**Федеральное агентство связи**

**Государственное бюджетное образовательное учреждение высшего образования Ордена Трудового Красного Знамени**

**«Московский технический университет связи и информатики»**

**Лабораторная работа № 3**

«Методы поиска подстроки в строке и пятнашки»

Выполнила: студентка группы БВТ1904

Хорикова Софья Гарегиновна

Проверил: Павликов Артём Евгеньевич

Москва, 2020

1. **Цель работы**

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

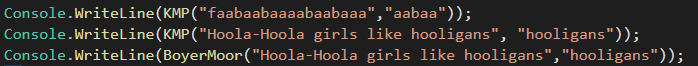
Задание 2

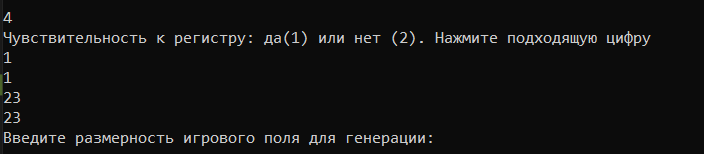
Написать программу, определяющую, является ли данное

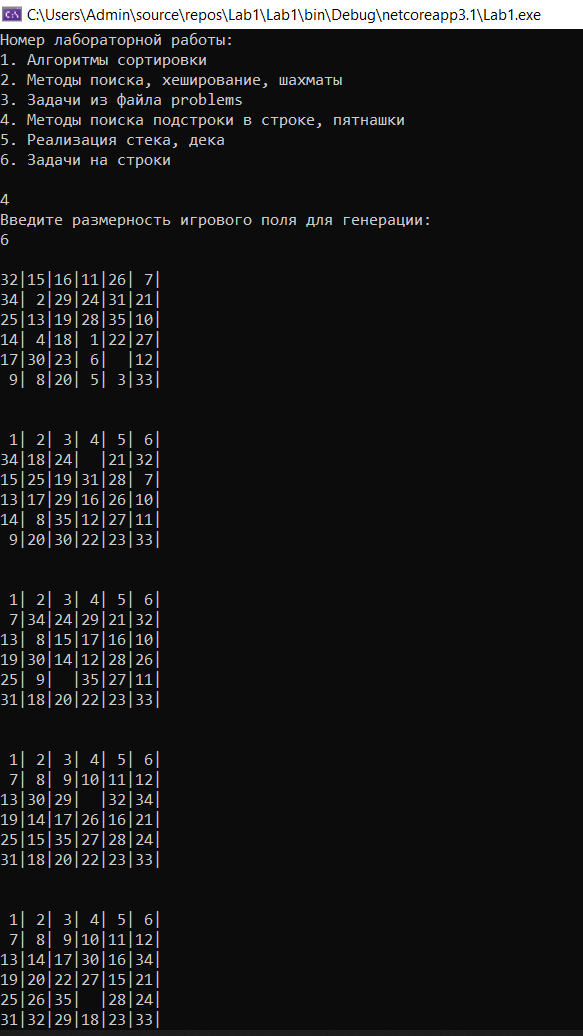
расположение чисел в игре «Пятнашки» решаемым, то есть можно ли из него за конечное число шагов перейти к правильному. Если это возможно, то необходимо найти хотя бы одно решение - последовательность движений, после которой числа будут расположены в правильном порядке.

