**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №2

«**Расстояние Левенштейна.**»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-31Б |  | преподаватель каф. ИУ5 |
| Ларкин Б. В. |  | Гапанюк Ю. Е. |
| Подпись и дата: |  | Подпись и дата: |
|  |  |  |

Москва, 2023 г.

Постановка задачи

Разработать программу для решения биквадратного уравнения.

1. Программа должна быть разработана в виде консольного приложения.
2. Программа должна запрашивать повторный ввод при неверном вводе с клавиатуры.
3. Программа должна выводить дистанцию Левенштейна для двух слов, а также путь достижения слова-результата.

Текст программы

**LevensteinsDistance.cp, C#**

using System;

using System.Text;

namespace LevensteinApp

{

class LevensteinDistance

{

static string GetInput (string prompt)

{

Console.Write(prompt);

string Inp = Console.ReadLine();

int count = 0;

foreach (var el in Inp) if (char.IsLetter(el)) count++;

while (count == 0) {

Console.WriteLine("Enter a non-empty string");

Inp = Console.ReadLine();

foreach (var el in Inp) if (char.IsLetter(el)) count++;

}

//Having taken in a string with letters, delete everything non-letter

StringBuilder Result = new StringBuilder(count);

foreach (var x in Inp) if (char.IsLetter(x)) { Result.Append(char.ToUpper(x)); };

return Result.ToString();

}

static void Print\_Matrix(int[,] Matrix, string WordSource, string WordRes, string title) {

Console.WriteLine("\n{0}\n", title);

Console.Write("\t\t");

foreach (var x in WordSource) { Console.Write("{0}\t",x); }

Console.Write("\n");

for (int i = 0; i < Matrix.GetLength(0); i++)

{

if (i != 0) { Console.Write("{0}\t", WordRes[i - 1]); }

else { Console.Write("\t"); }

for (int j = 0; j < Matrix.GetLength(1); j++)

{

Console.Write(Matrix[i, j] + "\t");

}

Console.WriteLine();

}

}

static void Run\_It\_Back(int curr\_line, int curr\_column, string WordSource, string WordRes, int[,] Matrix)

{

if (curr\_line == 0 & curr\_column == 0) { return; }

StringBuilder NewWordRes = new StringBuilder(WordRes);

if (curr\_column == 0)

{

char tmp = NewWordRes[curr\_line - 1];

NewWordRes.Remove(curr\_line - 1, 1);

Run\_It\_Back(curr\_line - 1, curr\_column, WordSource, NewWordRes.ToString(), Matrix);

Console.WriteLine("We add {0} to {1} and recieve {2}", tmp, NewWordRes.ToString(), WordRes);

}

else if (curr\_line == 0)

{

NewWordRes.Insert(curr\_column - 1, WordSource[curr\_column - 1]);

Run\_It\_Back(curr\_line, curr\_column - 1, WordSource, NewWordRes.ToString(), Matrix);

Console.WriteLine("We delete {0} out of {1} to recieve {2}", WordSource[curr\_column - 1], NewWordRes.ToString(), WordRes);

}

else

{

if (Matrix[curr\_line, curr\_column] == Matrix[curr\_line - 1, curr\_column] + 1)

{ //Adding a letter to source

char tmp = NewWordRes[curr\_line - 1];

NewWordRes.Remove(curr\_line - 1, 1);

Run\_It\_Back(curr\_line - 1, curr\_column, WordSource, NewWordRes.ToString(), Matrix);

Console.WriteLine("We add {0} to {1} and recieve {2}", tmp, NewWordRes.ToString(), WordRes);

}

else if (Matrix[curr\_line, curr\_column] == Matrix[curr\_line, curr\_column - 1] + 1)

{ //Deleting a leter from source

NewWordRes.Insert(curr\_column - 1, WordSource[curr\_column - 1]);

Run\_It\_Back(curr\_line, curr\_column - 1, WordSource, NewWordRes.ToString(), Matrix);

Console.WriteLine("We delete {0} out of {1} to recieve {2}", WordSource[curr\_column - 1], NewWordRes.ToString(), WordRes);

}

else if (WordSource[curr\_column-1] == WordRes[curr\_line-1])

{

if (Matrix[curr\_line, curr\_column] == Matrix[curr\_line - 1, curr\_column - 1])

{

Run\_It\_Back(curr\_line - 1, curr\_column - 1, WordSource, NewWordRes.ToString(), Matrix);

