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. 1304 ___ Ефремов А.А.

Преподаватель ___ Шевелева А.М.

Санкт-Петербург

2023

Задание.

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 (см. Рисунок 1 - Пример столешницы 7×7).

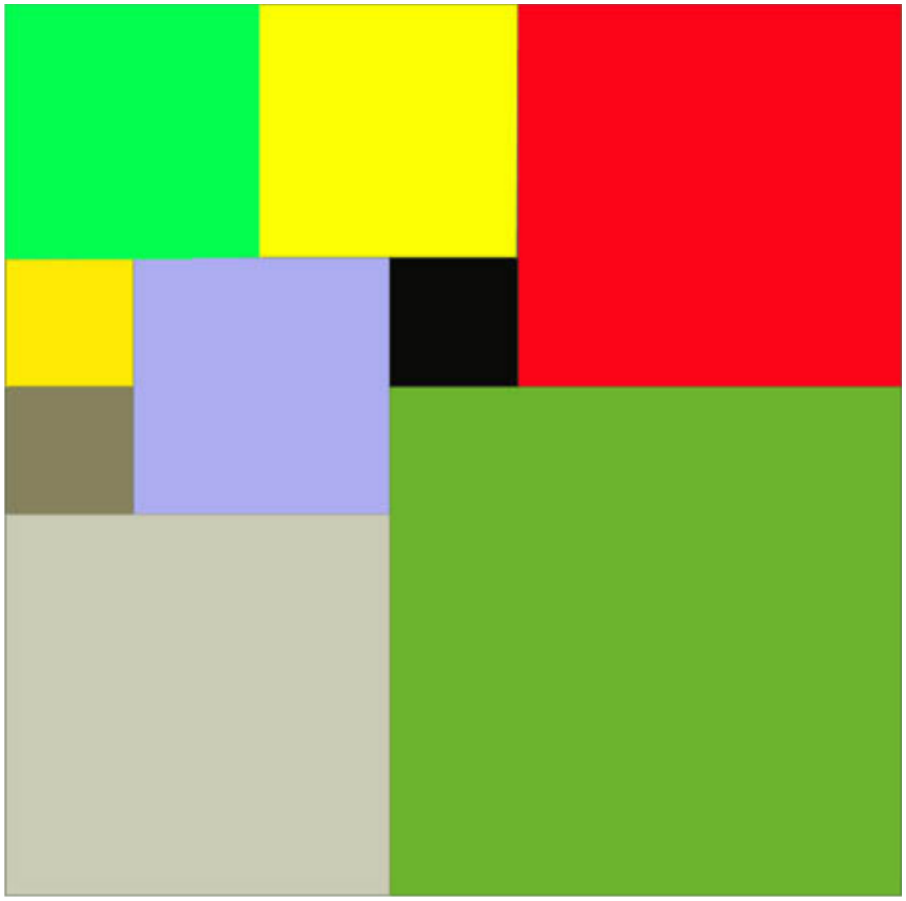

Рисунок 1 - Пример столешницы 7×7

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

Входные данные

Размер столешницы - одно целое число N(2≤N≤20).

Выходные данные

Одно число K, задающее минимальное количество обрезков(квадратов),

изкоторых 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≤x,y≤N) и длину сторонысоответствующего обрезка(квадрата).

Выполнение работы.

В ходе выполнения работы был создан класс *Square*, статические целочисленные переменные *N* и *square_counter*, статический вектор классов *Square final_array*, а также функции *overlay_check, backtracking* и *main*.

1) Класс *Square* содержит в поле с открытым доступом целочисленные переменные *x, y, w* – координаты по *x, y* и ширина квадрата соответственно. Также в поле содержится объявление конструктора класса Square, принимающий параметры *x, y, w*. Далее следует описание конструктора с инициализацией переменных.

2) Статическая целочисленная переменная *N* – размер столешницы, по условию задачи 2 <= *N* <= 20. Данная переменная имеет глобальную область видимости.

3) Статическая целочисленная переменная *square_counter* – счётчик (обрезков) квадратов. Данная переменная имеет глобальную область видимости.

