Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по домашнему заданию «Многопоточный поиск в файле»

Выполнил:
студент группы ИУ5-33
Сергеев МЮ
Подпись и дата:
29.12.20

Проверил:
Подпись и дата:

Москва, 2020 г.

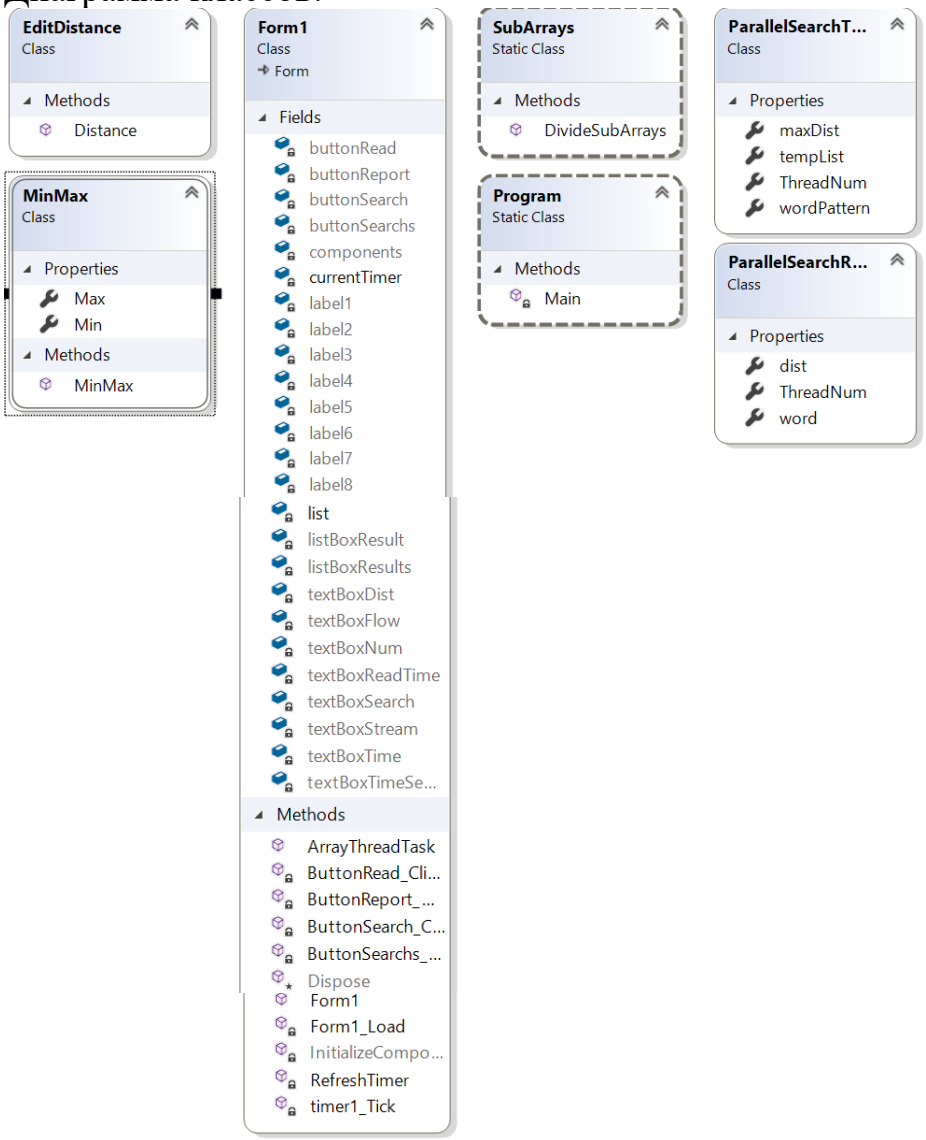
Задание:

Часть 1. Разработать программу, использующую делегаты.