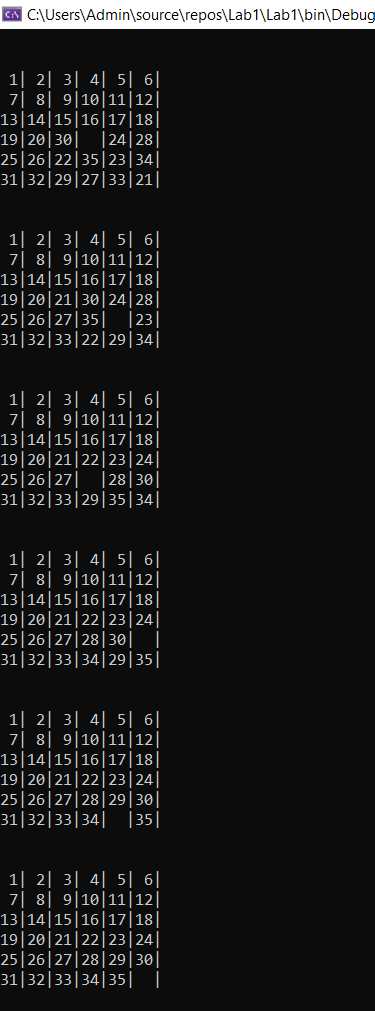
1. **Пример выполнения программы**

Код находится в конце ради удобства.









1. **Выводы**

В данной лабораторной работе были реализованы два метода поиска подстроки в строке.

Также была написана программа для решения игры «Пятнашки» на любой размер игрового поля. В ней еще учитываются нерешаемые комбинации, об этом случае выдается сообщение.

1. **Листинг программы**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

namespace Lab\_number

{

static class Lab3

{

public static void Start\_Lab3()

{

while (true)

{

/\*bool choose = false;

Console.WriteLine("Чувствительность к регистру: да(1) или нет (2). Нажмите подходящую цифру");

if (int.Parse(Console.ReadLine()) == 1) choose = true;

string str = "";

string word = "";

if (choose != true)

{

str.ToLower();

word.ToLower();

}

Console.WriteLine(KMP("faabaabaaaabaabaaa","aabaa"));

Console.WriteLine(KMP("Hoola-Hoola girls like hooligans", "hooligans"));

Console.WriteLine(BoyerMoor("Hoola-Hoola girls like hooligans","hooligans"));

\*/

Console.WriteLine("Введите размерность игрового поля для генерации:");

var size = int.Parse(Console.ReadLine());

List<int> list = new List<int>();

while (list.Count != size \* size)

{

Random rnd = new Random();

int next = rnd.Next(1, size \* size + 1);

if (!list.Contains(next))

{

list.Add(next);

}

}

Game game = new Game(list.ToArray()/\*new int[] { 10, 2, 4, 7, 13, 16, 1, 3, 11, 6, 12, 14, 9, 5, 15, 8 }\*/);

int[] order = game.Solve();

//Console.WriteLine("Полученная последовательность передвижений:");

//foreach (var item in order)

//{

// Console.Write(item + ", ");

//}

Console.WriteLine();

Console.WriteLine("Число перестановок: " + order.Length);

Console.WriteLine("Нажмите любую кнопку для повторения");

Console.ReadLine();

}

}

static int KMP(string str, string word)

{

bool flag = true;

var tmp = str + word;

char ch = ' '; //начальный символ

while (flag)

{

if (tmp.Contains(ch))

{

ch++; //если содержится такой символ, то переходим на следующий, пока не найдем уникальный

}

else

{

flag = false; //когда нашли уникальный символ - выходим

}

}

tmp = word + ch + str; //разделяем слово и строку уникальным символом

var prefs = new int[tmp.Length];

CalcPrefs(tmp);

for (int i = word.Length; i < prefs.Length; i++) //пока не совпадут длина слова и префикса

{

if (prefs[i] == word.Length)

{

return i - word.Length \* 2; //индекс начала слова

}

}

return -1;

void CalcPrefs(string word)

{

/\*

\* находим максимальную длину префикса для каждого символа

\*/

for (int i = 0, j = 1; j < word.Length; j++, i = 0)

{

string tempi = word[i].ToString(), tempj = word[j].ToString();

int ended = j;

while (tempi.Length <= j)

{

if (tempi.Equals(tempj) && tempj.Length > prefs[j])

{

prefs[j] = tempj.Length;

}

i++;

ended--;

tempi += word[i].ToString();

tempj = word[ended].ToString() + tempj;

