Министерство цифрового развития, связи и массовых коммуникаций Государственное образовательного учреждение высшего образования

Ордена Трудового Красного Знамени

«Московский технический университет связи и информатики»

Лабораторная работа № 3

«Методы поиска подстроки в строке»

Выполнил студент

группы БВТ1902

Шульпина Полина

Москва

2021

**Задание**

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  
3. Стандартная функция поиска

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.

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.

Например, для начального расположения элементов [1, 2, 3, 4, 5, 6, 7, 8, 13, 9, 11, 12, 10, 14, 15, 0] одним из возможных решений будет [15, 14, 10, 13, 9, 10, 14, 15] (последовательность шагов здесь: двигаем 15, двигаем 14, двигаем 10, ..., двигаем 15).

# Выполнение

Листинг программы:

package ru.shulpinchik;  
  
import java.util.\*;  
  
public class Lab3 {  
 public static int SearchString(String string, String word) {  
 int[] array = *PrefixFunction*(word);  
 int index = 0;  
 int i;  
 for (i = 0; i < string.length(); i++){  
 if (string.charAt(i) == word.charAt(index)){  
 index += 1;  
 } else if (index > 0){  
 index = array[index - 1];  
 }  
 if (index == word.length()){  
 return i - word.length()+1;  
 }  
 }  
 return -1;  
 }  
  
 public static int[] PrefixFunction(String word) {  
 String prefix = "";  
 String suffix = "";  
 int[] array = new int[word.length()]; //strvpf  
 int max = 0;  
 String lol = "" + word.charAt(0);  
 array[0] = 0;  
 for (int i = 1; i < word.length(); i++) {  
 lol = lol + word.charAt(i);  
 for (int j = 0; j < word.length() / 2 + 1; j++) {  
 prefix = prefix + word.charAt(j);  
 suffix = word.charAt(word.length() - 1 - j) + suffix;  
 if (prefix.equals(suffix)) {  
 max = prefix.length();  
 } else{  
 break;  
 }  
 }  
 array[i] = max;  
 max = 0;  
 }  
 return array;  
 }  
  
  
 public static int BMSearch(String T, String P) {  
 int i = P.length() - 1;  
 int j = P.length() - 1;  
 do {  
 if (P.charAt(j) == T.charAt(i)) {  
 if (j == 0) {  
 return i; // возвращаем индекс первого вхождения строки  
 } else {  
 i--;  
 j--;  
 }  
 } else {  
 i = i + P.length() - *min*(j, 1 + *last*(T.charAt(i), P));  
 j = P.length() - 1;  
 }  
 } while (i <= T.length() - 1);  
  
 return -1; // возвращаем -1, если вхождение не найдено  
 }  
  
 public static int last(char c, String P) { // берем символ и строку на вход  
 for (int i = P.length() - 1; i >= 0; i--) { // с конца строки идем посимвольно  
 if (P.charAt(i) == c) { // если какой-то символ с конца строки совпадает, то возвращаем индекс этого символа в строке  
 return i;  
 }  
 }  
 return -1;  
 }  
  
 public static int min(int a, int b) { // возвращение минимального числа  
 if (a < b)  
 return a;  
 else if (b < a)  
 return b;  
 else  
 return a;  
 }  
  
  
 static class attempts {//структура  
 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 static class graphSearch { // главный метод  
 public static boolean mrt(int[][] a, int[][] b) { // проверка на то что 2 матрицы равны  
 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) { // делаем так, чтобы в матрице хранились не ссылки на объекты, а новая матрица  
 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) { // делаем так, чтобы в динамическом массиве хранились не ссылки на динамический массив, находящийся в объекте, а новый динамический массив  
 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) { // проверка позиции. был у нас такой пример или нет  
 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) { // проверка на оптимальность  
 int counter = 0;  
 for (int i = 0; i < 4; i++) { // манхэттенское расстояние  
 for (int j = 0; j < 4; j++) { //  
 for (int l = 0; l < 4; l++) { //  
 if (array[0][l] == (4 \* i + j + 1)) { // проверяем, какой эл-т должен стоять  
 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++) { // линейный конфликт  
 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) {//Главный Метод  
 int[][] answer = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 0}}; // создали двумерный массив (матрицу), уже решенный  
  
