**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Московский государственный технический университет имени Н.Э. Баумана**
**(национальный исследовательский университет)»**
**(МГТУ им. Н.Э. Баумана)**

# Базовые компоненты интернет технологий

## Отчет по домашнему заданию

Студент: Булыгина С. А.
Группа: ИУ5Ц-51Б

Преподаватель: Гапанюк Ю. Е.

2019 г.

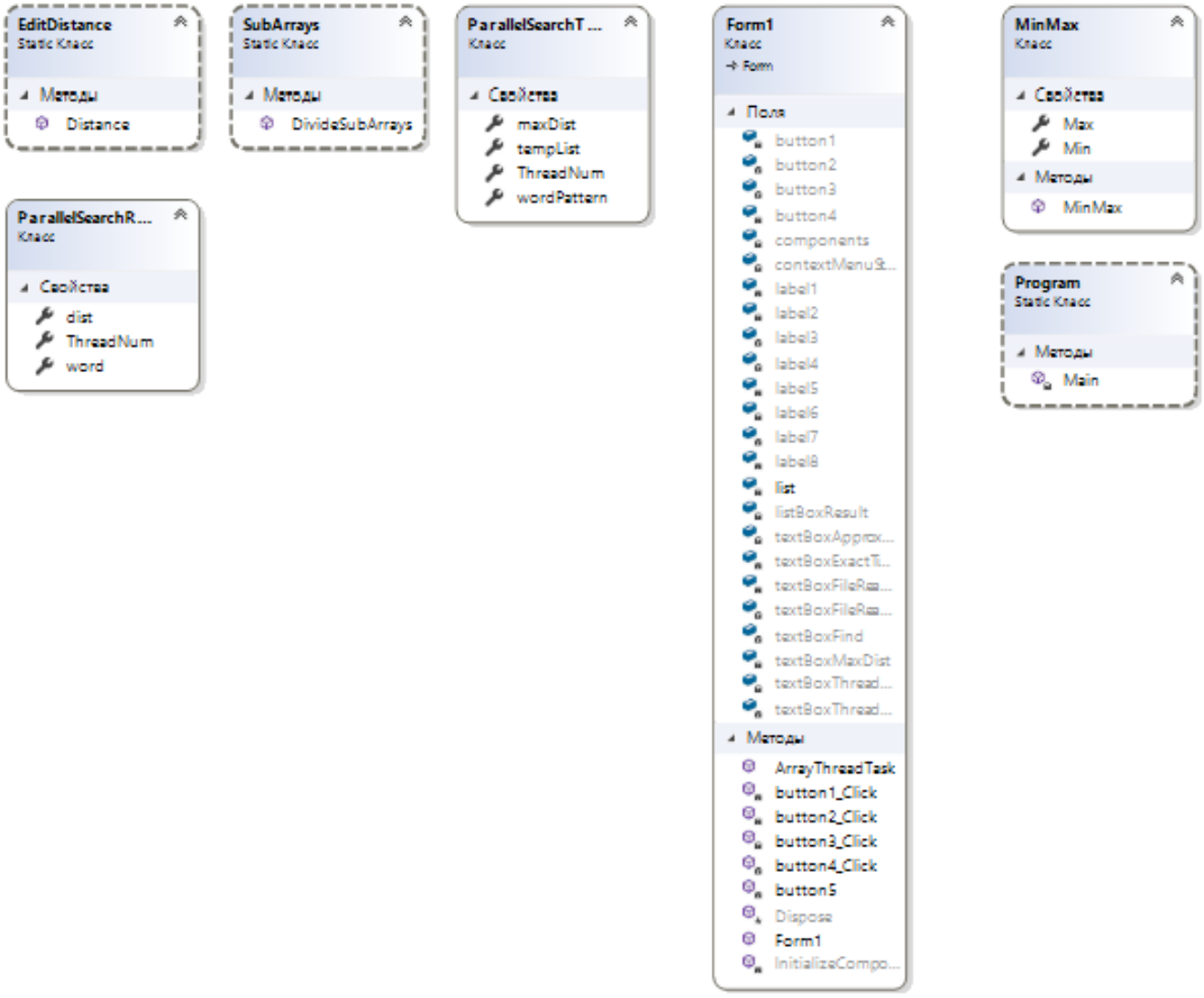
## Домашнее задание

Разработать программу, реализующую многопоточный поиск в файле.

