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

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по домашнему заданию «Разработать программу, реализующую многопоточный поиск в файле»

Выполнил:
студент группы ИУ5-31Б
Алехин Сергей
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Осликов С.П.
Подпись и дата:

г. Москва, 2019 г.

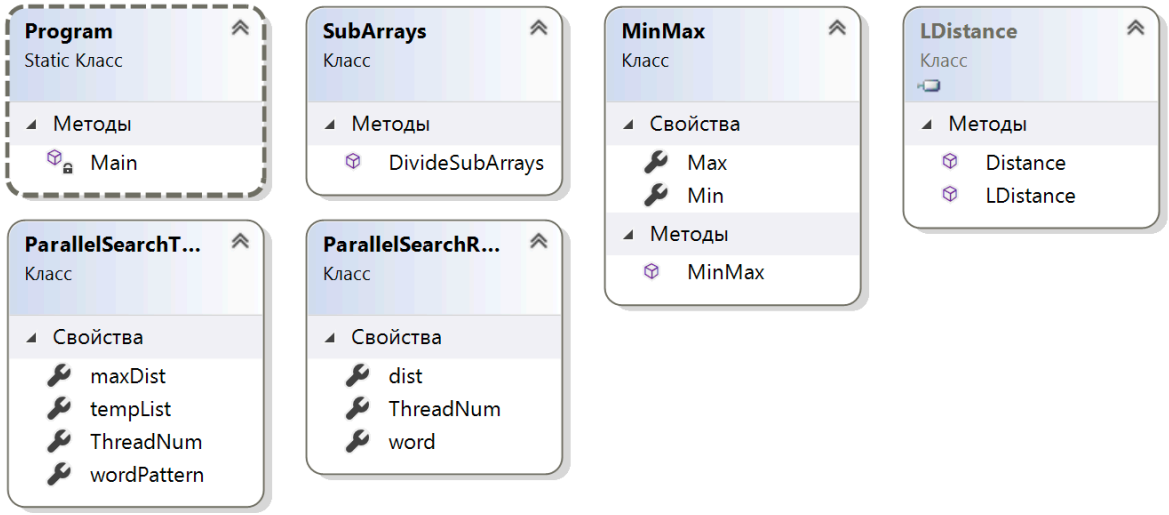
Задание лабораторной работы

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

MinMax.cs

```
namespace DZ
{
   class MinMax
   {
      public int Min { get; set; }
      public int Max { get; set; }
      public MinMax(int pmin, int pmax)
      {
         this.Min = pmin;
         this.Max = pmax;
      }
   }
}
```

SubArrays.cs

```
using System.Collections.Generic;
namespace DZ
{
   class SubArrays
   {
      public static List<MinMax> DivideSubArrays(int beginIndex, int endIndex, int subArraysCount)
      {
         List<MinMax> result = new List<MinMax>();
         if ((endIndex - beginIndex) <= subArraysCount)
         {
            result.Add(new MinMax(0, (endIndex - beginIndex)));
```

```
            }
            else
            {
                int delta = (endIndex - beginIndex) / subArraysCount;
                int currentBegin = beginIndex;
                while ((endIndex - currentBegin) >= 2 * delta)
                {
                    result.Add(new MinMax(currentBegin, currentBegin + delta));
                    currentBegin += delta;
                }
                result.Add(new MinMax(currentBegin, endIndex));
            }
            return result;
        }
    }
}
```

ParallelSearchResult.cs

```
namespace DZ
{
    public class ParallelSearchResult
    {
        public string word { get; set; }
        public int dist { get; set; }
        public int ThreadNum { get; set; }
    }
}
```

ParallelSearchThreadParam.cs

```
using System.Collections.Generic;
namespace DZ
{
    class ParallelSearchThreadParam
    {
        public List<string> tempList { get; set; }
        public string wordPattern { get; set; }
        public int maxDist { get; set; }
        public int ThreadNum { get; set; }
    }
}
```

