Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени

«Московский технический университет связи и информатики»

Кафедра “Математической кибернетики и информационных технологий”

дисциплина «Структуры и алгоритмы обработки данных»

Отчет по лабораторной работе №3

“Методы поиска подстроки в строке”

Выполнил: студенты группы БФИ1901

Козырев Сергей Владимирович

Проверил:

Кутейников Иван Алексеевич

Москва 2021

Оглавление

1.Задание на лабораторную работу. 3
2.Листинг программы (Lab3) 4
2.1 Листинг программы Boyer_Mur. 4
2.2 Листинг программы Game. 5
2.3 Листинг программы Knutt_Moris_Pratt. 5
2.4 Листинг программы ReshGame. 7
2.5 Листинг программы Table. 9
3.Вывод 12
Список использованных источников 13

1.Задание на лабораторную работу.

Задание на лабораторную работу показано на рисунке 1.

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.

Рисунок 1 – Задание на работу.

2.Листинг программы (Lab3)

2.1 Листинг программы Boyer_Mur.

```
package Lab3;
import java.util.*;
import java.util.Arrays;
public class Boyer_Mur {
    public static void main(String[] args) {
        Boyer_Mur a = new Boyer_Mur();
        ArrayList <String> names = new ArrayList<>();
        Scanner sc = new Scanner (System.in);
        String str = sc.nextLine();
        String template = "ja";
        names.add(str);
        long start = 0;
        long stop = 0;

        start = System.nanoTime();
        for (int i = 0; i < names.indexOf(i); i++){
            names.get(i).indexOf("ja");
            if (names.get(i) == " "){
                i++;
            }
        }
        stop = System.nanoTime();
        System.out.println("IndexOf: " + (stop-start));
        a.getFirstEntry(str,template);
}
    public int getFirstEntry(String str, String template) {
        long start = 0;
        long stop = 0;
        start = System.nanoTime();
        int sourceLen = str.length();
        int templateLen = template.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(template.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 && str.charAt(k) == template.charAt(j)) {
                k--;
                j--;
            }
            i += offsetTable.get(str.charAt(i));
        }
        stop = System.nanoTime();
        System.out.println("Boyer - Mur: " + (stop-start));
        if (k >= sourceLen - templateLen) {
            return -1;
        } else {
            return k + 1;
        }

    }
}
```

2.2 Листинг программы Game.

```
package Lab3;
import java.util.Scanner;

public class Game {
    public static void main(String[] args) {
        int[][] blocks = new int[][]{{1, 2, 3, 0}, {5, 6, 7, 8}, {9, 10, 11, 12},
 {13, 14, 15, 4}};
        Table initial = new Table(blocks);
        ReshGame solver = new ReshGame(initial);
        System.out.println("Minimum number of moves = " + solver.moves());
        for (Table board : solver.solution())
            System.out.println(board);
    }
}
```

2.3 Листинг программы Knutt_Moris_Pratt.

```
package Lab3;
import java.util.Arrays;
import java.util.Scanner;
import java.util.*;
import java.util.stream.Collectors;

public class Knut_Morris_Pratt {

    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
```

```
        System.out.println("Введите основную строку:");
        String text = sc.nextLine();
        System.out.println("Введите подстроку для поиска:");
        String sample = sc.nextLine();
        System.out.println(Arrays.toString(prefixFunction(sample)));
        System.out.println("Индекс с которого начинается подстрока в строке:");
        System.out.println(Arrays.toString(KMPSearch(text, sample).toArray()));
    }
    static int[] prefixFunction(String sample) {
        int [] values = new int[sample.length()];
        for (int i = 1; i < sample.length(); i++) {
            int j = 0;
            while (i + j < sample.length() && sample.charAt(j) == sample.charAt(i
 + j)) {
                values[i + j] = Math.max(values[i + j], j + 1);
                j++;
            }
        }
        return values;
    }

    public static ArrayList<Integer> KMPSearch(String text, String sample) {
        ArrayList<Integer> found = new ArrayList<>();

        int[] prefixFunc = prefixFunction(sample);

        int i = 0;
        int j = 0;

        while (i < text.length()) {
            if (sample.charAt(j) == text.charAt(i)) {
                j++;
                i++;
            }
            if (j == sample.length()) {
                found.add(i - j);
                j = prefixFunc[j - 1];
            } else if (i < text.length() && sample.charAt(j) != text.charAt(i)) {
                if (j != 0) {
                    j = prefixFunc[j - 1];
                } else {
                    i = i + 1;
                }
            }
        }

        return found;
    }
}
```

2.4 Листинг программы ReshGame.

```
package Lab3;
import java.util.*;
//Решатель для Game

public class ReshGame {  //  наш "решатель"

    private Table initial;
    private List<Table> result = new ArrayList<Table>();
    private class ITEM{
        private ITEM prevBoard;
        private Table board;

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

        public Table getBoard() {
            return board;
        }


    }

    public ReshGame(Table initial) {
        this.initial = initial;

        if(!isSolvable()) return;

        PriorityQueue<ITEM> priorityQueue = new PriorityQueue<ITEM>(10, new Compa
rator<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
            if(board.board.isGoal()) {
                itemToList(new ITEM(board, board.board));
                return;
```

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

                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;
        int measure = item.getBoard().h();
        while (true){
            c++;
            item2 = item2.prevBoard;
            if(item2 == null) {
                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, Table 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;
    }


    // result
    public Iterable<Table> solution() {
        return result;
    }


}
```

2.5 Листинг программы Table.

```
package Lab3;
import java.util.HashSet;
import java.util.Set;
//Поле для игры

public class Table {
    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 Table(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;

        Table board = (Table) 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<Table> 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<Table> boardList = new HashSet<Table>();
        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 Table 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 Table(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;
    }
}
```

3.Вывод

Выполнив данную лабораторную работу, мы научились работать с методами поиска подстрок в строке и применять их в поставленных задачах, а именно, реализовывать поиски с помощью алгоритмов Кнута Морриса Пратта, Бойера Мура, а также реализовали игру Пятнашки.

Список использованных источников

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

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