4) Статический целочисленный вектор *final_array*, элементами которого являются объекты класса *Square* – хранит в себе результат поиска минимального количества обрезков (квадратов). Данный вектор имеет глобальную область видимости.

5) Функция *overlay_check*, принимающая вектор *board*, в котором хранятся объекты класса *Square*, а также принимающая координаты *x* и *y* – координаты левого верхнего угла при расстановке нового квадрата. Функция возвращает булевый тип. В теле функции в цикле *for* просматриваются все установленные квадраты и сравниваются с координатами нового. Если происходит наложение квадратов, то проверка считается провальной, и

функция возвращает false, иначе true.

6) Функция *backtracking*, принимающая вектор *board*, в котором хранятся объекты класса *Square*, а также принимающая целочисленную площадь занятой квадратами области *square_sum*, целочисленное количество установленных квадратов *square_amount*, а также целочисленные минимальные допустимые значения координат нового квадрата *minx*, *miny*. Внутри тела функции происходит обход всех допустимых значений координат нового квадрата с помощью вложенных циклов *for*. Происходит проверка на наложение координат с уже существующими с помощью вызова функции *overlay_check*. Если проверка отрицательная, то цикл перебора координат продолжается. Если проверка положительная, то начинается вычисление доступной ширины квадрата *new_square_width*. Для этого вычисляется минимальное расстояние от координаты левого верхнего угла нового квадрата до границы столешницы. Далее с помощью цикла *for* проверяется не накладывается ли квадрат указанной ширины на существующие квадраты. В итоге переменная *new_square_width* хранит максимально допустимый размер ширины. Далее с помощью цикла for проходим по всем возможным размерам ширины нового квадрата от максимального до единицы. Внутри данного цикла создается объект класса *Square* – новый квадрат по указанным координатам. Далее копируется вектор *board* в новый вектор *board_copy* и именно в *board_copy* добавляется новый квадрат. Это необходимо для корректной рекурсивной работы программы. Далее происходит сравнение площади, покрываемой квадратами, и площади всей столешницы. Если площади совпадают, то происходит проверка количества затраченных квадратов. Если количество затраченных квадратов меньше чем значение счётчика квадратов, то такой результат становится новым значением счетчика, а финальный вектор обновляется. Если площади не совпадают, то также происходит проверка количества затраченных квадратов, только в этом случае при положительной проверке происходит рекурсия посредством вызова функции *backtracking* с новыми значениями. При отрицательной проверке

происходит возврат из тела функции, так как нет смысла продолжать итерацию.

7) Внутри главной функции main происходит ввод пользователем с консоли значения *N*. Далее идет инициализация переменной square_counter, посчитанная по эмпирическим соображениям. Целочисленная переменная *boardsize* инициализируется нулем и отвечает за «масштаб» границ квадрата. При анализе разбиений было выяснено, что многие разбиения по сути идентичны и отличаются только масштабом клетки, например, разбиение для столешницы 3x3 и 9x9 на Рисунке 2.

