

Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Ордена Трудового Красного Знамени федеральное государственное
бюджетное образовательное учреждение высшего образования
“Московский технический университет связи и информатики”

Кафедра: "Математическая кибернетика и информационные
технологии"

Лабораторная работа №3 по дисциплине “Структуры и алгоритмы
обработки данных” по теме “Методы поиска подстроки в строке”

Выполнил студент
группы БФИ1902
вариант №9
Крутиков С.С.

г. Москва, 2021

Содержание

1	Задание на лабораторную работу	3
2	Ход выполнения лабораторной работы	5
2.1	Листинги программы	5
2.2	Результаты выполнения программы.....	12
Вывод	14

1 Задание на лабораторную работу

Задание на лабораторную работу представлено 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 лабораторную работу

2 Ход выполнения лабораторной работы

2.1 Листинги программы

Код в классе Search, в котором реализованы вышеуказанные методы поиска подстроки в строке представлен на листинге 1.

Листинг 1 – Код в классе Search

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

public class Search {

    public static void main(String[] args) {

        Scanner vvod = new Scanner(System.in);
        System.out.println("Введите строку: ");
        String text = vvod.next();

        System.out.println("Введите подстроку для поиска: ");
        String sample = vvod.next();

        System.out.println("Алгоритм Кнута-Морриса-Пратта");
        System.out.println(Arrays.toString(KMPSearch(text,
sample).toArray()));

        System.out.println("Алгоритм Бойера-Мура");
        long start = 0;
        long stop = 0;
        Search_BM a = new Search_BM();
        ArrayList <String> names = new ArrayList<>();

        Scanner sc = new Scanner (System.in);
        System.out.println("Введите основную строку:");
        String str = sc.nextLine();
        System.out.println("Введите подстроку для поиска:");
        String template = sc.nextLine();
        names.add(str);

        start = System.nanoTime();
        int index1 = str.indexOf("ja");
        System.out.println("Мы ищем букву 'ja' в строке "+ str + ". Индекс
данной буквы " + index1);
        stop = System.nanoTime();
        System.out.println("IndexOf: " + (stop-start));

        a.getFirstEntry(str,template);

    }

    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;
}

static class Search_BM{
    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;
    }
}
}
}

```

Код в классе Main, в котором происходит решения задачи с пятнашками представлен на листинге 2. Этот класс также использует классы Board (отвечает за создание доски с пятнашками) и Solver (отвечает за поиск решений).

Листинг 2 – Код в классе Main

```

import java.util.ArrayList;
import java.util.Collections;

public class Main {

    public static void main(String[] args) {
        //Генерация фишек
        Pyatnashki p = new Pyatnashki();
        int [][] pole = new int[4][4];
        pole = p.genetation();
        System.out.println();
        System.out.println(Pyatnashki.number_of_riots(pole));
        Board initial = new Board(pole);
        Solver solver = new Solver(initial);
        System.out.println("Minimum number of moves = " + solver.moves());
        for (Board board : solver.solution())
            System.out.println(board);
    }

    static class Pyatnashki{
        public static int[][] genetation(){
            int[][] mas = new int[4][4];
            ArrayList<Integer> pole = new ArrayList();
            int j = 0;

            for (int i = 0; i < 16; i++) {
                pole.add(j);
                j++;
            }

            Collections.shuffle(pole);
            int x=0;

            for (int i = 0; i < 4; i++) {
                System.out.println();
                for (int k = 0; k < 4; k++) {
                    mas[i][k]=pole.get(x);
                    x++;
                }
            }
        }
    }
}

```

```

    }
}

for (int i = 0; i < 4; i++) {
    System.out.println();
    for (int k = 0; k < 4; k++) {
        System.out.print(mas[i][k]+ " ");
    }
}
return mas;
}

public static int number_of_riots(int [][] mas){
    int number_riots = 0;
    int r = 0;
    int one = 0;
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            one = mas[i][j];
            if (one==0){
                r=i;
            }
            for (int k = j; k < 4; k++) {
                for (int l = i; l < 3; l++) {
                    if (one>mas[k][l] && mas[k][l]!=0){
                        number_riots++;
                    }
                    if (mas[k][l]==0 || mas[k][l+1]==0){
                        r=k;
                    }
                }
            }
        }
    }
    return number_riots+r+1;
}
}
}

```

Код в классе Board, который отвечает за создание доски с пятнашками, представлен на листинге 3.

Листинг 3 – Код в классе Board

```

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;
    }
}
```

Код в классе Solver, который отвечает за поиск подходящий решений, представлен на листинге 4.

Листинг 4 – Код в Solver

```
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
    // предыдущие 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 решается 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;
}