//No Console output - there was no action performed because the letters matched

}

}

else

{ //The only option left - we change the letters

char tmp = NewWordRes[curr\_column - 1]; //We need to save that for the output

NewWordRes.Remove(curr\_line - 1, 1);

NewWordRes.Insert(curr\_line - 1, WordSource[curr\_column - 1]);

Run\_It\_Back(curr\_line - 1, curr\_column - 1, WordSource, NewWordRes.ToString(), Matrix);

Console.WriteLine("We replace {0} with {1} to recieve {2}", NewWordRes[curr\_line - 1], tmp, WordRes);

}

}

return;

}

static void Main()

{

Console.WriteLine("### Levenstein`s Distance Calculation Algorithm ###\n");

string Word1 = GetInput("Enter the word that is to be edited: ");

string Word2 = GetInput("Enter the result word: ");

Console.WriteLine("\nThe word {0} is to become the word {1}.\n", Word1, Word2);

int[,] VF\_Matrix = new int[Word2.Length + 1, Word1.Length + 1]; //Creating a Vagner-Fischer`s Matrix

//Suppose Word1 is the Source word, Word2 is the result

//Initializing the first line of the Matrix

for (int i = 0; i < Word1.Length + 1; ++i)

{

VF\_Matrix[0, i] = i;

}

//Initializing the first column of the Matrix

for (int j = 0; j < Word2.Length + 1; ++j)

{

VF\_Matrix[j, 0] = j;

}

//Filling the matrix up using VF method

for (int line = 1; line < VF\_Matrix.GetLength(0); ++line)

{

for (int column = 1; column < VF\_Matrix.GetLength(1); ++column)

{

//Whether we +1 the left-up num in the comparison depends on whether the letters of the current pair are equal

VF\_Matrix[line, column] = Math.Min(Math.Min(VF\_Matrix[line - 1, column] + 1, VF\_Matrix[line, column - 1] + 1), VF\_Matrix[line - 1, column - 1] + Convert.ToInt32(Word1[column-1] != Word2[line-1]));

}

}

//Printing the matrix to check

Print\_Matrix(VF\_Matrix, Word1, Word2, "Vagner - Fischer`s Matrix:");

//Got the Matrix, now the most faraway from (0,0) cell contains the Levenstein`s Distance.

Console.WriteLine("\nThe Levenstein`s Distance for the input pair of words = {0}\n", VF\_Matrix[Word2.Length, Word1.Length]);

//To trace the path of permutations, we`ll use a recurrent function

Run\_It\_Back(VF\_Matrix.GetLength(0)-1, VF\_Matrix.GetLength(1)-1, Word1, Word2, VF\_Matrix);

Console.Write("\nProgram finished. Press any key to terminate . . .");

Console.ReadKey();

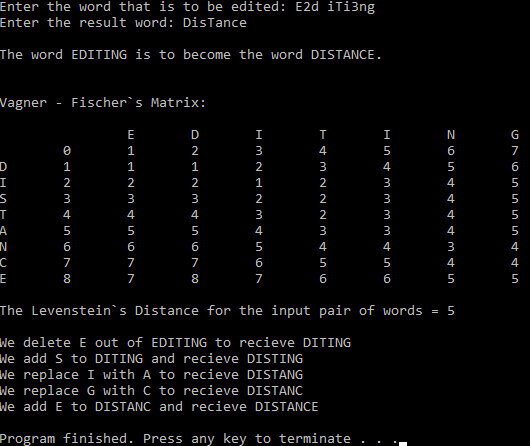
return;

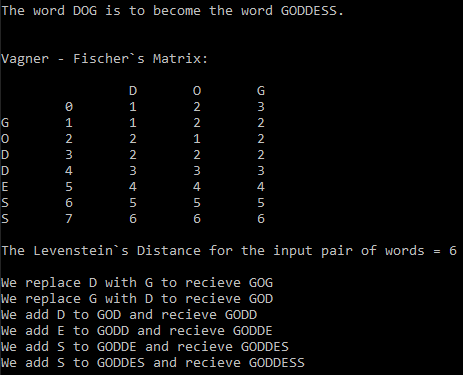
}

}

}

Анализ результатов

Ввод с пробелами и числами, дистанция для “EDITING - DISTANCE”:

“DOG - GODDESS”:

Ввод пустой строки: