**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ**

**Московский государственный технический университет**

**им. Н.Э. Баумана**

**(МГТУ им. Н.Э. Баумана)**

**Радиотехнический факультет (РТ)**

Домашнее задание

По дисциплине: «Базовые компоненты интернет технологий»

# Тема: «Пример многопоточного поиска в текстовом файле.»

Выполнил: Астанов Э М,

студент группы РТ5-31

Проверил: Гапанюк Ю.Е.,

Преподаватель каф. ИУ5

г. Москва 2017 г.

Описание задания 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.

Текст программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DamerauLivensteinLib

{

public static class DamerauLivenstein

{

public static List<Tuple<string, int>> ThreadDiv(

List<string> list, string word , int ThreadCount, int maxDist)

{

Dictionary<int,List<string>> DividedList = new Dictionary<int,List<string>>();

int WordsInThread = list.Count / ThreadCount;

int Ost = list.Count % ThreadCount;

for (int i = 0; i < ThreadCount; i++)

{

DividedList[i] = new List<string>();

for (int j = i \* WordsInThread; j <( i+1) \* WordsInThread; j++)

{

DividedList[i].Add(list[j]);

}

}

//разбили список слов на кол-во потоков

List<Tuple<string, int>> Result = new List<Tuple<string,int>>();

Task<List<Tuple<string, int>>>[] tasks = new Task<List<Tuple<string, int>>>[ThreadCount];

//Запуск потоков

for (int i = 0; i < ThreadCount; i++)

{

//Создание временного списка, чтобы потоки

//не работали параллельно с одной коллекцией

tasks[i] = new Task<List<Tuple<string,int>>>(

//Метод, который будет выполняться в потоке

TaskMethod,

//Параметры потока - кортеж из куска листа, макс.дист и слова для поиска

new Tuple<List<string>, int,string>(DividedList[i],maxDist,word));

//Запуск потока

tasks[i].Start();

}

//Ожидание завершения всех потоков

Task.WaitAll(tasks);

for (int i = 0; i < ThreadCount; i++)

{

foreach (var x in tasks[i].Result)

Result.Add(x);

}

//Вывод общего массива результатов

return Result;

}

public static List<Tuple<string,int>> TaskMethod(object param)

{

Tuple<List<string>, int, string> paramet = (Tuple<List<string>, int, string>)param;

List<Tuple<string, int>> tempList = new List<Tuple<string, int>>();

foreach (string str in paramet.Item1)

{

int dist = DamerauLivenstein.Distance(str.ToUpper(), paramet.Item3);

//Если расстояние меньше порогового, то слово добавляется в результат

if (dist <= paramet.Item2)

{

tempList.Add(new Tuple<string, int>(str, dist));

}

}

return tempList;

}

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);

}

}

}

return matrix[str1Len, str2Len];

}

}

}

private void buttonReport\_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' align='center'>");

b.AppendLine("<tr>");

b.AppendLine("<td>Время чтения из файла</td>");

b.AppendLine("<td>" + this.textBoxElapsedTime.Text + "</td>");

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

b.AppendLine("<tr>");

b.AppendLine("<td>Количество уникальных слов в файле</td>");

b.AppendLine("<td>" + this.textBoxCountWords.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 нечеткого поиска</td>");

b.AppendLine("<td>" + this.labelSearch.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>");

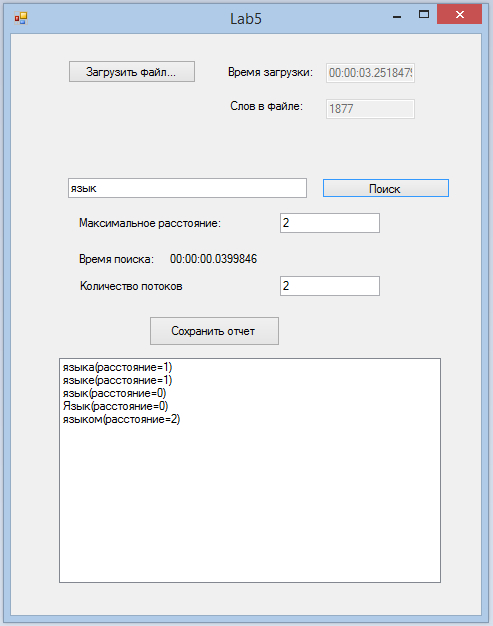
b.AppendLine("</html>");

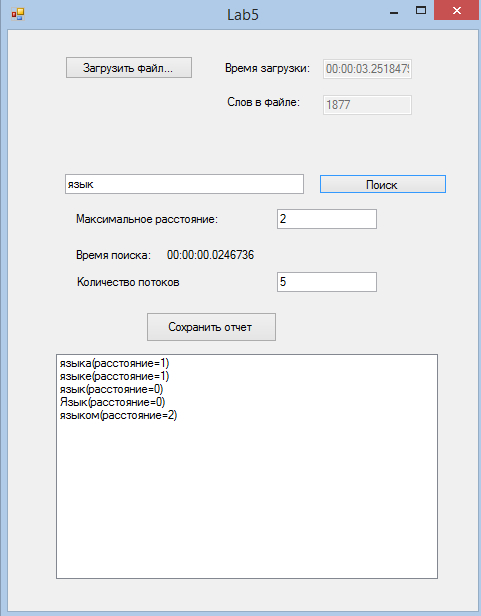
//Сохранение файла

File.AppendAllText(ReportFileName, b.ToString());

MessageBox.Show("Отчет сформирован. Файл: " + ReportFileName);

}





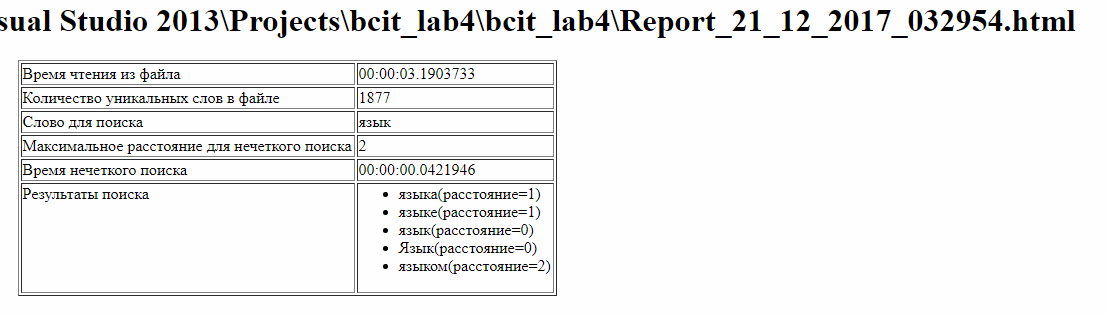


Диаграмма классов:

