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. 8304 ____ Бочаров Ф.Д.

Преподаватель ____ Размочаева Н.В.

Санкт-Петербург

2020

Вариант 4р - прямоугольник

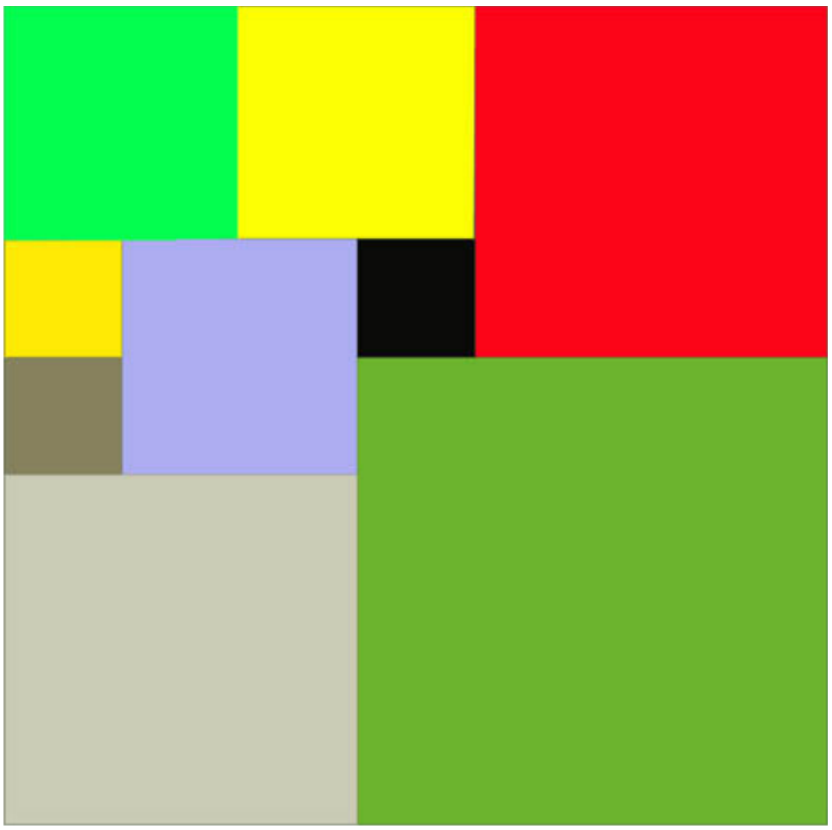
Цель работы

Ознакомиться и закрепить знания, связанные с алгоритмами поиска с возвратом.

Постановка задачи

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.

Описание алгоритма

Рекурсивный алгоритм, для каждой клетки квадрата (прямоугольника), для каждого возможного размещенного в этой клетке обрезка, рассматриваются все варианты расположения отрезков в следующей клетке обрезков всех возможных размеров.

Описание функций

1. `bool add_sq(size_t x, size_t y, size_t side_size, size_t sq_number, std::shared_ptr<size_t[]> sq_arr, size_t sq_width, size_t sq_height)`

добавляет квадрат в точку с координатами x, y.

2. `void delete_sq(size_t x, size_t y, size_t side_size, std::shared_ptr<size_t[]> sq_arr, size_t sq_width, size_t sq_height)`

удаляет квадрат из точки x, y.

3. `void fill_sq(std::shared_ptr<size_t[]> &sq_arr, size_t sq_width, size_t sq_height, size_t i, size_t sq_number, size_t &min_cnt, std::shared_ptr<size_t[]> &min_sq_arr, size_t &covers_cnt)`

Рекурсивная функция, основная, осуществляющая поиск решения.

