**Московский государственный технический университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Базовые компоненты интернет-технологий»

Отчет по домашнему заданию

Выполнил:

студент группы ИУ5-31

Стрихар Павел

Подпись и дата:

Проверил:

преподаватель каф. ИУ5

Гапанюк Ю. Е.

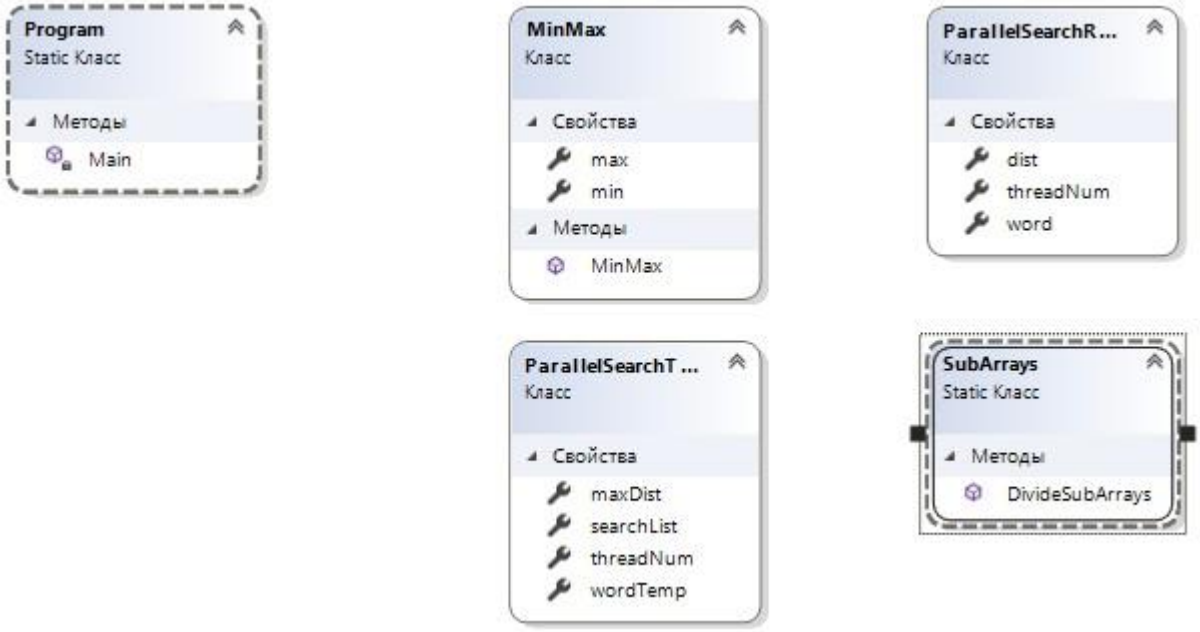
Подпись и дата:

Москва, 2020 г.

## Описание задания:

1) Программа должна быть разработана в виде приложения Windows Forms на языке C#. По желанию вместо Windows Forms возможно использование WPF.
2) В качестве основы используется макет, разработанный в лабораторных работах №4 и №5.
3) Реализуйте функцию поиска с использованием расстояния Левенштейна в многопоточном варианте. Количество потоков для запуска функции поиска вводится на форме в поле ввода (TextBox). В качестве примера используйте проект «Parallel» из примера «Введение в C#».
4) Реализуйте функцию записи результатов поиска в файл отчета. Файл отчета создается в формате .txt или .html. В качестве примера используйте проект «WindowsFormsFiles» (обработчик события кнопки «Сохранение отчета») из примера «Введение в C#».

# Диаграмма классов:



# Текст программы:

1. Forms1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Diagnostics;
using Lab5;

namespace Lab4
{
    public partial class Form1 : Form
    {
        List<string> word_list = new List<string>();
        public Form1()
        {
            InitializeComponent();

        }

        private void button1_Click(object sender, EventArgs e)
        {
            word_list.Clear();
            Stopwatch extime = new Stopwatch();
            string fileContent;
            string filePath;
            string[] strs;
            OpenFileDialog openFile = new OpenFileDialog();
            openFile.Filter = "Текстовый файл|*.txt";
            if (openFile.ShowDialog() == DialogResult.OK)
            {
                extime.Start();
                filePath = openFile.FileName;
                textBox2.Text = filePath.Split('\\').Last();
                fileContent = File.ReadAllText(filePath);
                char[] separators = new char[] { ' ', '.', ',', '!', '?', '/', '\t', '\n', '\f' };
                strs = fileContent.Split(separators);
                for (int i = 0; i < strs.Length; i++)
                {
                    strs[i] = strs[i].Trim();
                    if (!word_list.Contains(strs[i]) && (strs[i] != " ") && (strs[i].Length > 0))
                        word_list.Add(strs[i]);
                }
            }
            extime.Stop();
            textBox1.Text = extime.Elapsed.ToString();
        }

        private void label3_Click(object sender, EventArgs e)
        {

        }
        public static List<ParallelSearchResult> ArrayThreadTask(object t_param)
        {
            ParallelSearchThreadParam param = t_param as ParallelSearchThreadParam;
            string word_temp = param.wordTemp.Trim();
            List<ParallelSearchResult> result_list = new List<ParallelSearchResult>();
            foreach (string str in param.searchList)
            {
                int dist = Fisher.GetLen(str.ToUpper(), word_temp.ToUpper());
                if (dist <= param.maxDist)
                {
                    ParallelSearchResult temp_list = new ParallelSearchResult()
                    {
                        word = str,
                        dist = dist,
                        threadNum = param.threadNum
                    };
                    result_list.Add(temp_list);
                }
            }
            return result_list;
```

```
        }
        private void button2_Click(object sender, EventArgs e)
        {
            Stopwatch extime = new Stopwatch();
            string new_str = "\0";
            Random rnd = new Random();
            string word_temp = textBox3.Text.Trim();
            List<string> tempList = new List<string>();
            if (string.IsNullOrWhiteSpace(word_temp))
            {
                MessageBox.Show("Пожалуйста, введите слово для поиска!");
            }
            if (radioButton1.Checked == true)
            {
                extime.Start();
                for (int i = 0; i < word_list.Count; i++)
                {
                    if (word_list[i].ToUpper().Contains(word_temp.ToUpper()) &&
!listBox1.Items.Contains(word_list[i]))
                    {
                        new_str = word_list[i];
                        tempList.Add(new_str);
                    }
                }
            }
            else if (radioButton2.Checked == true)
            {
                if (!int.TryParse(textBox6.Text, out int thread_count))
                {
                    MessageBox.Show("Пожалуйста, введите количество потоков!");
                    return;
                }
                if (!int.TryParse(textBox5.Text, out int max_dist))
                {
                    MessageBox.Show("Пожалуйста, введите максимальное редакционное расстояние!");
                    return;
                }
                extime.Start();
                List<ParallelSearchResult> result_list = new List<ParallelSearchResult>();
                List<MinMax> div_list = SubArrays.DivideSubArrays(0, word_list.Count, thread_count);
                int countSub = div_list.Count;
                Task<List<ParallelSearchResult>>[] tasks = new Task<List<ParallelSearchResult>>[countSub];
                for(int i = 0; i < countSub; i++)
                {
                    List<string> tempTaskList = word_list.GetRange(div_list[i].min, div_list[i].max -
div_list[i].min);
                    tasks[i] = new Task<List<ParallelSearchResult>>(
                        ArrayThreadTask,
                        new ParallelSearchThreadParam()
                        {
                            searchList = tempTaskList,
                            maxDist = max_dist,
                            threadNum = i,
                            wordTemp = word_temp
                        });
                    tasks[i].Start();
                }
                Task.WaitAll();
                extime.Stop();
                for (int i = 0; i < countSub; i++)
                {
                    result_list.AddRange(tasks[i].Result);
                }
                label8.Text = "Вычисленное количество потоков: " + countSub.ToString();
                label8.Visible = true;
                listBox1.BeginUpdate();
                listBox1.Items.Clear();
                for (int i = 0; i < result_list.Count; i++)
                {
                    listBox1.Items.Add(result_list[i].word + " [расстояние = " +
                        result_list[i].dist.ToString() + " | поток = " +
result_list[i].threadNum.ToString() + "]");
                }
                listBox1.EndUpdate();
            }
            else
            {
                MessageBox.Show("Пожалуйста, выберите метод поиска слова!");
            }
            if (tempList.Count > 0)
            {
                listBox1.BeginUpdate();
```