Form1.cs

```
using LevenDistance;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace DZ
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        public List<string> List = new List<string>();
        private void ReadFromFile_Click(object sender, EventArgs e)
        {
            var currentFileDialog = new OpenFileDialog
```

```
            {
                InitialDirectory = Directory.GetCurrentDirectory(),
                Filter = "Текстовые файлы (*.txt)|*.txt"
            };
            currentFileDialog.ShowDialog();
            if (currentFileDialog.FileName == "")
            {
                MessageBox.Show("Необходимо выбрать файл!");
                return;
            }
            var downloadTime = new Stopwatch();
            downloadTime.Start();
            string text = File.ReadAllText(currentFileDialog.FileName, Encoding.GetEncoding(1251));
            foreach (var currentWord in text.Split())
            {
                if (!List.Contains(currentWord))
                {
                    List.Add(currentWord);
                }
            }
            MessageBox.Show("Чтение файла завершено");
            downloadTime.Stop();
            downloadTimeLabel.Text = downloadTime.Elapsed.TotalMilliseconds.ToString();
            numberOfUniqueWordsLabel.Text = List.Count.ToString();
        }
        private void FindWordButton_Click(object sender, EventArgs e)
        {
            if (!string.IsNullOrWhiteSpace(findWordTextBox.Text))
            {
                var findTime = new Stopwatch();
                findTime.Start();
                List<string> currentList = new List<string>();
                foreach (var currentWord in List)
                {
                    if (currentWord.Contains(findWordTextBox.Text))
                    {
                        currentList.Add(currentWord);
                    }
                }
                findTime.Stop();
                findWordTimeLabel.Text = findTime.Elapsed.TotalMilliseconds.ToString();
                listBox1.BeginUpdate();
                listBox1.Items.Clear();
                foreach (string str in currentList)
                {
                    listBox1.Items.Add(str);
                }
                listBox1.EndUpdate();
            }
            else
            {
                MessageBox.Show("Введите слово для поиска");
            }
        }
        private void findLevenDistanceButton_Click(object sender, EventArgs e)
        {
            List<string> currentList = new List<string>();

            int maxDistance;
            int threadCount;
            string word = findWordTextBox.Text.Trim();
            if ((int.TryParse(maxDistanceTextBox.Text, out maxDistance)) &&
(int.TryParse(threadNumberTextBox.Text, out threadCount)) && (!string.IsNullOrWhiteSpace(word)))
```

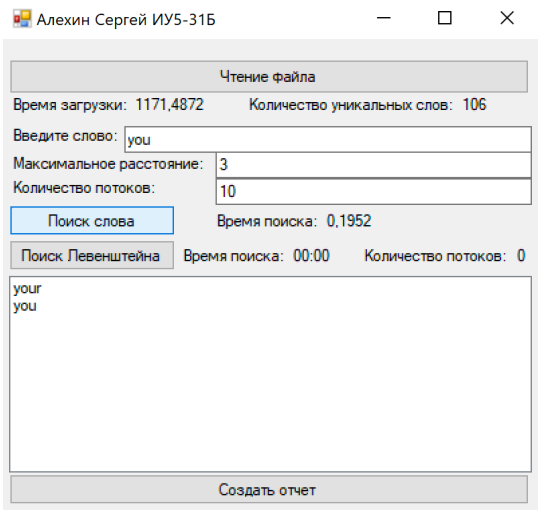
```
        {
            var findTime = new Stopwatch();
            findTime.Start();
            List<ParallelSearchResult> Result = new List<ParallelSearchResult>();
            List<MinMax> arrayDivList = SubArrays.DivideSubArrays(0, List.Count, threadCount);
            int count = arrayDivList.Count;
            Task<List<ParallelSearchResult>>[] tasks = new Task<List<ParallelSearchResult>>[count];
            for (int i = 0; i < count; i++)
            {
                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 = maxDistance,
                        ThreadNum = i,
                        wordPattern = word
                    });
                tasks[i].Start();
            }
            Task.WaitAll(tasks);
            for (int i = 0; i < count; i++)
            {
                Result.AddRange(tasks[i].Result);
            }
            listBox1.BeginUpdate();
            listBox1.Items.Clear();
            foreach (var x in Result)
            {
                string temp = x.word + "(расстояние=" + x.dist.ToString() + " поток=" + x.ThreadNum.ToString() +
")";
                listBox1.Items.Add(temp);
            }
            listBox1.EndUpdate();
            findTime.Stop();
            findLevenDistanceTimeLabel.Text = findTime.Elapsed.TotalMilliseconds.ToString();
            numberOfThreadLabel.Text = count.ToString();
        }
        else
        {
            MessageBox.Show("Введите слово или количество потоков или максимально расстояние");
        }
    }
    public static List<ParallelSearchResult> ArrayThreadTask(object paramObj)
    {
        ParallelSearchThreadParam param = (ParallelSearchThreadParam)paramObj;
        string wordUpper = param.wordPattern.Trim().ToUpper();
        List<ParallelSearchResult> Result = new List<ParallelSearchResult>();
        foreach (string str in param.tempList)
        {
            int dist = LDistance.Distance(str.ToUpper(), wordUpper);
            if (dist <= param.maxDist)
            {
                ParallelSearchResult temp = new ParallelSearchResult()
                {
                    word = str,
                    dist = dist,
                    ThreadNum = param.ThreadNum
                };
                Result.Add(temp);
            }
```

