**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:
студент группы ИУ5-31Б
Ларкин Б. В.
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю. Е.
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”:

```
Enter the word that is to be edited: E2d iTi3ng
Enter the result word: DisTance

The word EDITING is to become the word DISTANCE.


Vagner - Fischer`s Matrix:

                E       D       I       T       I       N       G
        0       1       2       3       4       5       6       7
D       1       1       1       2       3       4       5       6
I       2       2       2       1       2       3       4       5
S       3       3       3       2       2       3       4       5
T       4       4       4       3       2       3       4       5
A       5       5       5       4       3       3       4       5
N       6       6       6       5       4       4       3       4
C       7       7       7       6       5       5       4       4
E       8       7       8       7       6       6       5       5

The Levenstein`s Distance for the input pair of words = 5

We delete E out of EDITING to recieve DITING
We add S to DITING and recieve DISTING
We replace I with A to recieve DISTANG
We replace G with C to recieve DISTANC
We add E to DISTANC and recieve DISTANCE

Program finished. Press any key to terminate . . .
```

“DOG - GODDESS”:

```
The word DOG is to become the word GODDESS.


Vagner - Fischer`s Matrix:

                D       O       G
        0       1       2       3
G       1       1       2       2
O       2       2       1       2
D       3       2       2       2
D       4       3       3       3
E       5       4       4       4
S       6       5       5       5
S       7       6       6       6

The Levenstein`s Distance for the input pair of words = 6

We replace D with G to recieve GOG
We replace G with D to recieve GOD
We add D to GOD and recieve GODD
We add E to GODD and recieve GODDE
We add S to GODDE and recieve GODDES
We add S to GODDES and recieve GODDESS
```

Ввод пустой строки:

```
Enter the word that is to be edited:
Enter a non-empty string
```