Федеральное агентство связи

Ордена трудового Красного Знамени

федеральное государственное

бюджетное образовательное учреждение

высшего образования

«Московский технический университет связи и информатики»

Кафедра информатики

Отчет по лабораторной работе №3 по СиАОД

«Методы поиска подстроки в строке и задача про пятнашки»

Выполнил: студент группы БВТ1903

Клычёв Д.А

Проверил: Павликов А. Е.

Москва, 2021

# **1. Задание**

# **2. Ход работы**

Язык программирования, используемый для выполнения работы: C#, версия 9.0.

Для выполнения поставленных задач было создано решение в среде разработки MVS2019, включающее проект Lab1, исполняемый код которого представлен в классах Lab3.cs, Cell.cs, Field.cs, MatrixExtensions.cs, листинг представлен ниже:

#define strings

#define taken

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

namespace Lab3

{

static class Lab3

{

internal static void Run3Lab()

{

Random rnd = new Random();

Stopwatch sw = new Stopwatch();

#if strings

#region KMP

Console.WriteLine("Введите строку для поиска или 0 чтобы перейти к следующему заданию");

string str = Console.ReadLine();

while (str != "0")

{

Console.WriteLine("Введите подстроку для поиска");

string word = Console.ReadLine();

//str = "aabaabaaaabaabaaab";

//word = "aabaab";

var start = DateTime.Now;

int res = KMP(str, word);

var end = DateTime.Now;

Console.WriteLine($"Затраченное время на выполнение: {start - end}");

Console.WriteLine($"Индекс начала искомой подстроки в строке: {res}");

Console.WriteLine("Введите строку для поиска или 0 чтобы выйти");

str = Console.ReadLine();

}

#endregion

#region Boyer

Console.WriteLine("Введите строку для поиска или 0 чтобы перейти к следующему заданию");

string str\_bo = Console.ReadLine();

while (str\_bo != "0")

{

Console.WriteLine("Введите подстроку для поиска");

string word = Console.ReadLine();

//str = "Hoola-hoola girls like hooligans";

//word = "h";

var start = DateTime.Now;

int res = BoyerMoor(str, word);

var end = DateTime.Now;

Console.WriteLine($"Затраченное время на выполнение: {start - end}");

Console.WriteLine($"Индекс начала искомой подстроки в строке: {res}");

Console.WriteLine("Введите строку для поиска или 0 чтобы выйти");

str = Console.ReadLine();

}

#endregion

#endif

#if taken

#region Taken

Console.WriteLine("Введите 1 чтобы ввести массив вручную, 2 сгенерировать случайный массив или 0 чтобы выйти");

var arr = new int?[16];

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

while (enter != 0)

{

if (enter == 1)

{

arr = new int?[16];

int count = 1;

var avialible = new List<int>(16);

for (int i = 0; i < avialible.Capacity; i++)

{

avialible.Add(i);

}

Console.WriteLine($"Введите исходную последовательность неповторяющихся чисел от 0 до 15 включительно для игры в пятнашки или 20 для перехода к следующему заданию");

while (count < 17)

{

int entered;

Console.WriteLine($"Введите {count} число");

try

{

entered = int.Parse(Console.ReadLine());

}

catch (FormatException)

{

Console.WriteLine($"Ошибка преобразования типов. Повторите ввод");

continue;

}

if (!arr.Contains(entered) && entered < 16 && entered >= 0)

{

arr[count - 1] = entered;

avialible.Remove(entered);

count++;

}

else

{

Console.WriteLine($"Введённое число уже есть в массиве или не входит в заданный диапазон! Введите другое число.");

}

Console.Write($"Доступные числа для ввода: ");

foreach (var item in avialible)

{

Console.Write(item + " ");

}

}

}

else if (enter == 2)

{

arr = new int?[16];

int count = 1;

var avialible = new List<int>(16);

for (int i = 0; i < avialible.Capacity; i++)

{

avialible.Add(i);

}

Random rand = new Random();

for (int i = 0; i < 16;)

{

var next = rand.Next(0, 16);

if (!arr.Contains(next))

{

arr[i] = next;

i++;

}

}

}

var start = DateTime.Now;

int[] res = Taken(arr /\*new int?[] {1,2,3,4,5,6,7,8,9,11,0,14,13,15,10,12}\*/);//закомментирована просто одна из допустимых начальных комбинаций

var end = DateTime.Now;

Console.WriteLine($"Полученная последовательность перестановок: ");

if (res != null)

{

foreach (var item in res)

{

Console.Write(item + " ");

}

}

else

{

Console.WriteLine("Нерешаемая комбинация.");

}

Console.WriteLine();

Console.WriteLine($"Затраченное время на выполнение: {start - end}");

Console.WriteLine("1: Ввести новый массив вручную\n2: Cгенерировать случайный массив\n0: Выйти\nЛюбое другое число: - повторить");

enter = int.Parse(Console.ReadLine());

}

#endregion

#endif  
}

/// <summary>

/// Алгоритм Боуера-Мура. Принцип поиска основан на смещении относительно текущего символа на

/// определённое "стоимостью" этого символа расстояния, которое определяется позицией этого символа в подстроке.

