МГТУ им. Баумана  
Факультет «Информатика и системы управления»  
Кафедра «Автоматизированные системы обработки информации и управления»  
Дисциплина «Базовые компоненты интернет-технологий»

Отчёт по домашнему заданию

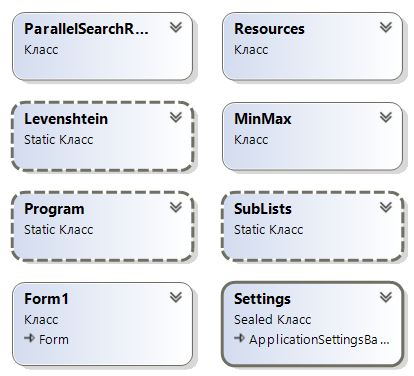


Описание задания:

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.

Диаграмма классов:



Текст программы:

Класс Levenshtein:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Files {

public static class Levenshtein {

public static int Distance(string str1, string str2) {

int strlen1 = str1.Length, strlen2 = str2.Length;

//проверка на пустую строку

if (strlen1 == 0 || strlen2 == 0) {

if (strlen1 != 0) {

return strlen1;

}

else {

return strlen2;

}

}

string s1 = str1.ToUpper(), s2 = str2.ToUpper();

int[,] matrix = new int[strlen1 + 1, strlen2 + 1];

//первая строка и столбец матрицы

for (int i = 0; i <= strlen1; i++) {

matrix[i, 0] = i;

}

for (int j = 0; j <= strlen2; j++) {

matrix[0, j] = j;

}

//остальные ячейки матрицы

for (int i = 1; i <= strlen1; i++) {

for (int j = 1; j <= strlen2; j++) {

int equal;

if (s1.Substring(i - 1, 1) == s2.Substring(j - 1, 1)) {

equal = 0;

}

else {

equal = 1;

}

matrix[i, j] = Math.Min(Math.Min(matrix[i - 1, j] + 1, matrix[i, j - 1] + 1), matrix[i - 1, j - 1] + equal);

if ((i > 1) && (j > 1) &&

(s1.Substring(i - 1, 1) == s2.Substring(j - 1, 1)) &&

(s1.Substring(i - 2, 1) == s2.Substring(j - 2, 1))) {

matrix[i, j] = Math.Min(matrix[i, j], matrix[i - 2, j - 2] + equal);

}

}

}

//возврат результата

return matrix[strlen1, strlen2];

}

public static void WriteDistance(string str1, string str2) {

Console.WriteLine(Distance(str1, str2));

Console.ReadLine();

}

}

}

Класс MinMax:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Files {

public class MinMax {

public int Min { get; set; }

public int Max { get; set; }

public MinMax(int min, int max) {

Min = min;

Max = max;

}

}

}

Класс SubLists:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Files {

public static class SubLists {

public static List<MinMax> SplitList(int firstIndex, int lastIndex, int subListCount) {

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

if (lastIndex - firstIndex <= subListCount) {

result.Add(new MinMax(0, lastIndex - firstIndex));

return result;

}

int subListSize = (lastIndex - firstIndex) / subListCount;

int currentFirstIndex = firstIndex;

while (lastIndex - currentFirstIndex >= 2\*subListSize) {

result.Add(new MinMax(currentFirstIndex, currentFirstIndex + subListSize));

currentFirstIndex += subListSize;

}

result.Add(new MinMax (currentFirstIndex, lastIndex));

return result;

}

}

}

Класс ParallelSearchResult:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Files {

public class ParallelSearchResult {

public string Word { get; set; }

public int Dist { get; set; }

public int Thread { get; set; }

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

Word = w;

Dist = d;

Thread = t;

}

}

}

Класс Form1:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.IO;

using System.Linq;

using System.Text;

using System.Diagnostics;

using System.Windows.Forms;

using System.Threading.Tasks;

namespace Files {

public partial class Form1 : Form {

public Form1() {

InitializeComponent();

}

/// <summary>

/// Список слов WordList, содержащихся в файле.

/// </summary>

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

/// <summary>

/// Простой поиск слов в WordList.

/// </summary>

public void SimpleSearch() {

Stopwatch sw = new Stopwatch();

sw.Start();

if (WordList.Contains(WordSearchBox.Text) &&

//В простом поиске не добавляем в экранный список слова, которые там уже есть

(!WordListBox.Items.Contains(WordSearchBox.Text))) {

WordListBox.BeginUpdate();

WordListBox.Items.Add(WordSearchBox.Text);

WordListBox.EndUpdate();

}

sw.Stop();

SimpleSearchTime.Text = sw.Elapsed.ToString();

}

/// <summary>

///

/// </summary>

/// <param name="ParamObj"></param>

/// <returns></returns>

public List<ParallelSearchResult> TempListMethod(object ParamObj) {

Tuple<List<string>, int, int> Param = (Tuple<List<string>, int, int>)ParamObj;

List<string> TempList = Param.Item1;

int MaxDistance = Param.Item2;

int CurrentThread = Param.Item3;

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

for (int i = 0; i < TempList.Count; i++) {

int Dist = Levenshtein.Distance(WordSearchBox.Text, TempList[i]);

if (Dist <= MaxDistance) {

ParallelSearchResult Temp = new ParallelSearchResult(TempList[i], Dist, CurrentThread);

Result.Add(Temp);

}

}

return Result;

}

/// <summary>

/// Разделение поиска слов алгоритмом Вагнера-Фишера на паралелльные потоки. Сбор и вывод результатов.

/// </summary>

/// <param name="ThreadCount">Число потоков</param>

/// <param name="MaxDistance">Максимальное расстояние Левенштейна</param>

public void ParallelSearch(int ThreadCount, int MaxDistance) {

Stopwatch sw = new Stopwatch();

sw.Start();

//Создание списка индексов для разбиения списка слов на подсписки

List<MinMax> NewWordLists = SubLists.SplitList(0, WordList.Count, ThreadCount);

//Инициализация списка результатов работы потоков

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

int count = NewWordLists.Count;

//Создание нужного числа потоков

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

for (int i = 0; i < count; i++) {

//Разбиение списка слов на подсписки

List<string> TempTaskList = WordList.GetRange(NewWordLists[i].Min, NewWordLists[i].Max - NewWordLists[i].Min);

//Инициализация потоков методом TempListMethod с передачей необходимых данных в виде кортежа

tasks[i] = new Task<List<ParallelSearchResult>>(TempListMethod, new Tuple<List<string>, int, int>(TempTaskList, MaxDistance, i));

//Запуск потоков

tasks[i].Start();

}

//Ожидание конца работы всех потоков

Task.WaitAll(tasks);

sw.Stop();

ParallelSearchTime.Text = sw.Elapsed.ToString();

//Сбор результатов работы всех потоков в один список

for (int i = 0; i < count; i++) {

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

}

//Вывод результатов поиска на экранный список слов

WordListBox.BeginUpdate();

WordListBox.Items.Clear();

foreach (ParallelSearchResult res in SearchResultList) {

string s = res.Word + " (Поток: " + res.Thread + ") (Расстояние: " + res.Dist + ")";

WordListBox.Items.Add(s);

}

WordListBox.EndUpdate();

}

/// <summary>

/// Создание списка слов WordList, содержащихся в файле. Активация остальных элементов управления формы.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void FileSelectionDialog\_FileOk(object sender, CancelEventArgs e) {

//Сокращение полного имени файла для менее громоздкого отображения на экране

FileNameLabelShown.Text = "..." + FileSelectionDialog.FileName.Remove(0, FileSelectionDialog.FileName.LastIndexOf('\\') + 1);

Stopwatch sw = new Stopwatch();

sw.Start();

string FileContents = File.ReadAllText(FileSelectionDialog.FileName, Encoding.GetEncoding(1251));

string[] FileWords = FileContents.Split(' ', '\n', '\r', '!', ',', '?', '—', '.', '-', ':', ';', ')', '(', '"', '\t', '/');

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

foreach (string s in FileWords) {

if (!list.Contains(s) && s != "") {

list.Add(s);

}

}

WordList = list;

sw.Stop();

ElapsedListCreationTime.Text = sw.Elapsed.ToString() /\*" тиков"\*/;

label1.Text = "Уникальных слов в файле: " + WordList.Count.ToString();

WordSearchButton.Enabled = true;

WordSearchBox.Enabled = true;

WordListBox.Enabled = true;

SimpleSearchButton.Enabled = true;

ParallelSearchButton.Enabled = true;

}

/// <summary>

/// Вызов диалога выбора текстового файла.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void FileSelectionButton\_Click(object sender, EventArgs e) {

FileSelectionDialog.ShowDialog();

}

/// <summary>

/// Поиск слова в списке WordList и вывод найденных слов в WordListBox.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void WordSearchButton\_Click(object sender, EventArgs e) {

if (SimpleSearchButton.Checked) {

SimpleSearch();

}

else if (ParallelSearchButton.Checked) {

if (!int.TryParse(MaxDistanceBox.Text, out int MaxDistance)) {

MaxDistance = 1;

MaxDistanceBox.Text = "1";

}

if (!int.TryParse(NumberOfThreadsBox.Text, out int NumberOfThreads)) {

NumberOfThreads = 5;

NumberOfThreadsBox.Text = "5";

}

ParallelSearch(NumberOfThreads, MaxDistance);

}

else {

throw new Exception("Это исключение не должно появляться никогда.");

}

MakeReportButton.Enabled = true;

}

/// <summary>

/// Реализация функционала клавиши Enter при вводе слова для поиска в окне WordSearchBox

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void WordSearchBox\_Enter(object sender, EventArgs e) {

AcceptButton = WordSearchButton;

}

/// <summary>

/// Предотвращение некорректной работы клавиши Enter при неактивном окне WordSearchBox

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void WordSearchBox\_Leave(object sender, EventArgs e) {

AcceptButton = null;

}

private void ParallelSearchButton\_CheckedChanged(object sender, EventArgs e) {

MaxDistanceBox.Enabled = ParallelSearchButton.Checked;

NumberOfThreadsBox.Enabled = ParallelSearchButton.Checked;

}

private void MakeReportButton\_Click(object sender, EventArgs e) {

sfd.FileName = "Report\_" + DateTime.Now.ToString("yyyy\_MM\_ddThh\_mm\_ss");

sfd.DefaultExt = ".txt";

sfd.Filter = "Текстовые файлы (\*.txt)|\*.txt";

if (sfd.ShowDialog() == DialogResult.OK) {

string ReportFileName = sfd.FileName;

StringBuilder s = new StringBuilder();

s.AppendLine("Отчёт");

s.AppendLine();

s.Append("Время создания списка: ");

s.AppendLine(ElapsedListCreationTime.Text);

s.AppendLine(label1.Text);

s.Append("Искомое слово: ");

s.AppendLine(WordSearchBox.Text);

s.Append("Время простого поиска: ");

s.AppendLine(SimpleSearchTime.Text);

s.Append("Количество потоков: ");

s.AppendLine(NumberOfThreadsBox.Text);

s.Append("Максимальное расстояние: ");

s.AppendLine(MaxDistanceBox.Text);

s.Append("Время параллельного поиска: ");

s.AppendLine(ParallelSearchTime.Text);

s.AppendLine();

s.AppendLine("Результаты поиска:");

foreach (string str in WordListBox.Items) {

s.AppendLine(str);

}

File.AppendAllText(ReportFileName, s.ToString());

