МГТУ имени Баумана

Факультет «Информатика и Системы управления»

Кафедра «Автоматизированные системы обработки информации и управления»

Дисциплина «Базовые компоненты интернет технологий»

Отчет по лабораторной работе №4 и 5

Выполнила

студентка группы

ИУ5-34б

Слободчикова Юлия

Москва 2020

Описание задания:

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 находится 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).

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 Дамерау-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.

Текст программы:

Код формы:

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


namespace lab4
{
    public partial class Form1 : Form
    {
        ///<summary>
        ///Список слов
        ///</summary>
        List<string> list = new List<string>();

        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            OpenFileDialog fd = new OpenFileDialog();
            fd.Filter = "текстовые файлы|*.txt";
            if (fd.ShowDialog() == DialogResult.OK) //не выбранно корректное имя
            {
                Stopwatch t_load = new Stopwatch(); //измерение времени выполнения
потоков
                t_load.Start();

                // Чтение файла в виде одной строки
                string text = File.ReadAllText(fd.FileName);
                //Разделительные символы для разбиения полученной строки
                char[] separators = new char[] { ' ', '.', ',', '!', '?', '/', '\t', '\n'
};
                string[] textArray = text.Split(separators); //разделение строки на
подстроки
                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_load.Stop();
                this.textBoxFileReadTime.Text = t_load.Elapsed.ToString(); //свойство
Elapsed возвращает измеренное время
                this.textBoxFileReadCount.Text = list.Count.ToString(); //кол-во
уникальных слов
                MessageBox.Show("Файл прочитан");
            }
            else {
                MessageBox.Show("Необходимо выбрать файл");
            }
        }

        private void textBoxFileReadTime_TextChanged(object sender, EventArgs e)
        {

        }

        private void buttonSearch_Click(object sender, EventArgs e)
        {
            //Слово для поиска
            string word = this.textBoxFind.Text.Trim(); //удаление пробелов в слове,
которое ввели
            // Если слово для поиска не пусто и в тексте есть слова
            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_search = new Stopwatch();
                t_search.Start();
                foreach (string str in list)                    //перебор всех слов в
спике
                {
                    if (str.ToUpper().Contains(wordUpper)) //Contains проверяет
совпадение
                    {
                        tempList.Add(str);                 //тогда добавляес во временыый
список
                    }
                }
                t_search.Stop();
                this.textBoxSearchTime.Text = t_search.Elapsed.ToString();
                this.listBoxResult.BeginUpdate(); //для начала обновления данных списка
вызываем метод
                //Очистка списка
                this.listBoxResult.Items.Clear();
                //Вывод результатов поиска
                foreach(string str in tempList)
                {
                    this.listBoxResult.Items.Add(str); //добавление элементов в список,
выводимый на экран
                }
                this.listBoxResult.EndUpdate(); //для завершения обновления даннхы
            }
            else
            {
                MessageBox.Show("Необходимо выбрать файл и ввести слово для поиска");
            }
        }
```

```
        private void buttonParallelSearch_Click(object sender, EventArgs e)
        {
            //Слово для поиска
            string word = this.textBoxFind.Text.Trim();          //удаление пробелов в
введенном слове

            //Если слово для поиска не пусто
            if (!string.IsNullOrWhiteSpace(word) && list.Count > 0)
            {
                //максимальное расстояние
                int maxDist;
                if (!int.TryParse(this.textBoxMaxDist.Text.Trim(), out maxDist))
//провека на введение числа
                {
                    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;
                }

                //количество потоков
                int ThreadCount;
                if (!int.TryParse(this.textBoxThreadCount.Text.Trim(), out ThreadCount))
//провека на введение числа
                {
                    MessageBox.Show("Необходимо указать количество потоков");
                    return;
                }

                //время поиска
                Stopwatch timer = new Stopwatch();
                timer.Start();

                //-------------------------------------------------
                // Начало параллельного поиска
                //-------------------------------------------------

