Московский государственный технический

университет им. Н.Э. Баумана.

Факультет «Информатика и системы управления»

Кафедра ИУ5. Курс «Базовые компоненты интернет технологий»

Отчет по домашнему заданию

«Многопоточный поиск в файле»

Выполнил:
студент группы ИУ5-33
Терентьев Владислав

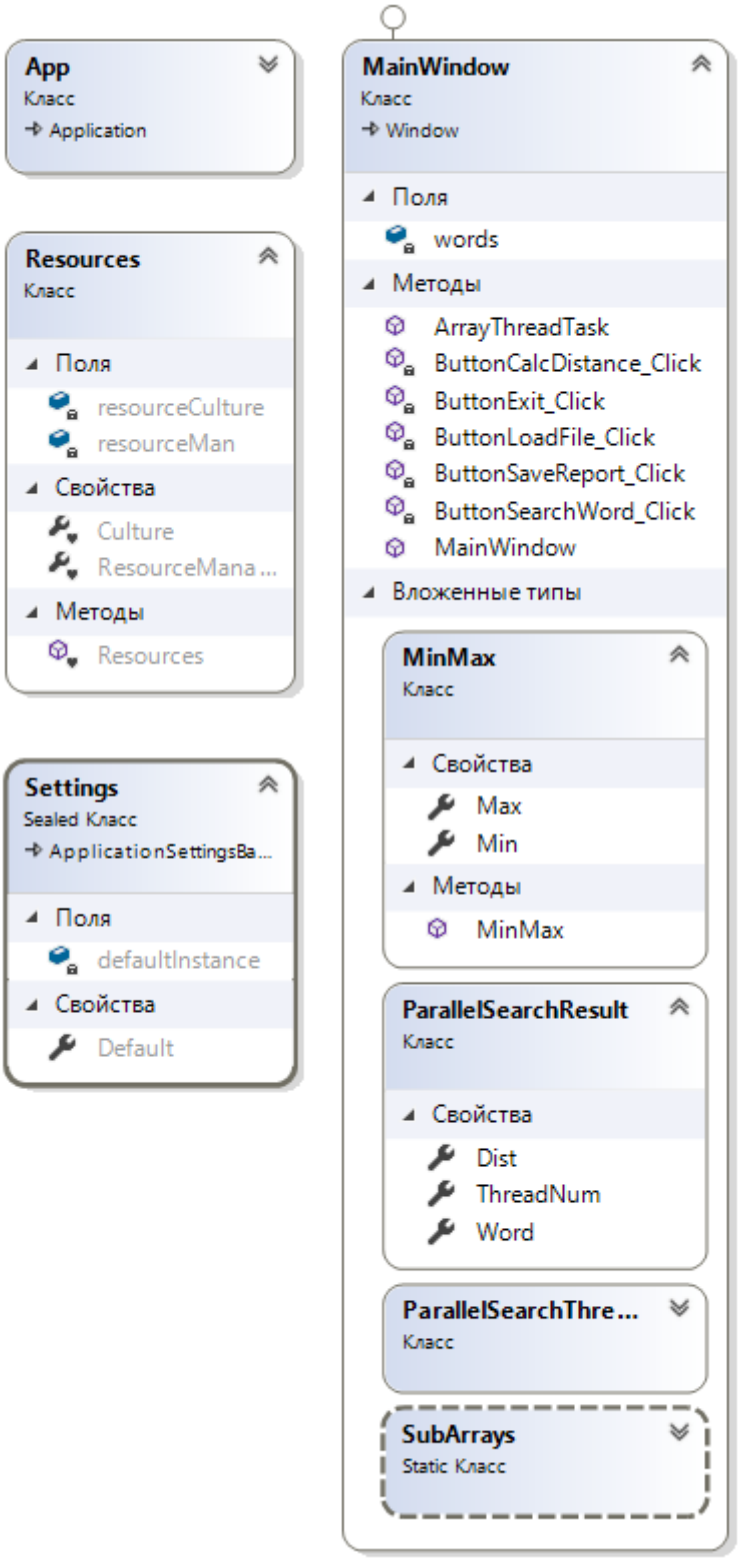
Москва, 2019 г.