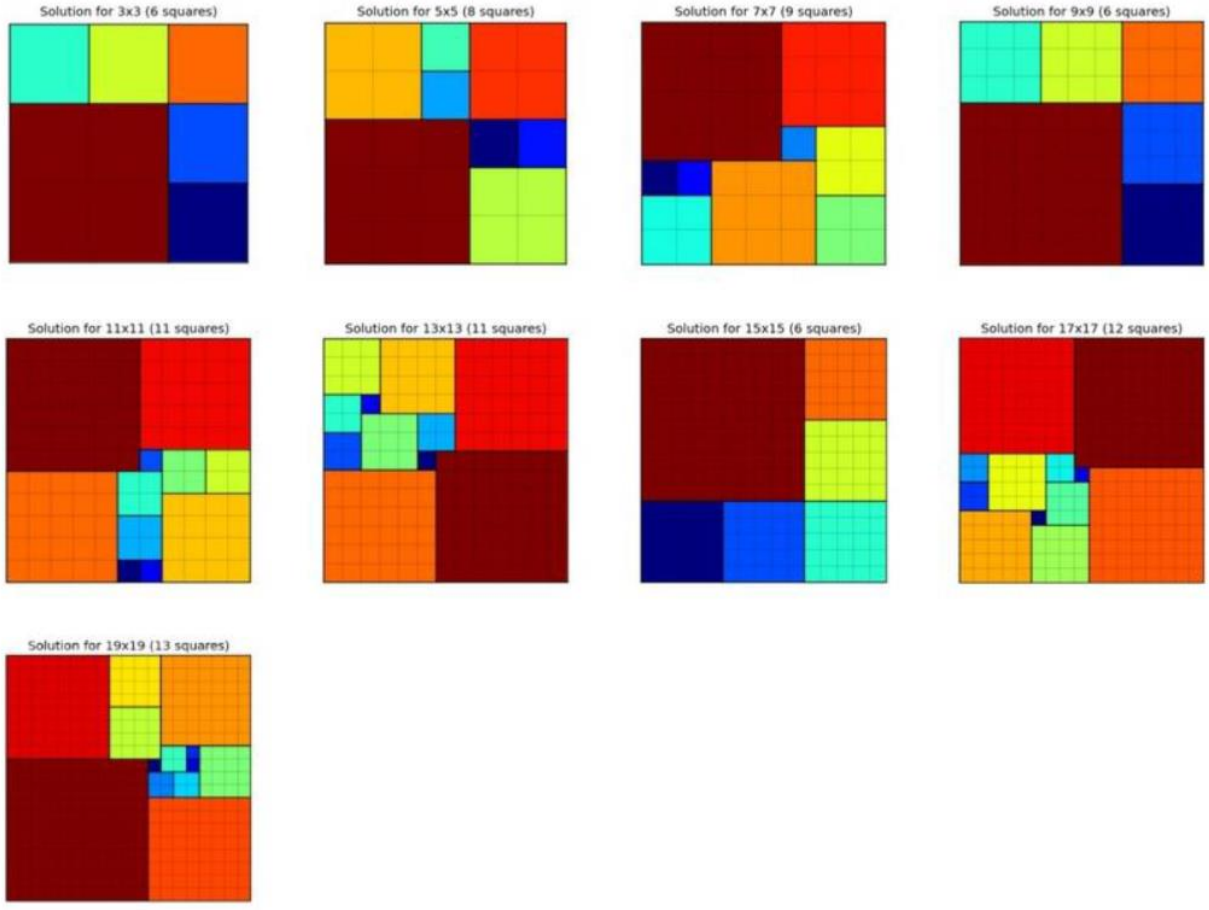

Рисунок 2 – Нечётный разбиение до N = 20.
Источник – https://laurentlessard.com/bookproofs/squaring-the-square

Таким образом, можно увеличить скорость работы программы с помощью приведения квадрата к меньшим размерам и последующим домножением результата на коэффициент реального масштаба, чем и является переменная *boardsize*.

С помощью цикла *for* проходимся по делителям числа N и вычисляем его наибольший делитель, тем самым мы нашли разложение числа N на максимальное число и минимальное. Далее сокращаем число N на этот максимальный делитель.

Далее инициализируем вектор *board*, элементами которого являются объекты класса *Square*. Заполняем это вектор тремя первыми квадратами, покрывающими самые большие участки столешницы.

Далее следует вызов функции *backtracking* с передачей необходимых аргументов и вывод результатов в консоль.

Программа успешно подходит для решения усложненных версий задачи на платформе Stepik.

Исходный код программы указан в приложении А.

Выводы.

В ходе выполнения работы был изучен, реализован на языке программирования $C++$, и применён на практике метод решения задач на тему «Поиск с возвратом». Вычисления были организованы таким образом, чтобы как можно раньше выявлять неподходящие варианты и ускорить работу. Это позволило значительно уменьшить время нахождения решения и успешно пройти задание за отведенное на платформе Stepik время.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

//Статическая переменная, описывающая размер столешницы
static int N;
//Статическая переменная, описывающая минимальное количество
квадратов, необходимых для столешницы
static int square_counter;

