МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Построение и анализ алгоритмов»
Тема: Поиск с возвратом

Студент гр. 7383 _____ Александров Р.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Познакомиться с алгоритмом поиска с возвратом и его реализацией.

Основные теоретические положения.

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).

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.

Постановка задачи.

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).

Реализация задачи.

В ходе работы были написаны классы Runner, SquarePoints и Solution, в котором находятся необходимые методы.

Класс Runner начинает работу программы.

Класс SquarePoints хранит в себе координаты x и y, их геттеры и сеттеры.

Класса Solution хранит двумерный массив mainField, представляющий поле квадрата; переменные numBest и numWorking, считающие количество

квадратов в лучшем случае и в ходе работы соответственно; ArrayLists bestSln и workingSln, содержащие координаты x, y и размеры сторон квадратов. в лучшем случае и в ходе работы соответственно; пременную countComplexity, считающую количество операций в алгоритме для оценки сложности.

Метод void fillSquare(SquarePoints s, int sideSize) заполняет двумерный массив в выбранных координатах единицей, добавляет в ArrayList координаты x, y и размер стороны квадрата.

Метод void removeFilled(SquarePoints s, int sideSize) заполняет двумерный массив в выбранных координатах нулем, удаляет данные из ArrayList.

Метод SquarePoints findEmpty() возвращает объект SquarePoints, в котором точка квадрата по x и y равна нулю.

Метод boolean IsEmpty(int x, int y) проверяет координаты x и y в двумерном массиве на пустоту.

Метод boolean checkOversize(SquarePoints square) проверяет, вышли ли координаты x или y за размер квадрата.

Метод void findAllPlaces(), подставляет в двумерный массив квадраты разных размеров, от N-1 до 1, сохраняет лучшее решение.

Метод int getSmallerSize(SquarePoints square, int smallSquareSize) возвращает меньший размер квадрата, который можно вставить.

Метод void printAnswer() выводит ответ.

Метод void setThreeSquares() при инициализации объекта Solution ставит в двумерный массив 3 квадрата, стороны которых зависит от того, делится ли N на 2, на 3 или на 5. Первый квадрат ставится с верхнего левого угла, второй квадрат снизу и третий квадрат справа от первого.

Исследование алгоритма.

Количество операций считалось по вызову метода подстановки квадратов разных размеров от N-1 до 1. Результаты представлены в табл. 1.

Таблица 1 - Результаты работы алгоритма

Делимость	Длина стороны квадрата	Количество операций
На 2	2	2
	20	3
	26	3
На 3	12	3
	18	3
	21	65
На 5	5	16
	25	1123
	35	3680
Простые числа	17	7056
	29	475994
	43	34113375

Из исследования видно, что алгоритм поиска с возвратом не превышает значение 2^N .

Выводы.