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). 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

Program.cs

```
using System;
using System.Windows.Forms;
namespace Lab7
{
   static class Program
   {
      ///<summary>
      ///Главная точка входа для приложения.
      ///</summary>
      [STAThread]
```

```
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Form1.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace Lab7
{
    public partial class Form1 : Form
    {
        public Form1() { InitializeComponent(); }
        List<string> list = new List<string>();// Список слов
        private void ButtonRead_Click_1(object sender, EventArgs e)
        {
            OpenFileDialog fd = new OpenFileDialog(); fd.Filter = "текстовые
файлы|*.txt"; if (fd.ShowDialog() == DialogResult.OK)
            {
                Stopwatch t_load = new Stopwatch(); t_load.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_load.Stop();
        textBoxReadTime.Text = t_load.Elapsed.ToString();
        textBoxNum.Text = list.Count.ToString();
        MessageBox.Show("Файл успешно прочитан");
    }
}
/// Текущее состояние таймера
/// </summary>
TimeSpan currentTimer = new TimeSpan();
/// <summary>
/// Обновление текущего состояния таймера
/// </summary>
private void RefreshTimer()
{
    //Обновление поля таймера в форме
    textBoxReadTime.Text = currentTimer.ToString();
}
private void ButtonSearch_Click(object sender, EventArgs e)
{
    //Слово для поиска
    string word = this.textBoxSearch.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_search = new Stopwatch();
        t_search.Start(); foreach (string str in list)
        {
            if (str.ToUpper().Contains(wordUpper)) { tempList.Add(str); }
        }
        t_search.Stop();
        textBoxTime.Text = t_search.Elapsed.ToString();
        listBoxResult.BeginUpdate();
        //Очистка списка
        listBoxResult.Items.Clear();
        //Вывод результатов поиска
```

```
            foreach (string str in tempList)
            {
                listBoxResult.Items.Add(str);
            }
            listBoxResult.EndUpdate();
        }
        else
        {
            MessageBox.Show("Необходимо выбрать файл и ввести слово для
поиска");
        }
    }
    private void timer1_Tick(object sender, EventArgs e)
    {
        //Добавление к текущему состоянию таймера интервала в одну
секунду
        currentTimer = currentTimer.Add(new TimeSpan(0, 0, 1));
        //Обновление текущего состояния таймера
        RefreshTimer();
    }
    private void Form1_Load(object sender, EventArgs e)
    {
        //Обновление текущего состояния таймера
        RefreshTimer();
    }
    private void ButtonSearchs_Click(object sender, EventArgs e)
    {
        //Слово для поиска
        string word = textBoxSearch.Text.Trim();
        //Если слово для поиска не пусто
        if (!string.IsNullOrWhiteSpace(word) && list.Count > 0)
        {
            int maxDist;
            if (!int.TryParse(textBoxDist.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.textBoxFlow.Text.Trim(), out ThreadCount))
            {
                MessageBox.Show("Необходимо указать количество потоков");
                return;
            }
            Stopwatch timer = new Stopwatch(); timer.Start();
            //Начало параллельного поиска
            //Результирующий список
            List<ParallelSearchResult> Result = new List<ParallelSearchResult>();
            //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<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  одной коллекцией
                List<string> tempTaskList = list.GetRange(arrayDivList[i].Min, arrayDivList[i].Max - arrayDivList[i].Min);
                tasks[i] = new Task<List<ParallelSearchResult>>(
                //Метод, который будет выполняться в потоке
                ArrayThreadTask,
                //Параметры потока
                new ParallelSearchThreadParam()
                {
                    tempList = tempTaskList,
                    maxDist = maxDist,
```

```
                    ThreadNum = i,
                    wordPattern = word
                });
                //Запуск потока
                tasks[i].Start();
            }
            Task.WaitAll(tasks);
            timer.Stop();
            //Объединение результатов
            for (int i = 0; i < count; i++)
            {
                Result.AddRange(tasks[i].Result);
            }
            // Завершение параллельного поиска
            timer.Stop();
            //Вывод результатов
            //Время поиска
            textBoxTimeSearchs.Text = timer.Elapsed.ToString();
            //Вычисленное количество потоков
            textBoxStream.Text = count.ToString();
            //Начало обновления списка результатов
            listBoxResults.BeginUpdate();
            //Очистка списка
            listBoxResults.Items.Clear();
            //Вывод результатов поиска
            foreach (var x in Result)
            {
                string temp = x.word + "(расстояние=" + x.dist.ToString() + "
поток=" + x.ThreadNum.ToString() + ")";
                listBoxResults.Items.Add(temp);
            }
            //Окончание обновления списка результатов
            listBoxResults.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)
        {
            ParallelSearchThreadParam param =
(ParallelSearchThreadParam)paramObj;
            //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 = 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 ButtonReport_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>" + textBoxTime.Text + "</td>");
               b.AppendLine("</tr>"); b.AppendLine("<tr>");
               b.AppendLine("<td>Количество уникальных слов в файле</td>");
               b.AppendLine("<td>" + textBoxNum.Text + "</td>");
               b.AppendLine("</tr>");
               b.AppendLine("<tr>");
               b.AppendLine("<td>Слово для поиска</td>");
               b.AppendLine("<td>" + textBoxSearch.Text + "</td>");
               b.AppendLine("</tr>"); b.AppendLine("<tr>");
               b.AppendLine("<td>Максимальное расстояние для нечеткого
поиска</td>");
               b.AppendLine("<td>" + textBoxDist.Text + "</td>");
               b.AppendLine("</tr>"); b.AppendLine("<tr>");
               b.AppendLine("<td>Время четкого поиска</td>");
               b.AppendLine("<td>" + textBoxTime.Text + "</td>");
               b.AppendLine("</tr>"); b.AppendLine("<tr>");
               b.AppendLine("<td>Время нечеткого поиска</td>");
               b.AppendLine("<td>" + textBoxTimeSearchs.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 listBoxResults.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);
        }
    }
  }
}
```