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).
4. Реализуйте функцию записи результатов поиска в файл отчета. Файл отчета создается в формате .txt или .html

Пример реализации ДЗ рассмотрен в учебном пособии, глава «Пример многопоточного поиска в текстовом файле с использованием технологии Windows Forms».

# Диаграмма классов



# Текст программы

## Program.cs

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

namespace DZ_Bulygina
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Console.Title = "Булыгина Светлана, ИУ5Ц-51Б";
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

## Form1.Designer.cs

```
namespace DZ_Bulygina
{
    partial class Form1
    {
        /// Обязательная переменная конструктора.
        private System.ComponentModel.IContainer components = null;
        /// Освободить все используемые ресурсы.
        /// <param name="disposing">истинно, если управляемый ресурс должен быть удален;
иначе ложно.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Код, автоматически созданный конструктором форм Windows

        /// Требуемый метод для поддержки конструктора — не изменяйте
        /// содержимое этого метода с помощью редактора кода.
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.label1 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.label3 = new System.Windows.Forms.Label();
            this.label4 = new System.Windows.Forms.Label();
            this.label5 = new System.Windows.Forms.Label();
            this.label6 = new System.Windows.Forms.Label();
            this.label7 = new System.Windows.Forms.Label();
            this.label8 = new System.Windows.Forms.Label();
            this.contextMenuStrip1 = new
System.Windows.Forms.ContextMenuStrip(this.components);
            this.textBoxFileReadTime = new System.Windows.Forms.TextBox();
```

```
this.textBoxFileReadCount = new System.Windows.Forms.TextBox();
this.textBoxFind = new System.Windows.Forms.TextBox();
this.textBoxExactTime = new System.Windows.Forms.TextBox();
this.textBoxMaxDist = new System.Windows.Forms.TextBox();
this.textBoxThreadCount = new System.Windows.Forms.TextBox();
this.textBoxApproxTime = new System.Windows.Forms.TextBox();
this.textBoxThreadCountAll = new System.Windows.Forms.TextBox();
this.button1 = new System.Windows.Forms.Button();
this.button2 = new System.Windows.Forms.Button();
this.button3 = new System.Windows.Forms.Button();
this.listBoxResult = new System.Windows.Forms.ListBox();
this.button4 = new System.Windows.Forms.Button();
//this.button5 = new System.Windows.Forms.Button();
this.SuspendLayout();
//
// label1
this.button1.Location = new System.Drawing.Point(40, 498);
this.button1.Margin = new System.Windows.Forms.Padding(4);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(157, 37);
this.button1.TabIndex = 0;
this.button1.Text = "Чтение из файла";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click += new System.EventHandler(this.button1_Click);
//
this.label1.AutoSize = true;
this.label1.Location = new System.Drawing.Point(137, 24);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(130, 13);
this.label1.TabIndex = 0;
this.label1.Text = "Время чтения из файла:";
//
// label2
this.button2.Location = new System.Drawing.Point(12, 99);
this.button2.Name = "button2";
this.button2.Size = new System.Drawing.Size(119, 80);
this.button2.TabIndex = 18;
this.button2.Text = "Чёткий поиск";
this.button2.UseVisualStyleBackColor = true;
this.button2.Click += new System.EventHandler(this.button2_Click);
//
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(137, 53);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(203, 13);
this.label2.TabIndex = 1;
this.label2.Text = "Количество уникальных слов в файле:";
//
// label3
this.button3.Location = new System.Drawing.Point(12, 186);
this.button3.Name = "button3";