1. Постановка задачи

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

2. Диаграмма классов



3. Текст программы

3.1. MainWindow.xaml

```
<Window x:Class="homework.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d"
        Title="Терентьев Владислав ИУ5-33" Height="450" Width="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="25"></RowDefinition>
            <RowDefinition Height="25"></RowDefinition>
            <RowDefinition Height="35"></RowDefinition>
            <RowDefinition Height="50"></RowDefinition>
            <RowDefinition Height="25"></RowDefinition>
            <RowDefinition Height="25"></RowDefinition>
            <RowDefinition Height="25"></RowDefinition>
            <RowDefinition Height="25"></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition Height="Auto"></RowDefinition>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="200"></ColumnDefinition>
            <ColumnDefinition Width="Auto"></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
        </Grid.ColumnDefinitions>
        <Button x:Name="ButtonLoadFile" Content="Чтение из файла" Grid.Column="0"
Grid.Row="0" Grid.RowSpan="2" HorizontalAlignment="Center" VerticalAlignment="Center"
Width="120" Height="30" Click="ButtonLoadFile_Click"/>
        <Label x:Name="LabelCountWords" Grid.Column="2" Grid.Row="1"
HorizontalAlignment="Left" VerticalAlignment="Center"/>
        <TextBox x:Name="TextBoxCurrentWord" TextWrapping="Wrap" Grid.Column="1"
Grid.Row="2" Grid.ColumnSpan="2" Width="280" Height="20" HorizontalAlignment="Left"
VerticalAlignment="Center"/>
        <Button x:Name="ButtonSearchWord" Content="Четкий поиск" Grid.Column="0"
Grid.Row="3" HorizontalAlignment="Center" VerticalAlignment="Center" Width="120"
Height="30" Click="ButtonSearchWord_Click"/>
        <ListBox x:Name="ListBoxResult" Grid.Column="0" Grid.Row="8" Grid.ColumnSpan="3"
Margin="10,10,10,0"/>
        <Label x:Name="LabelTimeSearch" Grid.Column="2" Grid.Row="3"
HorizontalAlignment="Left" VerticalAlignment="Center"/>
        <Label x:Name="LabelTimer" Grid.Column="2" Grid.Row="0"
HorizontalAlignment="Left" VerticalAlignment="Center"/>
        <TextBox x:Name="TextBoxMaxDistance" TextWrapping="Wrap" Grid.Column="2"
Grid.Row="4" Width="35" HorizontalAlignment="Left" VerticalAlignment="Center"/>
        <Button x:Name="ButtonCalcDistance" Grid.Column="0" Grid.Row="4" Grid.RowSpan="4"
HorizontalAlignment="Center" VerticalAlignment="Center" Width="120" Height="60"
Click="ButtonCalcDistance_Click">
            <TextBlock>Параллельный<LineBreak/>нечеткий поиск</TextBlock>
        </Button>
        <Label x:Name="LabelTimeCalc" Grid.Column="2" Grid.Row="7"
HorizontalAlignment="Left" VerticalAlignment="Center"/>
        <TextBox x:Name="TextBoxThreadCount" TextWrapping="Wrap" Grid.Column="2"
Grid.Row="5" Width="70" HorizontalAlignment="Left" VerticalAlignment="Center"/>
        <Label x:Name="LabelThreadCount" Grid.Column="2" Grid.Row="6"
HorizontalAlignment="Left" VerticalAlignment="Center"/>
        <Label Content="Время чтения из файла: " Grid.Column="1" Grid.Row="0"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
        <Label Content="Количество уникальных слов в файле: " Grid.Column="1"
Grid.Row="1" HorizontalAlignment="Center" VerticalAlignment="Center"/>
        <Label Content="Слово для поиска: " Grid.Column="0" Grid.Row="2"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
        <Label Content="Время нечеткого поиска: " Grid.Column="1" Grid.Row="7"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
        <Label Content="Время четкого поиска: " Grid.Column="1" Grid.Row="3"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
        <Label Content="Максимальное расстояние для нечеткого поиска: " Grid.Column="1"
Grid.Row="4" HorizontalAlignment="Center" VerticalAlignment="Center"/>
```

```
        <Label Content="Количество потоков: " Grid.Column="1" Grid.Row="5"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
        <Label Content="Вычисленное количество потоков: " Grid.Column="1" Grid.Row="6"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
        <Button x:Name="ButtonSaveReport" Content="Сохранение отчета"
HorizontalAlignment="Left" Grid.Row="9" VerticalAlignment="Bottom" Width="125"
Margin="10" Height="25" Click="ButtonSaveReport_Click"/>
        <Button x:Name="ButtonExit" Content="Выход" HorizontalAlignment="Right"
Grid.Column="2" Grid.Row="9" VerticalAlignment="Bottom" Width="125" Margin="10"
Height="25" Click="ButtonExit_Click"/>
    </Grid>
</Window>
```

