**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по домашнему заданию

«Многопоточный поиск в файле»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-33 |  | Гапанюк Ю.Е. |
| Миронова Александра |  |  |
| Подпись и дата:  29.12.2020 |  | Подпись и дата: |

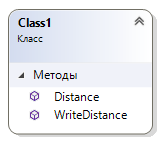
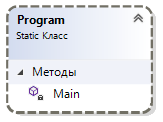
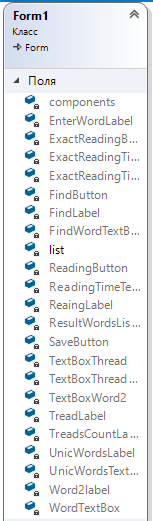
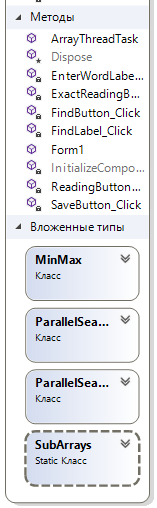
Москва, 2020 г.

**Задание:**

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#».

**Диаграмма классов:**

**Текст программы:**

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.IO;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Work\_with\_files2

{

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.IO;

using System.Diagnostics;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Damerau\_Levenshtein;

namespace Work\_with\_files2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

//-------------------------------------------------

List<string> list = new List<string>(); // Лист для записи слов в из файла

//-------------------------------------------------

public class ParallelSearchResult // Результаты параллельного поиска

{

public string word // Найденное слово

{

get; set;

}

public int dist // Расстояние

{

get; set;

}

public int ThreadNum // Номер потока

{

get; set;

}

}

//-------------------------------------------------

// Параметры которые передаются в поток для параллельного поиска

class ParallelSearchThreadParam

{

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

}

//-------------------------------------------------

// Выполняется в параллельном потоке для поиска строк

public static List<ParallelSearchResult> ArrayThreadTask(object paramObj)

{

ParallelSearchThreadParam param = (ParallelSearchThreadParam)paramObj;

string wordUpper = param.wordPattern.Trim().ToUpper(); // Слово для поиска в верхнем регистре

List<ParallelSearchResult> Result = new List<ParallelSearchResult>(); // Результаты поиска в одном потоке

foreach (string str in param.tempList) // Перебор всех слов во временном списке данного потока

{

int dist = Class1.Distance(str.ToUpper(), wordUpper); // Вычисление расстояния Дамерау-Левенштейна

if (dist <= param.maxDist) // Если расстояние меньше порогового, то слово добавляется в результат

{

ParallelSearchResult temp = new ParallelSearchResult()

{

word = str,

dist = dist,

ThreadNum = param.ThreadNum

};

Result.Add(temp);

}

}

return Result;

}

//-------------------------------------------------

public class MinMax // Хранение минимального и максимального значений диапазона

{

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 // Класс для деления массива на последовательности

{ // Деление массива на последовательности

/// <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)

{

List<MinMax> result = new List<MinMax>(); // Результирующий список пар с индексами подмассивов

//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

{

int delta = (endIndex - beginIndex) / subArraysCount; // Размер подмассива

int currentBegin = beginIndex; // Начало отсчета

//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;

}

}

//------------кнопочки-----------------------------

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

{

OpenFileDialog fd = new OpenFileDialog();

fd.Filter = "текстовые файлы|\*.txt";

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

{

Stopwatch t = new Stopwatch(); // Таймер

t.Start();

string text = File.ReadAllText(fd.FileName); // Чтение файла в строку

string[] textWords = text.Split(new char[] { ' ', ',', '.', '"', ':', '[', ']', '!', '?', '/', '\t','\n' }, StringSplitOptions.RemoveEmptyEntries); //Разделительные символы чтения файла

foreach (string strTemp in textWords) // Удаление пробелов в начале и конце строки

{

string str = strTemp.Trim();

if (!list.Contains(str)) // Добавление строки в список, если строка не содержится в списке

{

list.Add(str);

