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

Гинатуллин Айгиз

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:

Lab3

import {KMPSearch} from "./KMPSearch.js";  
import {BMHSearch} from "./BMHSearch.js";  
import {graphSearch} from "./graphSearch.js";  
  
  
const sought = 'string'  
const str = 'aaasdadadaaastringr1214'  
  
***console***.log(KMPSearch(str, sought))  
***console***.log(BMHSearch(str, sought))  
  
let inv = 0  
let arr=[5,1,2,3,9,6,7,4,13,10,11,8,14,15,0,12]  
//let arr=[7,3,5,12,6,8,14,13,2,11,9,1,0,10,4,15]  
//let arr = [1, 2, 3, 4, 5, 6, 7, 8, 13, 9, 11, 12, 10, 14, 15, 0]  
//let arr = [5,1,3,4,0,2,6,8,7,10,15,11,9,13,14,12]  
//let arr = [11,9,4,6,3,15,7,13,2,10,0,8,5,12,1,14]  
//let arr = [5,9,8,14,0,6,12,3,13,11,1,10,15,2,7,4]  
// let arr = [7,1,4,15,10,12,3,14,5,6,0,11,2,13,8,9]  
for(let i = 0; i < 16; i++) {  
 if (arr[i])  
 for (let j = 0; j < i; ++j)  
 if (arr[j] > arr[i])  
 ++inv;  
}  
for (let i = 0; i < 16; ++i) {  
 if (arr[i] === 0)  
 inv += 1 + i / 4;  
}  
  
let arr1 = ***Array***();  
let k = 0  
for (let i = 0; i < 4; i++) {  
 arr1[i] = ***Array***();  
 for (let j = 0; j < 4; j++) {  
 arr1[i][j] = arr[k];  
 k++  
 }  
}  
if (inv & 1) {  
 ***console***.log("Решения нет")  
} else {  
 ***console***.log("Решение есть")  
 ***console***.log(graphSearch(arr1).join(","));  
}

BMHSearch

export function BMHSearch(source, template) {  
 let index;  
 let templateMask = [],  
 i = 0, j, c;  
 while (i < template.length - 1) {  
 templateMask[template.charAt(i)] = template.length - 1 - i;  
 i++  
 }  
 i = 0;  
 while (i < source.length) {  
 for (j = template.length - 1; j >= 0; j--)  
 if (template[j] !== source[i + j]) {  
 break;  
 }  
 if (j < 0) {  
 i++;  
 index = i;  
 } else {  
 c = templateMask[source.charAt(i + j)];  
 if (!c)  
 c = template.length - 1 + 1;  
 c += j - template.length - 1;  
 if (c <= 0)  
 c = 1;  
 i += c;  
 }  
 }  
 return index - 1;  
}

graphSearch

export const graphSearch = (array) => {  
 console.log(optimal(array,0))  
 let queue = [],chekPosition=[]  
 const answer = [[1, 2, 3, 4],  
 [5, 6, 7, 8],  
 [9, 10, 11, 12],  
 [13, 14, 15, 0]]  
  
 let act, opt;  
 queue.push(  
 {  
 array: array,  
 path: [],  
 lastAct: act,  
 optimal: opt  
 }  
 )  
 let index = 0, l = 0  
 while (queue.length > 0) {  
 const current = queue.shift()  
 index++  
 chekPosition.push(current.array)  
 if (JSON.stringify(current.array) === JSON.stringify(answer)) {  
 console.log("кол-во действий:"+current.path.length)  
 return current.path  
 }  
 let indexOfZeros;  
 for (let i = 0; i < 4; i++) {  
 for (let j = 0; j < 4; j++)  
 if (current.array[i][j] === 0) {  
 indexOfZeros = [  
 i, j  
 ]  
 break  
 }  
 }  
  
 if (indexOfZeros[0] < 3 && current.act !== 2) {  
 let newArray = JSON.parse(JSON.stringify(current.array))  
 newArray[indexOfZeros[0]][indexOfZeros[1]] = newArray[indexOfZeros[0] + 1][indexOfZeros[1]]  
 newArray[indexOfZeros[0] + 1][indexOfZeros[1]] = 0  
 const action = newArray[indexOfZeros[0]][indexOfZeros[1]];  
 let newPath = JSON.parse(JSON.stringify(current.path))  
 newPath.push(action)  
 if (finder(chekPosition, newArray)) {  
 queue.push(  
 {  
 array: newArray,  
 path: newPath,  
 act: 1,  
 opt: optimal(newArray,newPath)  
 }  
 )  
 }  
 //console.log("1 ", current.path)  
 }  
 if (indexOfZeros[0] > 0 && current.act !== 1) {  
 let newArray = JSON.parse(JSON.stringify(current.array))  
 newArray[indexOfZeros[0]][indexOfZeros[1]] = newArray[indexOfZeros[0] - 1][indexOfZeros[1]]  
 newArray[indexOfZeros[0] - 1][indexOfZeros[1]] = 0  
 const action = newArray[indexOfZeros[0]][indexOfZeros[1]];  
 let newPath = JSON.parse(JSON.stringify(current.path))  
 newPath.push(action)  
 if (finder(chekPosition, newArray)) {  
 queue.push(  
 {  
 array: newArray,  
 path: newPath,  
 opt: optimal(newArray,newPath),  
 act: 2  
 }  
 )  
 }  
 //console.log("2 ", current.path)  
 }  
 if (indexOfZeros[1] < 3 && current.act !== 4) {  
 let newArray = JSON.parse(JSON.stringify(current.array))  
 newArray[indexOfZeros[0]][indexOfZeros[1]] = newArray[indexOfZeros[0]][indexOfZeros[1] + 1]  
 newArray[indexOfZeros[0]][indexOfZeros[1] + 1] = 0  
 const action = newArray[indexOfZeros[0]][indexOfZeros[1]];  
 let newPath = JSON.parse(JSON.stringify(current.path))  
 newPath.push(action)  
  
