Федеральное агенство связи
Ордена Трудового Красного Знамени федеральное государственное
бюджетное учреждение высшего образования
«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и
информационных технологий

Лабораторная работа №3

по дисциплине: «Методы поиска подстроки в строке.»

Выполнил студент
группы БФИ1902
Рахимов Е.К.
Проверила:
Мосева М.С.

Москва, 2021 г.

Оглавление

1. Цель лабораторной работы

Цель данной лабораторной работы — научиться использовать методы .

2. Задание на лабораторную работу

Задание 1

Реализовать методы поиска подстроки в строке. Добавить возможность ввода строки и подстроки с клавиатуры. Предусмотреть возможность существования пробела. Реализовать возможность выбора опции чувствительности или нечувствительности к регистру. Оценить время работы каждого алгоритма поиска и сравнить его со временем работы стандартной функции поиска, используемой в выбранном языке программирования.

Алгоритмы:

1. Кнута-Морриса-Пратта
2. Упрощенный Бойера-Мура

Задание 2 «Пятнашки»

Игра в 15, пятнашки, такен — популярная головоломка, придуманная в 1878 году Ноем Чепмэном. Она представляет собой набор одинаковых квадратных костяшек с нанесёнными числами, заключённых в квадратную коробку. Длина стороны коробки в четыре раза больше длины стороны костяшек для набора из 15 элементов, соответственно в коробке остаётся незаполненным одно квадратное поле. Цель игры — перемещая костяшки по коробке, добиться упорядочивания их по номерам, желательно сделав как можно меньше перемещений.

На рисунках выше изображены различные позиции элементов в задаче:

1. Левый рисунок — одна из возможных начальных позиций элементов.

2. Средний рисунок — одна из «нерешаемых» позиций.

3. Правый рисунок — позиция, где все элементы расставлены в правильном порядке.

Задача: написать программу, определяющую, является ли данное расположение «решаемым», то есть можно ли из него за конечное число шагов перейти к правильному. Если это возможно, то необходимо найти хотя бы одно решение - последовательность движений, после которой числа будут расположены в правильном порядке.

Входные данные: массив чисел, представляющий собой расстановку в порядке «слева направо, сверху вниз». Число 0 обозначает пустое поле. Например, массив [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0] представляет собой «решенную» позицию элементов.

Выходные данные: если решения нет, то функция должна вернуть пустой массив []. Если решение есть, то необходимо представить решение — для каждого шага записывается номер передвигаемого на данном шаге элемента.

Например, для начального расположения элементов [1, 2, 3, 4, 5, 6, 7, 8, 13, 9, 11, 12, 10, 14, 15, 0] одним из возможных решений будет [15, 14, 10, 13, 9, 10, 14, 15] (последовательность шагов здесь: двигаем 15, двигаем 14, двигаем 10, …, двигаем 15).