4. `void start_algorithm(size_t sq_width, size_t sq_height)`

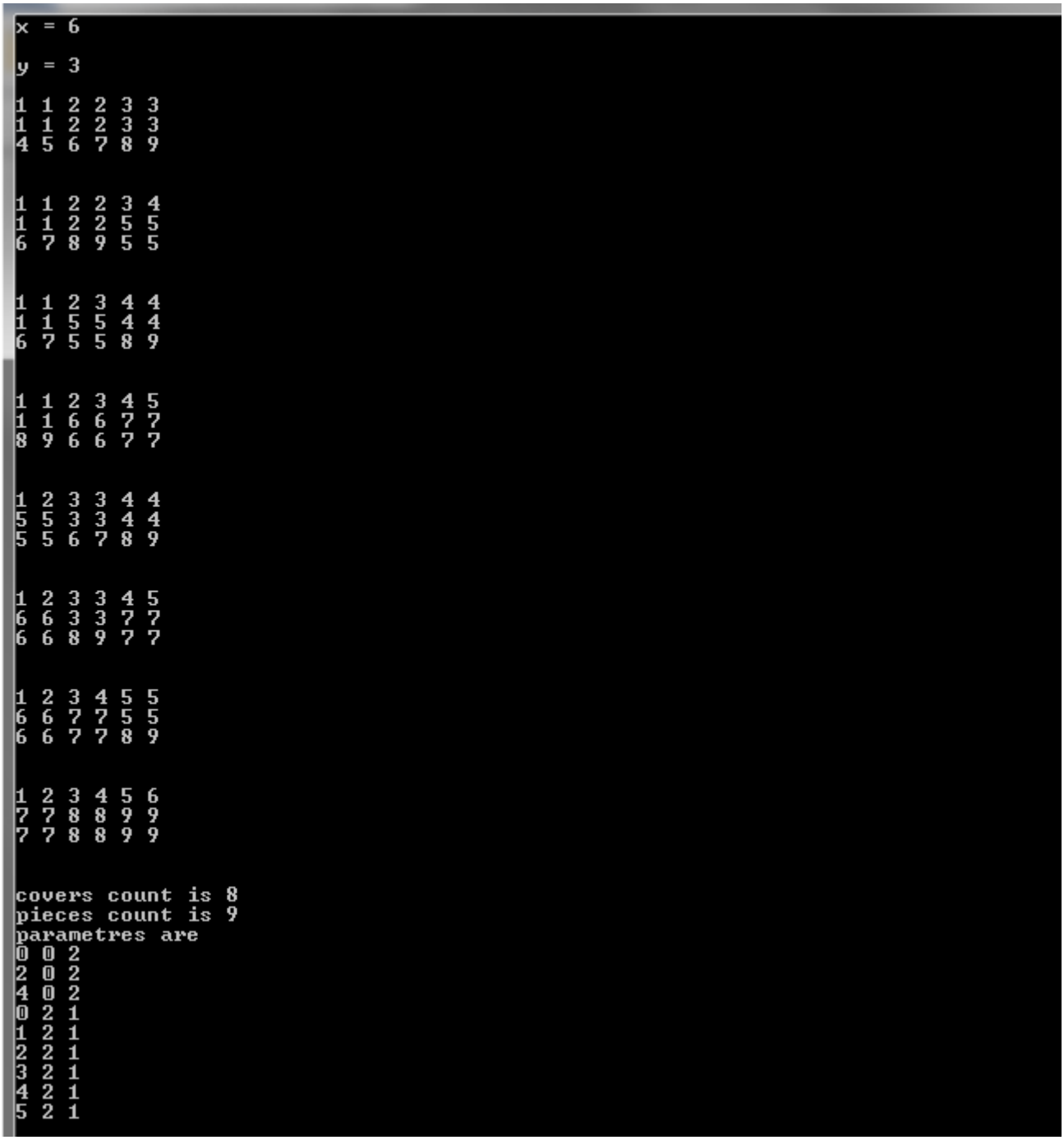
функция передает в основную параметры и обрабатывает результат.

