

Защищено:
Гапанюк Ю.Е.

Демонстрация:
Гапанюк Ю.Е.

"__" _____ 2017 г.

"__" _____ 2017 г.

Отчет по лабораторной работе № 4 и №5 по курсу Базовые компоненты интернет-технологий

12
(количество листов)

Студент группы ИУ5-33
Коционова Анна

Москва МГТУ 2017

СОДЕРЖАНИЕ

1. Описание задания лабораторной работы
2. Текст программы
3. Диаграмма классов
4. Результат работы программы

1. Описание задания лабораторной работы

Лабораторная работа №4

Разработать программу, реализующую работу с файлами.

1. Программа должна быть разработана в виде приложения Windows Forms на языке C#. По желанию вместо Windows Forms возможно использование WPF.
2. Добавить кнопку, реализующую функцию чтения файла в список слов `List<string>`.
3. Для выбора имени файла используется класс `OpenFileDialog`, который открывает диалоговое окно с выбором файла. Ограничить выбор только файлами с расширением «.txt».
4. Для чтения из файла рекомендуется использовать статический метод `ReadAllText()` класса `File` (пространство имен `System.IO`). Содержимое файла считывается методом `ReadAllText()` в виде одной строки, далее делится на слова с использованием метода `Split()` класса `string`. Слова сохраняются в список `List<string>`.
5. При сохранении слов в список `List<string>` дубликаты слов не записываются. Для проверки наличия слова в списке используется метод `Contains()`.
6. Вычислить время загрузки и сохранения в список с использованием класса `Stopwatch` (пространство имен `System.Diagnostics`). Вычисленное время вывести на форму в поле ввода (`TextBox`) или надпись (`Label`).
7. Добавить на форму поле ввода для поиска слова и кнопку поиска. При нажатии на кнопку поиска осуществлять поиск введенного слова в списке. Слово считается найденным, если оно входит в элемент списка как подстрока (метод `Contains()` класса `string`).
8. Добавить на форму список (`ListBox`). Найденные слова выводить в список с использованием метода «название_списка.Items.Add()». Вызовы метода «название_списка.Items.Add()» должны находиться между вызовами методов «название_списка.BeginUpdate()» и «название_списка.EndUpdate()».
9. Вычислить время поиска с использованием класса `Stopwatch`. Вычисленное время вывести на форму в поле ввода (`TextBox`) или надпись (`Label`).

Лабораторная работа №5

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 Дameraу-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. Текст программы

Form

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

namespace Lab4_3sem
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            List<string> list = new List<string>(); // список слов

            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();

                    string text = File.ReadAllText(fd.FileName); // метод чтения файла в виде
строки

                    char[] separators = new char[] { ' ', '.', ',', '!', '?', '/', '\t', '\n' }; // Разделительные символы для чтения из файла
                    string[] textArray = text.Split(separators); // Делим на слова
                    foreach (string strTemp in textArray)
                    {

                        string str = strTemp.Trim(); // Удаление пробелов в начале и конце
строки

                        if (!list.Contains(str)) // проверка
                        {
                            list.Add(str); // Добавление строки в список, если строка не
содержится в списке
                        }
                        t.Stop();
                        this.textBoxFileReadTime.Text = t.Elapsed.ToString();
                        this.textBoxFileReadCount.Text = list.Count.ToString();
                    }
                }
                else
                {
                    MessageBox.Show("Необходимо выбрать файл");
                }
            }
        }
    }
}
```

```

private void buttonExit_Click_1(object sender, EventArgs e)
{
    this.Close();
}

private void buttonExact_Click_1(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 buttonApprox_Click_1(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 buttonSaveReport_Click_1(object sender, EventArgs e)
{
    //Имя файла отчета
    string TempReportFileName = "Report_" +
    DateTime.Now.ToString("dd_MM_yyyy_hhmmss");
    //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);
    }
}
}

```

Класс EditDistance

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