3.2. MainWindow.xaml.cs

```
using System.Collections.Generic;
using System.Threading.Tasks;
using System.Windows;
using Microsoft.Win32;
using System.Diagnostics;
using System.IO;
using laba5_lib;
using System.Text;
using System;

namespace homework
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        public class ParallelSearchResult
        {
            public string Word { get; set; }

            public int Dist { get; set; }

            public int ThreadNum { get; set; }
        }

        public class MinMax
        {
            public int Min { get; set; }
            public int Max { get; set; }

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

        public static 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;
        }
    }

    class ParallelSearchThreadParam
    {
        public List<string> TempList { get; set; }

        public string WordPattern { get; set; }

        public int MaxDist { get; set; }

        public int ThreadNum { get; set; }
    }

    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 = LevenshteinDistance.Distance(str.ToUpper(), wordUpper);
            if (dist <= param.MaxDist)
            {
                ParallelSearchResult temp = new ParallelSearchResult()
                {
                    Word = str,
                    Dist = dist,
                    ThreadNum = param.ThreadNum
                };

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

    readonly List<string> words = new List<string>();

    private void ButtonLoadFile_Click(object sender, RoutedEventArgs e)
    {
        OpenFileDialog fd = new OpenFileDialog
        {
            Filter = "Текстовые Файлы|*.txt"
        };
        bool? result = fd.ShowDialog();
        if (result == true)
        {
            Stopwatch timer = new Stopwatch();
```

```
                timer.Start();
                string text = File.ReadAllText(fd.FileName);
                char[] separators = new char[] { ' ', '.', ',', '?', '!', '\"', '<', '>',
'/', '\t', '\n' };
                string[] textArray = text.Split(separators);
                foreach (string strTemp in textArray)
                {
                    string str = strTemp.Trim();
                    if (str != "")
                    {
                        if (!words.Contains(str)) words.Add(str);
                    }
                }
                timer.Stop();
                LabelTimer.Content = timer.Elapsed.ToString();
                LabelCountWords.Content = words.Count.ToString();
            }
            else
            {
                MessageBox.Show("Необходимо выбрать файл");
            }
        }

        private void ButtonSearchWord_Click(object sender, RoutedEventArgs e)
        {
            string wordSearch = TextBoxCurrentWord.Text.Trim();
            if (!string.IsNullOrWhiteSpace(wordSearch) && words.Count > 0)
            {
                string wordUpper = wordSearch.ToUpper();
                List<string> tempList = new List<string>();
                Stopwatch timer = new Stopwatch();
                timer.Start();
                foreach (string str in words)
                {
                    if (str.ToUpper().Contains(wordUpper))
                    {
                        tempList.Add(str);
                    }
                }
                timer.Stop();
                LabelTimeSearch.Content = timer.Elapsed.ToString();
                ListBoxResult.ItemsSource = tempList;
            }
            else
            {
                MessageBox.Show("Выберите файл и введите слово для поиска");
            }
        }

        private void ButtonCalcDistance_Click(object sender, RoutedEventArgs e)
        {
            string currentWord = TextBoxCurrentWord.Text.Trim();
            if (!string.IsNullOrWhiteSpace(currentWord) && words.Count > 0)
            {
                if (!int.TryParse(TextBoxMaxDistance.Text.Trim(), out int maxDistance))
                {
                    MessageBox.Show("Введите максимальное расстояние");
                    return;
                }
                if (maxDistance < 1 || maxDistance > 5)
                {
                    MessageBox.Show("Максимальное расстояние должно быть в диапазоне от 1
до 5");
                    return;
                }
```