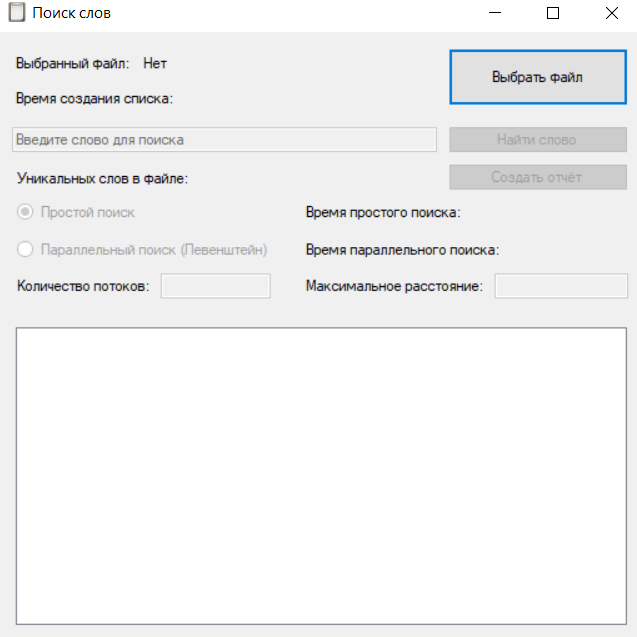
}

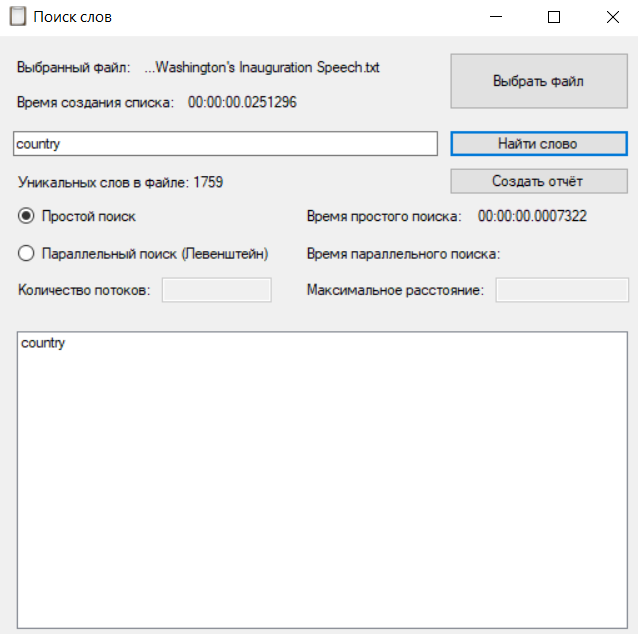
}

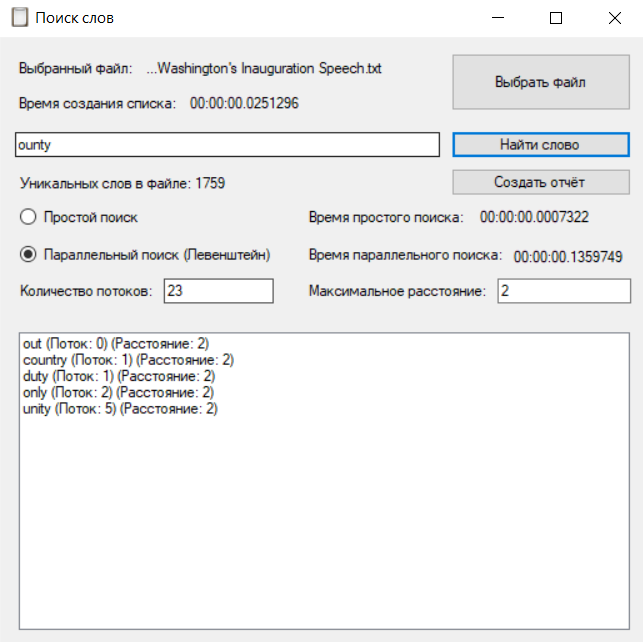
}

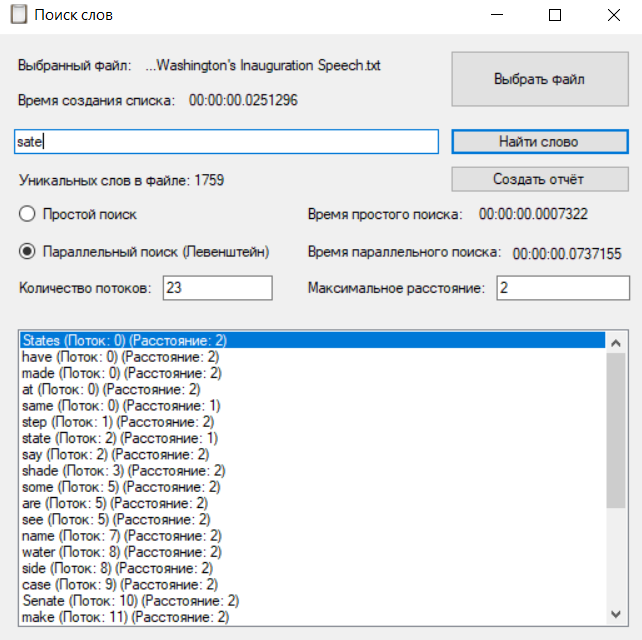
}

Экранные формы:

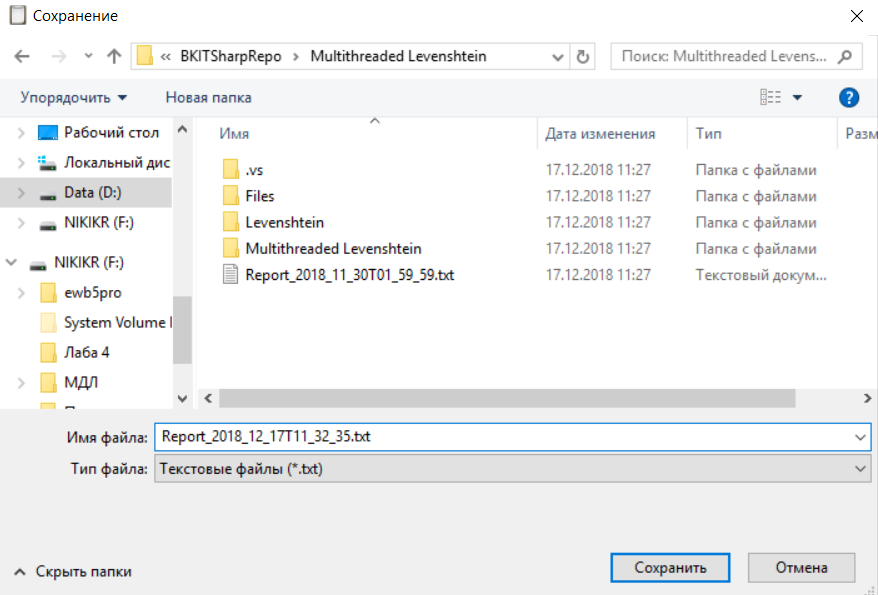








Нажатие кнопки «Создать отчёт»:



Файл отчёта:

