Отчет по домашнему заданию по курсу C#

8

(количество листов)

Студент группы ИУ5-32

Яценко Антон

Дата: 06.12.2017

Руководитель:

Гапанюк Ю.Е.

Подпись:

Дата:

Домашнее задание

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

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.

Текст программы:

Form1:

```
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 ab5library;
using System.Threading.Tasks;

namespace lab4.csh
{

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

        private void buttonLoadFile_Click(object sender, EventArgs e)
        {
            OpenFileDialog fd = new OpenFileDialog();
            fd.Filter = "текстовые файлы|*.txt";
            if (fd.ShowDialog() == DialogResult.OK)
            {
                Stopwatch t = new Stopwatch();
                t.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 (!listBoxResult.Items.Contains(str)) listBoxResult.Items.Add(str);
                }
                t.Stop();
                this.textBoxFileReadTime.Text = t.Elapsed.ToString();
                this.textBoxFileReadCount.Text = listBoxResult.Items.Count.ToString();
            }
            else
            {
                MessageBox.Show("Необходимо выбрать файл");
            }
        }

        private void buttonExact_Click(object sender, EventArgs e)
        {
            string word = this.textBoxFind.Text.Trim();
            if (!string.IsNullOrWhiteSpace(word) && listBoxResult.Items.Count > 0)
            {
                string wordUpper = word.ToUpper();
                List<string> tempList = new List<string>();
                Stopwatch t = new Stopwatch();
                t.Start();
                foreach (string str in listBoxResult.Items)
                {
                    if (str.ToUpper().Contains(wordUpper))
                    {
                        tempList.Add(str);
                    }
                }
                t.Stop();
                this.textBoxExactTime.Text = t.Elapsed.ToString();
                this.listBoxResult.BeginUpdate();
                this.listBoxResult.Items.Clear();
                foreach (string str in tempList)
                {
                    this.listBoxResult.Items.Add(str);
                }
                this.listBoxResult.EndUpdate();
            }
            else
            {
                MessageBox.Show("Необходимо выбрать файл и ввести слово для поиска");
            }
        }


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

        private void buttonSaveReport_Click(object sender, EventArgs e)
        {
            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);
            }

        }

        private void buttonApprox_Click(object sender, EventArgs e)
        {
            string word = this.textBoxFind.Text.Trim();
            if (!string.IsNullOrWhiteSpace(word) && listBoxResult.Items.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.textBoxThreads.Text.Trim(), out ThreadCount))
                {
                    MessageBox.Show("Необходимо указать количество потоков"); return;
                }

                string wordUpper = word.ToUpper();
                List<Tuple<string, int>> tempList = new List<Tuple<string, int>>();

                Stopwatch t = new Stopwatch();
                t.Start();

                List<ParallelSearchRes> Result = new List<ParallelSearchRes>();
                List<MinMax> arrayDivList = SubArrays.DivideSubArrays(0,
listBoxResult.Items.Count, ThreadCount);
                int count = arrayDivList.Count;
                Task<List<ParallelSearchRes>>[] tasks = new
Task<List<ParallelSearchRes>>[count];

                for (int i = 0; i < count; i++)
                {
                    List<string> tempTaskList = new List<string>();
                    for (int i1=arrayDivList[i].Min;i1<=arrayDivList[i].Max -
arrayDivList[i].Min; i1++ )
                    {
                        tempTaskList.Add(listBoxResult.Items[i1].ToString());
                    }
                    tasks[i] = new Task<List<ParallelSearchRes>>(
Program.ArrayThreadTask, new ParallelSearchThreadParam() { tempList = tempTaskList,
maxDist = maxDist, ThreadNum = i, wordPattern = word });
                    tasks[i].Start();
                }
                Task.WaitAll(tasks);

                /*foreach (string str in listBoxResult.Items)
                {
                    //Вычисление расстояния Дамерау-Левенштейна
                    int dist = ab5library.EditDistanse.Distance(str.ToUpper(),
wordUpper);
                    //Если расстояние меньше порогового, то слово добавляется в результат
                    if (dist <= maxDist)
                    {
                        tempList.Add(new Tuple<string, int>(str, dist));
                    }
                }*/

                t.Stop();
                for (int i = 0; i < count; i++) { Result.AddRange(tasks[i].Result); }

                this.textBoxApproxTime.Text = t.Elapsed.ToString();
                this.listBoxResult.BeginUpdate();
                this.listBoxResult.Items.Clear();
                foreach (var x in Result)
                {
```

```
                    string temp = x.word + "(расстояние=" + x.dist.ToString() + " поток="
+ x.ThreadNum.ToString() + ")";
                    this.listBoxResult.Items.Add(temp);
                }
                /*foreach (var x in tempList)
                {
                    string temp = x.Item1 + "(расстояние=" + x.Item2.ToString() + ")";
                    this.listBoxResult.Items.Add(temp);
                }*/
                this.listBoxResult.EndUpdate();
            }
            else
            {
                MessageBox.Show("Необходимо выбрать файл и ввести слово для поиска");
            }
        }

    }
}
```

SubArrays:

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

namespace lab4.csh
{
    class SubArrays // для разделения массива на подмассивы
    {
        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;
        }

    }
}
```

Program:

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Threading.Tasks;

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

        public static List<ParallelSearchRes> ArrayThreadTask(object paramObj) //
выполняется в параллельном потоке для поиска строк
        {
            ParallelSearchThreadParam param = (ParallelSearchThreadParam)paramObj;
            string wordUpper = param.wordPattern.Trim().ToUpper(); // слово для поиска в
верхнем регистре
            List<ParallelSearchRes> Result = new List<ParallelSearchRes>(); // результаты
поиска в одном потоке
            foreach (string str in param.tempList) // перебор всех слов во временном
списке данного потока
            {
                int dist = ab5library.EditDistanse.Distance(str.ToUpper(), wordUpper); //
вычисление расстояния Дамерау-Левенштейна
                if (dist <= param.maxDist) // условие добавления слова в результат
                {
                    ParallelSearchRes temp = new ParallelSearchRes() { word = str, dist =
dist, ThreadNum = param.ThreadNum };
                    Result.Add(temp);
                }
            }
            return Result;
        }
    }
}
```

ParallelReseachParam:

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

namespace lab4.csh
{
    class ParallelSearchThreadParam // парамаетры которые передаются в поток для
параллельного поиска
    {
        public List<string> tempList { get; set; } // массив для поиска
        public string wordPattern { get; set; } // слово для поиска
        public int maxDist { get; set; } // максимальное расстояние для нечеткого поиска
        public int ThreadNum { get; set; } // номер потока
    }
}
```

ParallelReseachRes:

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

namespace lab4.csh
{
    class ParallelSearchRes // для хранения информации о найденных словах
    {
        public string word { get; set; } // найденное слово
        public int dist { get; set; } // расстояние Дамерау-Левенштейна
        public int ThreadNum { get; set; } // номер потока
    }
}
```

MinMax:

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

namespace lab4.csh
{
    class MinMax
    {
        public int Min { get; set; }
        public int Max { get; set; }
        public MinMax(int pmin, int pmax) { this.Min = pmin; this.Max = pmax; }
    }
}
```

Экранные формы: