Министерство цифрового развития, связи и массовых коммуникаций

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»



Отчет по лабораторной работе №3

по дисциплине «Структура и алгоритмы обработки данных»

по теме «Методы поиска подстроки в строке»

Выполнила: студент группы

БВТ1902

Соколова А.Ю.

Москва

2021 г

Оглавление

[Цель работы 3](#_Toc72677660)

[Задание 1 3](#_Toc72677661)

[Код программы 3](#_Toc72677662)

[Задание 2 «Пятнашки» 3](#_Toc72677663)

[Код программы 3](#_Toc72677664)

[Снимки экрана работы программ 3](#_Toc72677665)

[Вывод 3](#_Toc72677666)

## Цель работы

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.

## Задание 1

Алгоритмы:

1. Кнута-Морриса-Пратта
2. Упрощенный Бойера-Мура

## Код программы

public class BM {

static int NO\_OF\_CHARS = 256;

static int max (int a, int b) { return (a > b)? a: b; }

static void badCharHeuristic( char []str, int size,int badchar[])

{

for (int i = 0; i < NO\_OF\_CHARS; i++)

badchar[i] =size;

// Заполните фактическое значение последнего появления символа (индексы таблицы - ascii, а значения - индекс появления)

for (int i = 0; i < size; i++)

badchar[(int) str[i]] = i;

}

/\* Функция поиска по шаблону \*/

static void search( char txt[], char pat[])

{

int m = pat.length;

int n = txt.length;

int badchar[] = new int[NO\_OF\_CHARS];

/\* Заполните массив неверных символов,

вызвав функцию предварительной обработки

badCharHeuristic () для данного шаблона \*/

badCharHeuristic(pat, m, badchar);

int s = 0; // сдвиг шаблона относительно текста

while(s <= (n - m))

{

int j = m-1;

/\* Продолжайте уменьшать индекс j шаблона,

пока символы шаблона и текста совпадают

на этом сдвиге s \*/

while(j >= 0 && pat[j] == txt[s+j])

j--;

/\* Если шаблон присутствует в текущем сдвиге,

то индекс j станет -1 после указанного выше цикла. \*/

if (j < 0)

{

System.out.println("Patterns occur at shift = " + s);

/\* Перенести шаблон так, что следующий символ в тексте

совпадет с последним вхождением его в шаблоне. Условие

s + m <n необходимо для случая, когда в конце текста

встречается шаблон. \*/

s += (s+m < n)? m-badchar[txt[s+m]] : 1;

}

else

/\*Сдвиньте шаблон так, чтобы плохой символ в тексте совпал с

последним его появлением в шаблоне. Функция max используется,

чтобы убедиться, что мы получаем положительный сдвиг. Мы можем

получить отрицательный сдвиг, если последнее появление плохого

символа в шаблоне находится справа от текущего символа. \*/

s += max(1, j - badchar[txt[s+j]]);

}

}

public static void main(String[] args) {

long start = System.nanoTime();

Scanner scan=new Scanner(System.in);

System.out.println("Введите подстроку");

String pattern = scan.nextLine();

System.out.println("Введите строку");

String text = scan.nextLine();

System.out.println("Введите 0 или 1 для выбора опции чувствительности или нечувствительности к регистру");

int k = scan.nextInt();

if (k==0) {

char pat[] = pattern.toCharArray();

char txt[] = text.toCharArray();

search(txt, pat);

} else{

String pattern1 = pattern.toLowerCase();

String text1 = text.toLowerCase();

char pat[] = pattern1.toCharArray();

char txt[] = text1.toCharArray();

search(txt, pat);

}

long finish = System.nanoTime();

long elapsed = finish - start;

System.out.println("Прошло времени, мс: " + elapsed/ 1000000);

}

}

public class KMP {

//создание массива шаблонов.

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++;

}

}

}

return compliedPatternArray;

}

//сравнение символов в шаблоне и текстовом массиве.

