**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет**

**имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

Факультет «Информатика и вычислительная техника»

Кафедра ИУ5 «Системы обработки информации и управления»

**Отчет по лабораторной работе № 5**

Задание: разработать программу, реализующую вычисление расстояния Левенштейна с использованием алгоритма Вагнера-Фишер

Студент Левин Михаил Александрович

*фамилия, имя, отчество*

Группа ИУ5-32Б

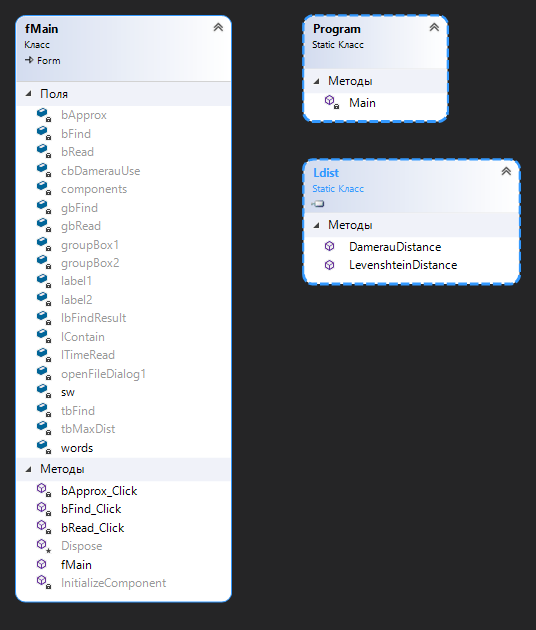
Москва, 2020 г.

# Задание

Разработать программу, реализующую вычисление расстояния Левенштейна с использованием алгоритма Вагнера-Фишера.

1. Программа должна быть разработана в виде библиотеки классов на языке C#.
2. Использовать самый простой вариант алгоритма без оптимизации.
3. Дополнительно возможно реализовать вычисление расстояния Дамерау-Левенштейна (с учетом перестановок соседних символов).
4. Модифицировать предыдущую лабораторную работу, вместо поиска подстроки используется вычисление расстояния Левенштейна.
5. Предусмотреть отдельное поле ввода для максимального расстояния. Если расстояние Левенштейна между двумя строками больше максимального, то строки считаются несовпадающими и не выводятся в список результатов.

# Диаграмма классов



# Текст программы

**fMain.cs**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

using System.Windows.Forms;

namespace Lab4

{

public partial class fMain : Form

{

private List<string> words;

private Stopwatch sw = new Stopwatch();

public fMain()

{

InitializeComponent();

}

private void bRead\_Click(object sender, EventArgs e)

{

openFileDialog1.InitialDirectory = Directory.GetCurrentDirectory();

openFileDialog1.Filter = "txt files (\*.txt)|\*.txt";

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

//Get the path of specified file

var filePath = openFileDialog1.FileName;

words = new List<string>();

sw.Reset();

sw.Start();

words.AddRange(File.ReadAllText(filePath).Split(new char[] { ' ', '.', ',', '!', '?', '/', '\t', '\n' },

StringSplitOptions.RemoveEmptyEntries));

sw.Stop();

lTimeRead.Text = $"Время считывания файла: {sw.Elapsed}";

}

}

private void bFind\_Click(object sender, EventArgs e)

{

if (words == null)

{

MessageBox.Show("Сначала необходимо считать какой-либо текстовый файл!");

return;

}

sw.Reset();

sw.Start();

lbFindResult.Items.Clear();

lbFindResult.BeginUpdate();

foreach (var word in words)

{

if (word.ToUpper().Contains(tbFind.Text.ToUpper()))

{

lbFindResult.Items.Add(word);

}

}

lbFindResult.EndUpdate();

sw.Stop();

lContain.Text = $"Поиск занял: {sw.Elapsed}";

}

private void bApprox\_Click(object sender, EventArgs e)

{

//Слово для поиска

string word = this.tbFind.Text.Trim();

//Если слово для поиска не пусто

if (!string.IsNullOrWhiteSpace(word) && words.Count > 0)

{

int maxDist;

if (!int.TryParse(this.tbMaxDist.Text.Trim(), out maxDist))

{

MessageBox.Show("Необходимо указать максимальное расстояние");

return;

}

if (maxDist < 1 || maxDist > 5)

{

MessageBox.Show("Максимальное расстояние должно быть в диапазоне от 1 до 5");

return;

}

//Слово для поиска в верхнем регистре

string wordUpper = word.ToUpper();