 ArrayList<attempts> jija = new ArrayList<>(); // динамический массив нашей структуры  
 ArrayList<Integer> a = new ArrayList<>(0); //Путь динамический, чтобы можно было его изменять, т.к. мы не знаем, какой длины будет путь  
 attempts quese1 = new attempts(arr1, a, 0); // создаем объект, состоящий из положения, пути и оптимальности  
 jija.add(quese1); // закидываем объект в динамический массив объектов  
 ArrayList<attempts> chekPosition = new ArrayList<>(); // динамический массив нашей структуры проверенных позиций (для оптимальности)  
 while (true) { // бесконечный цикл  
 attempts current; // объект нашей структуры  
 current = jija.remove(0); // объект каррэнт приравниваем к первому объекту и удаляем первый объект из динамического массива  
 chekPosition.add(current); // закидываем объект в проверенные позиции, потому что дальше мы его проверим  
 if (*mrt*(current.array, answer)) { // проверяем, является ли каррэнт (матрица в объекте, на которой находимся) ответу  
 System.*out*.println(current.path);// если равняется, выводим путь и заканчиваем метод  
 return;  
 }  
 int[] indexOfZeros = {0, 0}; // матрица, состоящая из 2х эл-тов, к-рая показывает местоположение 0  
 for (int i = 0; i < 4; i++) { // пробегаемся по матрице, находим нулевой эл-т и сохраняем его индекс в indexOfZeros  
 for (int j = 0; j < 4; j++)  
 if (current.array[i][j] == 0) {  
 indexOfZeros[0] = i;  
 indexOfZeros[1] = j;  
 break;  
 }  
 }  
 if (indexOfZeros[0] < 3) { // проверка на то, что строки меньше 3  
 int[][] newArray = *Go*(current.array); // создаем новую матрицу, к-рую приравниваем к матрице, на которой находимся (кар.эр)  
 newArray[indexOfZeros[0]][indexOfZeros[1]] = newArray[indexOfZeros[0] + 1][indexOfZeros[1]]; // эл-т, находящийся по индексу 0, приравниваем к соседнему эл-ту  
 newArray[indexOfZeros[0] + 1][indexOfZeros[1]] = 0; // соседний эл-т приравниваем 0  
 int action = newArray[indexOfZeros[0]][indexOfZeros[1]]; // создаем эл-т, к-рый находится там, где находился 0  
 ArrayList<Integer> newPath = *Go1*(current.path); // создаем новый путь  
 newPath.add(action); // в путь закидываем последний эл-т (число поменянное с 0)  
 if (*finder*(chekPosition, newArray) && *finder*(jija, newArray)) { // проверка на то, была у нас такая позиция или она находится в очереди на проверку. если уже были, то не вывполняем, если не были, то выполняем  
 jija.add(new attempts(newArray, newPath, *optimal*(newArray))); // закидываем в динам.массив объектов объект, хранящий в себе позицию, путь и оптимальность  
 }  
 }  
 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*(jija, newArray)) {  
 jija.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*(jija, newArray)) {  
 jija.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*(jija, newArray)) {  
 jija.add(new attempts(newArray, newPath, *optimal*(newArray)));  
 }  
 }  
 jija.sort(new Comparator<attempts>() { // сортировка. сортируем по оптимальности  
 @Override  
 public int compare(attempts o1, attempts o2) {  
 return o1.opt - o2.opt;  
 }  
 });  
 }  
 }  
 }  
  
  
 public static void main(String[] args) {  
  
 Scanner scan = new Scanner(System.*in*);  
 System.*out*.println("Введите строку:");  
 String str1 = scan.nextLine();  
 System.*out*.println("Введите подстроку:");  
 String str2 = scan.nextLine();  
  
  
  
  
 long time1 = System.*nanoTime*();  
 System.*out*.println("Поиск Кнута-Морриса-Пратта: " + *SearchString*(str1, str2) );  
  
 long time2 = System.*nanoTime*();  
 System.*out*.println("Поиск упрощенный Бойера-Мура: " + *BMSearch*(str1, str2) );  
  
 long time3 = System.*nanoTime*();  
 System.*out*.println("Стандартный поиск: " + str1.indexOf(str2) );  
  
  
 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 = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0};  
 double inv = 0; // проверка на то, есть решение или нет  
 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("Решения нет:");  
 }  
 }  
  
}

**Снимки экрана выполнения программы**

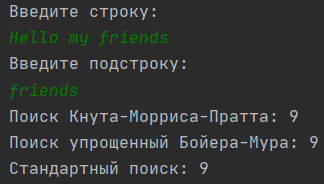


Рис. 1 Поиск, когда подстрока есть в строке

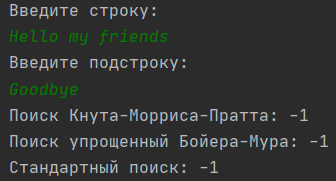


Рис. 2 Поиск, когда подстроки нет в строке



Рис. 3 При вводе строки {5, 1, 2, 3, 9, 6, 7, 4, 13, 10, 11, 8, 14, 15, 0, 12} из примера, вывод соответствует выводу задания



Рис. 4 При вводе строки {1,2,3,4,5,6,7,8,9,10,11,12,13,15,14,0} из примера, вывод соответствует выводу задания

# Вывод

В ходе выполнения лабораторной работы, я реализовала разные методы поиска подстроки в строке и выполнила задание с пятнашками.

Поиск Кнута-Морриса-Пратта выполняется при помощи префиксной функции и сдвига подстроки по ней.

Поиск Бойера-Мура выполняется при помощи таблицы смещений для каждого символа.

Пятнашки реализованы с помощью алгоритма A\*.