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.8304 _____ Холковский К.В.
Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2019

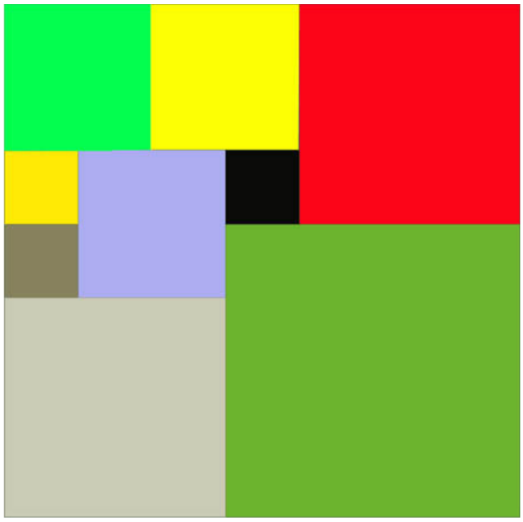
Задание.

Вариант 3р

Рекурсивный бэктрекинг. Исследование времени выполнения от размера квадрата.

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 ($2 \leq N \leq 20$).

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 ($1 \leq x, y \leq N$) и длину стороны соответствующего обрезка(квадрата).

Цель работы.

Написать программу, находящую минимально разложение заданного квадрата квадратами с меньшим ребром.

Описание алгоритма.

Ищем не закрашенный элемент, пытаемся вставить квадрат длинны 1, далее пытаемся вставить новый квадрат. Если же квадрат вставить не получается, то, если все закрашено, сравниваем с минимальным (на данный момент) результатом, после чего переходим к предыдущему варианту разбиения и увеличиваем сторону квадрата. Алгоритм имеет экспоненциальную сложность.

Описание функций и структур данных.

`void func(int * arr, std::vector<elem>& tmp, std::vector<elem>& min, int& count)` – рекурсивная функция поиска минимального разложения, где int* arr – это квадрат, который нужно закрасить, `std::vector<elem>& tmp` - ссылка на вектор, хранящий текущее разбиение, `std::vector<elem>& min` – ссылка на вектор, хранящий минимальное(на некоторый момент) разбиение, `int& count` – ссылка на переменную, хранящую количество итераций.

`struct elem {int x; int y; int len;};` - структура для хранения координат и размеров закрашенных квадратов, где x и y – координаты, len – длинна стороны квадрата.

Исследование количества итераций.

Данные исследования смотри в таблице 1 и на рисунке 1.

Таблица 1 – Результаты исследования

Размер стороны	Количество итераций
2	0
3	3
5	16
7	73
11	1656
13	3874
17	45324
19	123137
23	802133
29	8359021
31	17990778

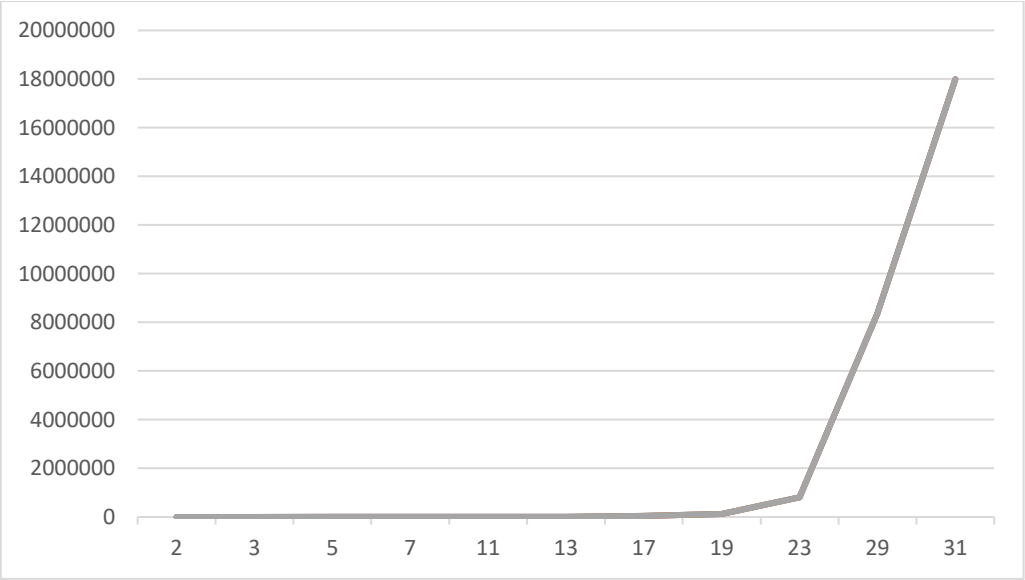


Рисунок 1 – Зависимость количества итераций от размера стороны

По полученным данным можно сказать, что малому изменению стороны соответствует большое изменение кол-ва итераций.