//Временные результаты поиска

List<Tuple<string, int>> tempList = new List<Tuple<string, int>>();

sw.Reset();

sw.Start();

foreach (string str in words)

{

//Вычисление расстояния

int dist = cbDamerauUse.Checked ? Ldist.LevenshteinDistance(str.ToUpper(), wordUpper) : Ldist.DamerauDistance(str.ToUpper(),wordUpper);

//Если расстояние меньше порогового, то слово добавляется в результат

if (dist <= maxDist)

{

tempList.Add(new Tuple<string, int>(str, dist));

}

}

sw.Stop();

this.lContain.Text = sw.Elapsed.ToString();

this.lbFindResult.BeginUpdate();

//Очистка списка

this.lbFindResult.Items.Clear();

//Вывод результатов поиска

foreach (var x in tempList)

{

string temp = x.Item1 + "(расстояние=" + x.Item2 + ")";

this.lbFindResult.Items.Add(temp);

}

this.lbFindResult.EndUpdate();

}

else

{

MessageBox.Show("Необходимо выбрать файл и ввести слово для поиска");

}

}

}

}

**LDist.cs**

using System;

namespace LDistLib

{

/// <summary>

/// Алгоритм Левенштейна с использованием алгоритма Вагнера-Фишера

/// </summary>

/// <param name="string1"></param>

/// <param name="string2"></param>

/// <returns></returns>

public static class Ldist

{

public static int LevenshteinDistance(string source, string dist)

{

//проверяем аргументы на null значения

if (source == null) throw new ArgumentNullException("source");

if (dist == null) throw new ArgumentNullException("dist");

//если строки равны - то нет необходимости что-то делать

if (source == dist) return 0;

//создаем прямогольную матрицу количество элементов произведения колич-во символов в строках+2

int[,] d = new int[source.Length + 1, dist.Length + 1];

//заполняем первую строку и первых столбец значениями порядковых номеров символов в строках

for (int i = 0; i <= source.Length; i++) d[i, 0] = i;

for (int j = 0; j <= dist.Length; j++) d[0, j] = j;

//Алгоритм Вагнера-Фишера

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

{

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

{

int diff = (source[i - 1] == dist[j - 1]) ? 0 : 1;

d[i, j] = Math.Min(

Math.Min(

d[i - 1, j] + 1, //цена удаления символа

d[i, j - 1] + 1), //цена вставки символа

d[i - 1, j - 1] + diff); //цена замены символа на символ

}

}

return d[source.Length, dist.Length];

}

/// <summary>

/// Вычисление расстояния Дамерау-Левенштейна

/// </summary>

/// <param name="source"></param>

/// <param name="dist"></param>

/// <returns></returns>

public static int DamerauDistance(string source, string dist)

{

//проверяем аргументы на null значения

if (source == null) throw new ArgumentNullException("source");

if (dist == null) throw new ArgumentNullException("dist");

//если строки равны - то нет необходимости что-то делать

if (source == dist) return 0;

//создаем прямогольную матрицу количество элементов произведения колич-во символов в строках+2

int[,] d = new int[source.Length + 1, dist.Length + 1];

//заполняем первую строку и первый столбец значениями индексов

for (int i = 0; i <= source.Length; i++) d[i, 0] = i;

for (int j = 0; j <= dist.Length; j++) d[0, j] = j;

//Алгоритм Вагнера-Фишера (с учётом перестановок)

//цикл по символам источника

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

{

//цикл по символам строки назначения

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

{

//если символы НЕ равны - то добавляем к стоимости единицу

int diff = (source[i - 1] == dist[j - 1]) ? 0 : 1;

//записываем минальную стоимость операции (удаления, вставки, замены)

d[i, j] = Math.Min(

Math.Min(

d[i - 1, j] + 1, //цена удаления символа

d[i, j - 1] + 1), //цена вставки символа

d[i - 1, j - 1] + diff); //цена замены символа на символ

//если совпали условия для перестановки символов - добавляем в стравнение стоимость перестаовки и берём снова минимальное значение

if (i > 1 && j > 1 && source[i - 1] == dist[j - 2] && source[i - 2] == dist[j - 1])

d[i, j] = Math.Min(d[i, j], d[i - 2, j - 2] + diff); //перестановка

}

}

return d[source.Length, dist.Length];

