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

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Базовые компоненты интернет-технологий»

Отчет по домашнему заданию

Работу выполнила: Корнеева Анна, ИУ5-34

г. Москва, 2017 г.

# 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

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 System.Threading.Tasks;

namespace ДЗ1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

/// <summary> /// Список слов

/// </summary>

List<string> list = new List<string>();

private void label1\_Click(object sender, EventArgs e)

{

}

private void label3\_Click(object sender, EventArgs e)

{

}

private void label5\_Click(object sender, EventArgs e)

{

}

private void buttonClose\_Click\_1(object sender, EventArgs e)

{

this.Close();

}

private void button1\_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 button2\_Click(object sender, EventArgs e)

{

//Слово для поиска

string word = this.textBoxFind.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 = 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");

}

}

/// <summary>

/// Выполняется в параллельном потоке для поиска строк

/// </summary>

//получает в качестве параметра объект класса

ParallelSearchThreadParam, осуществляет перебор всех слов в массиве для поиска, который был передан данному потоку 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 button3\_Click(object sender, EventArgs e)

{

//Слово для поиска string word = this.textBoxFind.Text.Trim();

//Если слово для поиска не пусто if (!string.IsNullOrWhiteSpace(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"); //потоки, на которые разделяется массив слов исходного файла

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 - класс, используюшийся для параллельного поиска

(задача)

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(); }

//ожидание завершения работы всех потоков, чтобы получить результаты поиска

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");

}

}

private void button4\_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);

}

}

private void label6\_Click(object sender, EventArgs e)

{

}

}

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

//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];

}

}

/// <summary>

/// Результаты параллельного поиска

/// </summary>

//содержит входной массив слов и слово для поиска, максимальное расстояние для нечеткого поиска и номер потока

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; }

}

/// <summary>

/// Параметры которые передаются в поток для параллельного поиска

/// </summary>

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; }

}

/// <summary>

/// Хранение минимального и максимального значений диапазона

/// </summary>

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;

} }

//Для деления массива на подмассивы

public static class SubArrays

{

/// <summary>

/// Деление массива на последовательности(подмассивы)

/// </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

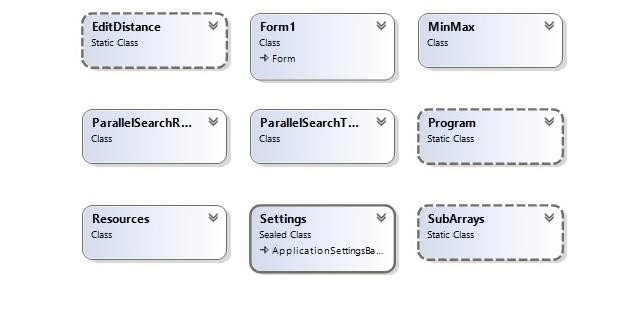
result.Add(new MinMax(currentBegin, endIndex)); }

//Возврат списка результатов return result; }

}

}

# Диаграмма классов



# Результат