this.button3.Size = new System.Drawing.Size(119, 80);
this.button3.TabIndex = 19;
this.button3.Text = "Параллельный нечёткий поиск";
this.button3.UseVisualStyleBackColor = true;
this.button3.Click += new System.EventHandler(this.button3_Click);
//
this.label3.AutoSize = true;
this.label3.Location = new System.Drawing.Point(137, 82);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(101, 13);
this.label3.TabIndex = 2;
this.label3.Text = "Слово для поиска:";
//
// label4
```

```
//
this.label4.AutoSize = true;
this.label4.Location = new System.Drawing.Point(137, 116);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(124, 13);
this.label4.TabIndex = 3;
this.label4.Text = "Время чёткого поиска:";
//
// label5
//
this.label5.AutoSize = true;
this.label5.Location = new System.Drawing.Point(137, 149);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(266, 13);
this.label5.TabIndex = 4;
this.label5.Text = "Максимальное расстояние для нечёткого поиска: ";
//
// label6
//
this.label6.AutoSize = true;
this.label6.Location = new System.Drawing.Point(137, 182);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(113, 13);
this.label6.TabIndex = 5;
this.label6.Text = "Количество потоков:";
//
// label7
//
this.label7.AutoSize = true;
this.label7.Location = new System.Drawing.Point(137, 214);
this.label7.Name = "label7";
this.label7.Size = new System.Drawing.Size(186, 13);
this.label7.TabIndex = 6;
this.label7.Text = "Вычисленное количество потоков: ";
//
// label8
//
this.label8.AutoSize = true;
this.label8.Location = new System.Drawing.Point(137, 249);
this.label8.Name = "label8";
this.label8.Size = new System.Drawing.Size(135, 13);
this.label8.TabIndex = 7;
this.label8.Text = "Время нечёткого потока:";
//
// contextMenuStrip1
//
this.contextMenuStrip1.Name = "contextMenuStrip1";
this.contextMenuStrip1.Size = new System.Drawing.Size(61, 4);
//
// textBoxFileReadTime
//
this.textBoxFileReadTime.Location = new System.Drawing.Point(273, 24);
this.textBoxFileReadTime.Name = "textBoxFileReadTime";
this.textBoxFileReadTime.ReadOnly = true;
this.textBoxFileReadTime.Size = new System.Drawing.Size(100, 20);
this.textBoxFileReadTime.TabIndex = 9;
//
// textBoxFileReadCount
//
this.textBoxFileReadCount.Location = new System.Drawing.Point(346, 53);
this.textBoxFileReadCount.Name = "textBoxFileReadCount";
this.textBoxFileReadCount.ReadOnly = true;
this.textBoxFileReadCount.Size = new System.Drawing.Size(100, 20);
this.textBoxFileReadCount.TabIndex = 10;
//
```

```
// textBoxFind
//
this.textBoxFind.Location = new System.Drawing.Point(245, 82);
this.textBoxFind.Name = "textBoxFind";
this.textBoxFind.Size = new System.Drawing.Size(100, 20);
this.textBoxFind.TabIndex = 11;
//
// textBoxExactTime
//
this.textBoxExactTime.Location = new System.Drawing.Point(268, 116);
this.textBoxExactTime.Name = "textBoxExactTime";
this.textBoxExactTime.ReadOnly = true;
this.textBoxExactTime.Size = new System.Drawing.Size(100, 20);
this.textBoxExactTime.TabIndex = 12;
//
// textBoxMaxDist
//
this.textBoxMaxDist.Location = new System.Drawing.Point(410, 149);
this.textBoxMaxDist.Name = "textBoxMaxDist";
this.textBoxMaxDist.Size = new System.Drawing.Size(100, 20);
this.textBoxMaxDist.TabIndex = 13;
//
// textBoxThreadCount
//
this.textBoxThreadCount.Location = new System.Drawing.Point(257, 182);
this.textBoxThreadCount.Name = "textBoxThreadCount";
this.textBoxThreadCount.Size = new System.Drawing.Size(100, 20);
this.textBoxThreadCount.TabIndex = 14;
//
// textBoxApproxTime
//
this.textBoxApproxTime.Location = new System.Drawing.Point(330, 214);
this.textBoxApproxTime.Name = "textBoxApproxTime";
this.textBoxApproxTime.ReadOnly = true;
this.textBoxApproxTime.Size = new System.Drawing.Size(100, 20);
this.textBoxApproxTime.TabIndex = 15;
//
// textBoxThreadCountAll
//
this.textBoxThreadCountAll.Location = new System.Drawing.Point(279, 249);
