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 _____ Нгуен К.Х.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2019

Цель работы.

Исследование алгоритмов поиска с возвратом, реализация программы заполнения квадрата минимальным количеством квадратов.

Задание

У Вовы много квадратных обрезков доски. Их стороны (размер) изменяются от 11 до N−1N−1, и у него есть неограниченное число обрезков любого размера. Но ему очень хочется получить большую столешницу - квадрат размера N*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 NN(2≤N≤202≤N≤20).

Выходные данные

Одно число K, задающее минимальное количество обрезков(квадратов), из которых можно построить столешницу(квадрат) заданного размера NN. 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 стороны соответствующего обрезка(квадрата).

Экспериментальные результаты.

Была написана программа на языке c++, реализующая алгоритм заполнения. Были использованы функции:

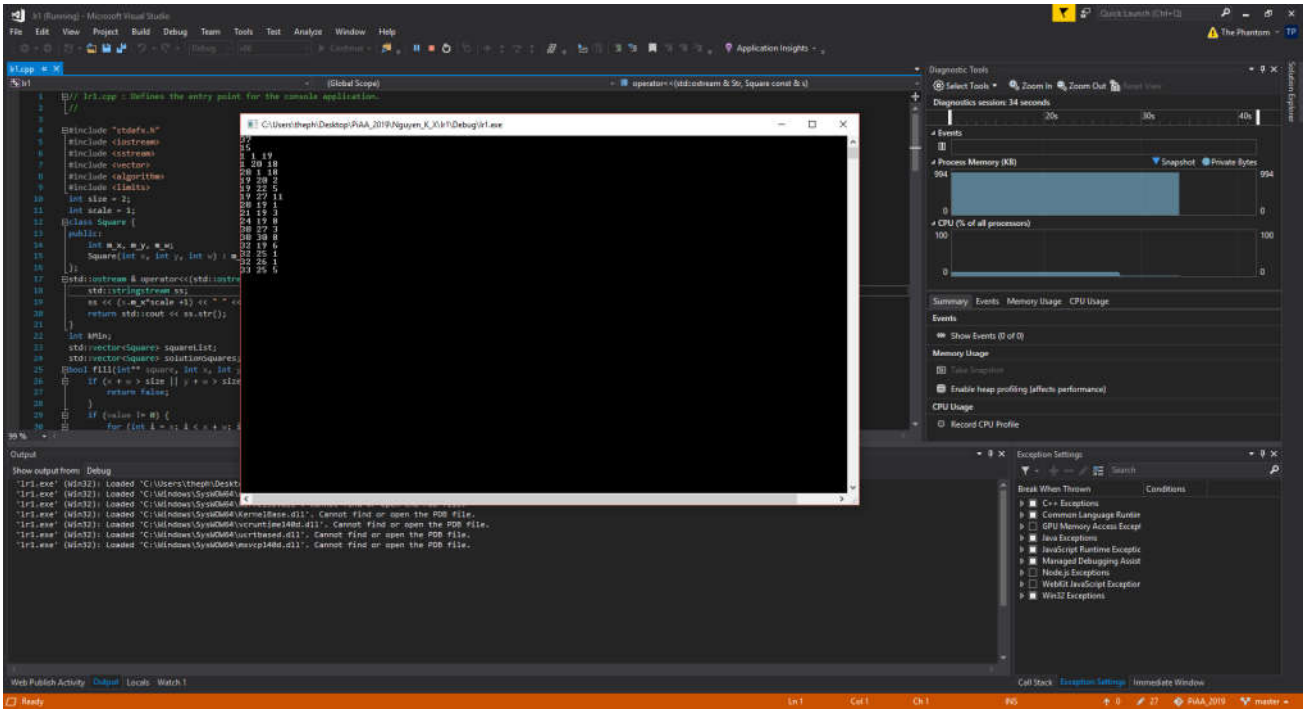
- Fill - изменить каждую ячейку массива в квадрате с координатой верхнего левого угла (x, y) и длиной стороны w на value
- Backtracking - найти количество квадратов и их расположение с помощью алгоритма поиска с возвратом