                //Результирующий список
                List<ParallelSearchResult> Result = new List<ParallelSearchResult>();
//записываем все подходящие по параметрам слова

                //Деление списка на фрагменты для параллельного запуска в потоках
                List<MinMax> arrayDivList = SubArrays.DivideSubArrays(0, list.Count,
ThreadCount); //вызываемый и прописанный класс начало списка, кол-во эл списка, поток
                int count = arrayDivList.Count; //колво-во элементов в разбиении

                //Количество потоков соответствует количеству фрагментов массива
                Task<List<ParallelSearchResult>>[] tasks = new
Task<List<ParallelSearchResult>>[count]; // массив tasks объектов класса Task (класс для
работы с потоками )

                //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); //Копирование элементов на заданном диапазоне
индексов
```

```
                    tasks[i] = new Task<List<ParallelSearchResult>>(   //создание
объектов класса Task
                        //Метод, который будет выполняться в потоке
                        ArrayThreadTask, // первый параметр исполняемый в потоке метод
                        //Параметры потока
                        new ParallelSearchThreadParam() //параметр типа Object передается
как кортеж
                        {
                            tempList = tempTaskList,
                            maxDist = maxDist,
                            ThreadNum = i,
                            wordPattern = word
                        });

                    //Запуск потока
                    tasks[i].Start();
                }

                Task.WaitAll(tasks); //ждем завершения работы всех потоков, чтобы
получить результаты поиска
                //Метод WaitAll завершит работу только после того как отработают все
потоки массива tasks

                timer.Stop();

                //Объединение результатов
                for (int i = 0; i < count; i++)
                {
                    Result.AddRange(tasks[i].Result); //добавляет в список все элементы
другого списка.
                }

                //-------------------------------------------------
                // Завершение параллельного поиска
                //-------------------------------------------------

                timer.Stop();

                //Вывод результатов

                //Время поиска
                // this.textBoxApproxTime.Text = timer.Elapsed.ToString();

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

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

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

                //Вывод результатов поиска
                foreach (var x in Result)
                {
                    string temp = x.word + " (расстояние = " + x.dist.ToString() + "
поток = " + x.ThreadNum.ToString() + ") ";
                    this.listBoxParallelResult.Items.Add(temp);
                }

                //Окончание обновления списка результатов
                this.listBoxParallelResult.EndUpdate();
            }
            else
```

```
            {
                MessageBox.Show("Необходимо выбрать файл и ввести слово для поиска");
            }
        }

        /// <summary>
        /// Выполняется в параллельном потоке для поиска строк
        /// </summary>
        public static List<ParallelSearchResult> ArrayThreadTask(object paramObj)
//возвращает тип List<int>
        {
            ParallelSearchThreadParam param = (ParallelSearchThreadParam)paramObj;
//получение параметров

            //Слово для поиска в верхнем регистре
            string wordUpper = param.wordPattern.Trim().ToUpper();

            //Результаты поиска 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 = DistanceLevenstein.Distance(str.ToUpper(), wordUpper);

                //Если расстояние меньше порогового, то слово добавляется в результат
                if (dist <= param.maxDist)
                {
                    ParallelSearchResult temp = new ParallelSearchResult()
                    {
                        word = str,            //слово
                        dist = dist,           //разница
                        ThreadNum = param.ThreadNum //какой поток
                    };

                    Result.Add(temp); //сохранение
                }
            }
            return Result;
        }

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

            //Диалог сохранения файла отчета
            SaveFileDialog fd = new SaveFileDialog(); //при вызове пользователю
предлагается сохранить файл
            fd.FileName = TempReportFileName;
            fd.DefaultExt = ".html"; //расширение файла по умолчанию
            fd.Filter = "HTML Reports|*.html";

            if (fd.ShowDialog() == DialogResult.OK) //если имя корректное, если нет
завершение работы
            {
                string ReportFileName = fd.FileName;