3. Ход лабораторной работы

3.1 Листинг программы

```
package com.company.Lab3;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;

public class StartLab3
{
    public static void main(String[] args) {

        String slovo="abcdsawdwfsdqw";
        String textForFind="dw";

        System.out.println("Кнут-Морр-Прт ="+
(Arrays.toString(KNUT_MOR_PRAT(slovo, textForFind).toArray())));
        System.out.println("Боер-Мур = "+ Boer_Mur(slovo,textForFind));


        int[][] blocks = new int[][]{{1, 2, 3, 0}, {5, 6, 7, 8}, {9, 10, 11,
12}, {13, 14, 15, 4}};
        Board initial = new Board(blocks);
        Solver solver = new Solver(initial);
        System.out.println("Minimum number of moves = " + solver.moves());
        for (Board board : solver.solution())
            System.out.println(board);
    }
    //Кнута-Морриса-Пратта
    static int[] prefFunc(String textForFind) {
        int [] values = new int[textForFind.length()];
        for (int i = 1; i < textForFind.length(); i++) {
            int j = 0;
            while (i + j < textForFind.length() && textForFind.charAt(j) ==
textForFind.charAt(i + j)) {
                values[i + j] = Math.max(values[i + j], j + 1);
                j++;
            }
        }
        return values;
    }
    public static ArrayList<Integer> KNUT_MOR_PRAT(String slovo, String
textForFind) {
        ArrayList<Integer> found = new ArrayList<>();

        int[] prefFunc = prefFunc(textForFind);
        int i = 0;
        int d = 0;
        while (i < slovo.length()) {
            if (textForFind.charAt(d) == slovo.charAt(i)) {
                d++;
                i++;
```

```
            }
            if (d == textForFind.length()) {
                found.add(i - d);
                d = prefFunc[d - 1];
            } else if (i < slovo.length() && textForFind.charAt(d) !=
slovo.charAt(i)) {
                if (d != 0) {
                    d = prefFunc[d - 1];
                } else {
                    i = i + 1;
                }
            }
        }

        return found;
    }


    /**
     * Возвращает индекс первого вхождения строки textForFind в строку Slovo,
или
     * -1, в случае если вхождение не найдено.
     *
     * @param Slovo
     *            исходная строка, в которой ищется вхождение шаблона.
     * @param textForFind
     *            шаблон строки, которая ищется в строке Slovo.
     * @return индекс первого вхождения строки textForFind в строку Slovo, или -
1,
     *         в случае если вхождение не найдено.
     */
    public static int Boer_Mur(String Slovo, String textForFind) {
        int sourceLen = Slovo.length();
        int templateLen = textForFind.length();
        if (templateLen > sourceLen) {
            return -1;
        }
        HashMap<Character, Integer> offsetTable = new HashMap<Character,
Integer>();
        for (int i = 0; i <= 255; i++) {
            offsetTable.put((char) i, templateLen);
        }
        for (int i = 0; i < templateLen - 1; i++) {
            offsetTable.put(textForFind.charAt(i), templateLen - i - 1);
        }
        int i = templateLen - 1;
        int j = i;
        int k = i;
        while (j >= 0 && i <= sourceLen - 1) {
            j = templateLen - 1;
            k = i;
            while (j >= 0 && Slovo.charAt(k) == textForFind.charAt(j)) {
                k--;
                j--;
            }
            i += offsetTable.get(Slovo.charAt(i));
        }
        if (k >= sourceLen - templateLen) {
```

```
            return -1;
        } else {
            return k + 1;
        }
    }




}
```

```
package com.company.Lab3;

import java.util.HashSet;
import java.util.Set;


public class Board {
    private int[][] blocks; //   Наше поле. пустое место будем обозначать нулем.
    private int zeroX;    // это нам пригодится в будущем - координаты нуля
    private int zeroY;
    private int h; //  мера

    public Board(int[][] blocks) {
        int[][] blocks2 = deepCopy(blocks);   //   копируем, так как нам нужно
быть уверенными в неизменяемости
        this.blocks = blocks2;

        h = 0;
        for (int i = 0; i < blocks.length; i++) {  //  в этом цикле определяем
координаты нуля и вычисляем h(x)
            for (int j = 0; j < blocks[i].length; j++) {
                if (blocks[i][j] != (i*dimension() + j + 1) && blocks[i][j] !=
0) {  // если 0 не на своем месте - не считается
                    h += 1;
                }
                if (blocks[i][j] == 0) {
                    zeroX = (int) i;
                    zeroY = (int) j;
                }
            }
        }
    }


    public int dimension() {
        return blocks.length;
    }

    public int h() {
        return h;
    }

    public boolean isGoal() {  //   если все на своем месте, значит это искомая
```

```
позиция
        return h == 0;
    }


    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Board board = (Board) o;

        if (board.dimension() != dimension()) return false;
        for (int i = 0; i < blocks.length; i++) {
            for (int j = 0; j < blocks[i].length; j++) {
                if (blocks[i][j] != board.blocks[i][j]) {
                    return false;
                }
            }
        }

        return true;
    }

    public Iterable<Board> neighbors() {  // все соседние позиции
        // меняем ноль с соседней клеткой, то есть всего 4 варианта
        // если соседнего нет (0 может быть с краю), chng(...) вернет null
