**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Базовые компоненты интернет-технологий»

Отчет по домашнему заданию

«Разработка программы, реализующей многопоточный поиск в файле»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-34Б |  | преподаватель каф. ИУ5 |
| Сергеев Илья |  | Гапанюк Ю. Е. |
|  |  |  |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2018 г.

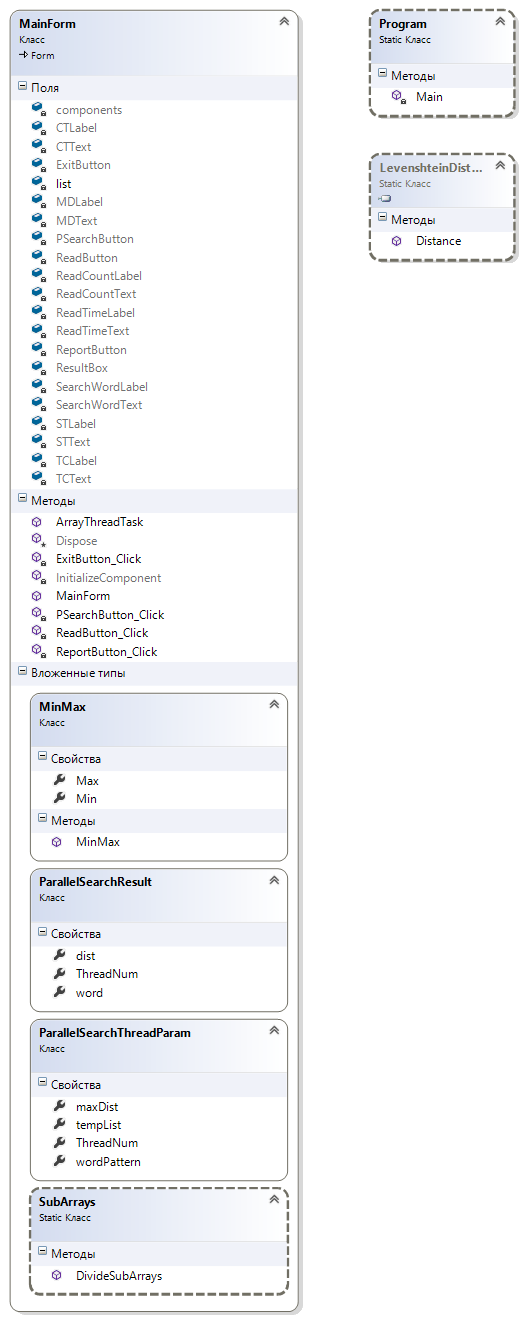
**Описание задания**

Разработать программу, реализующую работу с коллекциями.

1. Программа должна быть разработана в виде приложения Windows Forms на языке C#. По желанию вместо Windows Forms возможно использование WPF.
2. Реализуйте функцию поиска с использованием расстояния Левенштейна в многопоточном варианте. Количество потоков для запуска функции поиска вводится на форме в поле ввода (TextBox).
3. Реализуйте функцию записи результатов поиска в файл отчета. Файл отчета создается в формате .html.

**Диаграмма классов**

Диаграмма классов генерируется автоматически в среде Visual Studio:



**Текст программы (листинг)**

Листинг модуля формы MainForm:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using LevenshteinLibrary;

namespace threading

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

/// <summary>

/// Word list

/// </summary>

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

/// <summary>

/// Multi-thread search class

/// </summary>

public class ParallelSearchResult

{

/// <summary>

/// Found word

/// </summary>

public string word { get; set; }

/// <summary>

/// Distance

/// </summary>

public int dist { get; set; }

/// <summary>

/// Thread number

/// </summary>

public int ThreadNum { get; set; }

}

/// <summary>

/// Multi-threading params class

/// </summary>

class ParallelSearchThreadParam

{

/// <summary>

/// Searching array

/// </summary>

public List<string> tempList { get; set; }

/// <summary>

/// Searching word

/// </summary>

public string wordPattern { get; set; }

/// <summary>

/// Max distance

/// </summary>

public int maxDist { get; set; }

/// <summary>

/// Thread number

/// </summary>

public int ThreadNum { get; set; }

}

/// <summary>

/// Search strings

/// </summary>

public static List<ParallelSearchResult> ArrayThreadTask(object OBJ)