/// </summary>

/// <param name="str">Входная строка</param>

/// <param name="word">Образ</param>

/// <returns>Индекс начала подстроки</returns>

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 исходный образ чтобы было удобнее манипулировать

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;

}

/// <summary>

/// Алгоритм КНута-Морриса-Прата. Принцип основан на подсчёте префикс-функции для исходной строки и образа

/// </summary>

/// <param name="str">Исходная строка</param>

/// <param name="word">Образ</param>

/// <returns>Индекс начала подстроки</returns>

static int KMP(string str, string word)

{

bool flag = true;

var tmp = str + word;

char ch = ' ';

while (flag)

{

if (tmp.Contains(ch))

{

ch++;

}

else

{

flag = false;

}

}

tmp = word + ch + str;

var prefs = new int[tmp.Length];

CalcPrefs(tmp);

for (int i = word.Length; i < prefs.Length; i++)

{

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

{

return i - word.Length \* 2;

}

}

return -1;

///функция подсчёта префикса для строки

void CalcPrefs(string word)

{

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;

}

}

}

}

/// <summary>

/// Пятнашки. Суть кратко: на входе одномерный массив [0;15], 0 символизирует пустую клетку, остальные числа свои клетки

/// На основе массива составляется игровое поле 4х4, если комбинация решаемая, каждый элемент последовательно ставится на своё место. Иначе возвращается null

/// </summary>

/// <param name="arr"></param>

/// <returns></returns>

static int[] Taken(int?[] arr)

{

var counter = 0;

///сначала определяем, решаема ли комбинация

for (int i = 0; i < arr.Length - 1; i++)

{

if (arr[i] == 0)

{

continue;

}

for (int j = i + 1; j < arr.Length; j++)

{

if (arr[j] == 0)

{

continue;

}

if (arr[i] > arr[j])

{

counter++;

}

}

}

counter += (Array.IndexOf(arr, 0) / 4) + 1;

if (counter % 2 == 1)//если нерешаемая

{

return null;

}

///иначе создаём поле на основе массива

var field = new Field(arr);

field.PlaceFirstRow();//размещаем первый ряд

field.PlaceFirstCol();//размещаем первую колонку

field.PlaceSecondRow();//второй ряд

field.PlaceRest();//оставшиеся 3 клетки

Field.counter = 0;

return Field.movelist.ToArray();//возвращаем результат в виде массива

}

}

}

using System;

namespace Lab3

{

/// <summary>

/// Класс представляет отдельную клетку на поле, которая хранит информацию о своих соседях, о своём состоянии, текущих и целевых координатах

/// </summary>

public class Cell

{

internal bool Placed { get; set; }

internal bool Actual { get; set; }

internal int Value { private set; get; }

internal int Row { set; get; }

internal int Col { set; get; }

internal int TargRow { set; get; }

internal int TargCol { set; get; }

/// <summary>

/// Проверяет, на своём ли месте клетка

/// </summary>

internal bool CheckPlace()

{

if (ColIsPlaced() && RowIsPlaced())

{

Placed = true;

return true;

}

else

{

Placed = false;

return false;

}

}

internal Cell Up { set; get; }

internal Cell Right { set; get; }

internal Cell Down { set; get; }

internal Cell Left { set; get; }

internal string Previous { get; set; }

/// <summary>

/// Конструктор

/// </summary>

/// <param name="val">Числовое значение</param>

/// <param name="r">Ряд клетки на момент создания</param>

/// <param name="c">Столбец на момент создания</param>

public Cell(int val, int r, int c)

{

if (val == 0)

{

TargCol = 3;

TargRow = 3;

}

else

{

TargCol = (val - 1) % 4;

TargRow = (val - 1) / 4;

}

Value = val;

Row = r;

Col = c;

}

/// <summary>

/// Проверяет на соседство вызывающей клетки с параметром

/// </summary>

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

/// <returns></returns>

internal bool IsNear(Cell actual)

{

if (Up == actual || Right == actual || Left == actual || Down == actual)

{

return true;

}

return false;

}

internal bool RowIsPlaced()

{

return Row == TargRow;

}

internal bool ColIsPlaced()

{

return Col == TargCol;

}

/// <summary>

/// Определяет в каком направлении находится по отношению к цели с учётом приоритета

/// </summary>

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

/// <returns></returns>

internal Direction GetDirectionToMove(Cell actual)

{

if (Row > actual.Row && !Up.Placed)//если 0 снизу цели и над 0 неустановленная клетка

{

return Direction.Up;

}

if (Row < actual.Row && !Down.Placed)//если 0 сверху цели и под 0 неустановленная

{

return Direction.Down;

}

if (Col < actual.Col && !Right.Placed)//если 0 слева от цели и справа от 0 неустановленная

{

return Direction.Right;

}

else//если 0 справа от цели

{

return Direction.Left;

}

}

/// <summary>

/// Определяет с какой стороны от вызывающей находится переданная в параметр. Работает корректно если они действительно граничат.