                //Формирование отчета
                StringBuilder b = new StringBuilder();  //Отчет формируется в виде HTML-
таблицы.
                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 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("<tr>");
                b.AppendLine("<td>Максимальное расстояние для нечеткого поиска</td>");
                b.AppendLine("<td>" + this.textBoxMaxDist.Text + "</td>");
                b.AppendLine("</tr>");

                b.AppendLine("<tr valign='top'>");
                b.AppendLine("<td>Результаты параллельного поиска</td>");
                b.AppendLine("<td>");
                b.AppendLine("<ul>");

                foreach (var x in this.listBoxParallelResult.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()); //отчет сохраняется в
текстовый файл с помощью метода

                MessageBox.Show("Отчет сформирован. Файл: " + ReportFileName);
```

```
            }
        }
    }
     }
```

Код алгоритма Вагнера-Фишера

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

namespace lab4
{
    public static class DistanceLevenstein
    {
        /// <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
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];
        }
    }
     }
```

Класс МинМакс

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

namespace lab4
{
    /// <summary>
    /// Хранение минимального и максимального значений диапазона
    /// </summary>
    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;
        }
    }
     }
```

Класс результата параллельного поиска

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

namespace lab4
{
    /// <summary>
    /// Результаты параллельного поиска
    /// </summary>
    public class ParallelSearchResult
    {
        /// <summary>
        /// Найденное слово
        /// </summary>
        public string word { get; set; }

        /// <summary>
        /// Расстояние
        /// </summary>
        public int dist { get; set; }

        /// <summary>
        /// Номер потока
        /// </summary>
```

```
        public int ThreadNum { get; set; }
    }
     }
```

Класс потоков для параллельного поиска

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

namespace lab4
{
    /// <summary>
    /// Параметры которые передаются в поток для параллельного поиска
    /// </summary>
    class ParallelSearchThreadParam
    {
        /// <summary>
        /// Массив для поиска
        /// </summary>
        public List<string> tempList { get; set; }

        /// <summary>
        /// Слово для поиска
        /// </summary>
        public string wordPattern { get; set; }

        /// <summary>
        /// Максимальное расстояние для нечеткого поиска
        /// </summary>
        public int maxDist { get; set; }

        /// <summary>
        /// Номер потока
        /// </summary>
        public int ThreadNum { get; set; }
    }
     }
```

Класс деления массива на последовательности

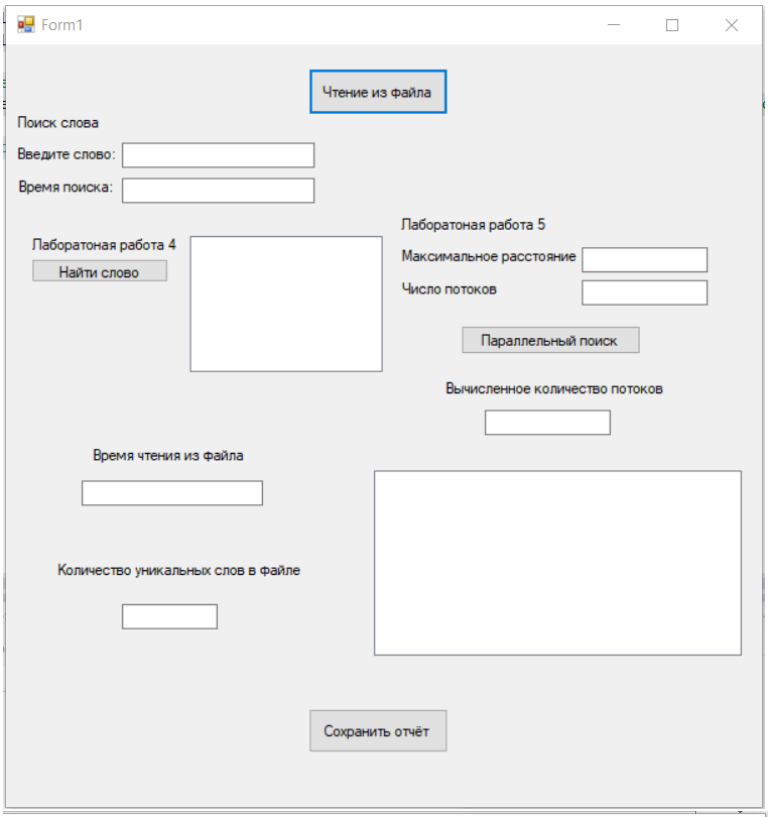
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab4
{
    /// <summary>
    /// Класс для деления массива на последовательности
    /// </summary>
    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)
```