//Класс, описывающий квадрат
//Содержит три поля типа int - координаты левого верхнего угла и
размер стороны квадрата
class Square{
public:
	int x;

	int y;

	int w;
	//Конструктор класса
	Square(int x, int y, int w);
};

//Описание констурктора класса Square, заполнение полей аргументами
Square::Square(int x, int y, int w) {
	this->x = x;
	this->y = y;
	this->w = w;
}

//Статическая переменная, описывающая финальный минимальный набор
квадратов необходимый для столешницы
static std::vector<Square> final_array;

//Проверка на наложение нового квадрата на уже существующий
//Принимает вектор уже существующих квадратов и координаты нового
//Возвращает значение true-false в зависимости от положительного или
отрицательного результата проверки
bool overlay_check(std::vector<Square> board, int x, int y) {
	for (auto square : board) {
		if (x >= square.x && x < square.x + square.w && y >=
square.y && y < square.y + square.w) {
			return false;
		}
	}
	return true;
}

//Реализация алгоритма поиска с возвратом, рекурсивный метод
```

```
//Принимает вектор уже расположенных квадратов, значение покрытой
квадратами площади, количество квадратов, минимальные возможные
координаты нового квадрата
//Итогом работы является заполненный вектор минимального набора
квадратов
void backtracking(std::vector<Square> board, int square_sum, int
square_amount, int minx, int miny) {
	for (int x = minx; x < N; x++) {
		for (int y = miny; y < N; y++) {
			if (overlay_check(board, x, y)) {

				int new_square_width = fmin(N - x, N - y);
				for (auto square : board) {
					if (square.x + square.w > x and square.y >
y) {
						new_square_width =
fmin(new_square_width, square.y - y);
					}
				}
				for (int i = new_square_width; i > 0; i--) {
					Square sq(x, y, i);
					std::vector<Square> board_copy;
					std::copy(board.begin(), board.end(),
back_inserter(board_copy));
					board_copy.push_back(sq);
					if (square_sum + pow(sq.w, 2) == pow(N, 2))
{
						if (square_amount + 1 <
square_counter) {
							square_counter = square_amount +
1;
							final_array.clear();
							std::copy(board_copy.begin(),
board_copy.end(), back_inserter(final_array));
						}
					}
					else {
						if (square_amount + 1 <
square_counter) {
							backtracking(board_copy,
square_sum + pow(sq.w, 2), square_amount + 1, x, y + i);
						}
						else {
							return;
						}
					}
				}
				return;
			}
		}
		miny = N / 2;
	}
}

//Функция вывода ответа на консоль
//Принимает масштаб квадратов (размер стороны)
void printing(int boardsize) {
	std::cout << final_array.size() << std::endl;
```

```
	for (auto i : final_array) {
		std::cout << i.x * boardsize << " " << i.y * boardsize << "
" << i.w * boardsize << std::endl;
	}
}

//Заполнение вектора тремя начальными квадратами
//Принимает вектор уже заполненных квадратов
void filling_basic_squares(std::vector<Square> &board) {
	Square s1(0, 0, (N + 1) / 2);
	board.push_back(s1);
	Square s2(0, (N + 1) / 2, N / 2);
	board.push_back(s2);
	Square s3((N + 1) / 2, 0, N / 2);
	board.push_back(s3);
}

//Масштабирование столешницы
//Принимает ссылку на размер стороны квадрата (масштаб квадрата)
//Итогом работы является вычисленный масштаб стороны квадрата и размер
столешницы
void scaling(int &boardsize) {
	for (int i = 1; i < N; i++) {
		if (N % i == 0) {
			boardsize = i;
		}
	}
	N /= boardsize;
}

//Решение задачи
//Инициализация основных переменных и вызывов функций
void solution() {
	std::cin >> N;

	square_counter = 2 * N + 1;
	int boardsize = 0;
	std::vector<Square> board;

	scaling(boardsize);
	filling_basic_squares(board);
	backtracking(board, pow((N + 1) / 2, 2) + 2 * pow(N / 2, 2), 3, N
/ 2, (N + 1) / 2);
	printing(boardsize);
}

//Главная функция
int main() {
	solution();
	return 0;
}
```