/// </summary>

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

/// <returns></returns>

internal Direction GetDirection(Cell actual)

{

if (Up == actual)//если 0 снизу цели

{

return Direction.Up;

}

if (Down == actual)//если 0 сверху цели

{

return Direction.Down;

}

if (Right == actual)//если 0 слева от цели

{

return Direction.Right;

}

else//если 0 справа от цели

{

return Direction.Left;

}

}

/// <summary>

/// Нужно ли вверх

/// </summary>

/// <returns></returns>

internal bool NeedUp()

{

if (Row > TargRow)

{

return true;

}

return false;

}

/// <summary>

/// Нужно ли влево

/// </summary>

/// <returns></returns>

internal bool NeedLeft()

{

if (Col > TargCol)

{

return true;

}

return false;

}

/// <summary>

/// Нужно ли вправо

/// </summary>

/// <returns></returns>

internal bool NeedRight()

{

if (Col < TargCol)

{

return true;

}

return false;

}

/// <summary>

/// Может ли вправо

/// </summary>

/// <returns></returns>

internal bool CanUp()

{

if (Up != null && !Up.Placed)

{

return true;

}

return false;

}

/// <summary>

/// Может ли вниз

/// </summary>

/// <returns></returns>

internal bool CanDown()

{

if (Down != null && !Down.Placed)

{

return true;

}

return false;

}

/// <summary>

/// Может ли влево

/// </summary>

/// <returns></returns>

internal bool CanLeft()

{

if (Left != null && !Left.Placed)

{

return true;

}

return false;

}

/// <summary>

/// Перечисление для определения направления в методах, где нужно узнать приоритет движения

/// </summary>

public enum Direction

{

Up = 0, Right = 1, Down = 2, Left = 3

}

/// <summary>

/// Возвращает предпочтительное направление движения перемещаемой клетки

/// </summary>

/// <returns></returns>

internal Direction GetDirectionToMove()

{

if (NeedLeft() && !Left.Placed)

{

return Direction.Left;

}

if (NeedRight() && !Right.Placed)

{

return Direction.Right;

}

if (NeedUp() && !Up.Placed)

{

return Direction.Up;

}

else

{

return Direction.Down;

}

}

}

}

using System;

using System.Threading;

namespace Lab3

{

/// <summary>

/// Класс расширений для двумерного массива, в основном в нём все взаимодействия с игровым полем

/// </summary>

public static class MatrixExtension

{

/// <summary>

/// Перемещает заданную клетку влево

/// </summary>

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

/// <param name="replaced">Перемещаемая клетка</param>

public static void MoveLeft(this Cell[,] map, Cell replaced)

{

Field.counter++;

Cell temp = replaced.Left;

Field.movelist.Add(temp.Value);

map[temp.Row, temp.Col] = replaced;

map[replaced.Row, replaced.Col] = temp;

temp.Col += 1;

replaced.Col -= 1;

replaced.Previous = "left";

temp.Previous = "right";

map.BindNeighbours();

map.Print();

Thread.Sleep(Field.delay);

}

public static void MoveDown(this Cell[,] map, Cell replaced)

{

Field.counter++;

Cell temp = replaced.Down;

Field.movelist.Add(temp.Value);

map[temp.Row, temp.Col] = replaced;

map[replaced.Row, replaced.Col] = temp;

temp.Row -= 1;

replaced.Row += 1;

replaced.Previous = "down";

temp.Previous = "up";

map.BindNeighbours();

map.Print();

Thread.Sleep(Field.delay);

}

/// <summary>

/// Параметром передаётся ПЕРЕДВИГАЕМАЯ в указанном направлении ячейка

/// </summary>

/// <param name="replaced"></param>

public static void MoveUp(this Cell[,] map, Cell replaced)

{

Field.counter++;

Cell temp = replaced.Up;

Field.movelist.Add(temp.Value);

map[temp.Row, temp.Col] = replaced;

map[replaced.Row, replaced.Col] = temp;

temp.Row += 1;

replaced.Row -= 1;

replaced.Previous = "up";

temp.Previous = "down";

map.BindNeighbours();

map.Print();

Thread.Sleep(Field.delay);

}

public static void MoveRight(this Cell[,] map, Cell replaced)

{

Field.counter++;

Cell temp = replaced.Right;

Field.movelist.Add(temp.Value);

map[temp.Row, temp.Col] = replaced;

map[replaced.Row, replaced.Col] = temp;

temp.Col -= 1;

replaced.Col += 1;

replaced.Previous = "right";

temp.Previous = "left";

map.BindNeighbours();

map.Print();