{

ParallelSearchThreadParam param = (ParallelSearchThreadParam)OBJ;

string UpperWord = param.wordPattern.Trim().ToUpper(); //up-cased word

List<ParallelSearchResult> Result = new List<ParallelSearchResult>(); //single threaded search results

foreach (string str in param.tempList) //trying words

{

int dist = LevenshteinDistance.Distance(str.ToUpper(), UpperWord); //calculating a Levenshtein distance

if (dist <= param.maxDist) //if distance is \*FINE\*

{

ParallelSearchResult temp = new ParallelSearchResult() //adding a result

{ word = str, dist = dist, ThreadNum = param.ThreadNum };

Result.Add(temp);

}

}

return Result;

}

/// <summary>

/// Min and Max class

/// </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;

}

}

/// <summary>

/// Sub-arrays division class

/// </summary>

public static class SubArrays

{

/// <summary>

/// Divides array into sub-arrays

/// </summary>

/// <param name="BIndex">beginning index</param>

/// <param name="EIndex">ending index</param>

/// <param name="Counter">reauired sub-arrays counter</param>

/// <returns>list of sub-arrays pairs</returns>

public static List<MinMax> DivideSubArrays( int BIndex, int EIndex, int Counter)

{

List<MinMax> result = new List<MinMax>(); //declaring resulting list

if ((EIndex - BIndex) <= Counter) //too few items!

result.Add(new MinMax(0, (EIndex - BIndex)));

else

{

int delta = (EIndex - BIndex) / Counter; //size of subarray

int CBegin = BIndex; //current begin index

while ((EIndex - CBegin) >= 2 \* delta)

{

result.Add( new MinMax(CBegin, CBegin + delta)); //building sub-array

CBegin += delta; //refreshing begin index

}

result.Add(new MinMax(CBegin, EIndex)); //reminder

}

return result;

}

}

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

{

OpenFileDialog OD = new OpenFileDialog(); //declaring new open dialog

OD.Filter = "Текстовые файлы|\*.txt"; //setting a filter

if (OD.ShowDialog() == DialogResult.OK) //if a file has been chosen

{

Stopwatch sw = new Stopwatch(); //declaring new stopwatch

sw.Start(); //starting the stopwatch

string text = File.ReadAllText(OD.FileName); //reading whole text from the file

char[] sep = new char[] {' ','.',',','!','?','/','\t','\n'}; //separating characters

string[] textArray = text.Split(sep); //splitting words

foreach (string strTemp in textArray) //preparing words

{

string str = strTemp.Trim(); //removing extra spaces

if (!list.Contains(str))

list.Add(str); //adding a word

}

sw.Stop(); //stopping the stopwatch

this.ReadTimeText.Text = sw.Elapsed.ToString(); //showing opening time

this.ReadCountText.Text = list.Count.ToString(); //showing words' counter

}

else

MessageBox.Show("It's necessary to choose a file!");

}

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

{

string word = this.SearchWordText.Text.Trim(); //searching word

if (!string.IsNullOrEmpty(word) && list.Count > 0) //checking a word

{

int MaxDist; //declaring a counter

if(!int.TryParse(this.MDText.Text.Trim(), out MaxDist)) //if distance is incorrect

{

MessageBox.Show("It's necessary to set a distance!");

return;

}

if (MaxDist < 1 || MaxDist > 5) //checking a correct range

{

MessageBox.Show("Max distance should be in [1..5]");

return;

}

int ThreadCount; //counter for threads

if (!int.TryParse(this.TCText.Text.Trim(), out ThreadCount)) //if number wof threads is incorrect

{

MessageBox.Show("It's necessary to set a number of threads");

return;

}

Stopwatch sw = new Stopwatch(); //declaring new stopwatch

sw.Start(); //starting a stopwatch

List<ParallelSearchResult> res = new List<ParallelSearchResult>(); //declaring a list

