**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»

Пояснительная записка по выполнению домашнего задания «Многопоточный поиск»

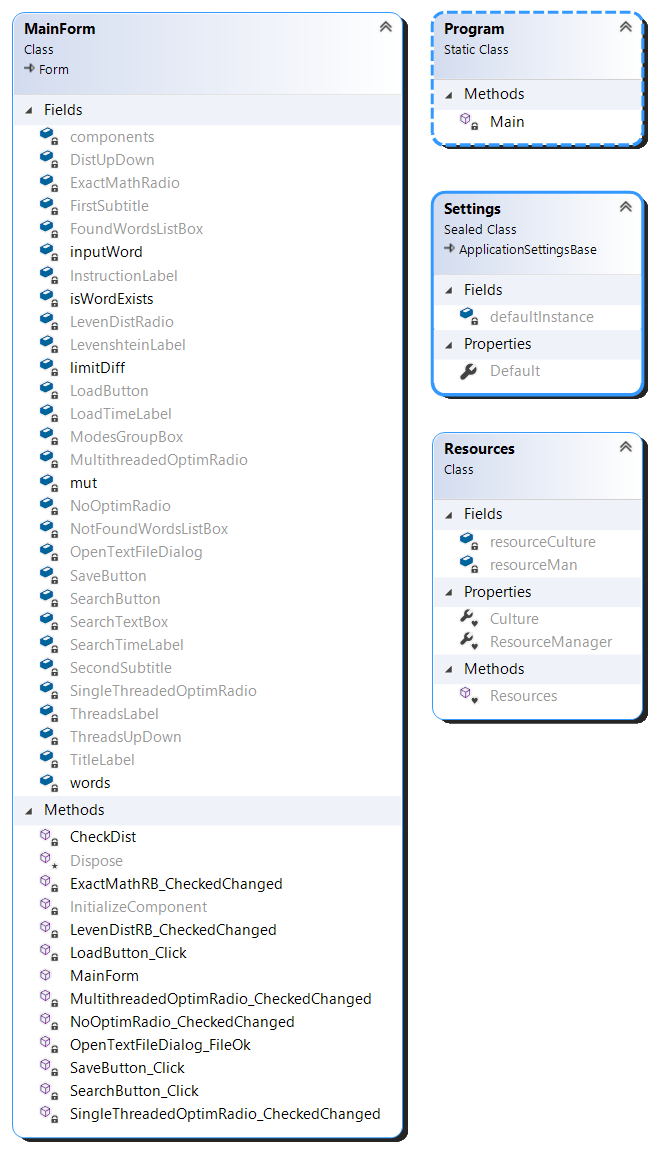
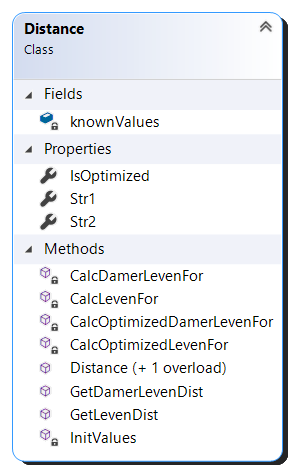
|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Бабин Артём |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2020 г.

**Условие 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). 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.Windows.Forms;

namespace BKIT\_DZ

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainForm());

}

}

}

**MainForm.cs:**

using System;

using System.IO;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Text;

using System.Windows.Forms;

using System.Threading;

using System.Threading.Tasks;

using BKIT\_LAB5;

namespace BKIT\_DZ

{

public partial class MainForm : Form

{

Mutex mut = new Mutex();

List<string> words;

string inputWord;

int limitDiff;

bool isWordExists;

public MainForm()

{

InitializeComponent();

}

private void OpenTextFileDialog\_FileOk(object sender, CancelEventArgs e)

{

words = new List<string>();

FoundWordsListBox.Items.Clear();

NotFoundWordsListBox.Items.Clear();

Stopwatch timer = new Stopwatch();

timer.Start();

string[] text = File.ReadAllText(OpenTextFileDialog.FileName, Encoding.UTF8).Split(' ');

foreach (string word in text)

{

if (!words.Contains(word))

words.Add(word);

}

timer.Stop();

LoadTimeLabel.Text = "Load time: " + timer.Elapsed.TotalMilliseconds.ToString() + " ms";

ExactMathRadio.Visible = true;

LevenDistRadio.Visible = true;

LoadTimeLabel.Visible = true;

SaveButton.Visible = true;