Thread.Sleep(Field.delay);

}

/// <summary>

/// Возвращает клетку поля с заданным значением

/// </summary>

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

/// <param name="val"></param>

/// <returns></returns>

public static Cell GetNum(this Cell[,] map, int val)

{

int size = 4;

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

{

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

{

if (map[i, j].Value == val)

{

return map[i, j];

}

}

}

return null;

}

/// <summary>

/// Двигает 0 по направлению к цели, пока не станет его соседом

/// </summary>

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

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

public static void MoveToNeighbourhood(this Cell[,] map, Cell actual)

{

var zero = map.GetNum(0);

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

{

switch (zero.GetDirectionToMove(actual))

{

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

map.MoveDown(zero);

break;

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

map.MoveLeft(zero);

break;

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

map.MoveUp(zero);

break;

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

map.MoveRight(zero);

break;

default:

break;

}

}

}

/// <summary>

/// Вывод карты на консоль на текущее состояние поля

/// </summary>

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

public static void Print(this Cell[,] map)

{

var size = 4;

Console.WriteLine(Field.counter);

Console.WriteLine();

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

{

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

{

if (map[i, j].Actual)

{

Console.ForegroundColor = ConsoleColor.Green;

}

if (map[i, j].Value.ToString().Length == 1)

{

if (map[i, j].Value == 0)

{

Console.ForegroundColor = ConsoleColor.Red;

}

Console.Write(" " + map[i, j].Value + "|");

}

else

{

Console.Write(map[i, j].Value + "|");

}

Console.ForegroundColor = ConsoleColor.White;

}

Console.WriteLine();

}

Console.WriteLine();

}

/// <summary>

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

/// </summary>

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

public static void BindNeighbours(this Cell[,] map)

{

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

{

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

{

var temp = map[i, j];

var col = temp.Col;

var row = temp.Row;

if (row != 0)

{

temp.Up = map[row - 1, col];

}

else

{

temp.Up = null;

}

if (col != 3)

{

temp.Right = map[row, col + 1];

}

else

{

temp.Right = null;

}

if (row != 3)

{

temp.Down = map[row + 1, col];

}

else

{

temp.Down = null;

}

if (col != 0)

{

temp.Left = map[row, col - 1];

}

else

{

temp.Left = null;

}

}

}

}

}

}

using System;

using System.Collections.Generic;

namespace Lab3

{

/// <summary>

/// Представляет поле 4х4, состоящее из клеток

/// </summary>

class Field

{

internal static long counter = 0;

internal static int delay = 300;

internal static List<int> movelist;

Cell[,] map = new Cell[4, 4];

/// <summary>

/// Клнструктор

/// </summary>

/// <param name="arr"></param>

public Field(int?[] arr)

{

int size = 4;

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

{

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

{

map[i, j] = new Cell((int)arr[j + i \* size], i, j);

}

}

map.BindNeighbours();

movelist = new List<int>();

}

/// <summary>

/// Метод для размещения первого ряда

/// </summary>

internal void PlaceFirstRow()

{

map.Print();

PlaceNum(map.GetNum(1));

PlaceNum(map.GetNum(2));

PlaceNum(map.GetNum(3));

var four = map.GetNum(4);

if (four.CheckPlace())

{

return;

}

four.TargRow = 1;

four.TargCol = 2;

var zero = map.GetNum(0);

if (zero.Row == 0 && zero.Col == 3 && zero.Down == four)

{

four.Actual = true;

map.MoveDown(zero);

four.Actual = false;

}

else if (!(four.Row == 0 && four.Col == 3))

{

PlaceNum(four);

BlackMagic1(four);

}

}

/// <summary>

/// Мтод размещения первой колонки

/// </summary>

internal void PlaceFirstCol()

{

var five = map.GetNum(5);

var nine = map.GetNum(9);

var thirteen = map.GetNum(13);

if (five.CheckPlace() && nine.CheckPlace() && thirteen.CheckPlace())

{

return;

}

var zero = map.GetNum(0);

PlaceNum(thirteen);

if (!nine.CheckPlace())

{

PlaceNum(nine);

}

if (zero.Right == five && zero.Down == nine)

{

five.Actual = true;

map.MoveRight(zero);

five.CheckPlace();

five.Actual = false;

}

if (!five.CheckPlace())

{

five.TargCol = 1;

five.TargRow = 2;

PlaceNum(five);

BlackMagic2(five);

}

}

/// <summary>

/// Метод размещения последних трёх клеток поля для одной из конечных конфигураций

/// </summary>

/// <param name="fourteen"></param>

private void BlackMagic5(Cell fourteen)

{

fourteen.Actual = true;

var zero = map.GetNum(0);

while (!zero.IsNear(fourteen))

{

switch (zero.GetDirectionToMove(fourteen))

{

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

map.MoveDown(zero);

break;

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

map.MoveLeft(zero);

break;

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

map.MoveUp(zero);

break;

