**Федеральное агентство связи**

**Государственное бюджетное образовательное учреждение**

**высшего образования**

**ордена Трудового Красного Знамени**

**«Московский технический университет связи и информатики»**

Кафедра

«Математическая кибернетика и информационные технологии»

**Лабораторная работа №3**

**по дисциплине «Структура и алгоритмы обработки данных»**

Выполнил студент

Группы БФИ1902

Леонов Н.Н.

Научный руководитель: Мкртчян Г.М.

Москва 2021

# Методы поиска подстроки в строке

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:

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.

Ниже будет представлен код лабораторной работе

import java.util.\*;

public class Boyer\_Mur {

public static void main(String[] args) { long start = 0;

long stop = 0;

Boyer\_Mur a = new Boyer\_Mur();

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 строку:"); String template = sc.nextLine();

String t = template; names.add(str);

start = System.*nanoTime*(); int index1 = str.indexOf(t);

System.*out*.println("Мы ищем символы:"+ template+ "в строке "+ str +".

Индекс данных символов: " + index1); stop = System.*nanoTime*();

System.*out*.println("IndexOf: " + (stop-start)); a.*getFirstEntry*(str,template);

}

public static 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));

}

System.*out*.println("Мы ищем символы:"+ template+ "в строке "+ str +".

Индекс данных символов: " + (k+1)); stop = System.*nanoTime*();

System.*out*.println("Boyer - Mur: " + (stop-start));

if (k >= sourceLen - templateLen) { return -1;

} else {

return k + 1;

}

}

}

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

public class KMPSearch {

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("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<>();

long start = 0; long stop = 0;

int[] prefixFunc = *prefixFunction*(sample);

int i = 0; int j = 0;

start = System.*nanoTime*(); 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;

}

}

}

stop = System.*nanoTime*(); System.*out*.println("KMPSearch: " + (stop-start)); return found;

}

}

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.\*;

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()){

состояния,

}

}

Board board1 = (Board) iterator.next();

//оптимизация. Очевидно, что один из соседей - это позиция

// которая была ходом раньше. Чтобы не возвращаться в

// 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;

}

}

public class Main {

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}};

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);

}

}

Ниже будет представлен результат выполнения работы на рисунках 1 - 4

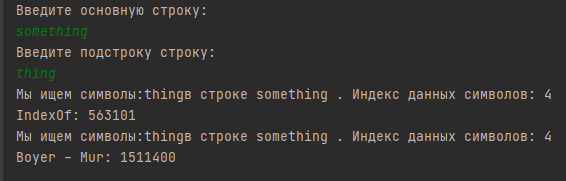


Рисунок 1 – Алгоритм Бойера – Мура

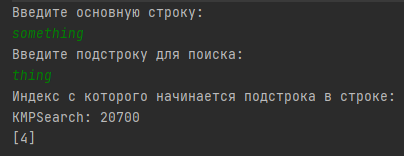


Рисунок 2 – Алгоритм Кнута – Морриса – Пратта

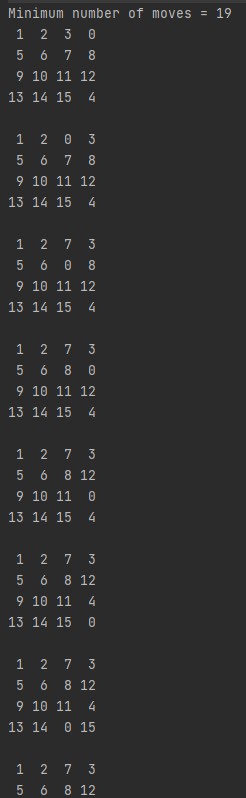


Рисунок 3 – Пятнашки

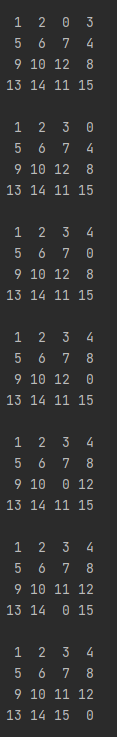


Рисунок 4 - Пятнашки