}

}

}

# Экранные формы

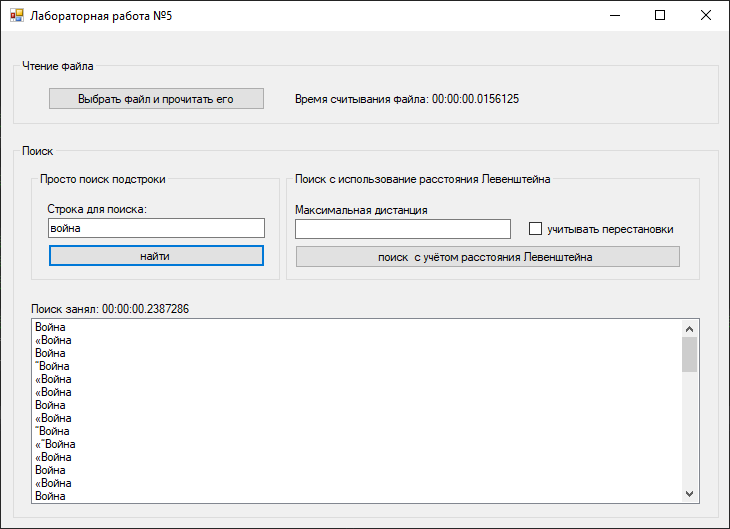


Рисунок 1 – результат простого поиска подстроки «война» в произведении Л.Н. Толстого «Война и мир»

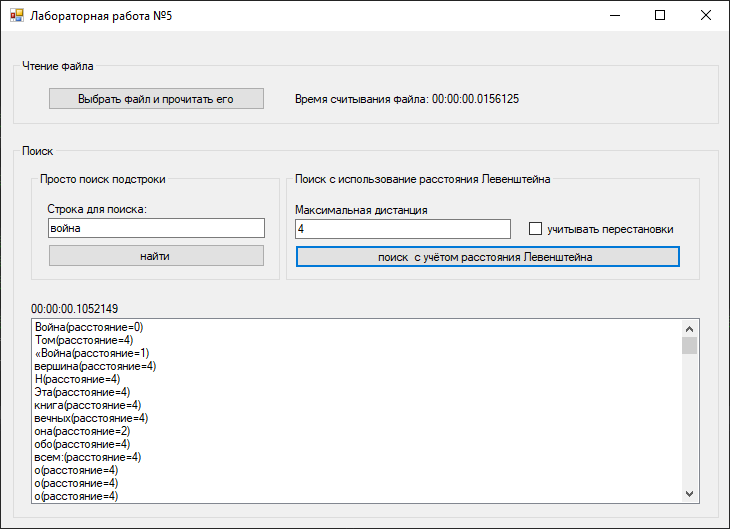


Рисунок 2 – результат поиска расстояния Левенштейна по слову «война» с максимальной дистанцией 4 св произведении Л.Н. Толстого «Война и мир»

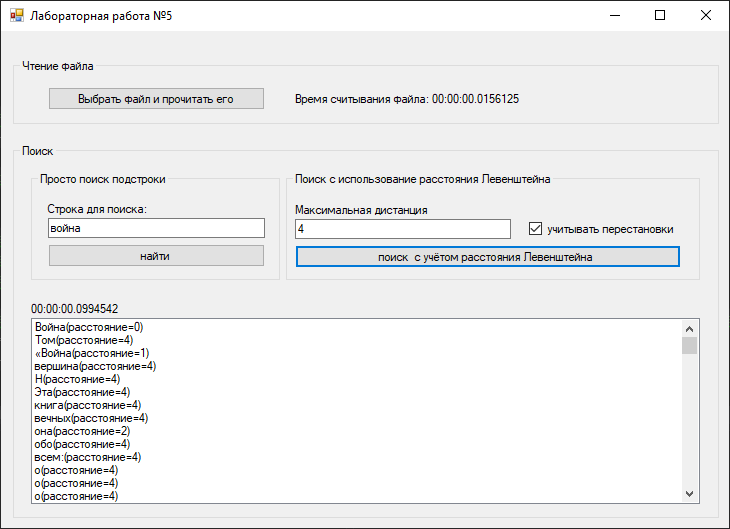


Рисунок 3 – результат поиска расстояния Левенштейна с учётом перестановок по слову «война» с максимальной дистанцией 4 св произведении Л.Н. Толстого «Война и мир»

# Контрольные вопросы:

1. Что такое расстояние Левенштейна?

Расстояние Левенштейна (редакционное расстояние, дистанция редактирования) — метрика, измеряющая разность между двумя последовательностями символов. Она определяется как минимальное количество односимвольных операций (а именно вставки, удаления, замены), необходимых для превращения одной последовательности символов в другую. В общем случае, операциям, используемым в этом преобразовании, можно назначить разные цены.

1. Приведите пример вычисления расстояния Левенштейна.

Пример вычисления расстояние Левенштейна:

Исходная строка: военный

Целевая строка: война

Расстояние Левенштейна: 4.

Составим таблицу последовательности действий, необходимо для получения из первой строки второй кратчайшим образом.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **М** | **М** | **R** | **M** | **R** | **D** | **D** |
| В | О | Е | Н | Н | Ы | Й |
| В | О | Й | Н | А |  |  |

Где первая строка – это строка действия над символами, D (англ. delete) — удалить, R (replace) — заменить, M (match) — совпадение.

Видим, что необходимо выполнить две замены, и два удаления – итого четыре действия.

1. Что такое расстояние Дамерау-Левенштейна?

Расстояние Дамерау — Левенштейна — это мера разницы двух строк символов, определяемая как минимальное количество операций вставки, удаления, замены и транспозиции (перестановки двух соседних символов), необходимых для перевода одной строки в другую. Является модификацией расстояния Левенштейна: к операциям вставки, удаления и замены символов, определённых в расстоянии Левенштейна добавлена операция транспозиции (перестановки) символов.

1. Приведите пример вычисления расстояния Дамерау-Левенштейна.

Пример вычисления расстояние Дамерау-Левештейна:

Исходная строка: рим

Целевая строка: рми

Расстояние Дамерау-Левенштейна: 1.

Требуется произвести одну перестановку двух соседних символ «и» и «м» в исходной строке, чтобы получить целевую строку.

1. В чем состоит поправка Дамерау?

Вводится новое возможное действие над символами – транспозиция (перестановка), суть которой заключается в том, что можно переставлять два рядом стоящих символа в исходной строке.

1. Для чего используется транспозиция в поправке Дамерау?

В случае возможности перестановки символов, это дает возможности уменьшить расстояние, так как заменяет два операции по изменению двух символов, на одну. Дамерау показал, что 80 % ошибок при наборе текста человеком являются транспозициями. Кроме того, расстояние Дамерау — Левенштейна используется и в биоинформатике.

1. Объясните алгоритм Вагнера-Фишера вычисления расстояния Дамерау-Левенштейна (на основе матрицы).

Обозначим исходную строку, как S1. Строку назначения, как S2.

Матрица занимает на один столбец и на одну строку больше, чем количество символов в S1 и S2. Пусть строки и столбцы матрицы нумеруются с нуля.

Нулевая строка матрицы заполняется ценой вставки соответствующего символа S2. Нулевой столбец матрицы заполняется ценой удаления соответствующего символа S1.

Далее идёт обход элементов матрицы построчно не затрагивая нулевые столбец и строку. При заполнении текущей ячейки рассматривается треугольник из действий: вставки символа, удаления символа, замены символа. Вычисляется вес этих действий и записывает в ячейку матрицы минимальный рассчитанный вес. При продвижении от начальной строки матрицы к конечной, рассчитываемого расстояние увеличивается. В конце прохода в прямом нижнем углу матрицы будет вычислено расстояние.

1. Как инициализируются начальные значения в матрице при вычислении расстояния Левенштейна? Почему?

Матрица занимает на один столбец и на одну строку больше, чем количество символов в S1 и S2. Пусть строки и столбцы матрицы нумеруются с нуля.

Нулевая строка матрицы заполняется ценой вставки соответствующего символа S2. Нулевой столбец матрицы заполняется ценой удаления соответствующего символа S1. Вес замены и вставки увеличивается пропорционально удалению символа в строке, это происходит, потому что необходимо учитывать накопление веса при продвижении в подсчете расстояния. Если бы эти значения были одинаковы, то при подсчете всегда этот текущий вес вставки или удаления был наименьшим и вес замены вообще бы не учитывался.

1. Как задать различные веса для операций удаления, добавления и замены?

В алгоритме Вагнера-Фишера, при выборе минимальной стоимости операции к каждой из них может быть добавляться своя цена. В нашем фрагменте кода, приведённом ниже, к каждой стоимости добавляется единица, которую можно заменить на другой вес в зависимости от операции.

d[i, j] = Math.Min(Math.Min(

d[i - 1, j] + 1, //цена удаления символа

d[i, j - 1] + 1), //цена вставки символа

d[i - 1, j - 1] + diff);

1. Как осуществить интеграцию разработанного метода в приложение Windows Forms.

Алгоритм, заключенный в метод, может быть описан в составе класса в виде статического метода. Сам класс может быть помещен в отдельную библиотеку, которая подключается к проектам по мере необходимости.