В ходе лабораторной работы был изучен и реализован алгоритм поиска с возвратом.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

```
import java.util.Scanner;

public class Runner {

    public static void main(String[] args) {
        var in = new Scanner(System.in);
        System.out.println("Enter size:");
        var size = in.nextInt();
        if (size < 2) {
            System.out.println("N < 2");
            return;
        }

        var solution = new Solution(size);
        solution.findAllPlaces();
        solution.printAnswer();
//        solution.printMainField();
    }
}

import java.util.ArrayList;

public class Solution {
    private int[][] mainField;
    private ArrayList<Integer> workingSln;
    private ArrayList<Integer> bestSln;
    private int numBest;  // numbers of best squares
    private int numWorking; // numbers of working squares
    private int squareSize;

    private long countComplexity;

    public Solution(int squareSize) {
        this.squareSize = squareSize;
        mainField = new int[squareSize][squareSize];
        numBest = (squareSize * squareSize) + 1;
        workingSln = new ArrayList<>();
        bestSln = new ArrayList<>();
        setThreeSquares();
    }
```

```
    private void fillSquare(SquarePoints s, int sideSize) {
        workingSln.add(s.getX());
        workingSln.add(s.getY());
        workingSln.add(sideSize);
        numWorking++;
        for (int x = s.getX(); x < s.getX() + sideSize; x++) {
            for (int y = s.getY(); y < s.getY() + sideSize; y++) {
                mainField[x][y] = 1;
            }
        }
    }

    private void removeFilled(SquarePoints s, int sideSize) {
        workingSln.remove(workingSln.size()-1);
        workingSln.remove(workingSln.size()-1);
        workingSln.remove(workingSln.size()-1);
        numWorking--;
        for (int x = s.getX(); x < s.getX() + sideSize; x++) {
            for (int y = s.getY(); y < s.getY() + sideSize; y++) {
                mainField[x][y] = 0;
            }
        }
    }

    private SquarePoints findEmpty() {
        for (int x = 0; x < squareSize; x++) {
            for (int y = 0; y < squareSize; y++) {
                if (IsEmpty(x,y)) {
                    return new SquarePoints(x,y);
                }
            }
        }
        return new SquarePoints(squareSize + 1, squareSize + 1);
    }

    private boolean IsEmpty(int x, int y) {
        return mainField[x][y] == 0;
    }

    private boolean checkOversize(SquarePoints square) {
          return (square.getX() > squareSize) || (square.getY() >
squareSize);
    }

    public void findAllPlaces() {
        for (int i = squareSize -1; i >= 1; i--) {
```

```
        countComplexity++;
            var freeSquare = findEmpty();
            if (checkOversize(freeSquare)) {
                if (numWorking < numBest) {
                    bestSln.clear();
                    for (int j = 0; j < numWorking * 3; j++) {
                        bestSln.add(workingSln.get(j));
                    }
                    numBest = numWorking;
                }
                break;
            }
            if (numWorking == (numBest - 1)) break;
            i = getSmallerSize(freeSquare, i);
            var newFreeSquare = findEmpty();
            fillSquare(newFreeSquare, i);
//            System.out.println("Filled");
//            printMainField();
            findAllPlaces();
            removeFilled(freeSquare, i);
//            System.out.println("Removed");
//            printMainField();
        }
    }

    // get such small square as we can use
          private int getSmallerSize(SquarePoints square, int
smallSquareSize) {
        int x;
        int y;
        for (x = square.getX(); x < squareSize; x++) {
            if (!IsEmpty(x, square.getY())) {
                break;
            }
        }
        for (y = square.getY(); y < squareSize; y++) {
            if (!IsEmpty(square.getX(), y)) {
                break;
            }
        }
//        System.out.println("Filled");
//        printMainField();
        int diffrenceInX = x - square.getX();
        int diffrenceInY = y - square.getY();
            if (diffrenceInX < smallSquareSize || diffrenceInY <
smallSquareSize) {
```

```
            if (diffrenceInX > diffrenceInY) {
                smallSquareSize = diffrenceInY;
            } else {
                smallSquareSize = diffrenceInX;
            }
        }
        return smallSquareSize;
    }


    public void printMainField() {
        for (int x = 0; x < squareSize; x++) {
            for (int y = 0; y < squareSize; y++) {
                System.out.print(mainField[x][y] + " ");
            }
            System.out.println();
        }
    }

    public void printAnswer() {
//        System.out.print("Complexity = " + countComplexity);
        System.out.println(numBest);
        for (int i = 0; i < numBest * 3; i++) {
            System.out.print(bestSln.get(i) + 1);
            System.out.print(" ");
            System.out.print(bestSln.get(++i) + 1);
            System.out.print(" ");
           System.out.print(bestSln.get(++i));
          System.out.println();
        }
    }
    public void setThreeSquares() {
        int firstSetting;
        int secondSetting;
        if (squareSize % 2 == 0) {
            firstSetting = squareSize /2;
            secondSetting = squareSize /2;
        } else if (squareSize % 3 == 0) {
            firstSetting = (2 * squareSize) / 3;
            secondSetting = squareSize / 3;
        } else  if (squareSize % 5 == 0) {
            firstSetting = (3 * squareSize) / 5;
            secondSetting = (2 * squareSize) / 5;
        } else {
            firstSetting = (squareSize / 2) + 1;
            secondSetting = squareSize - ((squareSize / 2) + 1);
```

```
        }
        fillSquare(new SquarePoints(), firstSetting);
        var bottomSquare = new SquarePoints();
        bottomSquare.setX(firstSetting);
        var rightSquare = new SquarePoints();
        rightSquare.setY(firstSetting);
        fillSquare(bottomSquare, secondSetting);
        fillSquare(rightSquare, secondSetting);
    }
}

public class SquarePoints {

    private int x;
    private int y;

    public SquarePoints() {

    }

    public SquarePoints(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public void setY(int y) {
        this.y = y;
    }

    public int getY() {
        return y;
    }
}
```