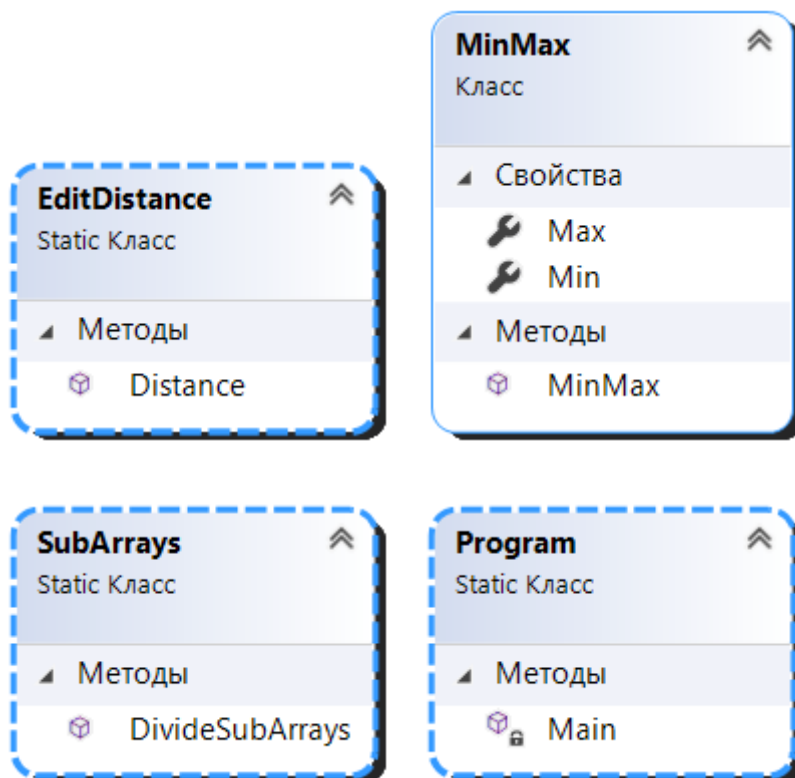


Отчёт по домашнему заданию

Волгина А. Д., ИУ5-33Б

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#».



Form1

Класс

→ Form

Поля

buttonApprox

buttonExact

buttonExit

buttonLoadFile

buttonSaveRep...

components

label1

label2

label3

label4

label5

label6

label7

label8

list

listBoxResult

textBoxApprox...

textBoxExactTi...

textBoxFileRea...

textBoxFileRea...

textBoxFind

textBoxMaxDist

textBoxThread...

textBoxThread...

timer1

Методы

ArrayThreadTask

buttonApprox_...

buttonExact_Cli...

buttonExit_Click

buttonLoadFile...

buttonSaveRep...

Dispose

Form1

InitializeCompo...

label3_Click

label4_Click

label5_Click

label6_Click

listBoxResult_S...

ParallelSearchR...

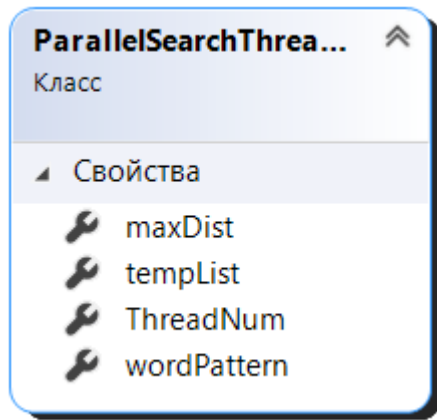
Класс

Свойства

dist

ThreadNum

word



Program.cs:

```
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());  
    }  
}  
}
```

Form1.cs:

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

```
namespace Lab4
```

```
{  
    public partial class Form1 : Form  
    {  
        public Form1()  
        {  
            InitializeComponent();  
        }  
        List<string> list = new List<string>();  
    }  
}
```

```

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();

        //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.Stop();
        this.textBoxFileReadTime.Text = t.Elapsed.ToString();
    }
}

```

```

        this.textBoxFileReadCount.Text = list.Count.ToString();
    }
    else
    {
        MessageBox.Show("Необходимо выбрать файл");
    }

}

private void label3_Click(object sender, EventArgs e)
{

}

private void buttonExact_Click(object sender, EventArgs e)
{
    //Слово для поиска
    string word = this.textBoxFind.Text.Trim();

    //Если слово для поиска не пусто
    if (!string.IsNullOrEmpty(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 = new Stopwatch();

t.Start();

foreach (string str in list)
{
    if (str.ToUpper().Contains(wordUpper))
    {
        tempList.Add(str);
    }
}

t.Stop();

this.textBoxExactTime.Text = t.Elapsed.ToString();

this.listBoxResult.BeginUpdate();

//Очистка списка

this.listBoxResult.Items.Clear();

//Вывод результатов поиска

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 label4_Click(object sender, EventArgs e)
```

```
{}
```

```
private void listBoxResult_SelectedIndexChanged(object sender,  
EventArgs e)
```

```
{
```

```
}
```

```
private void label5_Click(object sender, EventArgs e)
```

```
{
```

```
}
```

```
private void label6_Click(object sender, EventArgs e)
```

```
{
```

```
}
```

```
private void buttonApprox_Click(object sender, EventArgs e)
```

```
{
```

```
    //Слово для поиска
```

```
    string word = this.textBoxFind.Text.Trim();
```

```
    //Если слово для поиска не пусто
```

```

if (!string.IsNullOrEmpty(word) && list.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.textBoxThreadCount.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 с одной коллекцией

    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 = 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
this.textBoxApproxTime.Text = timer.Elapsed.ToString();

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

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

//Очистка списка
this.listBoxResult.Items.Clear();
```

```

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

        //Окончание обновления списка результатов
        this.listBoxResult.EndUpdate();
    }
    else
    {
        MessageBox.Show("Необходимо выбрать файл и ввести слово
для поиска");
    }
}

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 buttonExit_Click(object sender, EventArgs e)
```

```
{
```

```
Application.Exit();  
}
```

```
private void buttonSaveReport_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>" + 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);
    }

}

}

}

Class1.cs:

```

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

namespace Lab4
{
    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;
        }
    }

    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;
}
}

public static class EditDistance
{
    /// <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

```

случаев

```
//Элемент матрицы вычисляется как минимальный из трех
```

```
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];
```

```
}
```

```
}
```

```
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; }
}

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; }
}

```

}

Text.txt:

зима

весна

лето

осень

зимний

зам

весенний день

летний

осенний

веснушки

весна

Form1

Чтение из файла

Время чтения из файла: 00:00:00.0001767

Количество уникальных слов в файле: 11

Слово для поиска: зам

Чёткий поиск

Время чёткого поиска: 00:00:00.0000084

Параллельный нечёткий поиск

Максимальное расстояние для нечёткого поиска: 2

Количество потоков: 5

Вычисленное количество потоков: 5

Время нечёткого поиска: 00:00:00.0073234

зима(расстояние=2 поток=0)
зам(расстояние=0 поток=2)

Сохранить отчёт

Выход

Чтение из файла

Время чтения из файла: 00:00:00.0001713

Количество уникальных слов в файле: 11

Слово для поиска: весна

Чёткий поиск

Время чёткого поиска:

Параллельный нечёткий
поиск

Максимальное расстояние для нечёткого поиска: 1

Количество потоков: 5

Вычисленное количество потоков: 5

Время нечёткого поиска: 00:00:00.0064129

весна(расстояние=1 поток=0)

Сохранить отчёт

Выход