}

}

}

}

private static int BoyerMoor(string str, string word)

{

var alphabet = new int[256];

for (int i = 0; i < alphabet.Length; i++)

{

alphabet[i] = word.Length;

}

string reversed = new string(word.ToCharArray().Reverse().ToArray()); //переворачиваем слово

var chars\_cost = new Dictionary<char, int>();

for (int i = 0; i < reversed.Length; i++)

{

if (!chars\_cost.ContainsKey(reversed[i]))

{

chars\_cost.Add(reversed[i], i);

alphabet[reversed[i]] = i;

}

}

int l = word.Length - 1;

while (l <= str.Length)

{

if (word[word.Length - 1] != str[l])//если последний символ не == текущему

{

l += alphabet[str[l]];

}

else//если равен

{

var flag = false;

for (int i = l - 1, j = word.Length - 2; j >= 0; i--, j--)

{

if (str[i] != word[j])

{

l += alphabet[str[i]];

flag = true;

break;

}

}

if (!flag)

{

return l - word.Length + 1;

}

}

}

return -1;

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab\_number

{

public class Game

{

int size;

Cell[,] field;

List<int> movelist;

int[] arr;

public Game(int[] num)

{

arr = num;

size = (int)Math.Sqrt(num.Length);

if (num.Length % size == 0)

{

field = new Cell[size, size];

//инициализируется игровое поле

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

{

for (int j = 0; j < size; j++)

{

field[i, j] = new Cell(num[i \* size + j], i, j, size);

}

}

MakeNeighbours(field);

movelist = new List<int>();

}

else

{

Console.WriteLine("Недопустимое количество элементов, невозможно составить квадратное игровое поле.");

}

}

/// <summary>

/// Привязывает соседей со всех сторон для каждой клетки поля

/// </summary>

/// <param name="map"></param>

void MakeNeighbours(Cell[,] map)

{

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

{

for (int j = 0; j < size; j++)

{

var temp = map[i, j]; //текущий элемент

var col = temp.GetCol();

var row = temp.GetRow();

if (row != 0) //если ряд не первый

{

temp.SetUp(map[row - 1, col]); //устанавливаем соседа сверху

}

else

{

temp.SetUp(null);

}

if (col != size - 1)

{

temp.SetRight(map[row, col + 1]);

}

else

{

temp.SetRight(null);

}

if (row != size - 1)

{

temp.SetDown(map[row + 1, col]);

}

else

{

temp.SetDown(null);

}

if (col != 0)

{

temp.SetLeft(map[row, col - 1]);

}

else

{

temp.SetLeft(null);

}

}

}

}

/// <summary>

/// Решает заданное начальное поле

/// </summary>

/// <returns></returns>

public int[] Solve()

{

bool solvable = CheckSolvable(arr);

if (solvable)

{

PrintField(field);

for (int i = 0; i < size - 1; i++)

{

PlaceRow(i);

PrintField(field);

PlaceCol(i);

PrintField(field);

}

return movelist.ToArray();

}

else

{

Console.WriteLine("Нерешаемая комбинация.");

return new int[] { -1 };

}

}

/// <summary>

/// устанавливает строку

/// </summary>

/// <param name="row"></param>

void PlaceRow(int row)

{

int i;

for (i = 0; i < size - 1; i++)

{

PlaceCell(row, i);

}

PlaceLastInRow(row);

}

/// <summary>

/// устанавливает столбец

/// </summary>

/// <param name="col"></param>

void PlaceCol(int col)

{

int i;

for (i = 0; i < size - 1; i++)

{

PlaceCell(i, col);

}

PlaceLastInCol(col);

}

/// <summary>

/// для последнего значения в столбце, чтобы его последнее значение создавало уголок

/// </summary>

/// <param name="col"></param>

void PlaceLastInCol(int col)

{

Cell last = GetCell(field, 1 + col + (size - 1) \* size);

int targRow = last.GetTargRow();

Cell empty = GetCell(field, size \* size); //пустая

if (last.CheckPlace()) //если она уже стоит на своем месте

{

return;

}

last.SetTargCol(last.GetTargCol() + 1); //ставим целевое местоположение со смещением

last.SetTargRow(last.GetTargRow() - 1);

if (empty.GetRow() == size - 1 && empty.GetRight() == last) //если пустая на месте целевой, и она справа

{

MoveRight(field, empty);

}

else

{

PlaceCell(targRow, col);

SpinRight(last);

}

void SpinRight(Cell last) //делаем уголок

{

var empty = GetCell(field, size \* size);

MoveEmptyToClose(field, last);

switch (empty.GetDirection(last))

{

case Cell.Direction.Left://если 0 справа от цели

MoveUp(field, empty);

MoveLeft(field, empty);

goto case Cell.Direction.Down;

case Cell.Direction.Up://если 0 снизу цели

MoveRight(field, empty);

MoveUp(field, empty);

goto case Cell.Direction.Left;

case Cell.Direction.Down://если 0 сверху от цели

MoveLeft(field, empty);

MoveDown(field, empty);

MoveRight(field, empty);

MoveDown(field, empty);

MoveLeft(field, empty);

MoveUp(field, empty);

MoveUp(field, empty);

MoveRight(field, empty);

MoveRight(field, empty);

MoveDown(field, empty);

break;

default:

break;

}

last.SetTargCol(last.GetTargCol() - 1);

last.SetTargRow(last.GetTargRow() + 1);

last.CheckPlace();

}

}

/// <summary>

/// для последнего значения в строке, чтобы его последнее значение создавало уголок

/// </summary>

/// <param name="row"></param>

void PlaceLastInRow(int row)

{

Cell last = GetCell(field, (row + 1) \* size);

int targCol = last.GetTargCol();

Cell empty = GetCell(field, size \* size);

if (last.CheckPlace())

{

return;

}

last.SetTargCol(last.GetTargCol() - 1);

last.SetTargRow(last.GetTargRow() + 1);

if (empty.GetCol() == size - 1 && empty.GetDown() == last)

{

MoveDown(field, empty);

}

else if (!(last.GetRow() == 0 && last.GetCol() == size - 1))

{

PlaceCell(row, targCol);

SpinLeft(last);

}

void SpinLeft(Cell last)

{

var empty = GetCell(field, size \* size);

MoveEmptyToClose(field, last);

switch (empty.GetDirection(last))

{

case Cell.Direction.Left://если 0 справа от цели

MoveDown(field, empty);

MoveLeft(field, empty);

goto case Cell.Direction.Up;

case Cell.Direction.Up://если 0 снизу цели

MoveLeft(field, empty);

MoveUp(field, empty);

goto case Cell.Direction.Right;

case Cell.Direction.Right://если 0 слева от цели

MoveUp(field, empty);

MoveRight(field, empty);

MoveDown(field, empty);

MoveRight(field, empty);

MoveUp(field, empty);

MoveLeft(field, empty);

MoveLeft(field, empty);

MoveDown(field, empty);

break;

default:

break;

}

last.SetTargCol(last.GetTargCol() + 1);

last.SetTargRow(last.GetTargRow() - 1);

last.CheckPlace();

}

}

/// <summary>

/// Двигает пустую по направлению к цели, пока не станет его соседом

/// </summary>

/// <param name="map"></param>

/// <param name="actual"></param>

void MoveEmptyToClose(Cell[,] map, Cell actual)

{

var empty = GetCell(map, size \* size);

while (!empty.IsNear(actual))//двигаем 0 к цели пока не будем рядом

{

switch (empty.GetDirectionToMove(actual))