Тестирование.

Для 5:

```
5
8
d g g b b
e g g b b
f h c c c
a a c c c
a a c c c
```

Рисунок 2 – Полученный результат

```
* * * b b  d g * b b  d f g b b
* * * b b  e h * b b  e e * b b
* * c c c  f * c c c  e e c c c
a a c c c  a a c c c  a a c c c
a a c c c  a a c c c  a a c c c

d * * b b  d g g b b  d d * b b
* * * b b  e g g b b  d d * b b
* * c c c  f * c c c  * * c c c
a a c c c  a a c c c  a a c c c
a a c c c  a a c c c  a a c c c

d * * b b  d g g b b  d d * b b
e * * b b  e g g b b  d d * b b
* * c c c  f h c c c  e * c c c
a a c c c  a a c c c  a a c c c
a a c c c  a a c c c  a a c c c

d * * b b  d * * b b  d d * b b
e * * b b  e e * b b  d d * b b
f * c c c  e e c c c  e f c c c
a a c c c  a a c c c  a a c c c
a a c c c  a a c c c  a a c c c

d g * b b  d f * b b  d d g b b
e * * b b  e e * b b  d d * b b
f * c c c  e e c c c  e f c c c
a a c c c  a a c c c  a a c c c
a a c c c  a a c c c  a a c c c
```

Рисунок 3 – Промежуточные данные

Выводы.

В ходе выполнения данной работы была написана программа, находящая минимальное разложение заданного квадрата, и было проведено исследование кол-ва итерация от размера квадрата.

Исходный код

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

int size;

struct elem {
	int x;
	int y;
	int len;
};

void func(int * a, std::vector<elem>& tmp, std::vector<elem>& min, int&
count) {
	bool f1 = false;
	elem cur;

	for (int i = 0; i < size; ++i) {
		for (int j = 0; j < size; ++j)
			if (a[i * size + j] == 0) {
				f1 = true;
				cur = { i, j, 1 };
				break;
			}
		if (f1)
			break;
	}
	if (!f1)
	{
		min = tmp;
		return;
	}

	for (int tmp_len = 1; tmp_len < size && tmp.size() + 1 != min.size();
++tmp_len)
	{
		if (tmp_len > size - cur.x || tmp_len > size - cur.y)
			return;
		++count;
		for (int i = cur.x; i < cur.x + tmp_len; ++i)
			for (int j = cur.y; j < cur.y + tmp_len; ++j)
				if (a[i * size + j] == 1)
					return;

		for (int i = cur.x; i < cur.x + tmp_len; ++i)
			for (int j = cur.y; j < cur.y + tmp_len; ++j)
				a[i * size + j] = 1;

		cur.len = tmp_len;
tmp.push_back(cur);
		func(a, tmp, min, count);
		for (int i = cur.x; i < cur.x + tmp_len; ++i)
			for (int j = cur.y; j < cur.y + tmp_len; ++j)
				a[i * size + j] = 0;
		tmp.pop_back();
	}
}
int main() {
	int n;
    std::cin >> n;
    for (size = 2; size < n; ++size)
```

```
            if (n % size == 0)
                  break;
      std::vector<elem> tmp, min(size + 4);
      tmp.push_back({ 0, size / 2 + size % 2, size / 2});
      tmp.push_back({ size / 2 + size % 2, 0, size / 2});
        tmp.push_back({ size / 2, size / 2, size / 2 + size % 2 });
        int buf = n / size;
        int count = 0;
        if (size % 2 == 1)
        {
              int* my_arr = new int[(size / 2 + 1) * (size / 2 + 1)]();
              my_arr[(size / 2 + 1)*(size / 2 + 1)-1] = 1;
              size = size / 2 + size % 2;
              func(my_arr, tmp, min, count);
        }
        else
        {
              tmp.push_back({ 0, 0, 1 });
              min = tmp;
        }
        std::cout << min.size() << std::endl;
        for (size_t i = 0; i < min.size(); ++i)
              std::cout << min[i].x * buf << " " << min[i].y * buf << " " <<
min[i].len * buf << std::endl;
        std::cout << "Кол-во итераций: " << count << std::endl;
        return 0;
}
```