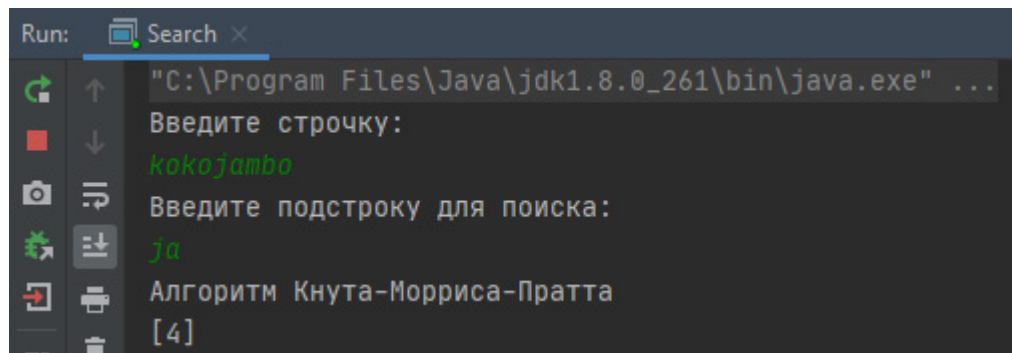
public int moves() {
    if (!isSolvable()) return -1;
    return result.size() - 1;
}

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

```

2.2 Результаты выполнения программы

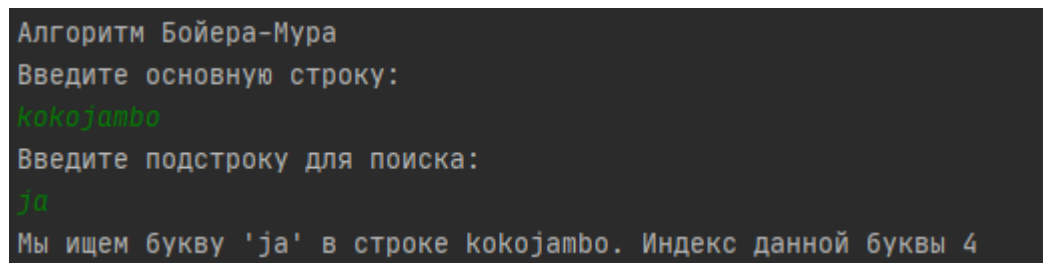
После ввода строки и подстроки для поиска программа выводит результат поиска с помощью алгоритма Кнута-Морриса-Пратта. Это показано на рисунке 2.



```
Run: Search ×
"C:\Program Files\Java\jdk1.8.0_261\bin\java.exe" ...
Введите строку:
kokojambo
Введите подстроку для поиска:
ja
Алгоритм Кнута-Морриса-Пратта
[4]
```

Рисунок 2 – Результат поиска с помощью Кнута-Морриса-Пратта

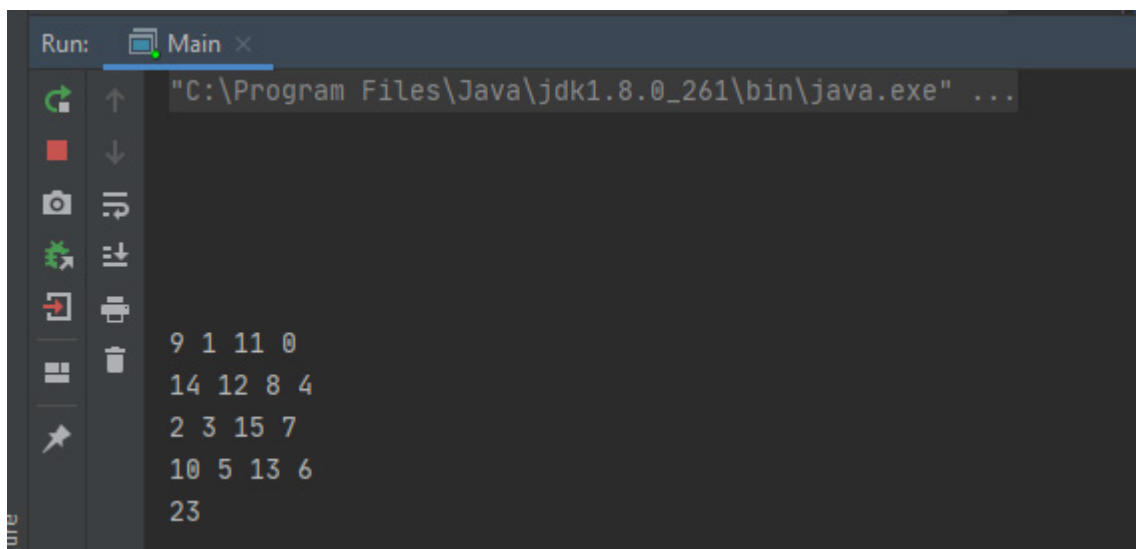
Затем происходит аналогичный запрос для метода Бойера-Мура. После ввода необходимых данных программа выводит результат. Это показано на рисунке 3.



```
Алгоритм Бойера-Мура
Введите основную строку:
kokojambo
Введите подстроку для поиска:
ja
Мы ищем букву 'ja' в строке kokojambo. Индекс данной буквы 4
```

Рисунок 3 – Результат поиска с помощью Бойера-Мура

Задача с пятнашками решает в классе Main. После запуска программы на экран выводится результат. Это показано на рисунке 4.



```
Run: Main ×
"C:\Program Files\Java\jdk1.8.0_261\bin\java.exe" ...

9 1 11 0
14 12 8 4
2 3 15 7
10 5 13 6
23
```

Рисунок 4 – Выполненное задание с пятнашками

Вывод

В результате выполнения данной лабораторной работы были получены навыки реализации алгоритмов поиска Кнута-Морриса-Пратта и Бойера-Мура.

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