EditDistance.cs

```
using System;
namespace Lab7
{
  internal class EditDistance
  {
    ///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();
      //Объявлениематрицы
      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];
         }
      }
   }
}
```

MinMax.cs

```
namespace Lab7
{
   public class MinMax
   {
      public int Min { get; set; }
```

```
        public int Max { get; set; }
        public MinMax(int pmin, int pmax)
        { Min = pmin; Max = pmax; }
    }
}
```

ParallelSearchResult

```
namespace Lab7
{
    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; }}
    }
```

ParallelSearchThreadParam

```
using System.Collections.Generic;
namespace Lab7
{
    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>
```

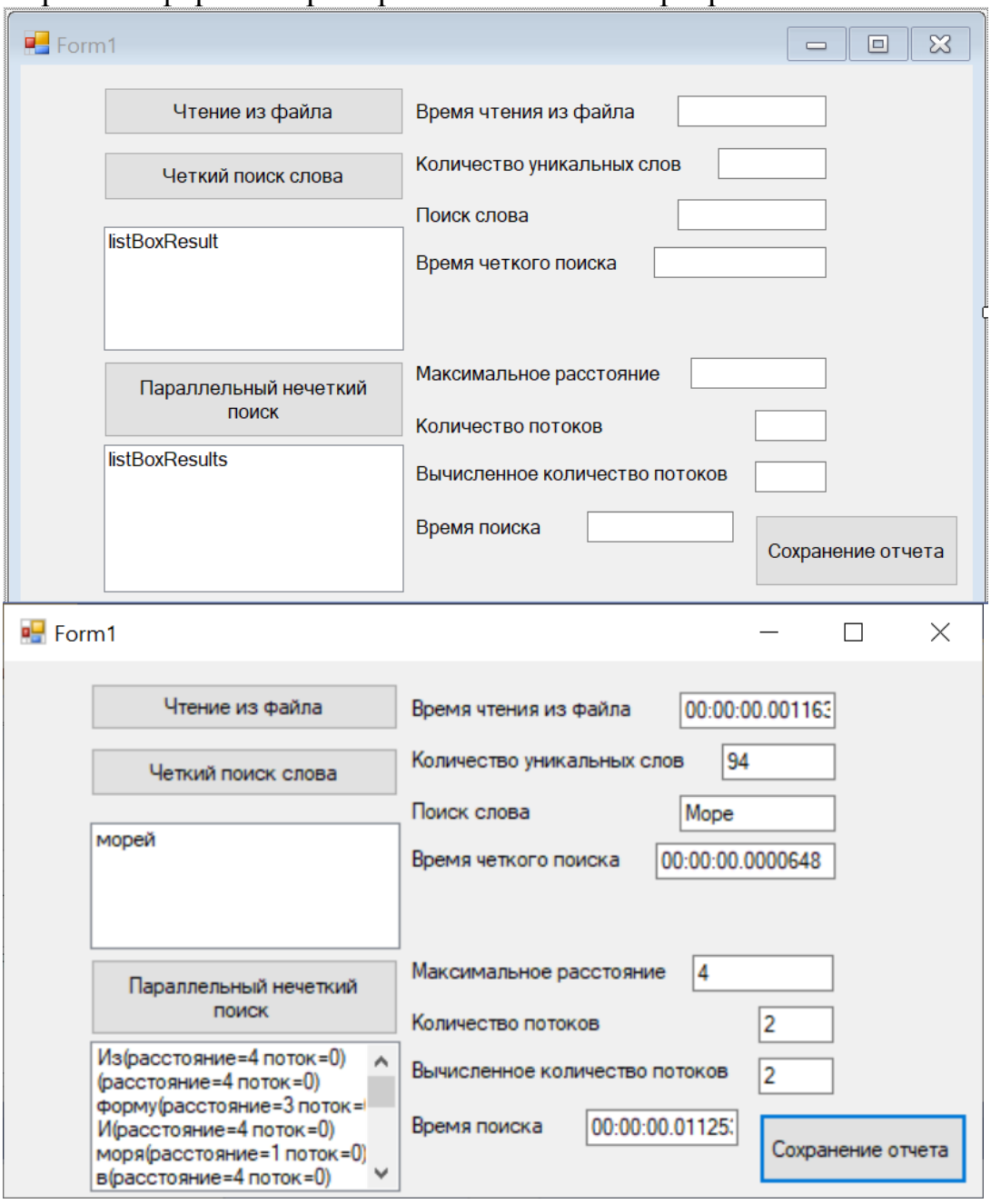
```
            public int maxDist { get; set; }
            ///<summary>
            ///Номер потока
            ///</summary>
            public int ThreadNum { get; set; }
        }
    }
```

SubArrays.cs

```
using System;
using System.Collections.Generic;
