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

Мартынов Николай Владимирович

Москва

2021

Оглавление

[Введение 3](#_Toc72438562)

[Листинг программы 4](#_Toc72438563)

[Вывод 10](#_Toc72438564)

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 Lab3

package com.company;

import java.util.\*;

public class Lab3 {

//БВТ1902 Мартынов Николай 16 вариант

public static int[] prefixFunction(String str) {

int[] prefixFunc = new int[str.length()];

for (int i = 1; i < str.length(); i++) {

int j = prefixFunc[i - 1];

while (j > 0 && str.charAt(i) != str.charAt(j)) {

j = prefixFunc[j - 1];

}

if (str.charAt(i) == str.charAt(j)) {

j += 1;

}

prefixFunc[i] = j;

}

return prefixFunc;

}

public static List<Integer> KMPSearch(String text, String pattern) {

int[] prefixFunc = prefixFunction(pattern);

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

int j = 0;

for (int i = 0; i < text.length(); i++) {

while (j > 0 && text.charAt(i) != pattern.charAt(j)) {

j = prefixFunc[j - 1];

}

if (text.charAt(i) == pattern.charAt(j)) {

j += 1;

}

if (j == pattern.length()) {

occurrences.add(i - j + 1);

j = prefixFunc[j - 1];

}

}

return occurrences;

}

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;

}

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;

}

public static int last(char c, String P)

{

for (int i=P.length()-1; i>=0; i--)

{

if (P.charAt(i) == c)

{

return i;

}

}

return -1;

}

public static int min(int a, int b)

{

if (a < b)

return a;

else if (b < a)

return b;

else

return a;

}

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: "+KMPSearch(str1,str2)+" time: "+(System.nanoTime()-time1) +"ns");

long time2 = System.nanoTime();

System.out.println("Поиск упрощенный Бойера-Мура: "+BMSearch(str1,str2)+" time: "+(System.nanoTime()-time2) +"ns");

long time3 = System.nanoTime();

System.out.println("Стандартный поиск: "+str1.indexOf(str2)+" time: "+(System.nanoTime()-time3) +"ns");

}

}

Класс puzzleSolver

package com.company;

import java.util.\*;

public class puzzleSolver {

private static class node{

int level = 0;

int [][] state = new int [4][4];

int blankRow;

int blankCol;

String move = "";

node parent = null;

node up = null;

node down = null;

node left = null;

node right = null;

}

private static class solutionData {

node solutionNode = null;

String path = "";

String startingBoard = "";

}

public static void main(String [] args){

if(args.length <= 0){

System.out.println("Пазла не было");

return;

}

else if(args.length != 16){

System.out.println("Пазл содержит неправильное количество цифр");

return;

}

System.gc();

System.out.println("\nАлгоритм A\*:");

try{

String board = "";

node root = new node();

root.move = "Start";

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

board = board + "\n";

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

root.state[i][q] = Integer.parseInt(args[4\*i + q]);

board = board + root.state[i][q] + "\t";

if(root.state[i][q] == 0){

root.blankRow = i;

root.blankCol = q;

}

}

}

solutionData solution = aStarH1(root, board);

if(solution != null)

printSolutionData(solution);

else

System.out.println("Решений не найдено");

}catch(OutOfMemoryError e){

System.out.println("Решений не найдено");

}

}

public static solutionData aStarH1(node root, String board){

ArrayList<node> unexpandedNodes = new ArrayList<node>();

unexpandedNodes.add(root);

solutionData breadthSolution = new solutionData();

int expandedNodes = 0;

node cur = null;

while(expandedNodes < Integer.MAX\_VALUE){

// получить узел с наименьшим результатом f(n) = g(n) + h(n)

int minVal = Integer.MAX\_VALUE;

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

node tmp = unexpandedNodes.get(i);

if(getMissplacedTiles(tmp) + tmp.level < minVal){

minVal = getMissplacedTiles(tmp) + tmp.level;

cur = tmp;

}

}

unexpandedNodes.remove(cur);

if (getMissplacedTiles(cur) == 0){

breadthSolution.solutionNode = cur;

breadthSolution.startingBoard = board;

breadthSolution.path = getPath(cur);

return breadthSolution;

