Федеральное государственное бюджетное образовательное учреждение   
высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»

(МГТУ им. Н.Э. Баумана)

**Отчет по домашнему заданию**

Выполнил:  
Студент группы ИУ5-33Б  
Богданов Д.А.  
Преподаватель:  
Гапанюк Ю.Е.

г. Москва, 2018

1. **Описание задания**

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.Diagnostics;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace laba4

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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 = 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;

}

}

public static int Distance(string str1Param, string str2Param)

{

if ((str1Param == null) || (str2Param == null)) return -1;

int str1Len = str1Param.Length; int str2Len = str2Param.Length;

if ((str1Len == 0) && (str2Len == 0)) return 0;

if (str1Len == 0) return str2Len;

if (str2Len == 0) return str1Len;

string str1 = str1Param.ToUpper();

string str2 = str2Param.ToUpper();

int[,] matrix = new int[str1Len + 1, str2Len + 1];

for (int i = 0; i <= str1Len; i++) matrix[i, 0] = i;

for (int j = 0; j <= str2Len; j++) matrix[0, j] = j;

for (int i = 1; i <= str1Len; i++)

{

for (int j = 1; j <= str2Len; j++)

{

int symbEqual = ((str1.Substring(i - 1, 1) == str2.Substring(j - 1, 1)) ? 0 : 1);

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

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

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

matrix[i, j] = Math.Min(Math.Min(ins, del), subst);

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 buttonREAD\_Click(object sender, EventArgs e)

{

OpenFileDialog fileDialog = new OpenFileDialog();

fileDialog.Filter = "текстовые файлы|\*.txt";

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

{

Stopwatch t = new Stopwatch();

t.Start();

string text = File.ReadAllText(fileDialog.FileName);

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

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

foreach (string strTemp in textArray)

{

string str = strTemp.Trim();

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 textBoxFileReadCount\_TextChanged(object sender, EventArgs e)

{

}

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

{

}

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

{

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

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

{

string wordUpper = word.ToUpper();

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.listBox1.BeginUpdate();

this.listBox1.Items.Clear();

foreach (string str in tempList)

{

this.listBox1.Items.Add(str);

}

this.listBox1.EndUpdate();

}

else

{

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

}

}

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

{

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

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

{

string wordUpper = word.ToUpper();

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

Stopwatch t = new Stopwatch();

int threads;

int MaxDist;

int.TryParse(potok.Text, out threads);

int.TryParse(textBox1.Text, out MaxDist);

t.Start();

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

List<MinMax> DivList = SubArrays.DivideSubArrays(0, list.Count, threads); //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);

t.Stop(); //stopping a stopwatch

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

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

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

this.textBoxExactTime.Text = t.Elapsed.ToString(); //showing search time

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

this.listBox1.BeginUpdate(); //ipdating a list

this.listBox1.Items.Clear(); //clearing a list

foreach (var x in res) //showing results

{

string temp = x.word;

this.listBox1.Items.Add(temp);

}

this.listBox1.EndUpdate();

}

else

{

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

}

}

}

}

**Примеры выполнения программы**