```
        foreach(string str in tempList)
        {
            listBox1.Items.Add(str);
        }
        listBox1.EndUpdate();
    }
    extime.Stop();
    textBox4.Text = extime.Elapsed.ToString();
}

private void radioButton2_CheckedChanged(object sender, EventArgs e)
{
    listBox1.BeginUpdate();
    listBox1.Items.Clear();
    listBox1.EndUpdate();
    if (radioButton2.Checked == true)
    {
        label6.Visible = true;
        textBox5.Visible = true;
        label7.Visible = true;
        textBox6.Visible = true;
    }
}

private void radioButton1_CheckedChanged(object sender, EventArgs e)
{
    listBox1.BeginUpdate();
    listBox1.Items.Clear();
    listBox1.EndUpdate();
    if (radioButton1.Checked == true)
    {
        label6.Visible = false;
        textBox5.Visible = false;
        label7.Visible = false;
        textBox6.Visible = false;
        label8.Visible = false;
    }
}

private void buttonCreateReport_Click(object sender, EventArgs e)
{
    string reportName = "Report_" + DateTime.Now.ToString("dd_MM_yyyy_hhmmss");
    SaveFileDialog fd = new SaveFileDialog();
    fd.FileName = reportName;
    fd.DefaultExt = ".html";
    fd.Filter = "HTML Reports|*.html";
    if (fd.ShowDialog() == DialogResult.OK)
    {
        StringBuilder rep = new StringBuilder();
        rep.AppendLine("<html>");
        rep.AppendLine("<head>");
        rep.AppendLine("<meta http-equiv='Content-Type' content='text/html;charset=UTF-8'/>");
        rep.AppendLine("<title> Отчет: " + fd.FileName + "</title>");
        rep.AppendLine("</head>");
        rep.AppendLine("<body>");
        rep.AppendLine("<h1>" + "Отчет: " + fd.FileName + "</h1>");
        rep.AppendLine("<table border='1'>");
        rep.AppendLine("<tr>");
        rep.AppendLine("<td>Время чтения из файла</td>");
        rep.Append("<td>" + textBox1.Text + "</td>\n" +
            "</tr>\n" +
            "<tr>\n" +
            "<td>Количество уникальных слов в файле</td>\n" +
            "<td>" + word_list.Count.ToString() + "</td>\n" +
            "</tr>\n" +
            "<tr>\n" +
            "<td>Слово для поиска</td>\n" +
            "<td>" + textBox3.Text + "</td>\n" +
            "</tr>\n");
        if (radioButton2.Checked == true)
        {
            rep.Append(
              "<tr>\n" +
              "<td>Максимальное редакционное расстояние</td>\n" +
              "<td>" + textBox5.Text + "</td>\n" +
              "</tr>\n" +
              "<tr>\n" +
              "<td>Время поиска методом редакционного расстояния</td>\n");
        }
        else if (radioButton1.Checked == true)
        {
            rep.Append(
```