{

case Cell.Direction.Down://если 0 сверху цели

MoveDown(map, empty);

break;

case Cell.Direction.Left://если 0 справа от цели

MoveLeft(map, empty);

break;

case Cell.Direction.Up://если 0 снизу цели

MoveUp(map, empty);

break;

case Cell.Direction.Right://если 0 слева от цели

MoveRight(map, empty);

break;

default:

break;

}

}

}

//методы, совершающие обмен между соседними

void MoveLeft(Cell[,] map, Cell replaced)

{

Cell temp = replaced.GetLeft();

movelist.Add(temp.GetValue());

map[temp.GetRow(), temp.GetCol()] = replaced;

map[replaced.GetRow(), replaced.GetCol()] = temp;

temp.SetCol(temp.GetCol() + 1);

replaced.SetCol(replaced.GetCol() - 1);

replaced.SetPrevious(Cell.Direction.Left);

temp.SetPrevious(Cell.Direction.Right);

MakeNeighbours(map);

}

void MoveDown(Cell[,] map, Cell replaced)

{

Cell temp = replaced.GetDown();

movelist.Add(temp.GetValue());

map[temp.GetRow(), temp.GetCol()] = replaced;

map[replaced.GetRow(), replaced.GetCol()] = temp;

temp.SetRow(temp.GetRow() - 1);

replaced.SetRow(replaced.GetRow() + 1);

replaced.SetPrevious(Cell.Direction.Down);

temp.SetPrevious(Cell.Direction.Up);

MakeNeighbours(map);

}

void MoveUp(Cell[,] map, Cell replaced)

{

Cell temp = replaced.GetUp();

movelist.Add(temp.GetValue());

map[temp.GetRow(), temp.GetCol()] = replaced;

map[replaced.GetRow(), replaced.GetCol()] = temp;

temp.SetRow(temp.GetRow() + 1);

replaced.SetRow(replaced.GetRow() - 1);

replaced.SetPrevious(Cell.Direction.Up);

temp.SetPrevious(Cell.Direction.Down);

MakeNeighbours(map);

}

void MoveRight(Cell[,] map, Cell replaced)

{

Cell temp = replaced.GetRight();

movelist.Add(temp.GetValue());

map[temp.GetRow(), temp.GetCol()] = replaced;

map[replaced.GetRow(), replaced.GetCol()] = temp;

temp.SetCol(temp.GetCol() - 1);

replaced.SetCol(replaced.GetCol() + 1);

replaced.SetPrevious(Cell.Direction.Right);

temp.SetPrevious(Cell.Direction.Left);

MakeNeighbours(map);

}

void PrintField(Cell[,] field)

{

Console.WriteLine();

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

{

for (int j = 0; j < size; j++)

{

if (field[i, j].GetValue().ToString().Length == 1)

{

if (field[i, j].GetValue() == size \* size)

{

Console.Write(" |");

}

else

{

Console.Write(" " + field[i, j].GetValue() + "|");

}

}

else

{

if (field[i, j].GetValue() == size \* size)

{

Console.Write(" |");

}

else

{

Console.Write(field[i, j].GetValue() + "|");

}

}

}

Console.WriteLine();

}

Console.WriteLine();

}

/// <summary>

/// размещение клетки

/// </summary>

/// <param name="row\_of\_num"></param>

/// <param name="col\_of\_num"></param>

void PlaceCell(int row\_of\_num, int col\_of\_num)

{

Cell placed\_cell = GetCell(field, row\_of\_num \* size + col\_of\_num + 1);

Cell empty = GetCell(field, size \* size);

placed\_cell.CheckPlace();

if (placed\_cell.GetPlaced()) //если размещена

{

return;

}

MoveEmptyToClose(field, placed\_cell);//пока не будем рядом с целью

while (!placed\_cell.GetPlaced())//двигаем пока не поставим на своё место

{

Cell.Direction destination\_side = placed\_cell.GetDirectionToMove();

/\*

определяем, в каком направлении двигаться

приоритет: лево-право, вверх-вниз.