}

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

{

Stopwatch timer = new Stopwatch();

isWordExists = false;

timer.Start();

if (ExactMathRadio.Checked)

isWordExists = words.Contains(SearchTextBox.Text);

else

{

if (MultithreadedOptimRadio.Checked)

{

limitDiff = Convert.ToInt32(DistUpDown.Value);

inputWord = SearchTextBox.Text;

int threadsNum = (int)ThreadsUpDown.Value;

int startIndex = 0;

int count = words.Count / threadsNum + ((words.Count % threadsNum != 0) ? 1 : 0);

Task[] tasks = new Task[threadsNum];

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

{

tasks[i] = new Task(CheckDist, words.GetRange(startIndex, count));

tasks[i].Start();

startIndex += count;

if (count > words.Count - startIndex)

count = words.Count - startIndex;

}

Task.WaitAll(tasks);

}

else

{

Distance dist = new Distance();

dist.IsOptimized = (SingleThreadedOptimRadio.Checked);

dist.Str1 = SearchTextBox.Text;

limitDiff = Convert.ToInt32(DistUpDown.Value);

foreach (string word in words)

{

dist.Str2 = word;

if (dist.GetLevenDist() <= limitDiff)

{

isWordExists = true;

break;

}

}

}

}

timer.Stop();

SearchTimeLabel.Text = "Search time: " + timer.Elapsed.TotalMilliseconds.ToString() + " ms";

ListBox targetListBox = (isWordExists) ? FoundWordsListBox : NotFoundWordsListBox;

targetListBox.BeginUpdate();

targetListBox.Items.Add(SearchTextBox.Text);

targetListBox.EndUpdate();

}

private void CheckDist(object words)

{

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

Distance dist = new Distance();

dist.IsOptimized = true;

dist.Str1 = inputWord;

foreach (string word in someWords)

{

if (!isWordExists)

{

dist.Str2 = word;

if (dist.GetLevenDist() <= limitDiff)

{

mut.WaitOne();

isWordExists = true;

mut.ReleaseMutex();

return;

}

}

else return;

}

}

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

{

OpenTextFileDialog.ShowDialog();

}

private void ExactMathRB\_CheckedChanged(object sender, EventArgs e)

{

if (ExactMathRadio.Checked)

{

LevenshteinLabel.Visible = false;

DistUpDown.Visible = false;

ModesGroupBox.Visible = false;

InstructionLabel.Visible = true;

SearchTextBox.Visible = true;

SearchButton.Visible = true;

SearchTimeLabel.Visible = true;

FoundWordsListBox.Visible = true;

NotFoundWordsListBox.Visible = true;

LoadTimeLabel.Visible = true;

FirstSubtitle.Visible = true;

SecondSubtitle.Visible = true;

}

}

private void LevenDistRB\_CheckedChanged(object sender, EventArgs e)

{

if (LevenDistRadio.Checked)

{

ModesGroupBox.Visible = true;

LevenshteinLabel.Visible = true;

DistUpDown.Visible = true;

InstructionLabel.Visible = true;

SearchTextBox.Visible = true;

SearchButton.Visible = true;

SearchTimeLabel.Visible = true;

FoundWordsListBox.Visible = true;

NotFoundWordsListBox.Visible = true;

LoadTimeLabel.Visible = true;

FirstSubtitle.Visible = true;

SecondSubtitle.Visible = true;

}

}

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

{

string log = words.Count + " unique words were loaded from \"" + OpenTextFileDialog.FileName + "\"\n" + LoadTimeLabel.Text + "\n";

log += "Found words:\n";

foreach (object items in FoundWordsListBox.Items)

log += items.ToString() + "\n";

log += "\nNot found words:\n";

foreach (object items in NotFoundWordsListBox.Items)

log += items.ToString() + "\n";

string fileName = "Log info for " + DateTime.Now.ToString("dd\_MM\_yyyy\_HHmmss") + ".txt";

File.WriteAllText(fileName, log);

MessageBox.Show("Done!");

}

private void SingleThreadedOptimRadio\_CheckedChanged(object sender, EventArgs e)

{

if (SingleThreadedOptimRadio.Checked)

{

ThreadsLabel.Visible = false;

ThreadsUpDown.Visible = false;

}

}

private void NoOptimRadio\_CheckedChanged(object sender, EventArgs e)

{

if (NoOptimRadio.Checked)

{

ThreadsLabel.Visible = false;

ThreadsUpDown.Visible = false;

}

}

private void MultithreadedOptimRadio\_CheckedChanged(object sender, EventArgs e)

{

if (MultithreadedOptimRadio.Checked)

{

ThreadsLabel.Visible = true;

ThreadsUpDown.Visible = true;