Код программы:

```
#include <iostream>
#include <sstream>
#include <vector>
#include <algorithm>
#include <limits>
int size = 2;
int scale = 1;
class Square {
public:
	int m_x, m_y, m_w;
	Square(int x, int y, int w) : m_x(x), m_y(y), m_w(w){}
};
std::ostream & operator<<(std::ostream & Str, Square const & s) {
	std::stringstream ss;
	ss << (s.m_x*scale +1) << " " << (s.m_y*scale +1) << " " << (s.m_w*scale);
	return std::cout << ss.str();
}
int kMin;
std::vector<Square> squareList;
std::vector<Square> solutionSquares;
bool fill(int** square, int x, int y, int w, int value) {
	if (x + w > size || y + w > size) {
		return false;
	}
	if (value != 0) {
		for (int i = x; i < x + w; i++) {
			for (int j = y; j < y + w; j++) {
				if (square[i][j] != 0) {
					return false;
				}
			}
		}
	}
	for (int i = x; i < x + w; i++) {
		for (int j = y; j < y + w; j++) {
			square[i][j] = value;
		}
	}
	return true;
}
void backtracking(int** square, int step) {
	if (step > kMin) {
		return;
	}
	bool startPointFound = false;
	int xRoot = 0, yRoot = 0;
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++) {
			if (square[i][j] == 0) {
				xRoot = i;
				yRoot = j;
				startPointFound = true;
				break;
			}
		}
		if (startPointFound) {
			break;
		}
	}
	if (!startPointFound) {
		//filled the square -> 1 solution
		if (step - 1 < kMin) {
			kMin = step - 1;
			solutionSquares = squareList;
```

```
                }
        }
        else {
                int maxSize = std::min(size - xRoot, size - yRoot);
                maxSize = std::min(maxSize, size - 1);
                for (int s = maxSize; s >= 1; s--) {
                        if (fill(square, xRoot, yRoot, s, step)) {
                                Square sNew (xRoot, yRoot, s);
                                squareList.push_back(sNew);
                                backtracking(square, step + 1);
                                squareList.pop_back();
                                fill(square, xRoot, yRoot, s, 0);
                        }
                }
        }
}
int main()
{
        static const int PRIMES[12] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 };
        int N_input;
        std::cin >> N_input;
        for (int i = 0; i < 12; i++) {
                if (N_input % PRIMES[i] == 0) {
                        size = PRIMES[i];
                        scale = N_input / PRIMES[i];
                        break;
                }
        }
        kMin = std::numeric_limits<int>::max();;
        int **board = new int*[size];
        for (int i = 0; i < size; i++) {
                board[i] = new int[size];
                for (int j = 0; j < size; j++) board[i][j] = 0;
        }

        fill(board, 0, 0, (size + 1) / 2, 1);
        fill(board, 0, (size + 1) / 2, size - (size + 1) / 2, 2);
        fill(board, (size + 1) / 2, 0, size - (size + 1) / 2, 3);
        squareList.push_back(Square(0, 0, (size + 1) / 2));
        squareList.push_back(Square(0, (size + 1) / 2, size - (size + 1) / 2));
        squareList.push_back(Square((size + 1) / 2, 0, size - (size + 1) / 2));

        backtracking(board, 4);

        std::cout<<kMin<<std::endl;
        for (Square s : solutionSquares) {
                std::cout << s << std::endl;
        }

        for (int i = 0; i < size; i++) {
                delete[] board[i];
        }
        delete[] board;
        getchar();
        getchar();
    return 0;
}
```

Результат работы программы



Выводы.

В результате работы программы был исследован алгоритм работы поиска с возвратом при помощи рекурсии. Также была написана программа, реализующая заполнение квадратной области минимальным количеством квадратов.