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

map.MoveRight(zero);

break;

default:

break;

}

}

map.MoveLeft(zero);

map.MoveDown(zero);

map.MoveRight(zero);

map.MoveRight(zero);

map.MoveUp(zero);

map.MoveLeft(zero);

map.MoveDown(zero);

map.MoveLeft(zero);

map.MoveUp(zero);

map.MoveRight(zero);

map.MoveRight(zero);

map.MoveDown(zero);

map.MoveLeft(zero);

map.MoveUp(zero);

fourteen.Actual = false;

}

/// <summary>

/// Метод размещения последних трёх клеток поля для одной из конечных конфигураций

/// </summary>

/// <param name="fourteen"></param>

private void BlackMagic4(Cell fourteen)

{

fourteen.Actual = true;

var zero = map.GetNum(0);

while (!zero.IsNear(fourteen))

{

switch (zero.GetDirectionToMove(fourteen))

{

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

map.MoveDown(zero);

break;

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

map.MoveLeft(zero);

break;

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

map.MoveUp(zero);

break;

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

map.MoveRight(zero);

break;

default:

break;

}

}

switch (zero.GetDirection(fourteen))

{

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

map.MoveUp(zero);

map.MoveLeft(zero);

goto case Cell.Direction.Down;

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

map.MoveLeft(zero);

map.MoveDown(zero);

map.MoveRight(zero);

break;

default:

break;

}

fourteen.Actual = false;

}

/// <summary>

/// Метод размещения последних трёх клеток поля

/// </summary>

internal void PlaceRest()

{

var zero = map.GetNum(0);

var ten = map.GetNum(10);

var eleven = map.GetNum(11);

var twelve = map.GetNum(12);

var fourteen = map.GetNum(14);

var fifteen = map.GetNum(15);

if (ten.CheckPlace() && eleven.CheckPlace() && twelve.CheckPlace())

{

PlaceNum(fourteen);

PlaceNum(fifteen);

return;

}

else

{

ten.TargRow = 3;

PlaceNum(ten);

if (ten.Up == fourteen)

{

BlackMagic5(fourteen);

}

if (zero.Down == ten && zero.Right == fourteen) //если требуется перемешать

{

fourteen.Actual = true;

map.MoveDown(zero);

map.MoveRight(zero);

map.MoveUp(zero);

map.MoveRight(zero);

map.MoveDown(zero);

map.MoveLeft(zero);

map.MoveLeft(zero);

map.MoveUp(zero);

map.MoveRight(zero);

fourteen.Actual = false;

}

fourteen.TargCol = 2;

PlaceNum(fourteen);

BlackMagic4(fourteen);

PlaceNum(eleven);

PlaceNum(twelve);

PlaceNum(fifteen);

}

}

/// <summary>

/// Метод размещения второго ряда

/// </summary>

internal void PlaceSecondRow()

{

var zero = map.GetNum(0);

var six = map.GetNum(6);

var seven = map.GetNum(7);

var eight = map.GetNum(8);

if (six.CheckPlace() && seven.CheckPlace() && eight.CheckPlace())

{

return;

}

PlaceNum(six);

PlaceNum(seven);

if (eight.Col == 3 && eight.Row == 2 && eight.Up == zero)

{

PlaceNum(eight);

}

if (!eight.CheckPlace())

{

eight.TargCol = 2;

eight.TargRow = 2;

PlaceNum(eight);

BlackMagic3(eight);

}

}

/// <summary>

/// Метод размещения одной из конечных конфигураций для второго ряда

/// </summary>

/// <param name="eight"></param>

private void BlackMagic3(Cell eight)

{

eight.Actual = true;

var zero = map.GetNum(0);

while (!zero.IsNear(eight))

{

switch (zero.GetDirectionToMove(eight))

{

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

map.MoveDown(zero);

break;

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

map.MoveLeft(zero);

break;

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

map.MoveUp(zero);

break;

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

map.MoveRight(zero);

break;

default:

break;

}

}

switch (zero.GetDirection(eight))

{

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

map.MoveDown(zero);

map.MoveLeft(zero);

goto case Cell.Direction.Up;

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

map.MoveLeft(zero);

map.MoveUp(zero);

goto case Cell.Direction.Right;

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

map.MoveUp(zero);

map.MoveRight(zero);

map.MoveDown(zero);

map.MoveRight(zero);

map.MoveUp(zero);

map.MoveLeft(zero);

map.MoveLeft(zero);

map.MoveDown(zero);

break;

default:

break;

}

eight.Actual = false;

}

/// <summary>

/// Метод размещения первого столбца поля для одной из конечных конфигураций

/// </summary>

/// <param name="five"></param>

private void BlackMagic2(Cell five)

{

five.Actual = true;

var zero = map.GetNum(0);

while (!zero.IsNear(five))

{

switch (zero.GetDirectionToMove(five))

{

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

map.MoveDown(zero);

break;