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;

}

public static void main(String[] args) {

long start = System.nanoTime();

Scanner scan=new Scanner(System.in);

System.out.println("Введите подстроку");

String pattern = scan.nextLine();

System.out.println("Введите строку");

String text = scan.nextLine();

System.out.println("Введите 0 или 1 для выбора опции чувствительности или нечувствительности к регистру");

int k = scan.nextInt();

if (k==0) {

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

if (foundIndexes.isEmpty()) {

System.out.println("Подстрока не найдена в данной строке");

} else {

System.out.println("Подстрока найдена в данной строке на позициях: " + foundIndexes);

}

} else{

pattern = pattern.toLowerCase();

text = text.toLowerCase();

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

if (foundIndexes.isEmpty()) {

System.out.println("Подстрока не найдена в данной строке");

} else {

System.out.println("Подстрока найдена в данной строке на позициях: " + foundIndexes);

}

}

long finish = System.nanoTime();

long elapsed = finish - start;

System.out.println("Прошло времени, мс: " + elapsed/ 1000000);

}

}

## Задание 2 «Пятнашки»

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.

## Код программы

public class Main {

public static void main(String[] args) {

long start = System.nanoTime();

int[] arr = {5, 1, 2, 3, 9, 6, 7, 4, 13, 10, 11, 8, 14, 15, 0, 12};

//int[] arr = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0};

//int[] arr = {7, 3, 5, 12, 6, 8, 14, 13, 2, 11, 9, 1, 0, 10, 4, 15};

// int[] arr = {1, 2, 3, 4, 5, 6, 7, 8, 13, 9, 11, 12, 10, 14, 15, 0};

// int[] arr = {5, 1, 3, 4, 0, 2, 6, 8, 7, 10, 15, 11, 9, 13, 14, 12};

// int[] arr = {11, 9, 4, 6, 3, 15, 7, 13, 2, 10, 0, 8, 5, 12, 1, 14};

// int[] arr = {5, 9, 8, 14, 0, 6, 12, 3, 13, 11, 1, 10, 15, 2, 7, 4};

// int[] arr = {7, 1, 4, 15, 10, 12, 3, 14, 5, 6, 0, 11, 2, 13, 8, 9};

// int[] arr = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,0};

double inv = 0;

for (int i = 0; i < 16; i++) {

if (arr[i] != 0)

for (int j = 0; j < i; ++j)

if (arr[j] > arr[i])

inv += 1;

}

for (int i = 0; i < 16; ++i) {

if (arr[i] == 0)

inv += 1 + i / 4;

}

int[][] arr1 = new int[4][4];

int k = 0;

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

arr1[i][j] = arr[k];

k++;

}

}

if(inv%2==0) {

System.out.println("Решение есть");

new graphSearch(arr1);

}

else{

System.out.println("Нет решения");

}

long finish = System.nanoTime();

long elapsed = finish - start;

System.out.println("Прошло времени, мс: " + elapsed/ 1000000);

}

}

class attempts {

public int[][] array;

public ArrayList<Integer> path = new ArrayList<>();

public int opt;

public attempts(int[][] arr, ArrayList<Integer> a, int b) {

this.array = arr;

this.path = a;

this.opt = b;

}

}

public class graphSearch {

public static boolean mrt(int[][] a, int[][] b) {

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

if (a[i][j] != b[i][j]) {

return false;

}

}

}

return true;

}

public static int[][] Go(int[][] a) {

int[][] b = new int[4][4];

for (int i = 0; i < 4; i++) {

for(int j =0;j<4;j++){

b[i][j]=a[i][j];

}

}

return b;

}

public static ArrayList<Integer> Go1(ArrayList<Integer> a) {

ArrayList<Integer> b = new ArrayList<>();

for (int i = 0; i <a.size(); i++) {

b.add(a.get(i));

}

return b;

}

public static boolean finder(ArrayList<attempts> array, int[][] sought) {

for (int i = 0; i < array.size(); i++) {

if (mrt(array.get(i).array, sought)) {

return false;

}

}

return true;

}

public static int optimal(int[][] array) {

int counter = 0;

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

for (int l = 0; l < 4; l++) {

if (array[0][l] == (4 \* i + j + 1)) {

counter += Math.abs(i) + Math.abs(j - l);

}

if (array[1][l] == (4 \* i + j + 1)) {

counter += Math.abs(i - 1) + Math.abs(j - l);

}

if (array[2][l] == (4 \* i + j + 1)) {

counter += Math.abs(i - 2) + Math.abs(j - l);

}

if (array[3][l] == (4 \* i + j + 1)) {

counter += Math.abs(i - 3) + Math.abs(j - l);

}

}

}

}

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 3; j++) {

if (array[i][j] > array[i][j + 1] && array[i][j] != 0 && array[i][j + 1] != 0) {

counter += 2;

}

}

}

//

return counter;

}

public graphSearch(int[][] arr1) {

int[][] answer = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 0}};

ArrayList<attempts> queue = new ArrayList<>();

ArrayList<Integer> a = new ArrayList<>(0);

attempts quese1 = new attempts(arr1, a, 0);

queue.add(quese1);

int l = 0;

ArrayList<attempts> chekPosition = new ArrayList<>();

while (true) {

attempts current;

current = queue.remove(0);

chekPosition.add(current);

l++;

if (mrt(current.array, answer)) {

System.out.println(current.path.size());

System.out.println(current.path);

return;

}

int[] indexOfZeros = {0, 0};

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++)

if (current.array[i][j] == 0) {

indexOfZeros[0] = i;

indexOfZeros[1] = j;

break;

}

}

if (indexOfZeros[0] < 3 ) {

int[][] newArray = Go(current.array);

newArray[indexOfZeros[0]][indexOfZeros[1]] = newArray[indexOfZeros[0] + 1][indexOfZeros[1]];

newArray[indexOfZeros[0] + 1][indexOfZeros[1]] = 0;

int action = newArray[indexOfZeros[0]][indexOfZeros[1]];

ArrayList<Integer> newPath = Go1(current.path);

newPath.add(action);

if (finder(chekPosition, newArray) && finder(queue, newArray)) {

queue.add(new attempts(newArray, newPath, optimal(newArray)));

}

}

if (indexOfZeros[0] > 0 ) {

int[][] newArray = Go(current.array);

newArray[indexOfZeros[0]][indexOfZeros[1]] = newArray[indexOfZeros[0] - 1][indexOfZeros[1]];

newArray[indexOfZeros[0] - 1][indexOfZeros[1]] = 0;

int action = newArray[indexOfZeros[0]][indexOfZeros[1]];

ArrayList<Integer> newPath = Go1(current.path);

newPath.add(action);

if (finder(chekPosition, newArray) && finder(queue, newArray)) {

queue.add(new attempts(newArray, newPath, optimal(newArray)));

}

}

if (indexOfZeros[1] < 3 ) {

int[][] newArray = Go(current.array);

newArray[indexOfZeros[0]][indexOfZeros[1]] = newArray[indexOfZeros[0]][indexOfZeros[1] + 1];

newArray[indexOfZeros[0]][indexOfZeros[1] + 1] = 0;

int action = newArray[indexOfZeros[0]][indexOfZeros[1]];

ArrayList<Integer> newPath = Go1(current.path);

newPath.add(action);

if (finder(chekPosition, newArray) && finder(queue, newArray)) {

queue.add(new attempts(newArray, newPath, optimal(newArray)));

}

}

if (indexOfZeros[1] > 0 ) {

int[][] newArray = Go(current.array);

newArray[indexOfZeros[0]][indexOfZeros[1]] = newArray[indexOfZeros[0]][indexOfZeros[1] - 1];

newArray[indexOfZeros[0]][indexOfZeros[1] - 1] = 0;

int action = newArray[indexOfZeros[0]][indexOfZeros[1]];

ArrayList<Integer> newPath = Go1(current.path);

newPath.add(action);

if (finder(chekPosition, newArray) && finder(queue, newArray)) {

queue.add(new attempts(newArray, newPath, optimal(newArray)));

}

}

queue.sort(new Comparator<attempts>() {

@Override

public int compare(attempts o1, attempts o2) {

return o1.opt - o2.opt;

}

});