```
                "<tr>\n" +
                "<td>Время поиска методом поиска подстроки</td>\n");
            }
            rep.Append(
                  "<td>" + textBox4.Text + "</td>\n" +
                  "</tr>\n" +
                  "<tr valign='top'>\n" +
                  "<td>Результаты поиска</td>\n" +
                  "<td>\n" +
                  "<ul>\n");
            foreach (var x in listBox1.Items)
            {
                rep.AppendLine("<li>" + x.ToString() + "</li>");
            }
            rep.AppendLine("</ul>");
            rep.AppendLine("</td>");
            rep.AppendLine("</tr>");
            rep.AppendLine("</table>");
            rep.AppendLine("</html>");
            rep.AppendLine("</body>");
            File.AppendAllText(fd.FileName, rep.ToString());
            MessageBox.Show("Отчет сформирован. Файл: " + fd.FileName);
        }
    }
}
```

2. ExtraClasses.cs

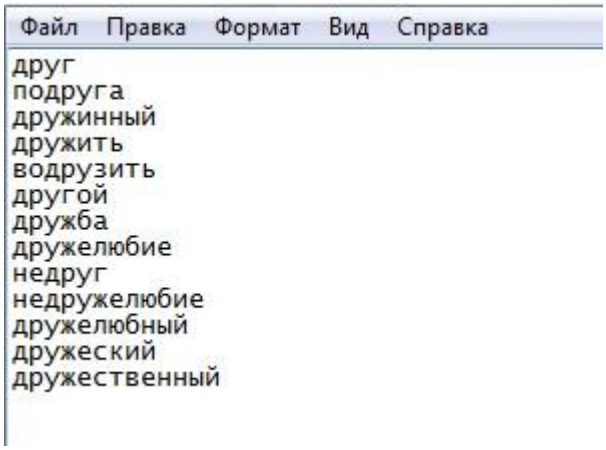
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab4
{
    public class ParallelSearchResult
    {
        public string word { get; set; }
        public int dist { get; set; }
        public int threadNum { get; set; }
    }
    public class ParallelSearchThreadParam
    {
        public List<string> searchList { get; set; }
        public string wordTemp { get; set; }
        public int maxDist { get; set; }
        public int threadNum { get; set; }
    }
    public class MinMax
    {
        public int min { get; set; }
        public int max { get; set; }

        public MinMax(int t_min, int t_max)
        {
            this.min = t_min;
            this.max = t_max;
        }
    }
    public static class SubArrays
    {
        /// <summary>
        /// Деление массива на последовательности
        /// </summary>
        /// <param name="beginIndex">Начальный индекс массива</param>
        /// <param name="endIndex">Конечный индекс массива</param>
        /// <param name="subArraysCount">Требуемое количество подмассивов</param>
        /// <returns>Список пар с индексами подмассивов</returns>
        public static List<MinMax> DivideSubArrays(int beginIndex, int endIndex, int subArraysCount)
        {
            //Результирующий список пар с индексами подмассивов
            List<MinMax> result = new List<MinMax>();

            //Если число элементов в массиве слишком мало для деления
            //то возвращается массив целиком
            if ((endIndex - beginIndex) <= subArraysCount)
            {
                result.Add(new MinMax(0, (endIndex - beginIndex)));
            }
            else
            {
                //Размер подмассива
                int delta = (endIndex - beginIndex) / subArraysCount;
                //Начало отсчета
                int currentBegin = beginIndex;
                //Пока размер подмассива укладывается в оставшуюся последовательность
                while ((endIndex - currentBegin) >= 2 * delta)
                {
                    //Формируем подмассив на основе начала последовательности
                    result.Add(new MinMax(currentBegin, currentBegin + delta));
                    //Сдвигаем начало последовательности вперед на размер подмассива
                    currentBegin += delta;
                }
```

```
                //Оставшийся фрагмент массива
                result.Add(new MinMax(currentBegin, endIndex));
            }
            //Возврат списка результатов
            return result;
        }
    }
}
```