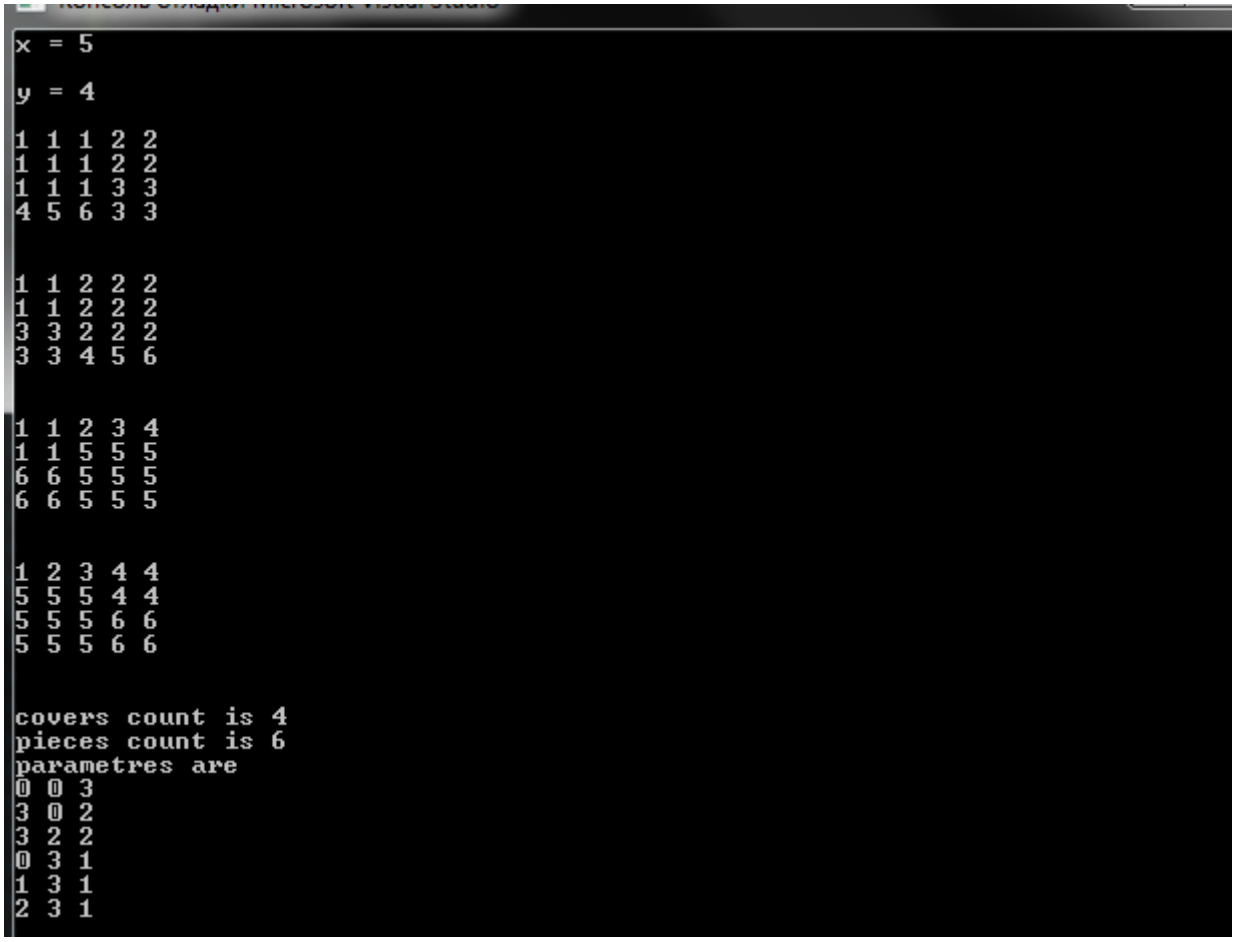
Вывод

Был получен опыт работы с алгоритмами поиска с возвратом, реализована программа, рассчитывающая минимальное разбиение квадрата (прямоугольника) на квадраты.

ПРИЛОЖЕНИЕ А.

ТЕСТИРОВАНИЕ.





ПРИЛОЖЕНИЕ В.

Исходный код.

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



void delete_sq(size_t x, size_t y, size_t side_size, std::shared_ptr<size_t[]> sq_arr, size_t
sq_width, size_t sq_height) {
	for (size_t j = y; j < y + side_size; j++)
		for (size_t i = x; i < x + side_size; i++)
			sq_arr[i + sq_width * j] = 0;
}

// вставляет квадрат в доску
bool add_sq(size_t x, size_t y, size_t side_size, size_t sq_number, std::shared_ptr<size_t[]>
sq_arr, size_t sq_width, size_t sq_height) {

	if (x + side_size >= sq_width + 1 || y + side_size >= sq_height + 1)
		return false;

	for (size_t j = y; j < y + side_size; j++)
		for (size_t i = x; i < x + side_size; i++)
```

