



**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Н. Э.  
БАУМАНА**

*Факультет «Информатика и системы управления»*

*Кафедра «Системы обработки информации и управления  
(ИУ5)»*

*Дисциплина: «Базовые компоненты Интернет-технологий»*

**Отчет по лабораторной работе № 5:**

**«Вычисление расстояния Дамерау-Левенштейна с  
использованием алгоритма Вагнера-Фишера»**

*Выполнила: Журавлева Полина Валерьевна*

*Группа: ИУ5-31Б*

*Преподаватель: Гапанюк Юрий Евгеньевич*

---

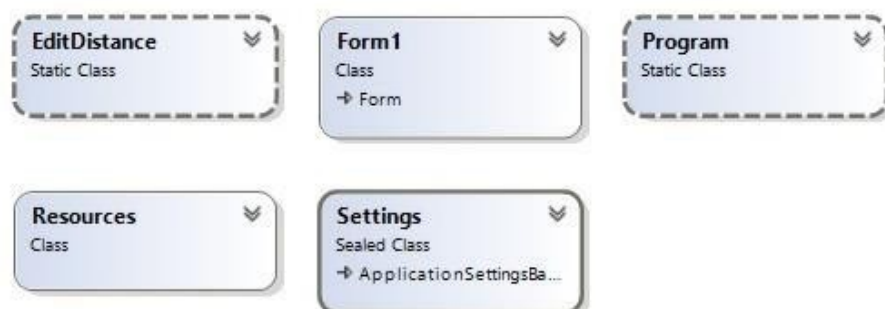
Дата:  
23.12.2018  
Подпись:

### **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 Дamerau-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:**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Diagnostics;

namespace Лаб5
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        /// <summary>
        /  Список слов
        /  </summary>
        List<string> list = new List<string>();

        private void label1_Click(object sender, EventArgs e)
        {

        }

        private void label3_Click(object sender, EventArgs e)
        {

        }

        private void label5_Click(object sender, EventArgs e)
        {
```

```

    }

    private void buttonClose_Click_1(object sender,
    EventArgs e)
    {
        this.Close();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        OpenFileDialog fd = new
        OpenFileDialog(); fd.Filter =
        "текстовые файлы|*.txt";
        if (fd.ShowDialog() == DialogResult.OK)
        {
            Stopwatch t = new Stopwatch();
            t.Start();
            //Чтение файла в виде строки
            string text = File.ReadAllText(fd.FileName);
            //Разделительные символы для чтения из
            файла char[] separators = new char[] { ' ', '.',
            ',', '!', '?', '/', '\t', '\n' }; string[] textArray =
            text.Split(separators); foreach (string strTemp
            in textArray)
            {
                //Удаление пробелов в начале и
                конце строки string str =
                strTemp.Trim();
                //Добавление строки в список, если строка не содержится в
                списке
                if (!list.Contains(str)) list.Add(str);
            }
            t.Stop();
            this.textBoxFileReadTime.Text =
            t.Elapsed.ToString();
            this.textBoxFileReadCount.Text =
            list.Count.ToString();
        }
        else
        {
            MessageBox.Show("Необходимо выбрать файл");
        }
    }
}

```

```

private void button2_Click(object sender, EventArgs e)
{
    //Слово для поиска
    string word = this.textBoxFind.Text.Trim();

    //Если слово для поиска не пусто
    if (!string.IsNullOrEmpty(word) && list.Count > 0)
    {
        //Слово для поиска в верхнем
        регистре string wordUpper =
        word.ToUpper(); //Временные
        результаты поиска List<string>
        tempList = new List<string>();
        Stopwatch t = new Stopwatch();
        t.Start();

        foreach (string str in list)
        {
            if (str.ToUpper().Contains(wordUpper))
            {
                tempList.Add(str);
            }
        }
        t.Stop();
        this.textBoxExactTime.Text =
        t.Elapsed.ToString();
        this.listBoxResult.BeginUpdate(); //Очистка
        списка
        this.listBoxResult.Items.Clear();
        //Вывод результатов поиска
        foreach (string str in tempList)
        {
            this.listBoxResult.Items.Add(str);
        }
        this.listBoxResult.EndUpdate();
    }
    else
    {
        MessageBox.Show("Необходимо выбрать файл и
        ввести слово для поиска");
    }
}

```

```

private void button3_Click(object sender, EventArgs e)
{
    //Слово для поиска
    string word =
    this.textBoxFind.Text.Trim(); //Если
    слово для поиска не пусто
    if (!string.IsNullOrEmpty(word) && list.Count > 0)
    {
        int maxDist;
        if (!int.TryParse(this.textBoxMaxDist.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>>();
        Stopwatch t = new Stopwatch();
        t.Start();
        foreach (string str in list)
        {
            //Вычисление расстояния Дамерау-
            Левенштейна
            int dist = EditDistance.Distance(str.ToUpper(),
            wordUpper); //Если расстояние меньше
            порогового, то слово
            добавляется в результат
            if (dist <= maxDist)
            {
                tempList.Add(new Tuple<string, int>(str, dist));
            }
        }
    }
}

```

```

t.Stop();
this.textBoxApproxTime.Text =
t.Elapsed.ToString();
this.listBoxResult.BeginUpdate(); /
/Очистка списка

this.listBoxResult.Items.Clear();
//Вывод результатов поиска
foreach (var x in tempList)
{
    string temp = x.Item1 + "(расстояние=" +
    x.Item2.ToString() +
    ")";
    this.listBoxResult.Items.Add(temp);
}
this.listBoxResult.EndUpdate();
}
else
{
    MessageBox.Show("Необходимо выбрать файл
и ввести слово для поиска");
}

