с возвратом**

Студент гр. 7304 _____ Давыдов А.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2018

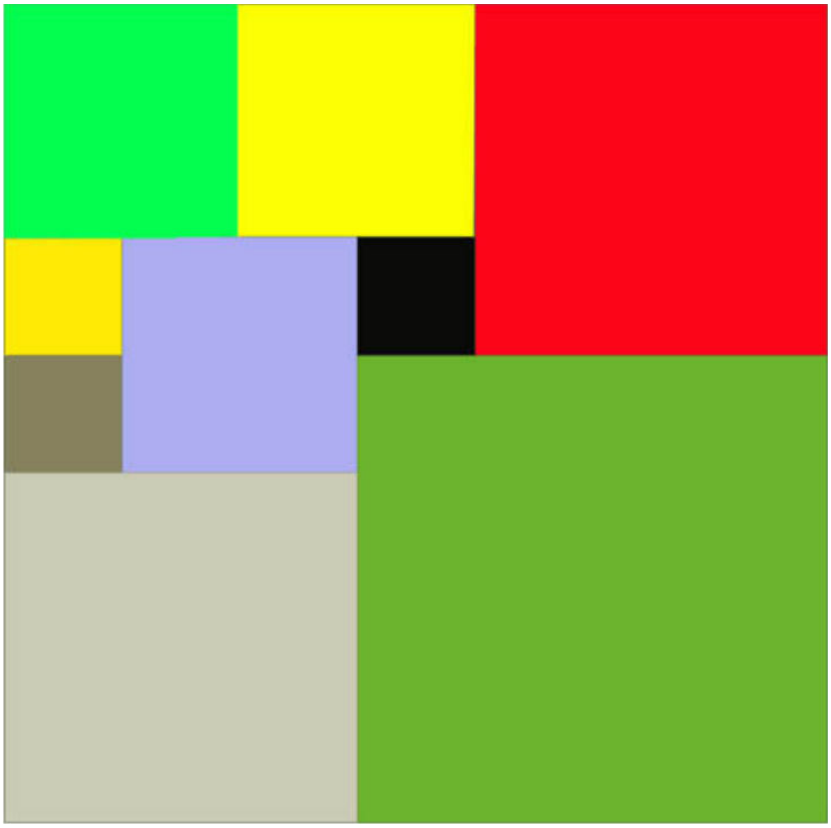
**Цель работы.**

Реализовать алгоритм квадрирования минимальным количеством с условием, что квадраты, которыми мы заполняем больший, могут повторяться.

**Задача**

У Вовы много квадратных обрезков доски. Их стороны (размер) изменяются от 1 до N−1, и у него есть неограниченное число обрезков любого размера. Но ему очень хочется получить большую столешницу - квадрат размера $N$ . Он может получить ее, собрав из уже имеющихся обрезков(квадратов).

Например, столешница размера 7×7 может быть построена из 9 обрезков.



Внутри столешницы не должно быть пустот, обрезки не должны выходить за пределы столешницы и не должны перекрываться. Кроме того, Вова хочет использовать минимально возможное число обрезков.

**Входные данные**

Размер столешницы - одно целое число $N(2 \le N \le 20)$.

**Выходные данные**

Одно число $K$, задающее минимальное количество обрезков(квадратов), из которых можно построить столешницу(квадрат) заданного размера $N$. Далее должны идти $K$ строк, каждая из

которых должна содержать три целых числа $x,y$ и $w$, задающие координаты левого верхнего угла ($1 \le x,y \le N$) и длину стороны соответствующего обрезка(квадрата).

**Экспериментальные результаты.**

1)Были реализованы функции для работы с двумерным массивом, создаваемым под размер стола.

```
 void init_table(short **table, int size)
{
    for(int i = 0; i < size; ++i)
        for(int j = 0; j < size; ++j)
            table[i][j] = 0;
}

//set coordinat system: i is y coordinat, j is x coordinat -> coorect call
table[y][x]
void next_cell(short **table, int size, int &x, int &y)
{
    for(int i = 0; i < size; ++i)
        for(int j = 0; j < size; ++j)
            if(table[i][j] == 0)
            {
                x = j;
                y = i;
                return;
            }
}


int max_side(short **table, int size, int x, int y)
{
    int max_side_x = 0;
    int max_side_y = 0;
    for(int i = y; i < size; ++i)
        if(table[i][x] == 0)
            ++max_side_y;
        else
            break;
    for(int i = x; i < size; ++i)
        if(table[y][i] == 0)
            ++max_side_x;
        else
            break;
    return min(max_side_x, max_side_y);
}


//color is number of next adding squard
void draw_square(short **table, int x, int y, int side, short color)
{
    for(int i = y; i < y + side; ++i)
        for(int j = x; j < x + side; ++j)
            table[i][j] = color;
}



void delete_square(short **table, int x, int y, int side)
{
```

```
    for(int i = y; i < y + side; ++i)
        for(int j = x; j < x + side; ++j)
                table[i][j] = 0; // 0 is empty field
}


bool fill_table(short **table, int size)
{
    for(int i = size - 1; i >= 0; --i)
        for(int j = size - 1; j >= 0; --j)
            if(table[i][j] == 0)
                return false;
    return true;
   }
```