 if (finder(chekPosition, newArray)) {  
 queue.push(  
 {  
 array: newArray,  
 path: newPath,  
 opt: optimal(newArray,newPath),  
 act: 3  
 }  
 )  
 }  
 //console.log("3 ", current.path)  
 }  
 if (indexOfZeros[1] > 0 && current.act !== 3) {  
 let newArray = JSON.parse(JSON.stringify(current.array))  
 newArray[indexOfZeros[0]][indexOfZeros[1]] = newArray[indexOfZeros[0]][indexOfZeros[1] - 1]  
 newArray[indexOfZeros[0]][indexOfZeros[1] - 1] = 0  
 const action = newArray[indexOfZeros[0]][indexOfZeros[1]];  
 let newPath = JSON.parse(JSON.stringify(current.path))  
 newPath.push(action)  
 if (finder(chekPosition, newArray)) {  
 queue.push(  
 {  
 array: newArray,  
 path: newPath,  
 opt: optimal(newArray,newPath),  
 act: 4  
 }  
 )  
 }  
 //console.log("4 ", current.path)  
 }  
 queue.sort((a, b) => {  
 return a.opt - b.opt  
 })  
 l++  
 //console.log(l)  
 // console.log(current.opt)  
 // console.log(current.path)  
 //console.log(current.array)  
 //console.log(chekPosition.length+" "+queue.length)  
 }  
  
  
}  
const finder = (array, sought) => {  
 let k=0  
 array.map(item => {  
 if (JSON.stringify(item) === JSON.stringify(sought)) {  
 //console.log(JSON.stringify(item.path[item.path.length - 1]), JSON.stringify(sought))  
 //console.log(2)  
 k++  
 return false  
 }  
 })  
 return k === 0;  
}  
  
const optimal = (array,pach) => {  
 let counter = 0  
 for (let i = 0; i < 4; i++) {  
 for (let j = 0; j < 4; j++) {  
 if (array[0].indexOf(4 \* i + j + 1) !== -1) {  
 counter += Math.abs(i) + Math.abs(j - array[0].indexOf(4 \* i + j + 1))  
 }  
 if (array[1].indexOf(4 \* i + j + 1) !== -1) {  
 counter += Math.abs(i - 1) + Math.abs(j - array[1].indexOf(4 \* i + j + 1))  
 }  
 if (array[2].indexOf(4 \* i + j + 1) !== -1) {  
 counter += Math.abs(i - 2) + Math.abs(j - array[2].indexOf(4 \* i + j + 1))  
 }  
 if (array[3].indexOf(4 \* i + j + 1) !== -1) {  
 counter += Math.abs(i - 3) + Math.abs(j - array[3].indexOf(4 \* i + j + 1))  
 }  
 }  
 }  
 const kon = [[0, 0, 0, 0],  
 [0, 0, 0, 0],  
 [0, 0, 0, 0],  
 [0, 0, 0, 0]]  
 for (let i = 0; i < 4; i++) {  
 for (let 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  
 }  
 }  
 }  
 //counter+=pach.length  
 // if (array[0][3] !== 4 && ((kon[0][1] !== 1&&kon[0][2] === 3) || (kon[1][2] !== 1&& kon[1][3] === 8)))  
 // counter += 2  
 // if (array[0][0] !== 1 && ((kon[0][1] !== 1&&kon[0][1] === 2) || (kon[1][0] !== 1&& kon[1][0] === 5)))  
 // counter += 2  
 // if (array[3][0] !== 13 && ((kon[2][0] !== 1&&kon[2][0] === 9) || (kon[3][1] !== 1&& kon[3][1] === 14)))  
 // counter += 2  
 // // //console.log(kon)  
 // // //console.log(array)  
 // if(array[3][3]!==12||array[3][3]!==15)  
 // counter+=2  
 return counter  
}

KMPSearch

const KMPMaskCreate = (str) => {  
 let maskArray = [str.length].fill(0)  
 let iteratorI = 1  
 let iteratorJ = 0  
 while (iteratorI < str.length) {  
 if (str[iteratorI] === str[iteratorJ]) {  
 maskArray[iteratorI] = iteratorJ + 1  
 iteratorJ += 1  
 iteratorI += 1  
 } else {  
 if (iteratorJ === 0) {  
 maskArray[iteratorI] = 0  
 iteratorI += 1  
 } else {  
 iteratorJ = maskArray[iteratorJ - 1]  
 }  
 }  
 }  
 return maskArray  
}  
  
export const KMPSearch = (str, soughtStr) => {  
 let i = 0  
 let j = 0  
 const soughtStringMask = KMPMaskCreate(soughtStr)  
 while (i < str.length) {  
 if (str[i] === soughtStr[j]) {  
 i++  
 j++  
 if (j === soughtStr.length)  
 return i - soughtStr.length  
 } else {  
 if (j > 0) {  
 j = soughtStringMask[j - 1]  
 } else  
 i++  
 }  
 }  
 return -1  
}

Вывод программы

12  
12  
Решение есть  
21  
кол-во действий:11  
15,14,13,9,5,1,2,3,4,8,12

# Вывод

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