}

```

```

private void button4_Click(object sender, EventArgs e)
{
    //Имя файла отчета
    string TempReportFileName = "Report_"
    +
    DateTime.Now.ToString("dd_MM_yyyy_hh
    mmss"); //Диалог сохранения файла
    отчета SaveFileDialog fd = new
    SaveFileDialog(); fd.FileName =
    TempReportFileName; fd.DefaultExt =
    ".html";
    fd.Filter = "HTML Reports|*.html";
    if (fd.ShowDialog() == DialogResult.OK)
    {
        string ReportFileName = fd.FileName;
        //Формирование отчета
        StringBuilder b = new StringBuilder();
        b.AppendLine("<html>");
    }
}

```

```

b.AppendLine("<head>");

b.AppendLine("<meta http-equiv='Content-Type'
content='text/html; charset = UTF - 8'/>");
b.AppendLine("<title>" + "Отчет: " + ReportFileName
+ "</title>");
b.AppendLine("</head>");
b.AppendLine("<body>");
b.AppendLine("<h1>" + "Отчет: " +
ReportFileName + "</h1>"); b.AppendLine("<table
border='1'>"); b.AppendLine("<tr>");
b.AppendLine("<td>Время чтения из
файла</td>");
b.AppendLine("<td>" + this.textBoxFileReadTime.Text
+ "</td>");
b.AppendLine("</tr>");
b.AppendLine("<tr>");
b.AppendLine("<td>Количество
уникальных слов в файле</td>");
b.AppendLine("<td>" +
this.textBoxFileReadCount.Text +
"</td>");
b.AppendLine("</tr>");
b.AppendLine("<tr>");
b.AppendLine("<td>Слово для поиска</td>");
b.AppendLine("<td>" + this.textBoxFind.Text +
"</td>");
b.AppendLine("</tr>");
b.AppendLine("<tr>");
b.AppendLine("<td>Максимальное расстояние
для нечеткого поиска </ td > ");
b.AppendLine("<td>" + this.textBoxMaxDist.Text +
"</td>");
b.AppendLine("</tr>");
b.AppendLine("<tr>");
b.AppendLine("<td>Время четкого поиска</td>");
b.AppendLine("<td>" + this.textBoxExactTime.Text
+ "</td>");
b.AppendLine("</tr>");
b.AppendLine("<tr>");
b.AppendLine("<td>Время нечеткого
поиска</td>");
b.AppendLine("<td>" + this.textBoxApproxTime.Text
+ "</td>");

```



```

        b.AppendLine("</tr>");
        b.AppendLine("<tr valign='top'>");
        b.AppendLine("<td>Результаты поиска</td>");
        b.AppendLine("<td>");
        b.AppendLine("<ul>");
        foreach (var x in this.listBoxResult.Items)
        {
            b.AppendLine("<li>" + x.ToString() + "</li>");
        }
        b.AppendLine("</ul>");
        b.AppendLine("</td>");
        b.AppendLine("</tr>");
        b.AppendLine("</table>");
        b.AppendLine("</body>");
        b.AppendLine("</html>");
        //Сохранение файла
        File.AppendAllText(ReportFileName, b.ToString());
        MessageBox.Show("Отчет сформирован. Файл: " +
ReportFileName);
    }

}

}

public static class EditDistance
{
    /  <summary>
    /  Вычисление расстояния Дамерау-Левенштейна
    /  </summary>
    public static int Distance(string str1Param, string
str2Param)
    {
        if ((str1Param == null) || (str2Param ==
null)) return -1; int str1Len =
str1Param.Length; int str2Len =
str2Param.Length;
        //Если хотя бы одна строка пустая, возвращается
        строки
        if ((str1Len == 0) && (str2Len ==
0)) return 0; if (str1Len == 0)
return str2Len; if (str2Len == 0)
return str1Len;

```

```

//Приведение строк к верхнему
регистру string str1 =
str1Param.ToUpper(); string str2 =
str2Param.ToUpper(); /
/Объявление матрицы
int[,] matrix = new int[str1Len + 1, str2Len + 1];
//Инициализация нулевой строки и нулевого
столбца матрицы for (int i = 0; i <= str1Len; i++)
matrix[i, 0] = i; for (int j = 0; j <= str2Len; j++)
matrix[0, j] = j;
//Вычисление расстояния Дамерау-Левенштейна
for (int i = 1; i <= str1Len; i++)
{
    for (int j = 1; j <= str2Len; j++)
    {
        //Эквивалентность символов, переменная
symbEqual соответствует m(s1[i], s2[j])
        int symbEqual = ((str1.Substring(i - 1, 1) ==
str2.Substring(j - 1, 1)) ? 0 : 1);
        int ins = matrix[i, j - 1] +
1; //Добавление int del =
matrix[i - 1, j] + 1; //Удаление
int subst = matrix[i - 1, j - 1] +
symbEqual; //Замена
//Элемент матрицы
вычисляется как минимальный из трех случаев
matrix[i, j] = Math.Min(Math.Min(ins, del), subst);
//Дополнение Дамерау по перестановке
соседних
СИМВОЛОВ
if ((i > 1) && (j > 1) &&
(str1.Substring(i - 1, 1) == str2.Substring(j - 2, 1))
&&
(str1.Substring(i - 2, 1) == str2.Substring(j - 1,
1)))
{
    matrix[i, j] = Math.Min(matrix[i, j], matrix[i
- 2, j - 2] + symbEqual);
}
}
}
//Возвращается нижний правый элемент
матрицы return matrix[str1Len, str2Len];
}
}

```

}