}

}

}

## Снимки экрана работы программ

Задача 1

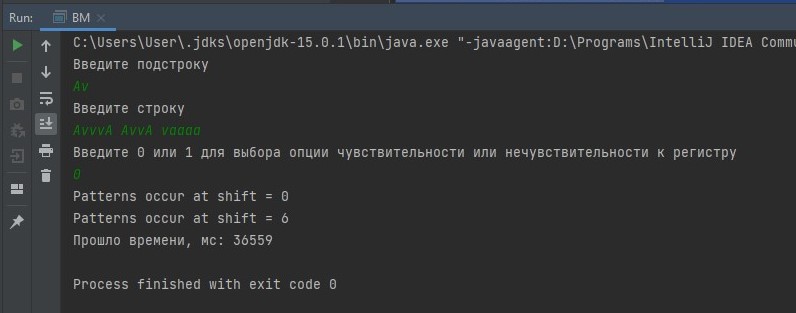


Рисунок 1 – Результат выполнения КМП

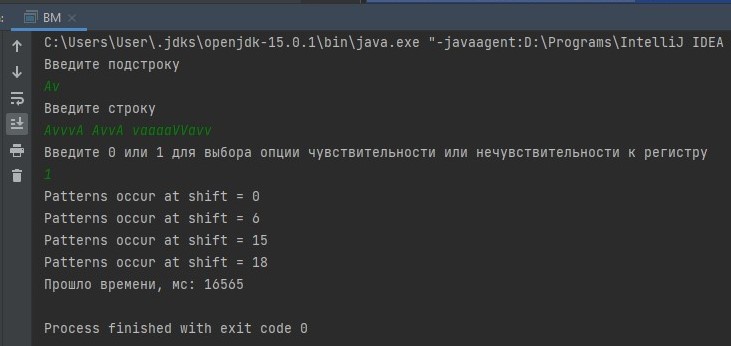


Рисунок 2 – Результат выполнения КМП

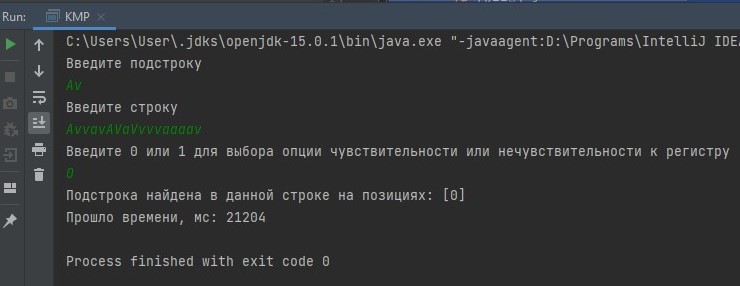


Рисунок 3 – Результат выполнения ВМ

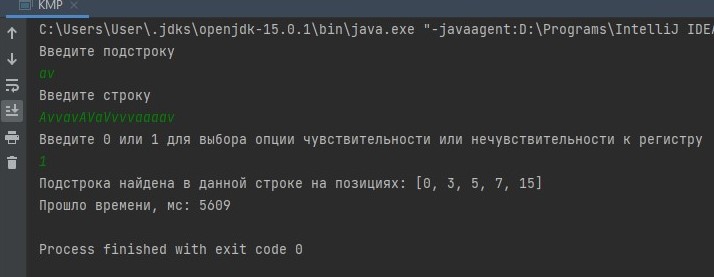


Рисунок 4 – Результат выполнения ВМ

Задача2

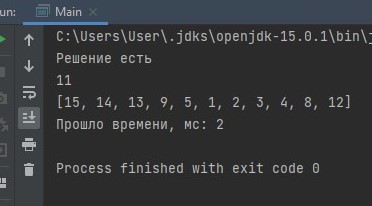


Рисунок 5

## Вывод

Я реализовала методы поиска подстроки в строке, добавила возможность ввода строки и подстроки с клавиатуры, предусмотрела возможность существования пробела, реализовала возможность выбора опции чувствительности или нечувствительности к регистру и оценила время работы каждого алгоритма поиска. А так же написала алгоритм решения «пятнашек».