}

}

t.Stop();

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

this.UnicWordsTextBox.Text = list.Count.ToString();

}

else

{

MessageBox.Show("Необходимо выбрать файл");

}

}

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

{

//Слово для поиска

string word = this.WordTextBox.Text.Trim();

//Если слово для поиска не пусто

if (!string.IsNullOrWhiteSpace(word) && list.Count > 0)

{

int maxDist;

if (!int.TryParse(this.TextBoxWord2.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.TextBoxThread.Text.Trim(), out ThreadCount))

{

MessageBox.Show("Необходимо указать количество потоков");

return;

}

Stopwatch timer = new Stopwatch();

timer.Start();

//-------------------------------------------------

// Начало параллельного поиска

//-------------------------------------------------

List<ParallelSearchResult> Result = new List<ParallelSearchResult>(); // Результирующий список

// Деление списка на фрагменты для параллельного запуска в потоках

List<MinMax> arrayDivList = SubArrays.DivideSubArrays(0, list.Count, ThreadCount); // !!1->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>>(

//Метод, который будет выполняться в потоке

ArrayThreadTask,

//Параметры потока

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.FindWordTextBox.Text = timer.Elapsed.ToString();

//Вычисленное количество потоков

this.TextBoxThreadCountAll.Text = count.ToString(); //!!

//Начало обновления списка результатов

this.ResultWordsListBox.BeginUpdate();

//Очистка списка

this.ResultWordsListBox.Items.Clear();

//Вывод результатов поиска

foreach (var x in Result)

{

string temp = x.word + "(расстояние=" + x.dist.ToString() + " поток=" + x.ThreadNum.ToString() + ")";

this.ResultWordsListBox.Items.Add(temp);

}

//Окончание обновления списка результатов

this.ResultWordsListBox.EndUpdate();

}

else

{

MessageBox.Show("Необходимо выбрать файл и ввести слово для поиска");

}

}

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

{

//Слово для поиска

string word = this.WordTextBox.Text.Trim();

//Если слово для поиска не пусто

if (!string.IsNullOrWhiteSpace(word) && list.Count > 0)

{

string wordUpper = word.ToUpper(); // Поиск в верхнем регистре

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.ExactReadingTimeTextBox.Text = t.Elapsed.ToString();

this.ResultWordsListBox.BeginUpdate();

this.ResultWordsListBox.Items.Clear(); // Очистка списка

foreach (string str in tempList) // Вывод результатов поиска

{

this.ResultWordsListBox.Items.Add(str);

}

this.ResultWordsListBox.EndUpdate();

}

else

{

MessageBox.Show("Необходимо выбрать файл и ввести слово для поиска");

}

}

private void SaveButton\_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.ReadingTimeTextBox.Text + "</td>");

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

b.AppendLine("<tr>");

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

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

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

b.AppendLine("<tr>");

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

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

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

b.AppendLine("<tr>");

b.AppendLine("<td>Максимальное расстояние для нечеткого поиска</td>");

b.AppendLine("<td>" + this.TextBoxWord2.Text + "</td>");

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

b.AppendLine("<tr>");

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

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

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

b.AppendLine("<tr>");

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

b.AppendLine("<td>" + this.FindWordTextBox.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.ResultWordsListBox.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;

//using System.Windows.Forms;

namespace Damerau\_Levenshtein

{

public class Class1

{

//-----------Вычисление расстояния Дамерау-Левенштейна

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);

//Добавление

int ins = matrix[i, j - 1] + 1;

//Удаление

int del = matrix[i - 1, j] + 1;

//Замена

int subst = matrix[i - 1, j - 1] + symbEqual;

//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];

}

//-----------Вывод расстояния Дамерау-Левенштейна в консоль

public static void WriteDistance(string str1Param, string str2Param)

{

int d = Distance(str1Param, str2Param);

Console.WriteLine("'" + str1Param + "','" + str2Param + "' -> " + d.ToString());

}

}

}

**Экранные формы с примерами выполнения программы:**