        Set<Board> boardList = new HashSet<Board>();
        boardList.add(chng(getNewBlock(), zeroX, zeroY, zeroX, zeroY + 1));
        boardList.add(chng(getNewBlock(), zeroX, zeroY, zeroX, zeroY - 1));
        boardList.add(chng(getNewBlock(), zeroX, zeroY, zeroX - 1, zeroY));
        boardList.add(chng(getNewBlock(), zeroX, zeroY, zeroX + 1, zeroY));

        return boardList;
    }

    private int[][] getNewBlock() { //  опять же, для неизменяемости
        return deepCopy(blocks);
    }

    private Board chng(int[][] blocks2, int x1, int y1, int x2, int y2) {  //  в
этом методе меняем два соседних поля

        if (x2 > -1 && x2 < dimension() && y2 > -1 && y2 < dimension()) {
            int t = blocks2[x2][y2];
            blocks2[x2][y2] = blocks2[x1][y1];
            blocks2[x1][y1] = t;
            return new Board(blocks2);
        } else
            return null;

    }


    public String toString() {
        StringBuilder s = new StringBuilder();
        for (int i = 0; i < blocks.length; i++) {
            for (int j = 0; j < blocks.length; j++) {
```

```
                s.append(String.format("%2d ", blocks[i][j]));
            }
            s.append("\n");
        }
        return s.toString();
    }

    private static int[][] deepCopy(int[][] original) {
        if (original == null) {
            return null;
        }

        final int[][] result = new int[original.length][];
        for (int i = 0; i < original.length; i++) {
            result[i] = new int[original[i].length];
            for (int j = 0; j < original[i].length; j++) {
                result[i][j] = original[i][j];
            }
        }
        return result;
    }
}
package com.company.Lab3;

import java.util.*;


public class Solver {  //  наш "решатель"

    private Board initial;     //
    private List<Board> result = new ArrayList<Board>();    // этот лист -
цепочка ходов, приводящих к решению задачи

    private class ITEM{     // Чтобы узнать длину пути, нам нужно помнить
предидущие позиции (и не только поэтому)
        private ITEM prevBoard;  // ссылка на предыдущий
        private Board board;   // сама позиция

        private ITEM(ITEM prevBoard, Board board) {
            this.prevBoard = prevBoard;
            this.board = board;
        }

        public Board getBoard() {
            return board;
        }


    }

    public Solver(Board initial) {
        this.initial = initial;

        if(!isSolvable()) return;  //  сначала можно проверить, а решаема ли
задача?

        //  очередь. Для нахождения приоритетного сравниваем меры
        PriorityQueue<ITEM> priorityQueue = new PriorityQueue<ITEM>(10, new
```

```
Comparator<ITEM>() {
            @Override
            public int compare(ITEM o1, ITEM o2) {
                return new Integer(measure(o1)).compareTo(new
Integer(measure(o2)));
            }
        });


        // шаг 1
        priorityQueue.add(new ITEM(null, initial));

        while (true){
            ITEM board = priorityQueue.poll(); //  шаг 2

            //   если дошли до решения, сохраняем весь путь ходов в лист
            if(board.board.isGoal()) {
                itemToList(new ITEM(board, board.board));
                return;
            }

            //   шаг 3
            Iterator iterator = board.board.neighbors().iterator(); // соседи
            while (iterator.hasNext()){
                Board board1 = (Board) iterator.next();

                //оптимизация. Очевидно, что один из соседей - это позиция
                // которая была ходом раньше. Чтобы не возвращаться в состояния,
                // которые уже были делаем проверку. Экономим время и память.
                if(board1!= null && !containsInPath(board, board1))
                    priorityQueue.add(new ITEM(board, board1));
            }

        }
    }

    //  вычисляем f(x)
    private static int measure(ITEM item){
        ITEM item2 = item;
        int c= 0;   // g(x)
        int measure = item.getBoard().h();  // h(x)
        while (true){
            c++;
            item2 = item2.prevBoard;
            if(item2 == null) {
                // g(x) + h(x)
                return measure + c;
            }
        }
    }

    //  сохранение
    private void itemToList(ITEM item){
        ITEM item2 = item;
        while (true){
            item2 = item2.prevBoard;
            if(item2 == null) {
                Collections.reverse(result);
```

```
                return;
            }
            result.add(item2.board);
        }
    }

    // была ли уже такая позиция в пути
    private boolean containsInPath(ITEM item, Board board){
        ITEM item2 = item;
        while (true){
            if(item2.board.equals(board)) return true;
            item2 = item2.prevBoard;
            if(item2 == null) return false;
        }
    }


    public boolean isSolvable() {
        return true;
    }

    public int moves() {
        if(!isSolvable()) return -1;
        return result.size() - 1;
    }


    // все ради этого метода - чтобы вернуть result
    public Iterable<Board> solution() {
        return result;
    }


}
```

3.2 Результат выполнения программы

Рисунок 1 – результат выполнения

Список использованных источников

1) ГОСТ 7.32-2017 Система стандартов по информации, библиотечному и издательскому делу. Отчёт о научно-исследовательской работе. Структура и правила оформления

2) ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание. Общие требования и правила составления