```



```

namespace Lab4_3sem
{
    /*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 длина другой 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
                    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];
        }

        /// <summary>
        /// Вывод расстояния Дамерау-Левенштейна в консоль
        /// </summary>
        public static void WriteDistance(string str1Param, string str2Param)
    }
}

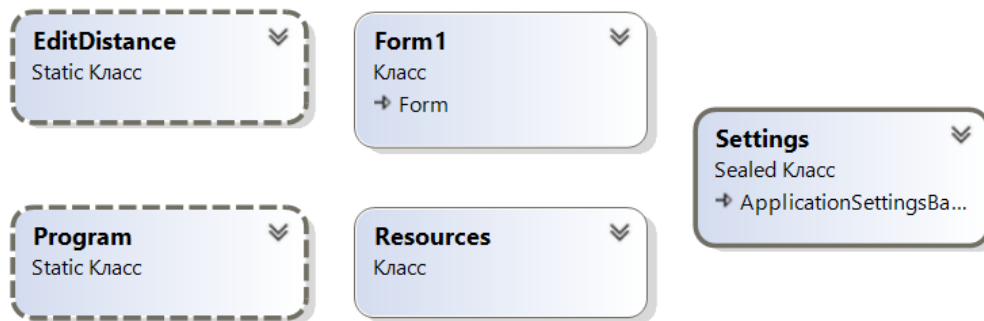
```

```

    {
        int d = Distance(str1Param, str2Param);
        Console.WriteLine("'" + str1Param + "', '" + str2Param + "' -> " +
d.ToString());
    }
}
}

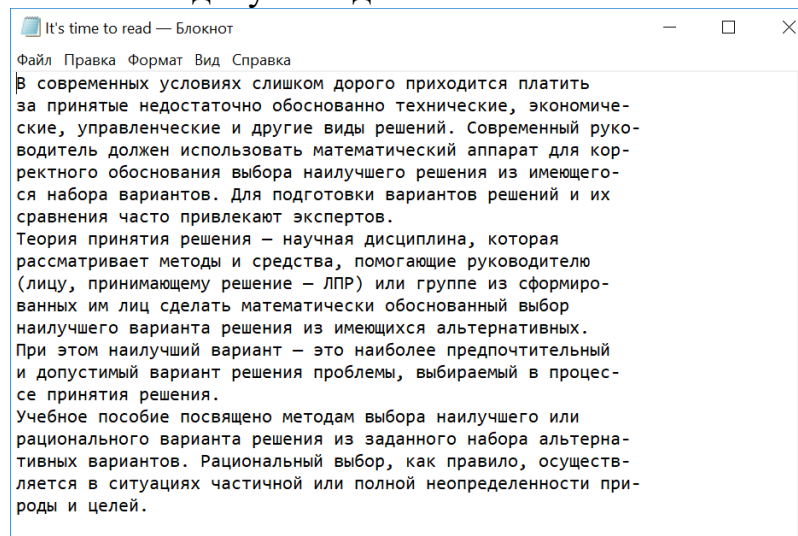
```

3. Диаграмма классов



4. Результат выполнения программы

Текстовый документ для чтения



Текст1

Чтение

Счетчик времени: 00:00:00.0003686

Число уникальных слов: 107

Слово для поиска: Решение

Четкий поиск

Время четкого поиска: 00:00:00.0000716

Параллельный нечеткий поиск

Максимальное расстояние для нечеткого поиска:

Время нечеткого поиска:

решение

Сохранить отчет

Выход

Текст1

Чтение

Счетчик времени: 00:00:00.0003686

Число уникальных слов: 107

Слово для поиска: Решение

Четкий поиск

Время четкого поиска: 00:00:00.0000716

Параллельный нечеткий поиск

Максимальное расстояние для нечеткого поиска: 3

Время нечеткого поиска: 00:00:00.0011336

решений(расстояние=1)
решения(расстояние=1)
решение(расстояние=0)

Сохранить отчет

Выход