**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №5

«Создание программы для нахождения расстояния Левенштейна с применением алгоритма Вагнера-Фишера»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-31 |  | преподаватель каф. ИУ5 |
| Агличеев Михаил |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

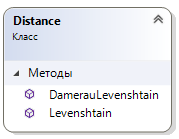
Москва, 2020г.

1. Описание задания

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.

2. UML-диаграмма классов



3. Листинг программы

Program.cs

using System;

using static Лабораторная\_\_\_5.Distance;

namespace Main

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Input 2 strings for comparison, each from new line");

string first = Console.ReadLine();

string second = Console.ReadLine();

Console.WriteLine("Resulting Levensthtein distance: {0}", Levenshtain(first, second));

Console.WriteLine("Resulting Damerau-Levensthtein distance: {0}", DamerauLevenshtain(first, second));

}

}

}

Distance.cs

using System;

namespace Лабораторная\_\_\_5

{

public class Distance

{

static public int Levenshtain(string first, string second)

{

int[,] d = new int[first.Length + 1, second.Length + 1];

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

{

d[i, 0] = i;

}

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

{

d[0, j] = j;

}

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

{

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

{

int substitutionCost = 0;

if (first[i - 1] != second[j - 1])

{

substitutionCost = 1;

}

d[i, j] = Math.Min(Math.Min(d[i - 1, j] + 1, // deletion

d[i, j - 1] + 1), // insertion

d[i - 1, j - 1] + substitutionCost); // substitution

}

}

return d[first.Length, second.Length];

}

static public int DamerauLevenshtain(string first, string second)

{

int[] alphabet = new int[255];

int[,] d = new int[first.Length + 2, second.Length + 2];

int maxDistance = first.Length + second.Length;

d[0, 0] = maxDistance;

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

{

d[i + 1, 0] = maxDistance;

d[i + 1, 1] = i;

}

for (int j = 0; j <= second.Length; ++j)

{

d[0, j + 1] = maxDistance;

d[1, j + 1] = j;

}

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

{

int db = 0;

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

{

int k = alphabet[second[j - 1] - 0]; // -0 converts char to int

int l = db;

int substitutionCost;

if (first[i - 1] == second[j - 1])

{

substitutionCost = 0;

db = j;

}

else

{

substitutionCost = 1;

}

d[i + 1, j + 1] = Math.Min(Math.Min(d[i, j + 1] + 1, // deletion

d[i + 1, j] + 1), // insertion

Math.Min(d[i, j] + substitutionCost, // substitution

d[k, l] + (i - k - 1) + 1 + (j - l - 1))); // transposition

}

alphabet[first[i - 1] - 0] = i;

}

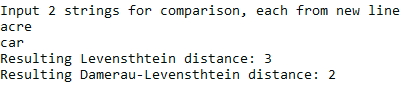
return d[first.Length + 1, second.Length + 1];

}

}

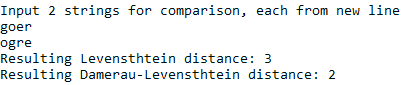
}

4. Результаты



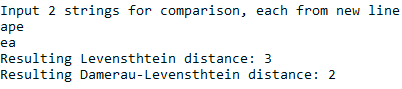
Левенштейн: acre->cre->care->car

Дамерау-Левенштейн: acre->care->car



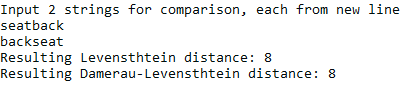
Левенштейн: goer->oer->ogr->ogre

Дамерау-Левенштейн: goer->oger->ogre



Левенштейн: ape->pe->e->ea

Дамерау-Левенштейн: ape->ae->ea



Данный пример показывает, что в расстоянии Дамерау-Левенштейна учитываются только перестановки только букв, а не групп букв.