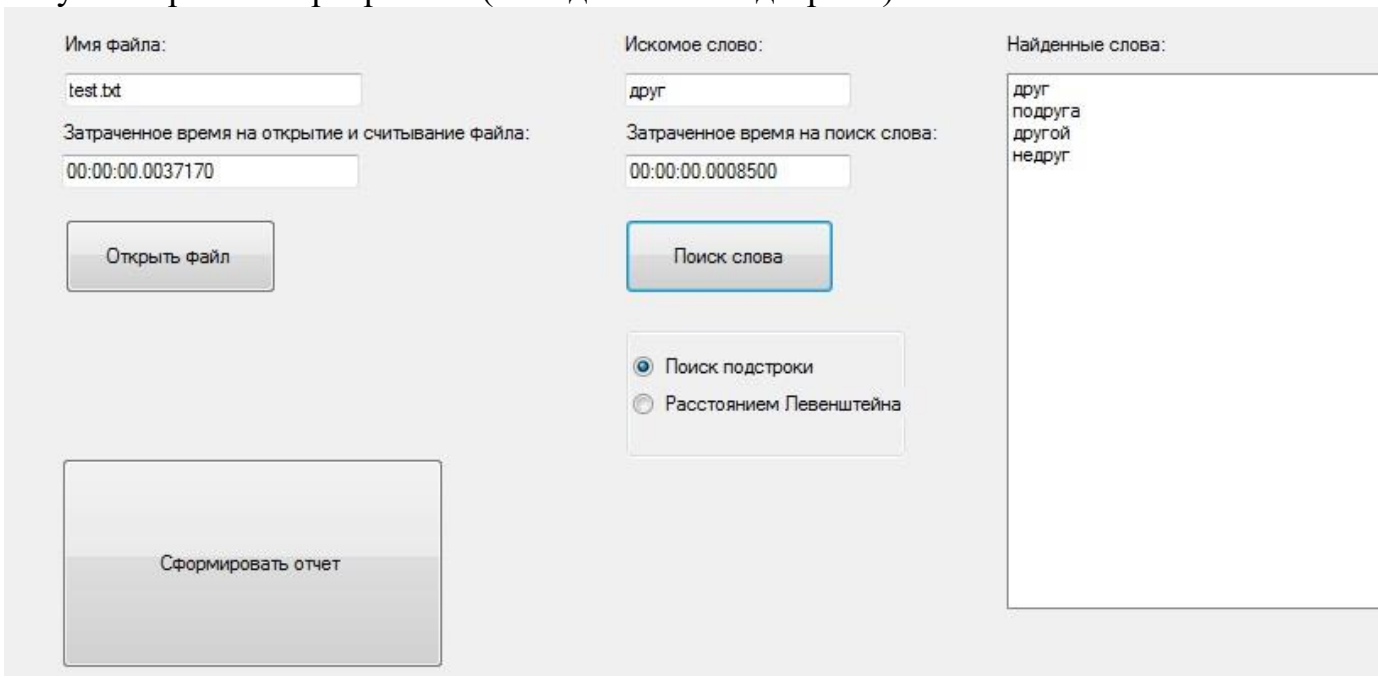
Использована библиотека Lab5, разработанная в лабораторной работе №5.

