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 «Математической кибернетики и информационных технологий»

Лабораторная работа №3 по дисциплине

«Структуры и алгоритмы обработки данных»

Выполнил:

студент группы БВТ1902

Клюев А.П.

Введение

Цель данной лабораторной – получить знания и навыки реализовав

методы поиска подстроки в строке. Добавив возможность ввода строки и

подстроки с клавиатуры. Предусмотрев возможность существования пробела.

Реализовав 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

Так же требуется написать программу «Пятнашки», 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.

Листинг программы

Класс Substring

import java.util.Arrays;  
import java.util.Scanner;  
  
public class Substring {  
  
 public static void main(String[] args) {  
 System.*out*.println("print string");  
 Scanner scanner = new Scanner(System.*in*);  
 String str = scanner.nextLine();  
 System.*out*.println("print substring");  
 String sub = scanner.nextLine();  
 System.*out*.println(*findStringKMP*(str, sub));  
 System.*out*.println(*findStringBM*(str, sub));  
 }  
  
 public static int findStringKMP(String str, String sub) {  
 long t1 = System.*nanoTime*();  
 char[] S = str.toCharArray();  
 char[] T = sub.toCharArray();  
 int[] prefix = *prefixFunc*(sub);  
 int index = 0;  
 for (int i = 0; i < str.length(); i++) {  
 if (S[i] == T[index]) {  
 index++;  
 } else {  
 if (index != 0)  
 index = prefix[index - 1];  
 }  
 if (index == sub.length()) {  
 long t2 = System.*nanoTime*();  
 System.*out*.println("KMP: "+ (t2-t1));  
 return i - index + 1;  
 }  
 }  
 long t2 = System.*nanoTime*();  
 System.*out*.println("KMP: "+ (t2-t1));  
 return -1;  
  
 }  
  
  
  
 public static int findStringBM(String str, String sub) {  
 long t1 = System.*nanoTime*();  
 char[] S = str.toCharArray();  
 char[] T = sub.toCharArray();  
 int[] basis = *basisFunc*(sub);  
 int index = basis.length-1;  
 int i = sub.length()-1;  
 while (i < str.length()){  
 if(S[i]==T[index]){  
 int tmp = index;  
 for(int j = i;j>=i-tmp;j--){  
 if(S[j]==T[index]){  
 index--;  
 }else{  
 i+=basis[sub.indexOf(S[j])]-1;  
 break;  
 }  
 if((index==0)){  
 if((S[j-1]==T[0])){  
 long t2 = System.*nanoTime*();  
 System.*out*.println("BM: "+ (t2-t1));  
 return j-1;  
 }else{  
 if(sub.indexOf(S[i])==-1) {  
 i+=sub.length();  
 break;  
 }else {  
 i += basis[sub.indexOf(S[i])];  
 break;  
 }  
 }  
 }  
 }  
 index = basis.length-1;  
 }else{  
 if(sub.indexOf(S[i])==-1) {  
 i+=sub.length();  
 }else {  
 i += basis[sub.indexOf(S[i])];  
 }  
 }  
 }  
 long t2 = System.*nanoTime*();  
 System.*out*.println("BM: "+ (t2-t1));  
 return -1;  
 }  
  
 public static int[] basisFunc(String sub) {  
 int[] result = new int[sub.length()];  
 int tmp = 1;  
 result[sub.length() - 2] = tmp;  
 for (int i = sub.length() - 3; i > -1; i--) {  
 for (int j = sub.length() - 2; j > i; j--) {  
 if (sub.charAt(i) == sub.charAt(j)) {  
 result[i] = result[j];  
 break;  
 }  
 }  
 if (result[i] ==0) result[i] = ++tmp;  
 }  
 int ind = sub.length()-1;  
 for (int j = 0; j < sub.length()-1; j++) {  
 if(sub.charAt(ind)==sub.charAt(j)){  
 result[ind]=result[j];  
 }  
 }  
 if(result[ind]==0) result[ind]=sub.length();  
 return result;  
 }

public static int[] prefixFunc(String sub) {  
 int len = sub.length();  
 int[] result = new int[len];  
 String str = "" + sub.charAt(0);  
 String prefix = "";  
 String suffix = "";  
 int maxl = 0;  
 result[0] = 0;  
 for (int k = 1; k < len; k++) {  
 str += sub.charAt(k);  
 for (int i = 0; i < str.length() -1; i++) {  
 prefix += str.charAt(i);  
 suffix = str.charAt(str.length() - 1 - i) + suffix;  
 if (suffix.equals(prefix)) {  
 maxl = prefix.length();  
 } else break;  
 }  
 result[k] = maxl;  
 maxl = 0;  
 suffix = "";  
 prefix = "";  
 }  
 return result;  
 }  
  
}

Класс Pyatnashki

public class Pyatnashki {  
 public static void main(String[] args) {  
 int[] arr = {5, 1, 2, 3, 9, 6, 7, 4, 13, 10, 11, 8, 14, 15, 0, 12};  
  
 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(inv);  
 new graphSearch(arr1);  
 }  
 else{  
 System.*out*.println("-1");  
 }  
 }  
}

Класс graphSearch

import java.util.\*;  
  
class attempts {  
 public int[][] array;  
 public ArrayList<Integer> path = new ArrayList<>();  
 //lastAct: act,  
 public int opt;  
  
 public attempts(int[][] arr, ArrayList<Integer> a, int b) {  
 this.array = arr;  
 this.path = a;  
 //lastAct: act,  
 this.opt = b;// you can access  
 }  
}  
  
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 void Vivod(int[][] a) {  
 for (int i = 0; i < 4; i++) {  
 System.*out*.println(Arrays.*toString*(a[i]));  
 }  
 }  
 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;  
// kon[i][j] = 1;  
 }  
 }  
 }  
  
 return counter;  
 }  
  
 public graphSearch(int[][] arr1) {  
 //Vivod(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<>();  
 //System.out.println(optimal(arr1));  
 while (true) {  
 attempts current;  
 current = queue.remove(0);  
 chekPosition.add(current);  
 l++;  
// System.out.println(l);  
 if (*mrt*(current.array, answer)) {  
 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;  
 }  
 }  
  
 //System.out.println(Arrays.toString(indexOfZeros));  
 if (indexOfZeros[0] < 3 /\*&& current.act != = 2\*/) {  
 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 /\*&& current.act != = 1\*/) {  
 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 /\*&& current.act != = 4\*/) {  
 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 /\*&& current.act != = 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)));  
 }  
 }  
 queue.sort(new Comparator<attempts>() {  
 @Override  
 public int compare(attempts o1, attempts o2) {  
 return o1.opt - o2.opt;  
 }  
 });  
  
 }  
 }  
}

Вывод

В данной лабораторной работе я реализовал 2 метода поиска подстроки в строке и решение пятнашек.