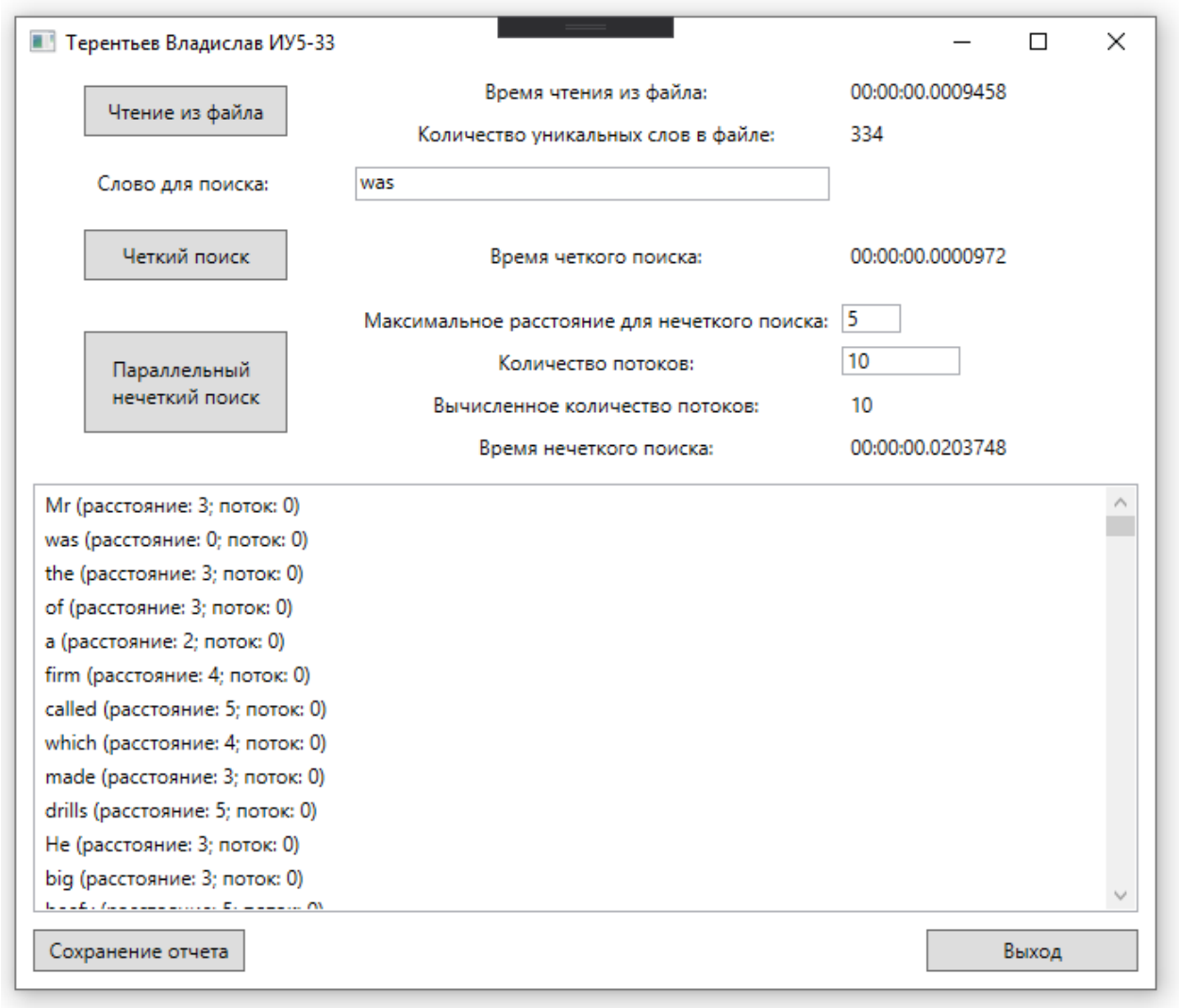
```
                if (!int.TryParse(TextBoxThreadCount.Text.Trim(), out int ThreadCount))
                {
                    MessageBox.Show("Необходимо указать количество потоков");
                    return;
                }
                Stopwatch timer = new Stopwatch();
                timer.Start();
                List<ParallelSearchResult> Result = new List<ParallelSearchResult>();
                List<MinMax> arrayDivList = SubArrays.DivideSubArrays(0, words.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 = words.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 = currentWord
                        });
                    tasks[i].Start();
                }
                Task.WaitAll(tasks);
                timer.Stop();
                for (int i = 0; i < count; i++)
                {
                    Result.AddRange(tasks[i].Result);
                }
                timer.Stop();
                LabelTimeCalc.Content = timer.Elapsed.ToString();
                LabelThreadCount.Content = count.ToString();
                List<string> tempList = new List<string>();
                foreach (var x in Result)
                {
                    string temp = x.Word + " (расстояние: " + x.Dist.ToString() + ";
поток: " + x.ThreadNum.ToString() + ")";
                    tempList.Add(temp);
                }
                LabelTimeCalc.Content = timer.Elapsed.ToString();
                ListBoxResult.ItemsSource = tempList;
            }
            else
            {
                MessageBox.Show("Выберите файл и введите слово для поиска");
            }
        }

        private void ButtonSaveReport_Click(object sender, RoutedEventArgs e)
        {
            string TempReportFileName = "Report_" +
DateTime.Now.ToString("dd_MM_yyyy_hhmmss");
            SaveFileDialog fd = new SaveFileDialog
            {
                FileName = TempReportFileName,