## Примеры выполнения программы:

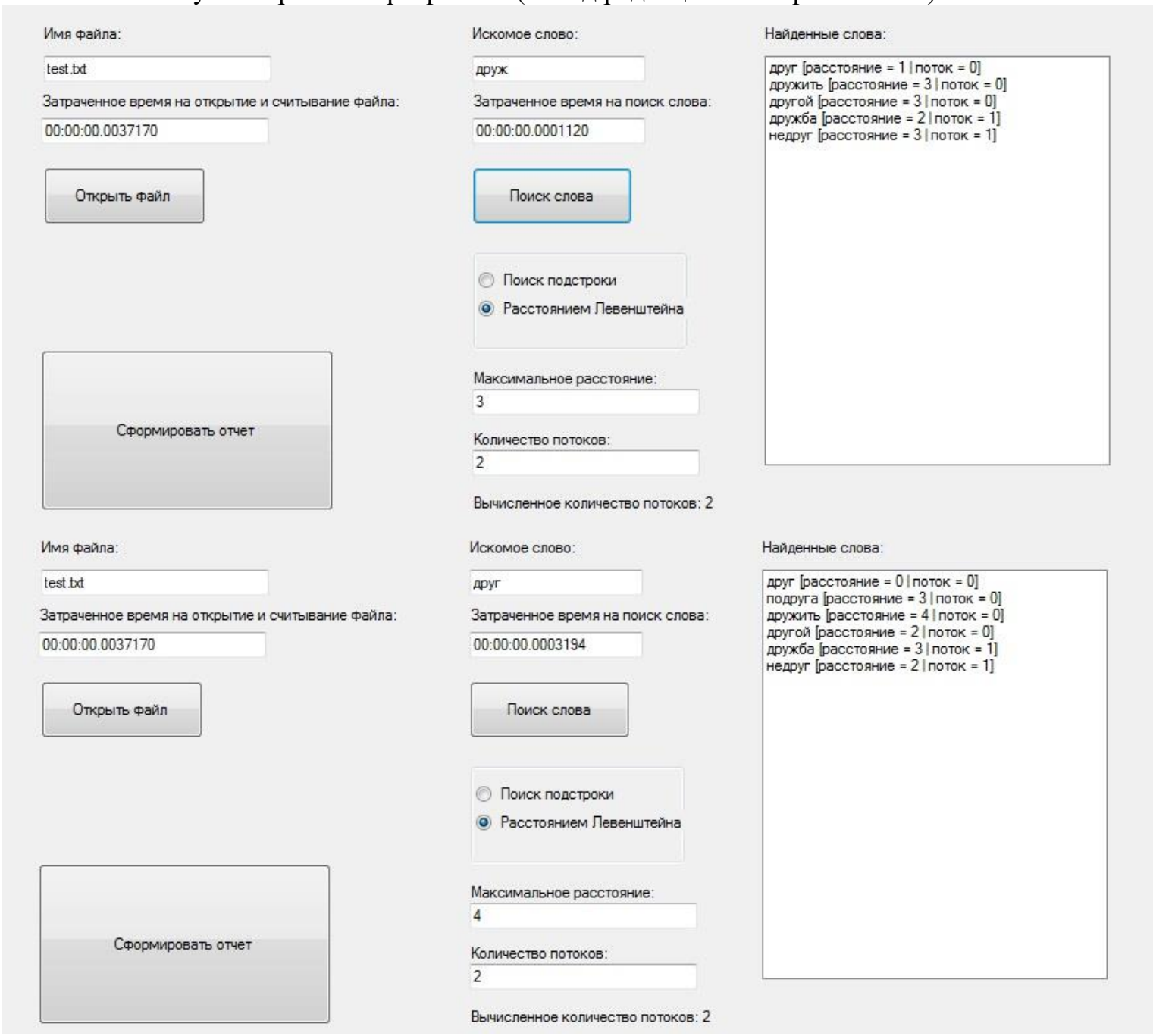
Исходный текст:




Результат работы программы (метод поиска подстроки):



Результат работы программы (метод редакционного расстояния):



Сформированные отчеты:

# Отчет: C:\Users\user\Desktop\Report_28_12_2020_044525.html

| | |
|---|---|
| Время чтения из файла | 00:00:00.0037170 |
| Количество уникальных слов в файле | 13 |
| Слово для поиска | друг |
| Максимальное редакционное расстояние | 4 |
| Время поиска методом редакционного расстояния | 00:00:00.0003194 |
| Результаты поиска | • друг [расстояние = 0 \| поток = 0]<br>• подруга [расстояние = 3 \| поток = 0]<br>• дружить [расстояние = 4 \| поток = 0]<br>• другой [расстояние = 2 \| поток = 0]<br>• дружба [расстояние = 3 \| поток = 1]<br>• недруг [расстояние = 2 \| поток = 1] |