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

map.MoveLeft(zero);

break;

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

map.MoveUp(zero);

break;

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

map.MoveRight(zero);

break;

default:

break;

}

}

switch (zero.GetDirection(five))

{

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

map.MoveDown(zero);

map.MoveLeft(zero);

goto case Cell.Direction.Up;

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

map.MoveLeft(zero);

map.MoveUp(zero);

map.MoveRight(zero);

map.MoveUp(zero);

map.MoveLeft(zero);

map.MoveDown(zero);

map.MoveDown(zero);

map.MoveRight(zero);

break;

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

map.MoveRight(zero);

map.MoveDown(zero);

goto case Cell.Direction.Left;

default:

break;

}

five.Actual = false;

}

private void BlackMagic1(Cell four)

{

var zero = map.GetNum(0);

four.Actual = true;

while (!zero.IsNear(four))

{

switch (zero.GetDirectionToMove(four))

{

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

map.MoveDown(zero);

break;

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

map.MoveLeft(zero);

break;

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

map.MoveUp(zero);

break;

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

map.MoveRight(zero);

break;

default:

break;

}

} switch (zero.GetDirection(four))

{

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

map.MoveDown(zero);

map.MoveLeft(zero);

goto case Cell.Direction.Up;

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

map.MoveLeft(zero);

map.MoveUp(zero);

goto case Cell.Direction.Right;

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

map.MoveLeft(zero);

map.MoveUp(zero);

map.MoveRight(zero);

map.MoveRight(zero);

map.MoveDown(zero);

map.MoveRight(zero);

map.MoveUp(zero);

map.MoveLeft(zero);

map.MoveLeft(zero);

map.MoveLeft(zero);

map.MoveDown(zero);

break;

default:

break;

}

four.Actual = false;

}

/// <summary>

/// Метод размещения элементарной клетки на поле

/// </summary>

/// <param name="target"></param>

private void PlaceNum(Cell target)

{

var zero = map.GetNum(0);

target.Actual = true;

target.CheckPlace();

if (target.Placed)

{

target.Actual = false;

return;

}

while (!target.Placed)//двигаем пока не поставим на своё место

{

map.MoveToNeighbourhood(target);//пока не будем рядом с целью

Cell.Direction destination\_side = target.GetDirectionToMove();

/\*

определяем, в каком направлении двигаться

приоритет: лево-право, вверх-вниз.

определяем кратчайший для передвижения с учётом уже размещённых

\*/

switch (zero.GetDirection(target))//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

map.MoveRight(zero);

map.MoveUp(zero);

break;

case Cell.Direction.Right://надо вправо

map.MoveRight(zero);

map.MoveUp(zero);

break;

case Cell.Direction.Down://надо вниз

map.MoveUp(zero);

break;

case Cell.Direction.Left://надо влево

if (zero.CanLeft())

{

map.MoveLeft(zero);

map.MoveUp(zero);

}

else

{

map.MoveRight(zero);

map.MoveUp(zero);

}

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 (zero.CanUp())

{

map.MoveUp(zero);

map.MoveRight(zero);

}

else

{

map.MoveDown(zero);

map.MoveRight(zero);

}

break;

case Cell.Direction.Right://надо вправо

if (zero.CanUp())

{

map.MoveUp(zero);

map.MoveRight(zero);

}

else

{

map.MoveDown(zero);

map.MoveRight(zero);

}

break;

case Cell.Direction.Down://надо вниз

if (zero.CanDown())

{

map.MoveDown(zero);

map.MoveRight(zero);

}

else if (zero.CanUp())

{

map.MoveUp(zero);

map.MoveRight(zero);

}

break;

case Cell.Direction.Left://надо влево

map.MoveRight(zero);

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

map.MoveDown(zero);

break;

case Cell.Direction.Right://надо вправо

map.MoveRight(zero);

map.MoveDown(zero);

break;

case Cell.Direction.Down://надо вниз

map.MoveRight(zero);

map.MoveDown(zero);

break;

case Cell.Direction.Left://надо влево

if (zero.CanLeft() && target.CanLeft())//и слева не установленная клетка и целевая может идти влево

{

map.MoveLeft(zero);

map.MoveDown(zero);

}

else

{

map.MoveRight(zero);

map.MoveDown(zero);

}

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

map.MoveUp(zero);

map.MoveLeft(zero);

break;

case Cell.Direction.Right://надо вправо

map.MoveLeft(zero);

break;

case Cell.Direction.Down://надо вниз

map.MoveDown(zero);

map.MoveLeft(zero);

break;

case Cell.Direction.Left://надо влево

if ((zero.CanDown() && zero.Previous != "up") || target.Row == 0)

{

map.MoveDown(zero);

map.MoveLeft(zero);

}

else

{

map.MoveUp(zero);

map.MoveLeft(zero);

}

break;

default:

break;

}

break;

default:

break;

}

target.CheckPlace();

}

target.Actual = false;

