Федеральное агентство 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: «Структуры и алгоритмы обработки данных»

на тему: «Методы поиска подстроки в строке»

Выполнил студент
группы БФИ1902
Гусев Н. С.
Проверил:
Мкртчян Г. М.

Москва, 2021 г.

Оглавление

1. Цель работы

Цель работы: рассмотреть работу различных методов поиска подстроки в строке.

2. Задание на лабораторную работу

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: Кнута-Морриса-Пратта, упрощенный Бойера-Мура.
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.

3. Листинг программы

Алгоритм Бойера-Мура:

```
import java.util.*;

public class BMAlg {
    public static List<Integer> match(String pattern, String text) {
        List<Integer> matches = new ArrayList<Integer>();
        int m = text.length();
        int n = pattern.length();
        Map<Character, Integer> rightMostIndexes =
preprocessForBadCharacterShift(pattern);
        int alignedAt = 0;
        while (alignedAt + (n - 1) < m) {
            for (int indexInPattern = n - 1; indexInPattern >= 0;
indexInPattern--) {
                int indexInText = alignedAt + indexInPattern;
                char x = text.charAt(indexInText);
                char y = pattern.charAt(indexInPattern);
                if (indexInText >= m)
                    break;
                if (x != y) {
                    Integer r = rightMostIndexes.get(x);
                    if (r == null) {
                        alignedAt = indexInText + 1;
                    }
                    else {
```

```
                        int shift = indexInText - (alignedAt + r);
                        alignedAt += shift > 0 ? shift : 1;
                    }
                    break;
                }
                else if (indexInPattern == 0) {
                    matches.add(alignedAt);
                    alignedAt++;
                }
            }
        }
        return matches;
    }
    private static Map<Character, Integer> preprocessForBadCharacterShift(
            String pattern) {
        Map<Character, Integer> map = new HashMap<Character, Integer>();
        for (int i = pattern.length() - 1; i >= 0; i--) {
            char c = pattern.charAt(i);
            if (!map.containsKey(c)) map.put(c, i);
        }
        return map;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Введите строку:");
        String text = scanner.nextLine();
        System.out.println("Введите шаблон:");
        String pattern = scanner.nextLine();

        List<Integer> matches = match(pattern, text);
        for (Integer integer : matches)
            System.out.println("Шаблон найден на позиции: " + integer);
    }
}
```

Алгоритм Кнута-Морриса-Пратта:

```
import java.util.*;
import java.util.stream.Collectors;

public class KMPAlg {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Введите строку:");
        String text = scanner.nextLine();
        System.out.println("Введите шаблон:");
        String pattern = scanner.nextLine();

        List<Integer> foundIndexes = performKMPSearch(text, pattern);

        if (foundIndexes.isEmpty()) {
            System.out.println("Шаблон не найден");
        } else {
            System.out.println("Шаблон найден в позициях: " +
foundIndexes.stream().map(Object::toString).collect(Collectors.joining(",
")));
        }
    }

    public static int[] compilePatternArray(String pattern) {
        int patternLength = pattern.length();
        int len = 0;
        int i = 1;
        int[] compliedPatternArray = new int[patternLength];
        compliedPatternArray[0] = 0;
```

```
        while (i < patternLength) {
            if (pattern.charAt(i) == pattern.charAt(len)) {
                len++;
                compliedPatternArray[i] = len;
                i++;
            } else {
                if (len != 0) {
                    len = compliedPatternArray[len - 1];
                } else {
                    compliedPatternArray[i] = len;
                    i++;
                }
            }
        }
        System.out.println("Скомпилированный массив шаблона: " +
Arrays.toString(compliedPatternArray));
        return compliedPatternArray;
    }

    public static List<Integer> performKMPSearch(String text, String pattern)
{
        int[] compliedPatternArray = compilePatternArray(pattern);

        int textIndex = 0;
        int patternIndex = 0;

        List<Integer> foundIndexes = new ArrayList<>();

        while (textIndex < text.length()) {
            if (pattern.charAt(patternIndex) == text.charAt(textIndex)) {
                patternIndex++;
                textIndex++;
            }
            if (patternIndex == pattern.length()) {
                foundIndexes.add(textIndex - patternIndex);
                patternIndex = compliedPatternArray[patternIndex - 1];
            }

            else if (textIndex < text.length() &&
pattern.charAt(patternIndex) != text.charAt(textIndex)) {
                if (patternIndex != 0)
                    patternIndex = compliedPatternArray[patternIndex - 1];
                else
                    textIndex = textIndex + 1;
            }
        }
        return foundIndexes;
    }
}
```

Пятнашки:

```
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;
    }
}

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;
    }
}

public class Main {
    public static void main(String[] args) {
        int[][] blocks = new int[][]{{1, 2, 3}, {4, 0, 5}, {7, 8, 6}};
        Board initial = new Board(blocks);
        FifteenSolver solver = new FifteenSolver(initial);
        System.out.println("Минимальное количество ходов: " +
solver.moves());
        for (Board board : solver.solution())
            System.out.println(board);
    }
}
```

4. Результат работы программы

```
"C:\Program Files\Java\jdk1.8.0_261\bin\java.exe" ...
Минимальное количество ходов: 2
 1  2  3
 4  0  5
 7  8  6

 1  2  3
 4  5  0
 7  8  6

 1  2  3
 4  5  6
 7  8  0
```

Рисунок 1 – Пятнашки

```
"C:\Program Files\Java\jdk1.8.0_261\bin\java.exe" ...
Введите строку:
bananas
Введите шаблон:
ana
Скомпилированный массив шаблона: [0, 0, 1]
Шаблон найден в позициях: 1, 3
```

Рисунок 2 – Алгоритм Кнута-Морриса-Пратта

```
"C:\Program Files\Java\jdk1.8.0_261\bin\java.exe" ...
Введите строку:
bananas
Введите шаблон:
ana
Шаблон найден на позиции: 1
Шаблон найден на позиции: 3
```

Рисунок 3 – Алгоритм Бойера-Мура

5. Вывод

Я рассмотрел различные методы поиска подстроки в строке и научился работать с ними.