определяем кратчайший для передвижения с учётом уже размещённых

\*/

switch (empty.GetDirection(placed\_cell))//определяем, с какой стороны граничит с целью 0

{

case Cell.Direction.Up://0 снизу

switch (destination\_side)//определяем в каком направлении мы хотели бы передвинуть целевую клетку

{

case Cell.Direction.Up://если надо вверх

MoveRight(field, empty);

MoveUp(field, empty);

break;

case Cell.Direction.Right://надо вправо

MoveRight(field, empty);

MoveUp(field, empty);

break;

case Cell.Direction.Down://надо вниз

MoveUp(field, empty);

break;

case Cell.Direction.Left://надо влево

if (empty.CanLeft())

{

MoveLeft(field, empty);

MoveUp(field, empty);

}

else

{

MoveRight(field, empty);

MoveUp(field, empty);

}

break;

default:

break;

}

break;

case Cell.Direction.Right://0 слева

switch (destination\_side)//определяем в каком направлении мы хотели бы передвинуть целевую клетку

{

case Cell.Direction.Up://если надо вверх

if (empty.CanUp())

{

MoveUp(field, empty);

MoveRight(field, empty);

}

else

{

MoveDown(field, empty);

MoveRight(field, empty);

}

break;

case Cell.Direction.Right://надо вправо

if (empty.CanUp())

{

MoveUp(field, empty);

MoveRight(field, empty);

}

else

{

MoveDown(field, empty);

MoveRight(field, empty);

}

break;

case Cell.Direction.Down://надо вниз

if (empty.CanDown())

{

MoveDown(field, empty);

MoveRight(field, empty);

}

else if (empty.CanUp())

{

MoveUp(field, empty);

MoveRight(field, empty);

}

break;

case Cell.Direction.Left://надо влево

MoveRight(field, empty);

break;

default:

break;

}

break;

case Cell.Direction.Down://0 сверху

switch (destination\_side)//определяем в каком направлении мы хотели бы передвинуть целевую клетку

{

case Cell.Direction.Up://если надо вверх

MoveDown(field, empty);

break;

case Cell.Direction.Right://надо вправо

MoveRight(field, empty);

MoveDown(field, empty);

break;

case Cell.Direction.Down://надо вниз

MoveRight(field, empty);

MoveDown(field, empty);

break;

case Cell.Direction.Left://надо влево

if (empty.CanLeft() && placed\_cell.CanLeft())//и слева не установленная клетка и целевая может идти влево

{

MoveLeft(field, empty);

MoveDown(field, empty);

}

else

{

MoveRight(field, empty);

MoveDown(field, empty);

}

break;

default:

break;

}

break;

case Cell.Direction.Left://0 справа

switch (destination\_side)//определяем в каком направлении мы хотели бы передвинуть целевую клетку

{

case Cell.Direction.Up://если надо вверх

MoveUp(field, empty);

MoveLeft(field, empty);

break;

case Cell.Direction.Right://надо вправо

MoveLeft(field, empty);

break;

case Cell.Direction.Down://надо вниз

MoveDown(field, empty);

MoveLeft(field, empty);

break;

case Cell.Direction.Left://надо влево

if ((empty.CanDown() && empty.GetPrevious() != Cell.Direction.Up) || placed\_cell.GetRow() == 0)

{

MoveDown(field, empty);

MoveLeft(field, empty);

}

else

{

MoveUp(field, empty);

MoveLeft(field, empty);

}

break;

default:

break;

}

break;

default:

break;

}

placed\_cell.CheckPlace();

}

}

/// <summary>

/// получение клетки из массива клеток

/// </summary>

/// <param name="field"></param>

/// <param name="num"></param>

/// <returns></returns>

Cell GetCell(Cell[,] field, int num)

{

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

{

for (int j = 0; j < size; j++)

{

if (field[i, j].GetValue() == num)

{

return field[i, j];

}

}

}

return null;

}