}

}

}

Результаты выполнения программы представлены на рисунках ниже:

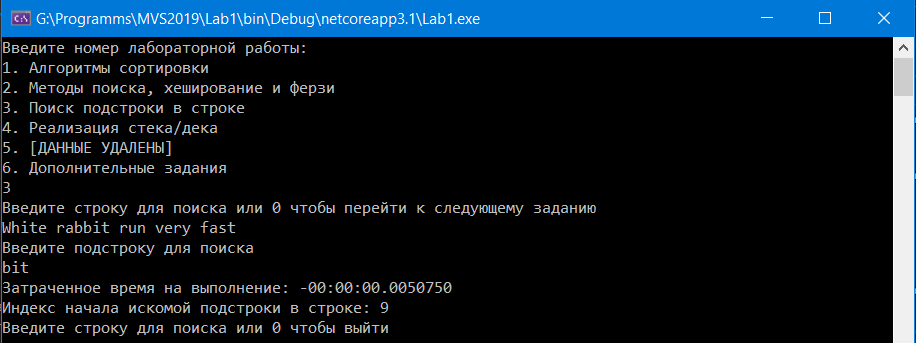


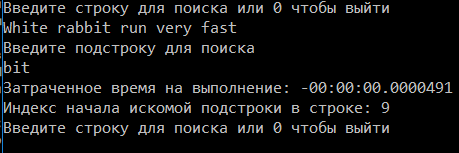
Рис. 1 – Результат поиска алгоритмом Кнутта-Морриса-Прата  
  


Рис. 2 – Результат поиска алгоритмом Боуера-Мура

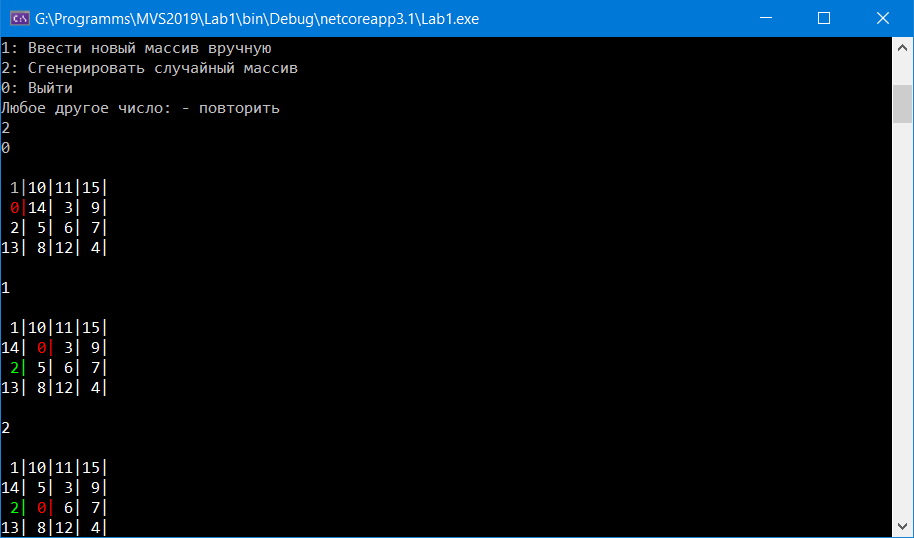


Рис. 3 – Начальная комбинация пятнашек

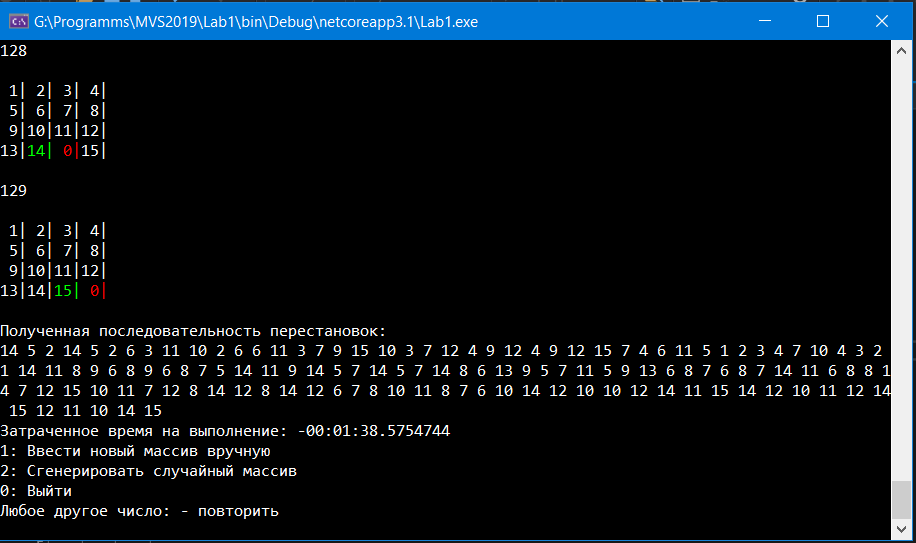


Рис. 4 – Результат работы программы

**3. Вывод**

В ходе данной работы были реализованы различные методы поиска подстроки в строке, время выполнения которых незначительно уступает встроенным методам поиска языка C#.