                DefaultExt = ".html",
                Filter = "HTML Reports|*.html"
            };
            bool? result = fd.ShowDialog();
            if (result == true)
            {
```

```
                string ReportFileName = fd.FileName;
                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>" + LabelTimer.Content + "</td>");
                b.AppendLine("</tr>");
                b.AppendLine("<tr>");
                b.AppendLine("<td>Количество уникальных слов в файле</td>");
                b.AppendLine("<td>" + LabelCountWords.Content + "</td>");
                b.AppendLine("</tr>");
                b.AppendLine("<tr>");
                b.AppendLine("<td>Слово для поиска</td>");
                b.AppendLine("<td>" + TextBoxCurrentWord.Text + "</td>");
                b.AppendLine("</tr>");
                b.AppendLine("<tr>");
                b.AppendLine("<td>Максимальное расстояние для нечеткого поиска</td>");
                b.AppendLine("<td>" + TextBoxMaxDistance.Text + "</td>");
                b.AppendLine("</tr>");
                b.AppendLine("<tr>");
                b.AppendLine("<td>Время четкого поиска</td>");
                b.AppendLine("<td>" + LabelTimeSearch.Content + "</td>");
                b.AppendLine("</tr>");
                b.AppendLine("<tr>");
                b.AppendLine("<td>Время нечеткого поиска</td>");
                b.AppendLine("<td>" + LabelTimeCalc.Content + "</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 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>");
                File.AppendAllText(ReportFileName, b.ToString());
                MessageBox.Show("Отчет сформирован. Файл: " + ReportFileName);
            }
        }

        private void ButtonExit_Click(object sender, RoutedEventArgs e)
        {
            Close();
        }
    }
}
```

4. Анализ результатов



5. Ссылка на репозиторий

https://github.com/iYroglif/newlabs