2) Была реализована функция simple_packing, реализующая перебор всех вариантов

```
//step is number of next adding square
//best table is table which will contain copy of table when we get new minimal
packing
void simple_packing(short **table, int size, short step,int &min, short
**best_table)
{
    if(step >= min)
        return;
    if(fill_table(table, size))
    {
        if(min >= step)
        {
            min = step;
            for(int i = 0; i < size; ++i)
                for(int j = 0; j < size; ++j)
                    best_table[i][j] = table[i][j];
        }
        return;
    }

    int x, y;
    next_cell(table, size, x, y);
    ++step; //before add new square

    for(int side = max_side(table, size, x, y); side >= 1; --side)
    {
        draw_square(table, x, y, side, step);
        simple_packing(table, size, step, min, best_table);
        delete_square(table, x, y, side);
    }
}
```

3) Были реализованы функции для вывода результатов работы предыдущей функции в случае, если длина стола – простое число, не кратное 2, 3, 5.

```
void print_coordinate_and_side(short **best_table, int size, short color)
{
    for(int i = 0; i < size; ++i)
        for(int j = 0; j < size; ++j)
        if(best_table[i][j] == color)
        {
            int side = 0;
            for(int k = j; k < size; ++k)
```

```
                if(best_table[i][k] == color)
                    ++side;
                else
                    break;
            cout << j + 1 << " " << i + 1 << " " << side << endl;
            return;
        }
}


void output_results(short **best_table, int size, int min)
{
    cout << min << endl;
    //print_table(best_table, size);
    for(short color = 1; color <= min; ++color)
        print_coordinate_and_side(best_table, size, color);
}
```

4) Были реализованы евристики для случаев, когда длина стороны столка кратна 2, 3, 5

```
void heuristik_result_2(short **best_table, int size)
{
    draw_square(best_table, 0, 0, size / 2, 1);
    draw_square(best_table, 0, size/2, size/2, 2);
    draw_square(best_table, size/2, 0, size/2, 3);
    draw_square(best_table, size/2, size/2, size/2, 4);
    output_results(best_table, size, 4);
}

void heuristik_result_3(short **best_table, int size)
{
    draw_square(best_table, 0, 0, (size * 2) / 3, 1);
    draw_square(best_table, 0, (size * 2) / 3, size / 3, 2);
    draw_square(best_table, size / 3, (size * 2) / 3, size / 3, 3);
    draw_square(best_table, (2 * size) / 3, (size * 2) / 3, size / 3, 4);
    draw_square(best_table, (size * 2) / 3, size / 3, size / 3, 5);
    draw_square(best_table, (size * 2)/ 3, 0, size / 3, 6);
    output_results(best_table, size, 6);
}

void heuristil_result_5(short **best_table, int size)
{
    draw_square(best_table, 0, 0, (size * 3) / 5, 1);
    draw_square(best_table, 0, (size * 3) / 5, (size * 2) / 5, 2);
    draw_square(best_table, (size * 2) / 5, (size * 3) / 5, (size * 2) / 5, 3);
    draw_square(best_table, (size * 4) / 5, (size * 3) / 5, size / 5, 4);
    draw_square(best_table, (size * 4) / 5, (size * 4) / 5, size / 5, 5);
    draw_square(best_table, (size * 3) / 5, 0, (size * 2) / 5, 6);
    draw_square(best_table, (size * 3) / 5, (size * 2) / 5, size / 5, 7);
    draw_square(best_table, (size * 4) / 5, (size * 2) / 5, size / 5, 8);
    output_results(best_table, size, 8);
}
```

5) Код функции main

```
int main()
{
    int size;
    cin >> size;
```

```
    int min = 16; //most bad packing as start value
    short step = 0;
    short **table = new short*[size];
    short **best_table = new short*[size];

    for(int i = 0; i < size; ++i)
        table[i] = new short[size];
    for(int i = 0; i < size; ++i)
        best_table[i] = new short[size];

    init_table(table, size);
    init_table(best_table, size);

    ////////// heuristiks //////////
    if(size % 2 == 0)
    {
        heuristik_result_2(best_table, size);
        return 0;
    }
    if(size % 3 == 0)
    {
        heuristik_result_3(best_table, size);
        return 0;
    }
    if(size % 5 == 0)
    {
        heuristil_result_5(best_table, size);
        return 0;
    }

    ////////////////////////////////

    //draw first three squares
    draw_square(table, 0, 0, int(size/2) + 1, 1);
    draw_square(table, int(size/2) + 1, 0, int(size/2), 2);
    draw_square(table, 0, int(size/2) + 1, int(size/2), 3);
    step = 3;

    ///////// call function simple_packing and output ///////////
    simple_packing(table, size, step, min, best_table);
    output_results(best_table, size, min);
   // print_table(best_table, size);
    ////////////////////////////////////////////////////////////

    ////////// test //////////
    //delete_square(table, 0, 0, int(size/2) + 1);
    //print_table(table, size);
    ////////////////////////

    for(int i = 0; i < size; ++i)
    {
        delete[] best_table[i];
        delete[] table[i];
    }
    delete[] table;
    delete[] best_table;

    return 0;
}
```

**Выводы.**

Был разработан и оптимизирован с помощью евристик алгоритм простого квадрирования, а также функции для вывода результатов работы алгоритма. В результате тестирования на диапазоне чисел от 2 до 40 были получены корректные результаты.