Кроме того, была реализована программа для решения игры «пятнашки». Эта программа решает все допустимые начальные комбинации игры, а в случае введения нерешаемой комбинации, её работа прекращается, а пользователь уведомляется о невозможности решения.

Дополнительные задачи:

Вернуть количество отдельных непустых подстрок текста, которые могут быть записаны как конкатенация некоторой строки с самой собой (т.е. она может быть записана, как a + a, где a - некоторая строка).

static int ConcatSum(string str)

{

int counter = 0;

var subs = new List<string>();

for (int i = 0; i < str.Length - 1; i++)

{

int j = i + 1;

while (j <= str.Length)

{

string left = str.Substring(i, j - i);

string right;

if (left.Length > (str.Length - j))

{

right = str.Substring(j);

}

else

{

right = str.Substring(j, left.Length);

}

if (left.Equals(right) && !subs.Contains(left))

{

subs.Add(left);

counter++;

}

j++;

}

}

return counter;

}

Дана строка s, вернуть самую длинную полиндромную подстроку в s.

static string LongestPalindrome(string input)

{

input = input.ToLower();

string res = string.Empty;

for (int i = 0, j = i - 1, k = i + 1; i < input.Length; i++)

{

j = i - 1;

k = i + 1;

while (j >= 0 && k < input.Length)

{

string temp = input.Substring(j, k - j + 1);

if (IsPalindrome(temp) && temp.Length > res.Length)

{

res = temp;

j--;

k++;

}

else

{

break;

}

}

j = i - 1;

k = i + 2;

while (j >= 0 && k < input.Length)

{

string temp = input.Substring(j, k - j + 1);

if (IsPalindrome(temp) && temp.Length > res.Length)

{

res = temp;

j--;

k++;

}

else

{

break;

}

}

}

return res;

bool IsPalindrome(string str)

{

if (str.Length % 2 == 0)

{

int index = str.Length / 2;

string left = str.Substring(0, index);

var right = new string(str.Substring(index).ToCharArray().Reverse().ToArray());

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

{

if (left[i] != right[i])

{

return false;

}

}

return true;

}

else

{

int index = str.Length / 2;

string left = str.Substring(0, index);

var right = new string(str.Substring(index + 1).ToCharArray().Reverse().ToArray());

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

{

if (left[i] != right[i])

{

return false;

}

}

return true;

}

}

}

Даны две строки: s1 и s2 с одинаковым размером, проверьте, может ли некоторая перестановка строки s1 “победить” некоторую перестановку строки s2 или наоборот. Строка x может “победить” строку y (обе имеют размер n), если x[i]> = y [i] (в алфавитном порядке) для всех i от 0 до n-1.  
static bool CoolestString(string first, string second)

{

if (first.Length != second.Length)//если начальные условия не выполняются

{

return false;

}

string tempf = first;

var listf = new List<string>();

string temps = second;

var lists = new List<string>();

do//в этом блоке циклов получаем всевозможные варианты перестановок символов в первом слове

{

var innerf = tempf;

do

{

tempf = new string(tempf.Append(tempf[0]).ToArray()).Remove(0, 1);

listf.Add(tempf);

} while (tempf != new string(innerf[innerf.Length - 1] + innerf.Remove(innerf.Length - 1, 1)));

var fch = tempf[0];

var lch = tempf[tempf.Length - 1];

tempf = tempf.Substring(1, tempf.Length - 2);

tempf = lch + tempf + fch;

listf.Add(tempf);

} while (tempf != first);

do//то же самое со вторым словом

{

var inners = temps;

do

{

temps = new string(temps.Append(temps[0]).ToArray()).Remove(0, 1);

lists.Add(temps);

} while (temps != new string(inners[inners.Length - 1] + inners.Remove(inners.Length - 1, 1)));

var fch = temps[0];

var lch = temps[temps.Length - 1];

temps = temps.Substring(1, temps.Length - 2);

temps = lch + temps + fch;

lists.Add(temps);

} while (temps != second);

foreach (var fs in listf)//проверяем, могут ли какие-то из перестановок в первом слове победить какие-то перестановки во втором

{

foreach (var ss in lists)

{

int counter = 0;

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

{

if (fs[i] >= ss[i])

{

counter++;

}

else

{

break;

}

}

if (counter == ss.Length)

{

return true;

}

}

}

foreach (var ss in lists)//то же самое наоборот

{

foreach (var fs in listf)

{

int counter = 0;

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

{

if (ss[i] >= fs[i])

{

counter++;

}

else

{

break;

}

}

if (counter == ss.Length)

{

return true;

}

}

}

return false;//если ничего не нашлось

}