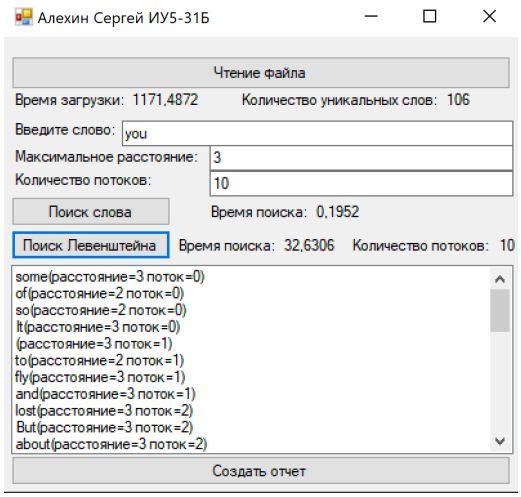
```
            }
            return Result;
        }
        private void createReportButton_Click(object sender, EventArgs e)
        {
            string TempReportFileName = "Report_" + DateTime.Now.ToString("dd_MM_yyyy_hhmmss");
            SaveFileDialog fd = new SaveFileDialog();
            fd.FileName = TempReportFileName;
            fd.Filter = "Text files|*.txt|HTML Reports|*.html;";
            if (fd.ShowDialog() == DialogResult.OK)
            {
                string ReportFileName = fd.FileName;
                if (Path.GetExtension(fd.FileName) == ".txt")
                {
                    StreamWriter sw = new StreamWriter(fd.FileName);
                    sw.WriteLine("Отчет: " + ReportFileName);
                    sw.WriteLine("Время чтения из файла: " + this.downloadTimeLabel.Text);
                    sw.WriteLine("Количество уникальных слов в файле: " + this.numberOfUniqueWordsLabel.Text);
                    sw.WriteLine("Слово для поиска: " + this.findWordTextBox.Text);
                    sw.WriteLine("Максимальное расстояние для нечеткого поиска: " +
this.maxDistanceTextBox.Text);
                    sw.WriteLine("Время четкого поиска: " + this.findWordTimeLabel.Text);
                    sw.WriteLine("Время нечеткого поиска: " + this.findLevenDistanceTimeLabel.Text);
                    sw.WriteLine("Результаты поиска: ");
                    foreach (var x in this.listBox1.Items)
                    {
                        sw.WriteLine(x.ToString());
                    }
                    sw.Close();
                }
                else
                {
                    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.downloadTimeLabel.Text + "</td>");
                    b.AppendLine("</tr>");
                    b.AppendLine("<tr>");
                    b.AppendLine("<td>Количество уникальных слов в файле</td>");
                    b.AppendLine("<td>" + this.numberOfUniqueWordsLabel.Text + "</td>");
                    b.AppendLine("</tr>");
                    b.AppendLine("<tr>");
                    b.AppendLine("<td>Слово для поиска</td>");
                    b.AppendLine("<td>" + this.findWordTextBox.Text + "</td>");
                    b.AppendLine("</tr>");
                    b.AppendLine("<tr>");
                    b.AppendLine("<td>Максимальное расстояние для нечеткого поиска</td>");
                    b.AppendLine("<td>" + this.maxDistanceTextBox.Text + "</td>");
                    b.AppendLine("</tr>");
                    b.AppendLine("<tr>");
                    b.AppendLine("<td>Время четкого поиска</td>");
                    b.AppendLine("<td>" + this.findWordTimeLabel.Text + "</td>");
                    b.AppendLine("</tr>");

                    b.AppendLine("<tr>");
```

```
                b.AppendLine("<td>Время нечеткого поиска</td>");
                b.AppendLine("<td>" + this.findLevenDistanceTimeLabel.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.listBox1.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>");
                File.AppendAllText(ReportFileName, b.ToString());
            }
            MessageBox.Show("Отчет успешно сформирован. Файл: " + ReportFileName);
        }
    }
  }
}
```

Примеры работы программы





Ссылка на репозиторий GitHub

https://github.com/SergeyBMSTU2018/AlekhinSergeyLabs.git