List<MinMax> DivList = SubArrays.DivideSubArrays(0, list.Count, ThreadCount); //dividing to subarrays

int counter = DivList.Count; //saving a counter

//Количество потоков соответствует количеству фрагментов массива

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

for (int i = 0; i < counter; i++) //launching threads

{

List<string> tempTaskList = list.GetRange(DivList[i].Min, DivList[i].Max - DivList[i].Min);

tasks[i] = new Task<List<ParallelSearchResult>>

(

ArrayThreadTask, //special method

new ParallelSearchThreadParam() //thread params

{

tempList = tempTaskList,

maxDist = MaxDist,

ThreadNum = i,

wordPattern = word

}

);

tasks[i].Start(); //launching a thread

}

Task.WaitAll(tasks);

sw.Stop(); //stopping a stopwatch

for (int i = 0; i < counter; i++) //preparing results

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

//timer.Stop(); //stopping a stopwatch

this.STText.Text = sw.Elapsed.ToString(); //showing search time

this.CTText.Text = counter.ToString(); //showing number of threads

this.ResultBox.BeginUpdate(); //ipdating a list

this.ResultBox.Items.Clear(); //clearing a list

foreach (var x in res) //showing results

{

string temp = x.word + " (расстояние = " + x.dist.ToString() + "; поток = " + (x.ThreadNum + 1).ToString() + ")";

this.ResultBox.Items.Add(temp);

}

this.ResultBox.EndUpdate();

}

else

MessageBox.Show("It's necessary to choose a file and a word to search!");

}

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

{

this.Close();

}

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

{

string TempReportFileName = "Report\_" + DateTime.Now.ToString("dd\_MM\_yyyy\_hhmmss"); //name of report file

SaveFileDialog fd = new SaveFileDialog(); //report saving dialog

fd.FileName = TempReportFileName;

fd.DefaultExt = ".html";

fd.Filter = "HTML Reports|\*.html";

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

{

string ReportFileName = fd.FileName;

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.ReadTimeText.Text + "</td>");

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

b.AppendLine("<tr>");

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

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

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

b.AppendLine("<tr>");

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

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

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

b.AppendLine("<tr>");

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

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

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

b.AppendLine("<tr>");

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

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

}

}

}

}

Листинг библиотеки классов:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace LevenshteinLibrary

{

public static class LevenshteinDistance

{

/// <summary>

/// Damerau–Levenshtein calculating class

/// </summary>

public static int Distance(string P1, string P2)

{

if ((P1 == null) || (P2 == null))

return -1; //in this case we will return -1

int L1 = P1.Length;

int L2 = P2.Length;

if ((L1 == 0) && (L2 == 0)) //if both zero-lenghted, distance is 0

return 0;

if (L1 == 0) //if first lenght equals 0 then distance equals L2

return L2;

if (L2 == 0) //if second lenght equals 0 then distance equals L1

return L1;

string UP1 = P1.ToUpper(); //upping cases of string 1

string UP2 = P2.ToUpper(); //upping cases of string 1

int[,] matrix = new int[L1 + 1, L2 + 1];

for (int i = 0; i <= L1; i++) //zero-row init

matrix[i, 0] = i;

for (int j = 0; j <= L2; j++) //zero-col init

matrix[0, j] = j;

for (int i = 1; i <= L1; i++) //encalculating the distance

for (int j = 1; j <= L2; j++)

{

int CharEqual = ((UP1.Substring(i - 1, 1) == UP2.Substring(j - 1, 1)) ? 0 : 1);

int InsertValue = matrix[i, j - 1] + 1; //adding

int DeleteValue = matrix[i - 1, j] + 1; //deleting

int subst = matrix[i - 1, j - 1] + CharEqual; //replacing

matrix[i, j] = Math.Min(Math.Min(InsertValue, DeleteValue), subst); //encalculating current item og the matrix

if ((i > 1) && (j > 1) && (UP1.Substring(i - 1, 1) == UP2.Substring(j - 2, 1)) &&

(UP1.Substring(i - 2, 1) == UP2.Substring(j - 1, 1)))

matrix[i, j] = Math.Min(matrix[i, j], matrix[i - 2, j - 2] + CharEqual); //Damerau addition

}

return matrix[L1, L2]; //result equals down-right item of the matrix

}

}

}

Листинг основной программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace threading

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainForm());

}

}

}

**Экранные формы с примерами выполнения программы (скриншоты)**

