Министерство цифрового развития, связи и массовых коммуникаций

Ордена Трудового Красного Знамени федеральное государственное бюджетное

образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и информационных технологий

Отчёт по лабораторной работе № 3

« Методы поиска подстроки в строке»

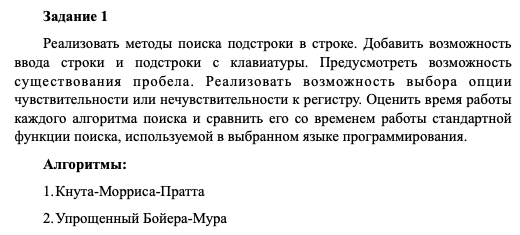
по дисциплине «Структуры и алгоритмы обработки данных»

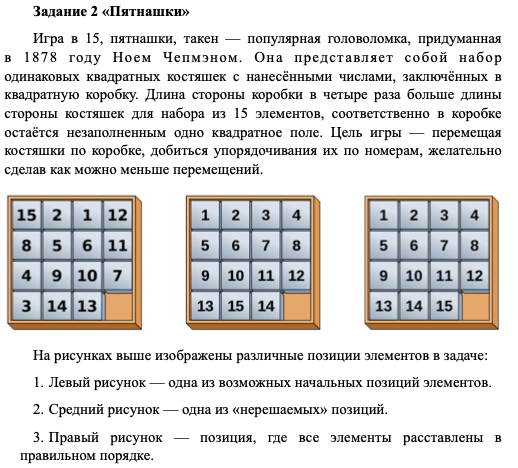
Выполнил: студент группы БВТ1902

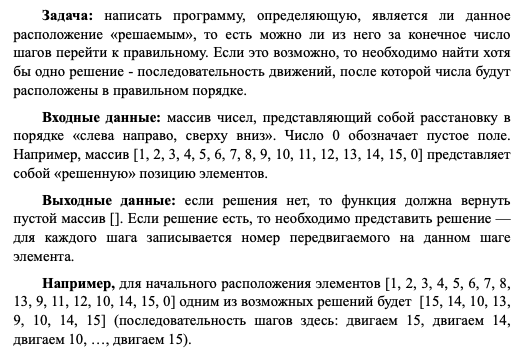
Холмовский Олег Васильевич

Москва

2021

Задание





Код программы

Алгоритм Кнута-Морриса-Пратта:

const readline = require('readline-sync');

*function* buildPatternTable(word) {

const patternTable = [0];

let prefixIndex = 0;

let suffixIndex = 1;

*while* (suffixIndex < word.length) {

*if* (word[prefixIndex] === word[suffixIndex]) {

patternTable[suffixIndex] = prefixIndex + 1;

suffixIndex += 1;

prefixIndex += 1;

} *else* *if* (prefixIndex === 0) {

patternTable[suffixIndex] = 0;

suffixIndex += 1;

} *else* {

prefixIndex = patternTable[prefixIndex - 1];

}

}

*return* patternTable;

}

*// O(m+n)*

*function* knuthMorrisPratt(text, word) {

*if* (word.length === 0) {

*return* 0;

}

let textIndex = 0;

let wordIndex = 0;

const patternTable = buildPatternTable(word);

*while* (textIndex < text.length) {

*if* (text[textIndex] === word[wordIndex]) {

*if* (wordIndex === word.length - 1) {

*return* textIndex - word.length + 1;

}

wordIndex += 1;

textIndex += 1;

} *else* *if* (wordIndex > 0) {

wordIndex = patternTable[wordIndex - 1];

} *else* {

wordIndex = 0;

textIndex += 1;

}

}

*return* -1;

}

const text = readline.question('Введите строку: ');

const word = readline.question('Введите подстроку: ');

console.log(knuthMorrisPratt(text, word));

Алгоритм Бойера-Мура:

const readline = require('readline-sync');

*function* makeCharTable(pattern) {

let table = [];

*for* (let i = 0; i < 65536; i++) {

table.push(pattern.length);

}

*for* (let i = 0; i < pattern.length - 1; i++) {

const charCode = pattern.charCodeAt(i);

table[charCode] = pattern.length - 1 - i;

}

*return* table;

}

*function* makeOffsetTable(pattern) {

let table = [];

table.length = pattern.length;

let lastPrefixPosition = pattern.length;

*for* (let i = pattern.length; i > 0; i--) {

*if* (isPrefix(pattern, i)) {

lastPrefixPosition = i;

}

table[pattern.length - i] = lastPrefixPosition - 1 + pattern.length;

}

*for* (let i = 0; i < pattern.length - 1; i++) {

const slen = suffixLength(pattern, i);

table[slen] = pattern.length - 1 - i + slen;

}

*return* table;

}

*function* isPrefix(pattern, p) {

*for* (let i = p, j = 0; i < pattern.length; i++, j++) {

*if* (pattern[i] != pattern[j]) {

*return* false;

}

*return* true;

}

}

*function* suffixLength(pattern, p) {

let len = 0;

*for* (

let i = p, j = pattern.length - 1;

i >= 0 && pattern[i] == pattern[j];

i--, j--

) {

len += 1;

}

*return* len;

}

*// O(m/n) or O(m+n)*

*function* boyerMooreSearch(text, pattern) {

*if* (pattern.length === 0) {

*return* -1;

}

let charTable = makeCharTable(pattern);

let offsetTable = makeOffsetTable(pattern);

*for* (let i = pattern.length - 1, j; i < text.length; ) {

*for* (j = pattern.length - 1; pattern[j] == text[i]; i--, j--) {

*if* (j === 0) {

*return* i;

}

}

const charCode = text.charCodeAt(i);

i += Math.max(offsetTable[pattern.length - 1 - j], charTable[charCode]);

}

*return* -1;

}

const text = readline.question('Введите строку: ');

const word = readline.question('Введите подстроку: ');

console.log(boyerMooreSearch(text, word));

Пятнашки:

const fifteenPuzzle = (array) => {

let queue = [],

chekPosition = [];

const answer = [

[1, 2, 3, 4],

[5, 6, 7, 8],

[9, 10, 11, 12],

[13, 14, 15, 0],

];

queue.push({

array: array,

path: [],

opt: 0,

});

*while* (queue.length > 0) {

const current = queue.shift();

chekPosition.push(current.array);

*if* (JSON.stringify(current.array) === JSON.stringify(answer)) {

*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.opt !== 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,

opt: optimal(newArray),

});

}

}

*if* (indexOfZeros[0] > 0 && current.opt !== 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),

});

}

}

*if* (indexOfZeros[1] < 3 && current.opt !== 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),

});

}

}

*if* (indexOfZeros[1] > 0 && current.opt !== 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;

let 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),

});

}

}

queue.sort((a, b) => {

*return* a.opt - b.opt;

});

}

};

const finder = (array, sought) => {

let k = 0;

array.map((item) => {

*if* (JSON.stringify(item) === JSON.stringify(sought)) {

k++;

*return* false;

}

});

*return* k === 0;

};

const optimal = (array) => {

let counter = 0;

*for* (let i = 0; i < 4; i++) {

*for* (let j = 0; j < 4; j++) {

*for* (let o = 0; o < 4; o++) {

*if* (array[o].indexOf(4 \* i + j + 1) !== -1) {

counter +=

Math.abs(i - o) + Math.abs(j - array[o].indexOf(4 \* i + j + 1));

}

}

}

}

*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;

}

}

}

*if* (array[3][3] !== 12 || array[3][3] !== 15) counter += 2;

*return* counter;

};

let inv = 0;

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

*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++;

}

}

console.log('fifteenPuzzle:');

console.log(arr1);

*if* (inv & 1) {

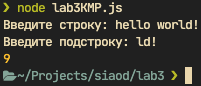
console.log('Решения нет');

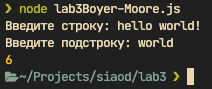
} *else* {

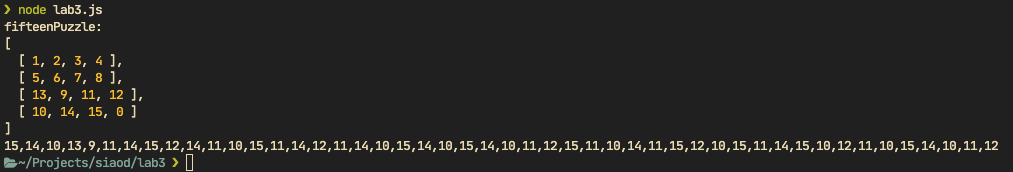
console.log(fifteenPuzzle(arr1).join(','));

}

Cнимки экрана работы программы







Вывод

В ходе выполнения лабораторной работы, мной были получены навыки реализации алгоритмов поиска подстроки в строке на примере алгоритмов Кнута-Морриса-Пратта и Бойера-Мура , а так же решения задачи ‘пятнашки’.