this.textBoxThreadCountAll.Name = "textBoxThreadCountAll";
this.textBoxThreadCountAll.ReadOnly = true;
this.textBoxThreadCountAll.Size = new System.Drawing.Size(100, 20);
this.textBoxThreadCountAll.TabIndex = 16;
//
// button1
//
this.button1.Location = new System.Drawing.Point(13, 13);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(118, 80);
this.button1.TabIndex = 17;
this.button1.Text = "Чтение из файла";
this.button1.UseVisualStyleBackColor = true;

//
// button2
//
this.button2.Location = new System.Drawing.Point(12, 99);
this.button2.Name = "button2";
this.button2.Size = new System.Drawing.Size(119, 80);
this.button2.TabIndex = 18;
this.button2.Text = "Чёткий поиск";
this.button2.UseVisualStyleBackColor = true;
//
// button3
```

```
//
this.button3.Location = new System.Drawing.Point(12, 186);
this.button3.Name = "button3";
this.button3.Size = new System.Drawing.Size(119, 80);
this.button3.TabIndex = 19;
this.button3.Text = "Параллельный нечёткий поиск";
this.button3.UseVisualStyleBackColor = true;
//
// listBoxResult
//
this.listBoxResult.FormattingEnabled = true;
this.listBoxResult.Location = new System.Drawing.Point(140, 283);
this.listBoxResult.Name = "listBoxResult";
this.listBoxResult.Size = new System.Drawing.Size(370, 160);
this.listBoxResult.TabIndex = 20;
//
// button4
//
this.button4.Location = new System.Drawing.Point(140, 450);
this.button4.Name = "button4";
this.button4.Size = new System.Drawing.Size(132, 44);
this.button4.TabIndex = 21;
this.button4.Text = "Сохранение";
this.button4.UseVisualStyleBackColor = true;
this.button4.Click += new System.EventHandler(this.button4_Click);
//
// button5
//
//this.button5.Location = new System.Drawing.Point(378, 450);
//this.button5.Name = "button5";
//this.button5.Size = new System.Drawing.Size(132, 44);
//this.button5.TabIndex = 22;
//this.button5.Text = "Выход";
//this.button5.UseVisualStyleBackColor = true;
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(624, 506);
//this.Controls.Add(this.button5);
this.Controls.Add(this.button4);
this.Controls.Add(this.listBoxResult);
this.Controls.Add(this.button3);
this.Controls.Add(this.button2);
this.Controls.Add(this.button1);
this.Controls.Add(this.textBoxThreadCountAll);
this.Controls.Add(this.textBoxApproxTime);
this.Controls.Add(this.textBoxThreadCount);
this.Controls.Add(this.textBoxMaxDist);
this.Controls.Add(this.textBoxExactTime);
this.Controls.Add(this.textBoxFind);
this.Controls.Add(this.textBoxFileReadCount);
this.Controls.Add(this.textBoxFileReadTime);
this.Controls.Add(this.label8);
this.Controls.Add(this.label7);
this.Controls.Add(this.label6);
this.Controls.Add(this.label5);
this.Controls.Add(this.label4);
this.Controls.Add(this.label3);
this.Controls.Add(this.label2);
this.Controls.Add(this.label1);
this.Name = "Form1";
this.Text = "Form1";
this.ResumeLayout(false);
```

```
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.Label label6;
        private System.Windows.Forms.Label label7;
        private System.Windows.Forms.Label label8;
        private System.Windows.Forms.ContextMenuStrip contextMenuStrip1;
        private System.Windows.Forms.TextBox textBoxFileReadTime;
        private System.Windows.Forms.TextBox textBoxFileReadCount;
        private System.Windows.Forms.TextBox textBoxFind;
        private System.Windows.Forms.TextBox textBoxExactTime;
        private System.Windows.Forms.TextBox textBoxMaxDist;
        private System.Windows.Forms.TextBox textBoxThreadCount;
        private System.Windows.Forms.TextBox textBoxApproxTime;
        private System.Windows.Forms.TextBox textBoxThreadCountAll;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.Button button3;
        private System.Windows.Forms.ListBox listBoxResult;
        private System.Windows.Forms.Button button4;
        //private System.Windows.Forms.Button button5;
    }
}
```

