**Отчёт по домашнему заданию по предмету «Базовые компоненты интернет-технологий»**

Подготовила: Гладова Анастасия

Группа: ИУ5-31

МГТУ имени Н.Э.Баумана

2017 г.

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.

**Диаграмма классов:**

**Текст программы:**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

using System.Windows.Forms;

//using DistanceLibrary;

using System.Threading.Tasks;

using System.Text;

using EditDistanceProject;

//https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/compiler-messages/cs0246

namespace dz

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

//Для хранения информации о найденных словах используется класс ParallelSearchResult:

/// <summary>

/// Результаты параллельного поиска

/// </summary>

public class ParallelSearchResult

{

/// <summary>

/// Найденное слово

/// </summary>

public string word;

/// <summary>

/// Расстояние

/// </summary>

public int dist;

/// <summary>

/// Номер потока

/// </summary>

public int threadNum;

public ParallelSearchResult(string w, int d, int t)

{

word = w;

dist = d;

threadNum = t;

}

}

/// <summary>

/// Параметры которые передаются в поток

/// для параллельного поиска

/// </summary>

public class ParallelSearchThreadParam

{

/// <summary>

/// Массив для поиска

/// </summary>

public List<string> wordsList;

/// <summary>

/// Слово для поиска

/// </summary>

public string wordToFind;

/// <summary>

/// Максимальное расстояние для нечеткого поиска

/// </summary>

public int maxDist;

/// <summary>

/// Номер потока

/// </summary>

public int threadNum;

//Параметры потока

public ParallelSearchThreadParam(List<string> wl, string wf, int m, int t)

{

wordsList = wl;

wordToFind = wf;

maxDist = m;

threadNum = t;

}

}

/// <summary>

/// Хранение минимального и максимального значений диапазона

/// </summary>

public class MinMax

{

public int min;

public int max;

public MinMax(int i, int j)

{

min = i;

max = j;

}

}

//Разделение массива на подмассивы реализуется с помощью метода DivideSubArrays статического класса SubArrays.

/// <summary>

/// Класс для деления массива на последовательности

/// </summary>

/// <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 List<MinMax> divideSubArrays(int wordsCount, int ThreadCount)

{

//Результирующий список пар с индексами подмассивов

List<MinMax> result = new List<MinMax>();

int countInThread = (int)Math.Ceiling(wordsCount / (double)ThreadCount);

int i = 0;

while (i < wordsCount)

{

MinMax mt = new MinMax(i, i + countInThread);

if (mt.max > wordsCount)

{

mt.max = wordsCount;

}

i += countInThread;

result.Add(mt);

}

return result;

}

/// <summary>

/// Выполняется в параллельном потоке для поиска строк

/// </summary>

public static List<ParallelSearchResult> arrayThreadTask(object p)

{

ParallelSearchThreadParam param = (ParallelSearchThreadParam)p;

//Слово для поиска в верхнем регистре

string wordUpper = param.wordToFind.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.wordsList)

{

//Вычисление расстояния Дамерау-Левенштейна

//int dist = Distance.CalculateDistance(str.ToUpper(), wordUpper);

int dist = EditDistance.Distance(str.ToUpper(), wordUpper);

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

if (dist <= param.maxDist)

{

ParallelSearchResult t = new ParallelSearchResult(str, dist, param.threadNum);

result.Add(t);

}

}

return result;

}

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

//кнопка загрузить файл

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

{

OpenFileDialog file = new OpenFileDialog();

if (file.ShowDialog() == DialogResult.OK)

{

Stopwatch time = new Stopwatch();

time.Start();

//Чтение файла в виде строки

string text = File.ReadAllText(file.FileName, Encoding.GetEncoding(1251));

textBox1.Text = text;

//Разделительные символы для чтения из файла

char[] seps = new char[] { ' ', '.', ',', '!', '?', '/', '\t', '\n', '(', ')' };

string[] textArray = text.Split(seps);

foreach (string word in textArray)

{

string trimmedWord = word.Trim();

if (!words.Contains(trimmedWord))

{

words.Add(trimmedWord);

}

}

time.Stop();

labelTime.Text = time.Elapsed.ToString();

}

else

{

MessageBox.Show("Необходимо выбрать файл");

}

}

private void Form1\_Load(object sender, EventArgs e)

{

}

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

{

//Кнопка «Закрыть окно»:

//Эта кнопка вызывает метод Close для текущего объекта, что приводит к закрытию окна.

this.Close();

}

//кнопка найти слово

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

{

string wordToFind = textBoxFind.Text.Trim();

textBoxFind.Text = wordToFind;

int maxDistance = Convert.ToInt32(textBoxMaxDistance.Text);

if (!string.IsNullOrEmpty(wordToFind) && words.Count > 0)

{

string wordToFindUpper = wordToFind.ToUpper();

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

//число потоков для одновременного поиска;

int threadCount;

if (!int.TryParse(textBox3.Text.Trim(), out threadCount))

{

threadCount = -1;

}

//Создание таймера

Stopwatch time = new Stopwatch();

//Запуск таймера

time.Start();

List<ParallelSearchResult> results = new List<ParallelSearchResult>();

List<MinMax> arrayDivList = divideSubArrays(words.Count, threadCount);

//Количество потоков соответствует количеству фрагментов массива

Task<List<ParallelSearchResult>>[] tasks = new Task<List<ParallelSearchResult>>[threadCount];

//Запуск потоков

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

{

//Создание временного списка, чтобы потоки не работали параллельно с одной коллекцией

List<string> tempTaskList = words.GetRange(arrayDivList[i].min, arrayDivList[i].max - arrayDivList[i].min);

//Метод, который будет выполняться в потоке - аrrayThreadTask

//Параметры потока передаются в виде кортежа, чтобы не создавать временный класс

tasks[i] = new Task<List<ParallelSearchResult>>(arrayThreadTask, new ParallelSearchThreadParam(tempTaskList, wordToFindUpper, maxDistance, i));

//Запуск потока

tasks[i].Start();

}

//Ожидание завершения всех потоков

Task.WaitAll(tasks);

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

{

results.AddRange(tasks[i].Result);

}

//Остановка таймера

time.Stop();

label7.Text = time.Elapsed.ToString();

textBox2.BeginUpdate();

textBox2.Items.Clear();

foreach (var x in results)

{

string temp = x.word + " (расстояние - " + x.dist + ", поток - " + x.threadNum + ")";

textBox2.Items.Add(temp);

}

textBox2.EndUpdate();

}

else

{

MessageBox.Show("Перепроверить.");

}

}

//кнопка отчет

private void buttonSave\_Click\_1(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.labelTime.Text + "</td>");

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

b.AppendLine("<tr>");

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 поиска</td>");

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

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

b.AppendLine("<tr>");

b.AppendLine("<td>Время поиска</td>");

b.AppendLine("<td>" + this.labelTimeFind.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.textBox2.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);

}

}

}

}

namespace EditDistanceProject

{

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];

}

}

}

**Экранные формы с примерами выполнения программы:**