}

else{

evaluateChildren(cur);

expandedNodes++;

if(cur.left != null)

unexpandedNodes.add(cur.left);

if(cur.right != null)

unexpandedNodes.add(cur.right);

if(cur.up != null)

unexpandedNodes.add(cur.up);

if(cur.down != null)

unexpandedNodes.add(cur.down);

}

}

return null;

}

public static void evaluateChildren(node curNode){

if(curNode.blankCol > 0){

curNode.left = new node();

curNode.left.move = "L";

curNode.left.level = curNode.level + 1;

curNode.left.blankCol = curNode.blankCol - 1;

curNode.left.blankRow = curNode.blankRow;

curNode.left.parent = curNode;

curNode.left.state = makeState(curNode.state, curNode.blankRow, curNode.blankCol, 'L');

}

if(curNode.blankCol < 3){

curNode.right = new node();

curNode.right.move = "R";

curNode.right.level = curNode.level + 1;

curNode.right.blankCol = curNode.blankCol + 1;

curNode.right.blankRow = curNode.blankRow;

curNode.right.parent = curNode;

curNode.right.state = makeState(curNode.state, curNode.blankRow, curNode.blankCol, 'R');

}

if(curNode.blankRow > 0){

curNode.up = new node();

curNode.up.move = "U";

curNode.up.level = curNode.level + 1;

curNode.up.blankCol = curNode.blankCol;

curNode.up.blankRow = curNode.blankRow - 1;

curNode.up.parent = curNode;

curNode.up.state = makeState(curNode.state, curNode.blankRow, curNode.blankCol, 'U');

}

if(curNode.blankRow < 3){

curNode.down = new node();

curNode.down.move = "D";

curNode.down.level = curNode.level + 1;

curNode.down.blankCol = curNode.blankCol;

curNode.down.blankRow = curNode.blankRow + 1;

curNode.down.parent = curNode;

curNode.down.state = makeState(curNode.state, curNode.blankRow, curNode.blankCol, 'D');

}

}

// возвращает новую матрицу

public static int[][] makeState(int[][] curState, int blankY, int blankX, char move){

int [][] newState = new int [4][];

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

newState[i] = curState[i].clone();

if(move == 'U'){

newState[blankY][blankX] = newState[blankY - 1][blankX];

newState[blankY - 1][blankX] = 0;

}

else if(move == 'D'){

newState[blankY][blankX] = newState[blankY + 1][blankX];

newState[blankY + 1][blankX] = 0;

}

else if(move == 'L'){

newState[blankY][blankX] = newState[blankY][blankX - 1];

newState[blankY][blankX - 1] = 0;

}

else{

newState[blankY][blankX] = newState[blankY][blankX + 1];

newState[blankY][blankX + 1] = 0;

}

return newState;

}

//сравнивает состояние цели и текущее состояние и возвращает количество пропущенных ячеек

public static int getMissplacedTiles(node curNode){

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

int tileCounter = 0;

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

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

if(goalState[row][col] != curNode.state[row][col])

tileCounter++;

}

}

return tileCounter;

}

//возвращает строку, которая показывает путь, который нужно пройти, чтобы решить пазл

public static String getPath(node solution){

LinkedList<node> solutionPath = new LinkedList<node>();

node cur = solution;

String path = "";

while(cur != null){

solutionPath.addFirst(cur);

cur = cur.parent;

}

while(!solutionPath.isEmpty()){

node tmp = solutionPath.removeFirst();

if(tmp.move != "Start")

path = path + tmp.move;

}

return path;

}

public static void printSolutionData(solutionData solution){

//1 2 3 4 5 6 7 8 13 9 11 12 10 14 15 0

//5 1 2 3 9 6 7 4 13 10 11 8 14 15 0 12

System.out.println(solution.startingBoard + "\n\nДействий:" + solution.path.length() + "\nШаги:" + solution.path);

}

}

Вывод

В результате выполненной лабораторной работы я реализовал два алгоритма поиска подстроки в строке (алгоритм Кнута-Морриса-Пратта и упрощенный алгоритм Бойера-Мура), сравнил со стандартной функцией поиска в java, а также реализовал программу, которая решает игру «Пятнашки» при помощи алгоритма A\*.