```
                        if (sq_arr[i + sq_width * j] != 0)
                                return false;

        for (size_t j = y; j < y + side_size; j++)
                for (size_t i = x; i < x + side_size; i++)
                        sq_arr[i + sq_width * j] = sq_number;

        return true;
}

// sq_arr - состояние доски sq_width и sq_height - её параметры
void fill_sq(std::shared_ptr<size_t[]>& sq_arr, size_t sq_width, size_t sq_height, size_t i,
size_t sq_number, size_t& min_cnt, std::shared_ptr<size_t[]>& min_sq_arr, size_t& covers_cnt) {

        if (i == sq_width * sq_height) {
                if (sq_number == min_cnt + 1) {
                        covers_cnt++;
                        for (size_t i = 0; i < sq_width * sq_height; i++) {
                                std::cout << sq_arr[i] << " ";
                                if ((i + 1) % sq_width == 0)
                                        std::cout << '\n';
                        }
                        std::cout << '\n';
                        std::cout << '\n';
                }
                if (sq_number < min_cnt) {
                        covers_cnt = 1;

                        for (size_t i = 0; i < sq_width * sq_height; i++) {
                                std::cout << sq_arr[i] << " ";
                                if ((i + 1) % sq_width == 0)
                                        std::cout << '\n';
                        }

                        std::cout << '\n';
                        std::cout << '\n';

                        min_cnt = sq_number - 1;

                        for (size_t i = 0; i < (sq_width * sq_height); i++) {

                                min_sq_arr[i] = sq_arr[i];


                        }
                }
                return;
        }

        if (sq_number > min_cnt + 1)
                return;

        for (size_t j = std::min(sq_width, sq_height) - 1; j > 0; j--) {
                if (add_sq(i % sq_width, i / sq_width, j, sq_number, sq_arr, sq_width, sq_height))
{

                        fill_sq(sq_arr, sq_width, sq_height, i + 1, sq_number + 1, min_cnt,
min_sq_arr, covers_cnt);
                        delete_sq(i % sq_width, i / sq_width, j, sq_arr, sq_width, sq_height);

                }
                else {
                        if (j == 1)
                                fill_sq(sq_arr, sq_width, sq_height, i + 1, sq_number, min_cnt,
min_sq_arr, covers_cnt);
```

```
                        continue;
                }

        }
        return;
}

// запускает рекурсию, обрабатывает её результаты
void start_algorithm(size_t sq_width, size_t sq_height) {

        std::shared_ptr<size_t[]> sq_arr(new size_t[sq_height * sq_width]);
        std::shared_ptr<size_t[]> min_sq_arr(new size_t[sq_height * sq_width]);



        for (size_t i = 0; i < (sq_height * sq_width); i++) {
                sq_arr[i] = 0;
        }

        size_t min_cnt = sq_height * sq_width;

        size_t covers_cnt = 0;

        fill_sq(sq_arr, sq_width, sq_height, 0, 1, min_cnt, min_sq_arr, covers_cnt);

        std::cout << "covers count is ";
        std::cout << covers_cnt << std::endl;

        std::vector<size_t> sqs_sizes(min_cnt);


        for (size_t j = 1; j <= min_cnt; j++) {
                for (size_t i = 0; i < (sq_height * sq_width); i++) {
                        if (min_sq_arr[i] == j) {
                                sqs_sizes.at(j - 1) += 1;
                        }
                        else {
                                continue;
                        }
                }
        }

        std::cout << "pieces count is " << min_cnt << std::endl;

        std::cout << "parametres are\n";

        for (size_t j = 1; j <= min_cnt; j++) {
                sqs_sizes.at(j - 1) = sqrt(sqs_sizes.at(j - 1));
                for (size_t i = 0; i < (sq_height * sq_width); i++) {
                        if (min_sq_arr[i] == j) {

                                std::cout << (i % sq_width) << " " << (i / sq_width) << " " <<
sqs_sizes.at(j - 1) << std::endl;
                                break;
                        }
                }
        }
}


int main() {

        size_t piece_size = 1;
        std::cout << "x = ";
        size_t sq_width;
```

```
        std::cin >> sq_width;
        std::cout << '\n';
        std::cout << "y = ";
        size_t sq_height;
        std::cin >> sq_height;
        std::cout << '\n';

        start_algorithm(sq_width, sq_height);

        return 0;
}
```