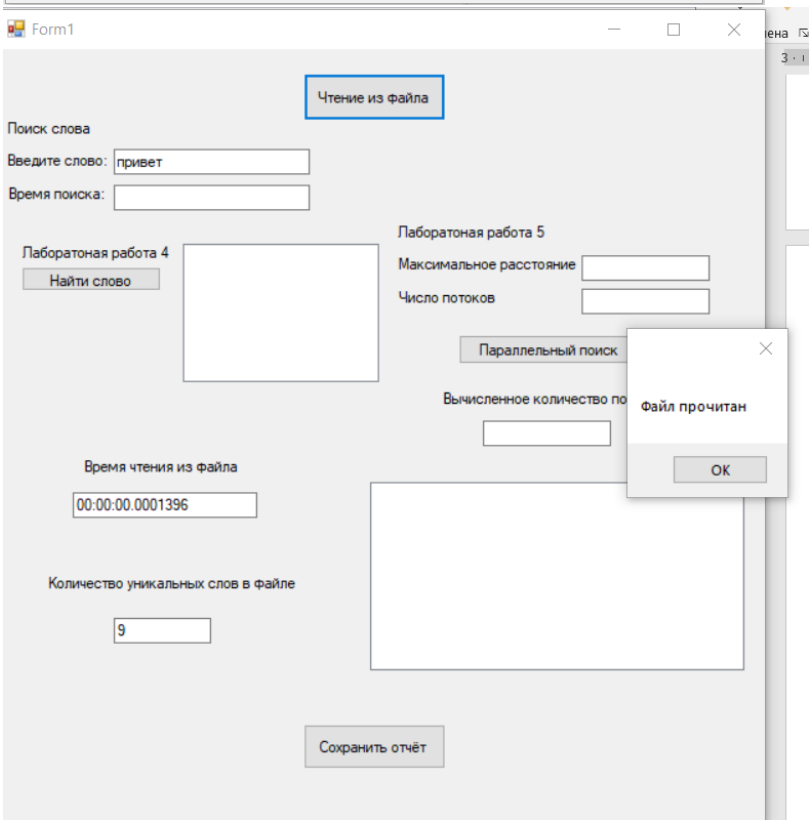
```
        {
            //Результирующий список пар с индексами подмассивов
            List<MinMax> result = new List<MinMax>();

            //Если число элементов в массиве слишком мало для деления
            //то возвращается массив целиком
            if ((endIndex - beginIndex) <= subArraysCount)
            {
                result.Add(new MinMax(0, (endIndex - beginIndex))); //возвращаем список
объектов класса МинМакс, которые хранят начальный и конечный индексы соотв диапазонов
массива
            }
            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;
        }
    }
     }
```