}

}

}

}

**Distance.cs:**

using System;

namespace BKIT\_LAB5

{

public class Distance

{

public string Str1 { get; set; }

public string Str2 { get; set; }

public bool IsOptimized { get; set; }

private int[,] knownValues;

public Distance()

{

Str1 = "";

Str2 = "";

IsOptimized = false;

}

public Distance(string s1, string s2, bool \_isOptimized)

{

Str1 = s1;

Str2 = s2;

IsOptimized = \_isOptimized;

}

public int GetLevenDist()

{

if (IsOptimized)

{

InitValues();

return CalcOptimizedLevenFor(Str1.Length, Str2.Length);

}

else

return CalcLevenFor(Str1.Length, Str2.Length);

}

public int GetDamerLevenDist()

{

if (IsOptimized)

{

InitValues();

return CalcOptimizedDamerLevenFor(Str1.Length, Str2.Length);

}

else

return CalcDamerLevenFor(Str1.Length, Str2.Length);

}

private void InitValues()

{

knownValues = new int[Str1.Length + 1, Str2.Length + 1];

for (int i = 0; i < Str1.Length + 1; ++i)

{

for (int j = 0; j < Str2.Length + 1; ++j)

{

if (i == 0)

knownValues[i, j] = j;

else if (j == 0)

knownValues[i, j] = i;

else

knownValues[i, j] = int.MaxValue;

}

}

}

private int CalcLevenFor(int i, int j)

{

if (Math.Min(i, j) == 0)

return Math.Max(i, j);

return Math.Min(Math.Min(CalcLevenFor(i, j - 1), CalcLevenFor(i - 1, j)) + 1, CalcLevenFor(i - 1, j - 1) + ((Str1[i - 1] != Str2[j - 1]) ? 1 : 0));

}

private int CalcDamerLevenFor(int i, int j)

{

if (Math.Min(i, j) == 0)

return Math.Max(i, j);

if (i > 1 && j > 1 && (Str1[i - 1] == Str2[j - 2]) && (Str1[i - 2] == Str2[j - 1]))

return Math.Min(Math.Min(CalcDamerLevenFor(i, j - 1), CalcDamerLevenFor(i - 1, j)) + 1, Math.Min(CalcDamerLevenFor(i - 1, j - 1) + ((Str1[i - 1] != Str2[j - 1]) ? 1 : 0), CalcDamerLevenFor(i - 2, j - 2) + 1));

return Math.Min(Math.Min(CalcDamerLevenFor(i, j - 1), CalcDamerLevenFor(i - 1, j)) + 1, CalcDamerLevenFor(i - 1, j - 1) + ((Str1[i - 1] != Str2[j - 1]) ? 1 : 0));

}

private int CalcOptimizedLevenFor(int i, int j)

{

if (i == 0 || j == 0)

return knownValues[i, j];

if (knownValues[i, j - 1] == int.MaxValue)

knownValues[i, j - 1] = CalcOptimizedLevenFor(i, j - 1);

if (knownValues[i - 1, j] == int.MaxValue)

knownValues[i - 1, j] = CalcOptimizedLevenFor(i - 1, j);

return Math.Min(Math.Min(knownValues[i, j - 1], knownValues[i - 1, j]) + 1, knownValues[i - 1, j - 1] + ((Str1[i - 1] != Str2[j - 1]) ? 1 : 0));

}

private int CalcOptimizedDamerLevenFor(int i, int j)

{

if (i == 0 || j == 0)

return knownValues[i, j];

if (knownValues[i, j - 1] == int.MaxValue)

knownValues[i, j - 1] = CalcOptimizedDamerLevenFor(i, j - 1);

if (knownValues[i - 1, j] == int.MaxValue)

knownValues[i - 1, j] = CalcOptimizedDamerLevenFor(i - 1, j);

if (i > 1 && j > 1 && (Str1[i - 1] == Str2[j - 2]) && (Str1[i - 2] == Str2[j - 1]))

return Math.Min(Math.Min(knownValues[i, j - 1], knownValues[i - 1, j]) + 1, Math.Min(knownValues[i - 1, j - 1] + ((Str1[i - 1] != Str2[j - 1]) ? 1 : 0), knownValues[i - 2, j - 2] + 1));

return Math.Min(Math.Min(knownValues[i, j - 1], knownValues[i - 1, j]) + 1, knownValues[i - 1, j - 1] + ((Str1[i - 1] != Str2[j - 1]) ? 1 : 0));

}

}

}

**Примеры выполнения программы:**

|  |  |
| --- | --- |
| *Пример* | *Вывод программы* |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |