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

Кафедра «Информатики»

Отчет по лабораторной работе №3

Выполнил: студент группы БВТ1901

Кускова А. Е.

Руководитель:

Мелехин А. А.

Москва 2021

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

Решение:

Реализуем заданные методы поиска, используя язык программирования Java.

Код алгоритма Кнута-Морриса-Пратта:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;


public class KMP_algoritm {
    public static void main(String[] args) {
        String text, sample;
        Scanner input = new Scanner(System.in);

        System.out.println("Введите строку: ");
        text = input.nextLine();
        System.out.println("Введите подстроку для поиска: ");
        sample = input.nextLine();

        System.out.println("Позиция первого вхождения: " +
       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;
            //пока не дошли до конца образца и если
            //символ в образце совпадает с символом начала образца
            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<>(); // - массив для
```

```
найденных вхождений
        // вычисляем префиксную функцию
        int[] prefixFunc = prefixFunction(sample);

        int i = 0, j = 0; // i - позиция внутри текста, j - внутри образца

        while (i < text.length()) {
            if (sample.charAt(j) == text.charAt(i)) {
                j++;
                i++;
            }
            if (j == sample.length()) { // если все символы образца совпали
                found.add(i - j); // записываем первый символ начала
вхождения
                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;
    }

    static ArrayList<Integer> search(String text, String sample){
        ArrayList<Integer> foundPosition = new ArrayList<>();
        for (int i = 0; i < text.length(); i++){
            int j = 0;
            while (j < sample.length() && i+j < text.length() &&
             sample.charAt(j) == text.charAt(i+j)){
                j++;
            }
            if (j == sample.length()){
                foundPosition.add(i);
            }
        }
        return foundPosition;
    }
}
```

Код алгоритма Бойера-Мура:

```
import java.util.Arrays;
import java.util.Scanner;
import java.util.TreeMap;

public class BM_algoritm {
    public static void main(String[] args) {
        String text, sample;
        Scanner input = new Scanner(System.in);

        System.out.println("Введите строку: ");
        text = input.nextLine();
        System.out.println("Введите подстроку для поиска: ");
        sample = input.nextLine();

        BM_algoritm(text, sample);
    }
```

```
    static void BM_algoritm(String text, String sample) {
        int textLen = text.length();
        int sampleLen = sample.length();

        TreeMap<Character, Integer> offsetTable = new TreeMap<Character,

        Integer>();

        for (int i = 0; i <= 255; i++) {
            offsetTable.put((char) i, sampleLen);
        }
        for (int i = 0; i < sampleLen - 1; i++) {
            offsetTable.put(sample.charAt(i), sampleLen - i - 1);
            System.out.println(sample.charAt(i) + "->" + (sampleLen- i - 1));
        }

        int i = sampleLen - 1;
        int j = i;
        int k = i;

        while (j >= 0 && i <= textLen - 1) {
            j = sampleLen - 1;
            k = i;
            while (j >= 0 && text.charAt(k) == sample.charAt(j)) {
                k -= 1;
                j -= 1;
            }
            i += offsetTable.get(text.charAt(i));
        }
        if (k >= textLen - sampleLen) {
            System.out.println("Не найдено");
        } else {
            System.out.println("Позиция первого вхождения: " + (k + 1));
        }
    }
}
```

Задание 2:

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.

Решение:

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

public class Board {
    public static void main(String[] args) {
        int[][] blocks = new int[][]{{1, 2, 3, 4}, {5, 6, 7, 8}, {13, 9, 11,
12}, {10, 14, 15, 0}};
        Board initial = new Board(blocks);
        Solver solver = new Solver(initial);

        System.out.println("Минимальное количество шагов = " +
```

```
solver.moves());
        System.out.println();
        for (Board board : solver.solution())
            System.out.println(board);

    }

    private int[][] blocks;
    private int zeroX;    // координаты нуля
    private int zeroY;
    private int h; //  мера

    public Board(int[][] blocks) {
        int[][] blocks2 = deepCopy(blocks);
        this.blocks = blocks2;
        h = 0;
        for (int i = 0; i < blocks.length; i++) {  //  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() {
        return deepCopy(blocks);
    }

    private Board chng(int[][] blocks2, int x1, int y1, int x2, int y2) {
        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;
    }
}


import java.util.*;


public class Solver {

    private Board initial;    //
    private List<Board> result = new ArrayList<Board>();
    private class ITEM{
        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();


                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;
        }

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

}
```

Вывод:

Реализовали алгоритмы Кнута-Морриса-Пратта и упрощенный Бойера-Мура, а также нахождение решения игры в пятнашки.