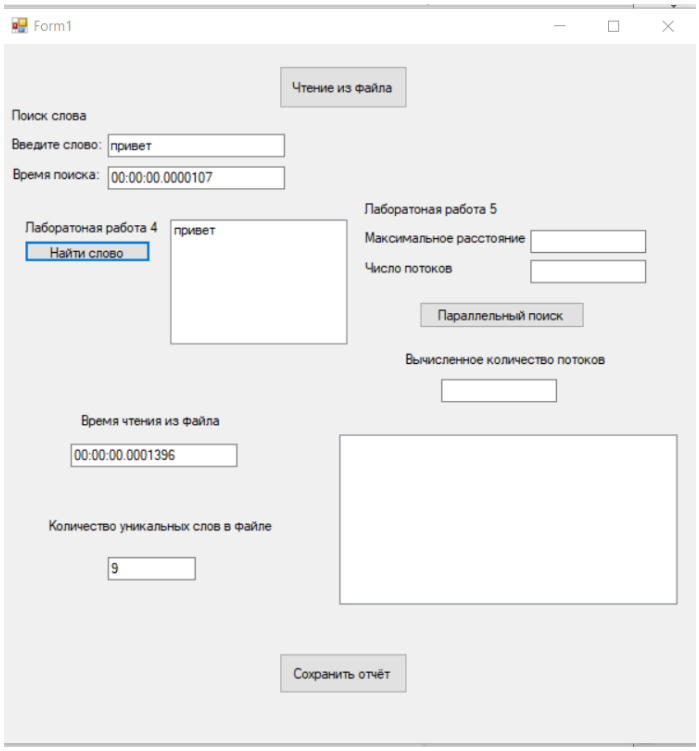
Запуск программы

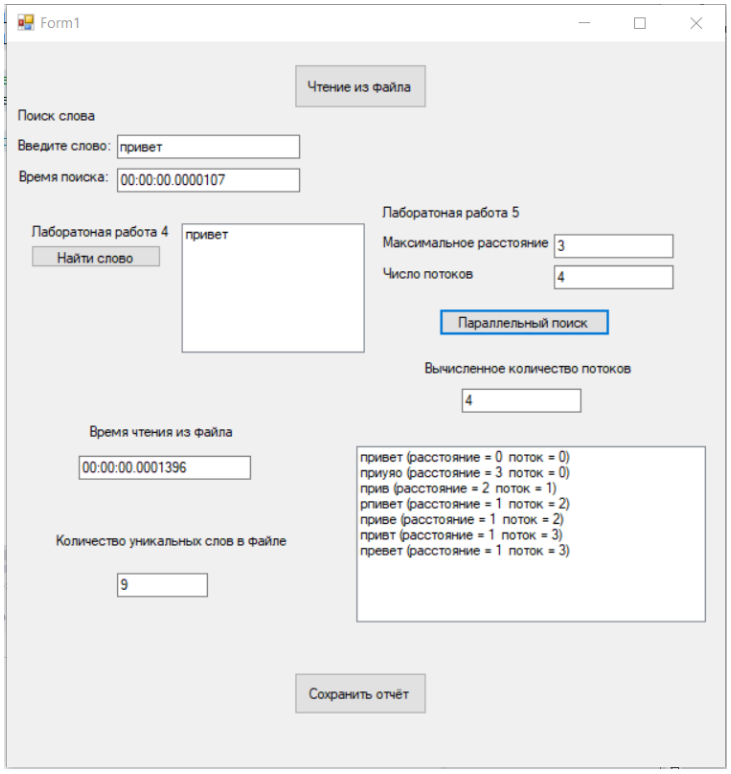
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace lab4
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
     }
```

Экранные формы с примерами выполнения программы:






Form1
Чтение из файла
Поиск слова
Введите слово:
привет
Время поиска:
00:00:00.0000107
Лабораторная работа 4
Найти слово
привет
Лабораторная работа 5
Максимальное расстояние
Число потоков
Параллельный поиск
Вычисленное количество потоков
Время чтения из файла
00:00:00.0001396
Количество уникальных слов в файле
9
Сохранить отчёт


Form1
Чтение из файла
Поиск слова
Введите слово:
привет
Время поиска:
00:00:00.0000107
Лабораторная работа 4
Найти слово
привет
Лабораторная работа 5
Максимальное расстояние
3
Число потоков
4
Параллельный поиск
Вычисленное количество потоков
4
Время чтения из файла
00:00:00.0001396
привет (расстояние = 0 поток = 0)
приуяо (расстояние = 3 поток = 0)
прив (расстояние = 2 поток = 1)
рпивет (расстояние = 1 поток = 2)
приве (расстояние = 1 поток = 2)
привт (расстояние = 1 поток = 3)
превет (расстояние = 1 поток = 3)
Количество уникальных слов в файле
9
Сохранить отчёт