## Form1.cs

```
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;
using System.Threading.Tasks;


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

        // Список слов

        List<string> list = new List<string>();


        private void button5(object sender, EventArgs e)
        {
            this.Close();
        }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog()
            {
                Multiselect = false,
                Filter = "текстовые файлы|*.txt"
            };

            var stopwatch = new Stopwatch();
            if (openFileDialog.ShowDialog() == DialogResult.OK)
            {

                stopwatch.Start();

                string text = File.ReadAllText(openFileDialog.FileName);

                char[] separators = new char[] { ' ', '.', ',', '!', '?', '/', '\t', '\n'
};

                foreach (var strTemp in text.Split(separators))
                {
                    string str = strTemp.Trim();
                    if (!list.Contains(str))
                    {
                        list.Add(str);
                    }
                }

                stopwatch.Stop();
                this.textBoxFileReadTime.Text = stopwatch.Elapsed.ToString() + " ms";
                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.IsNullOrWhiteSpace(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");
            }
        }


        public static List<ParallelSearchResult> ArrayThreadTask(object paramObj)
        {
            ParallelSearchThreadParam param = (ParallelSearchThreadParam)paramObj;
            //Слово для поиска в верхнем регистре
            string wordUpper = param.wordPattern.Trim().ToUpper(); //Результатыпоиска в
одном потоке
            List<ParallelSearchResult> Result = new List<ParallelSearchResult>();
            //Перебор всех слов во временном списке данного потока
            foreach (string str in param.tempList)
            {
                //Вычисление расстояния Дамерау-Левенштейна
                int dist = EditDistance.Distance(str.ToUpper(), wordUpper);
                //Если расстояние меньше порогового, то слово добавляется в результат
                if (dist <= param.maxDist)
                {
                    ParallelSearchResult temp = new ParallelSearchResult()
                    {
                        word = str,
                        dist = dist,
                        ThreadNum = param.ThreadNum
                    };
                    Result.Add(temp);
                }
            }
            return Result;
        }
        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.IsNullOrWhiteSpace(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;
                }
                int ThreadCount;
                if (!int.TryParse(this.textBoxThreadCount.Text.Trim(),
                out ThreadCount))
                {
                    MessageBox.Show("Необходимо указать количество потоков"); //потоки,
на которые разделяется массив слов исходного файла
                    return;
                }

                Stopwatch timer = new Stopwatch();
```

```
                timer.Start();//Начало параллельного поиска

                List<ParallelSearchResult> Result = new
List<ParallelSearchResult>();//Результирующий список

                //Деление списка на фрагменты для параллельного запуска в потоках

                List<MinMax> arrayDivList = SubArrays.DivideSubArrays(0, list.Count,
ThreadCount);
                int count = arrayDivList.Count;

                //Количество потоков соответствует количеству фрагментов массива

                //Task - класс, использующийся для параллельного поиска(задача)

                Task<List<ParallelSearchResult>>[] tasks = new
Task<List<ParallelSearchResult>>[count];
                //Запуск потоков
                for (int i = 0; i < count; i++)
                {
                    //Создание временного списка, чтобы потоки не работали параллельно с
одной коллекцией


                    List<string> tempTaskList = list.GetRange(arrayDivList[i].Min,
                    arrayDivList[i].Max - arrayDivList[i].Min);
                    tasks[i] = new Task<List<ParallelSearchResult>>
                        (ArrayThreadTask, new ParallelSearchThreadParam()
                        {
                            tempList = tempTaskList,
                            maxDist = maxDist,
                            ThreadNum = i,
                            wordPattern = word
                        });
                    //Запуск потока
                    tasks[i].Start();
                }
                //ожидание завершения работы всех потоков, чтобы получить результаты
поиска

                Task.WaitAll(tasks);
                timer.Stop();
                //Объединение результатов
                for (int i = 0; i < count; i++)
                {
                    Result.AddRange(tasks[i].Result);
                }

                timer.Stop();

                //Вывод результатов

                //Время поиска

                this.textBoxApproxTime.Text = timer.Elapsed.ToString();

                this.textBoxThreadCountAll.Text = count.ToString(); //Вычисленное
количество потоков

                this.listBoxResult.BeginUpdate();//Начало обновления списка результатов

                this.listBoxResult.Items.Clear(); //Очистка списка

                foreach (var x in Result) //Вывод результатов поиска
                {
```

```
                    string temp = x.word + "(расстояние=" + x.dist.ToString() + " поток="
+ x.ThreadNum.ToString() + ")";
                    this.listBoxResult.Items.Add(temp);
                }

                this.listBoxResult.EndUpdate();//Окончание обновления спискарезультатов
            }
            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_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)
```

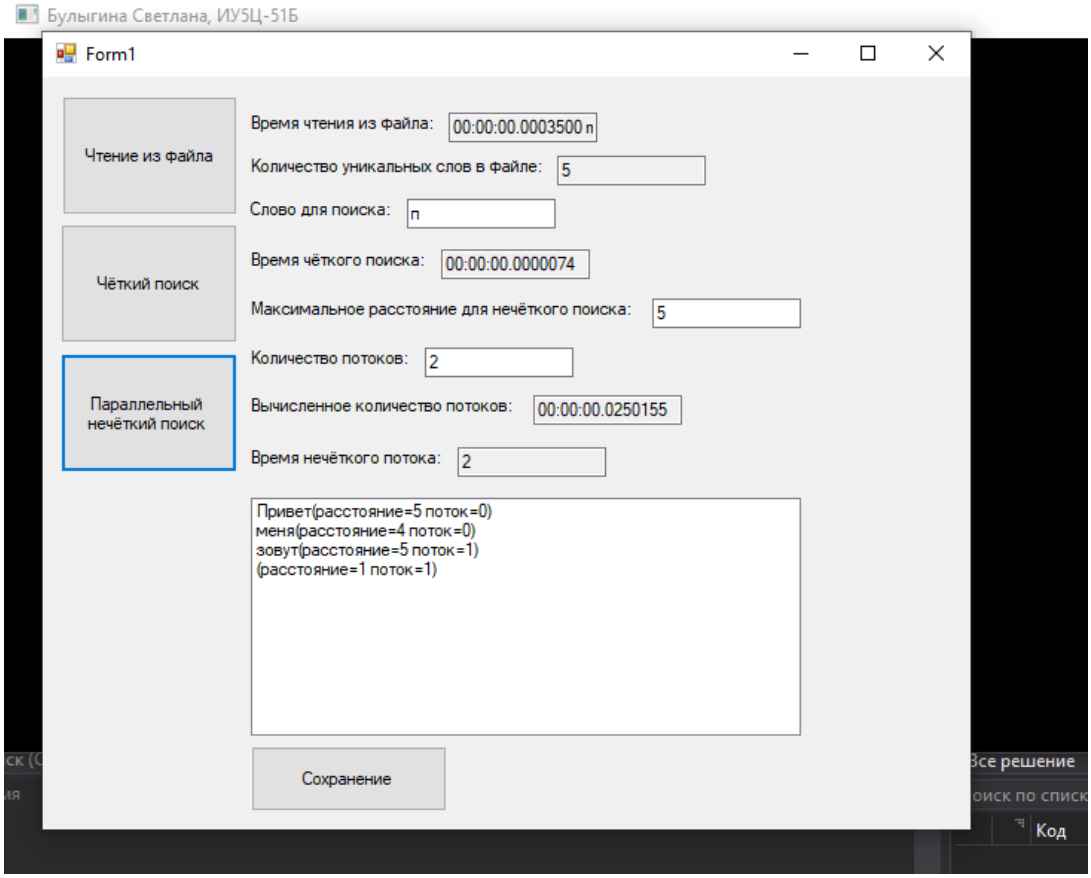
```
                {
                    b.AppendLine("<li>" + x.ToString() + "</li>");
                }
                b.AppendLine("</ul>");
                b.AppendLine("</td>");
                b.AppendLine("</tr>");
                b.AppendLine("</table>");
                b.AppendLine("</body>");
                b.AppendLine("</html>");
                //Сохранение файла
                File.AppendAllText(ReportFileName, b.ToString());
                MessageBox.Show("Отчет сформирован. Файл: " +
                ReportFileName);
            }
        }
    }
    public static class EditDistance
    {
        /// Вычисление расстояния Дамерау-Левенштейна
        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
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
                                                                  //Элементматрицы
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];
        }
    }
    /// Результаты параллельного поиска
    //содержит входной массив слов и слово для поиска, максимальное расстояние для
нечеткого поиска и номер потока
    public class ParallelSearchResult
    {
        /// Найденное слово
        public string word { get; set; }
        /// Расстояние
        public int dist { get; set; }
        /// Номер потока
        public int ThreadNum { get; set; }
    }
    /// Параметры которые передаются в поток для параллельного поиска
    class ParallelSearchThreadParam
    {
        /// Массив для поиска
        public List<string> tempList { get; set; }
        /// Слово для поиска
        public string wordPattern { get; set; }
        /// Максимальное расстояние для нечеткого поиска
        public int maxDist { get; set; }
        /// Номер потока
        public int ThreadNum { get; set; }
    }
    /// Хранение минимального и максимального значений диапазона
    public class MinMax
    {
        public int Min { get; set; }
        public int Max { get; set; }
        public MinMax(int pmin, int pmax)
        {
            this.Min = pmin;
            this.Max = pmax;
        }
    }
    //Для деления массива на подмассивы
    public static class SubArrays
    {
        /// Деление массива на последовательности(подмассивы)
        /// <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;
        }
    }
}
```

## Тест программы




## Ссылка на репозиторий исходных кодов GitHub

https://github.com/SvetikLana/BKIT-DZ_Bulygina