namespace Lab7
{
   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;
    }
  }
}
```

Экранные формы с примерами выполнения программы:



Отчет: C:\Users\maxim\OneDrive\Рабочий стол\Report_29_12_2020_073920.html

Время чтения из файла	00:00:00.0000648
Количество уникальных слов в файле	94
Слово для поиска	Море
Максимальное расстояние для нечеткого поиска	4
Время четкого поиска	00:00:00.0000648
Время нечеткого поиска	00:00:00.0112532
Результаты поиска	• Из(расстояние=4 поток=0) • (расстояние=4 поток=0) • форму(расстояние=3 поток=0) • И(расстояние=4 поток=0) • моря(расстояние=1 поток=0) • в(расстояние=4 поток=0) • шторм(расстояние=3 поток=0) • Знак(расстояние=4 поток=0) • мощи(расстояние=2 поток=0) • силы(расстояние=4 поток=0) • якорь(расстояние=3 поток=0) • свой(расстояние=4 поток=0) • курс(расстояние=3 поток=0) • зверь(расстояние=4 поток=0) • король(расстояние=4 поток=0) • Он(расстояние=3 поток=0) • был(расстояние=4 поток=0) • семи(расстояние=4 поток=0) • морей(расстояние=1 поток=0) • войны(расстояние=4 поток=0)