МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

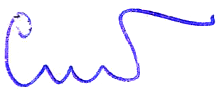
Допущен к защите

И.о. заведующего кафедрой ПМИ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.В. Разова

**Разработка программы парсинга википедии**

Курсовой проект

Выполнил студент группы ФИб-3302-51-00    / Д.А.Савин /

Руководитель доцент кафедры ПМИ   / Е.В.Котельников /

Работа защищена с оценкой     \_\_\_\_.\_\_\_\_.2021 г.

Члены комиссии:     /     /

    /     /

Киров 2021

Содержание

[Введение 3](#_Toc90996811)

[1 Анализ предметной области 4](#_Toc90996812)

[1.1 Основные понятия и определения 4](#_Toc90996813)

[1.2 Обзор методов для реализации программы 5](#_Toc90996814)

[1.3 Выводы по разделу 7](#_Toc90996815)

[2 Программная реализация программы парсинга википедии 8](#_Toc90996816)

[2.1 Реализация программы 8](#_Toc90996817)

[2.2 Экспериментальное исследование программы парсинга википедии 14](#_Toc90996818)

[2.2.1 Тестирование онлайн-метода 15](#_Toc90996819)

[2.2.2 Тестирование оффлайн-метода 15](#_Toc90996820)

[2.2.3 Выводы по разделу 15](#_Toc90996821)

[Заключение 16](#_Toc90996822)

[Библиографический список 17](#_Toc90996823)

[Приложения 18](#_Toc90996824)

[Приложение А. Листинг программы 18](#_Toc90996825)

# Введение

В википедии находится большое множество статей, и чтобы найти все статьи на выбранную тему, нужно вручную перебирать найденные статьи. Чтобы решить данную проблему, можно реализовать программу, которая будет искать в дампе википедии среди всех статей, статьи, подходящие по ключевым словам.

Проблема исследования – недостаток или отсутствие рациональных методов или способов поиска статей на википедии.

Целью курсового проекта является реализация программы парсинга википедии.

Для достижения цели необходимо решить следующие задачи.

1. Выполнить обзор методов, понадобившихся для реализации программы.
2. Программная реализация выбранных методов.
3. Экспериментальное исследование реализованных методов.

Курсовой проект включает два раздела. В первом разделе знакомство с предметной областью и обзор методов. Во втором разделе программная реализация программы парсинга википедии.

# Анализ предметной области

## Основные понятия и определения

Википедия (англ. Wikipedia, произносится [ˌwɪkɨˈpiːdiə] или [ˌwɪkiˈpiːdiə]) — общедоступная многоязычная универсальная интернет-энциклопедия со свободным контентом, реализованная на принципах вики.

Дампы резервных копий базы данных — полная копия всех вики-сайтов Викимедиа в виде источника викитекста и метаданных, встроенных в XML.

Дампы сжаты утилитой bzip2 основанной на алгоритме Барроуза — Уилера.

Программа сжатия файлов bzip2 работает с данными в блоках разного размера, обычно от 100 до 900 килобайт. Она основана на преобразовании или алгоритме Барроуза-Уиллера для преобразования всех повторяющихся последовательностей символов в идентичные строки букв. Затем программа использует переход кодирования Хаффмана к переднему преобразованию. bzip, который был предшественником bzip2, использовал арифметическое кодирование, но его преемник использует кодирование Хаффмана. Производительность bzip2 асимметрична. Имеет относительно быструю декомпрессию. Некоторое время в начале века программа учитывала возможности многопоточности и была нацелена на линейное повышение скорости на многоядерных или многопроцессорных компьютерах, но эта функциональность не была доступна в последующих версиях, выпущенных разработчик. [[1](#_Библиографический_список)]

Рабочая группа SGML Консорциума W3 предложила новый стандарт под названием XML (расширяемый язык разметки), который является подмножеством SGML. Цель XML - предоставить многие преимущества SGML, недоступные в HTML, и предоставить их на языке, который легче выучить и использовать, чем полный SGML. Эти преимущества включают произвольное расширение тегов и атрибутов документа, поддержку документов со сложной структурой и проверку структуры документа относительно дополнительной грамматики структуры документа, называемой дескриптором типа документа (DTD). Унаследованное от SGML, DTD определяет, какие элементы могут встречаться и как элементы могут быть вложены в XML-документ, соответствующий DTD.

Одно из преимуществ XML — его простота. XML-документ — это последовательность элементов, каждый из которых состоит из свободного текста и/или других элементов. Единственное ограничение - теги элементов должны совпадать, например, каждый <ADDRESS> должен иметь соответствующий </ADDRESS> и должен правильно размещаться. XML-документ, имеющий совпадающие и правильно вложенные теги называются правильно сформированными. Элементы в XML примерно соответствуют объектам в объектно-ориентированных или объектно-реляционных базах данных. Например, <PERSON> ... </PERSON> будет соответствовать объекту типа PERSON {...}. Вложенные элементы XML соответствуют поля объекта, например, элементы <NAME>, <PHONE> и <ADDRESS> в <PERSON> будут соответствуют полям NAME, PHONE и ADDRESS объекта PERSON. [[2](#_Библиографический_список)]

## Обзор методов для реализации программы

HttpWebRequest

HttpWebRequest – класс, изначально использовавшийся для осуществления HTTP запросов в .NET. Этот класс обладает богатым функционалом. Он позволяет контролировать каждый аспект объектов запроса: заголовки, таймауты, куки, протоколы и так далее. Стоит отметить, что данный класс не блокирует поток пользовательского интерфейса. Из богатой функциональности вытекает главный недостаток этого класса – сложность. Чтобы создать и прочитать простой GET запрос, необходимо как минимум 5 строчек кода. Это означает, что вероятность возникновения ошибки возрастает. Однако, несмотря на этот недостаток, HttpWebRequest хорошо подходит для создания низкоуровневых запросов. [[5](#_Библиографический_список)]

XmlReader

Расширение XmlReader — это объектно-ориентированный API, использующий реализацию libxml2, которая, в свою очередь, основана на реализации C# API XmlTextReader (http://dotgnu.org/pnetlib-doc/System/Xml/XmlTextReader.html). Расширение XmlReader — это парсер только для прямой передачи, основанный на потоке, но, в отличие от SAX, это парсер вытягивания, а не выталкивания. При перемещении по документу курсор анализатора располагается на разных узлах, позволяя вам получить доступ к информации с текущего узла. Он предлагает множество преимуществ по сравнению с расширение xml, включая дополнительные функциональные возможности.[3]

XmlReader предоставляет быстрый курсор только для переадресации для чтения XML-документов. Он скрывает сложности работы с базовыми данными, предоставляя информационный набор документа с помощью четко определенных методов. На первый взгляд, XmlReader очень похож на SAX (особенно ContentHandler), который также основан на потоковых документах, но две модели программирования принципиально отличаются. Еще в 1998 году, когда XML стал рекомендацией W3C, а затем в том же году Уровень DOM 1, разработчики быстро поняли, что DOM не соответствует потребностям каждого приложения, особенно при работе с большими документами. В результате SAX был разработан для обеспечения более чувствительной к производительности и ресурсам модели обработки. Читатели SAX отвечают за анализ потока символов и за передачу информационного набора документа в приложения-потребители через набор зарегистрированных интерфейсов. Модель push имеет ряд преимуществ перед DOM, но она имеет свои собственные сложности, большинство из которых связаны с управлением государством. XmlReader представляет собой компромисс между API-интерфейсами DOM и SAX. Он предлагает потоковая модель (например, SAX), которая поставляется с гораздо более простой моделью программирования (например, DOM). XmlReader очень похож на курсор firehose (прямой и только для чтения), который позволяет потребителю извлекать записи по одной и даже пропускать записи, не представляющие интереса. Модель вытягивания XmlReader имеет несколько преимуществ по сравнению с более распространенной моделью выталкивания, подобной SAX. Прежде всего, модель pull проще в использовании. Разработчикам просто проще думать о циклах while, чем о сложных машинах состояний. Хотя контекстуальный управление состоянием по-прежнему является проблемой с моделью вытягивания, с ней легче справиться с помощью процедурных методов, ориентированных на потребителя, которые более естественны для большинства разработчиков. [[4](#_Библиографический_список)]

## Выводы по разделу

Для поиска статей можно использовать дампы резервных копий базы данных википедии, встроенных в XML и сжатых bz2. Для реализации программы понадобятся такие классы, как HttpWebRequest, BZip2InputStream, XmlReader.

# Программная реализация программы парсинга википедии

## Реализация программы

Программную реализацию программы было решено писать на языке C#, в Microsoft Visual Studio 2022.

Интерфейс программы:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.1.1 – Интерфейс программы

Программа состоит из класса , который отвечает за парсинг данных с xml-файла, класса , отвечающего за распаковку дампа из архива в xml-файл и класса , с помощью которого формируется запрос на скачивание архива с дампом русскоязычной википедии.

Сначала требуется понять, как устроен xml-файл дампа википедии, для этого нужно посмотреть схему (рисунок 2.1.2):

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.1.2 – схема xml-файла

Функция получения статьи из xml-файла:

|  |
| --- |
| **public** IEnumerable<Page> ReadPages()  {  XNamespace nss = "http://www.mediawiki.org/xml/export-0.10/";  **while** (xmlReader.ReadToFollowing("page"))  {  XElement el = XNode.ReadFrom(xmlReader) **as** XElement;  XElement titleNode = el.Element(nss + "title");  XElement nsNode = el.Element(nss + "ns");  XElement idNode = el.Element(nss + "id");  XElement redirectNode = el.Element(nss + "redirect");  XElement revisionNode = el.Element(nss + "revision");  XElement textNode = revisionNode.Element(nss + "text");  **string** title = titleNode.Value;  **int** nsKey = **int**.Parse(nsNode.Value);  **int** id = **int**.Parse(idNode.Value);  **string** redirectTitle = redirectNode?.Attribute("title").Value;  **string** text = textNode.Value;  Page page = **new** Page(id, nsKey, redirectTitle, title, text);  **yield** **return** page;  }  } |

Далее требуется получить дамп википедии (если не скачан вручную), для этого нужно сформировать запрос:

|  |
| --- |
| **if** (keyWords.Text != "")  {  btnCancel.IsEnabled = **true**;  chooseSaveFolderBT.IsEnabled = **false**;  HttpWebRequest request = (HttpWebRequest)WebRequest.Create(URL);  Thread \_thread = **new** Thread(WriteToFile);  \_thread.Start();  **using** (HttpWebResponse response = (HttpWebResponse)request.GetResponse())  {  **using** (Stream receiveStream = response.GetResponseStream())  {  **await** **ProcessDump**(receiveStream);  }  }  }  **else**  {  MessageBox.Show("Вы не ввели ключевые слова. Искать нечего.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);  } |

После получения ответа можно парсить дамп:

|  |
| --- |
| **private** **async** Task **ProcessDump**(Stream inputStream)  {  btnOpenFile.IsEnabled = **false**;  btnDownloadFile1.IsEnabled = **false**;  progressBar.Visibility = Visibility.Visible;  **try**  {  **string**[] tempKeyWordsStringA = keyWords.Text.Split(' ');  **for** (**int** i = **0**; i < tempKeyWordsStringA.Length; i++)  {  KeyWords.Add(tempKeyWordsStringA[i]);  }  **using** (BZip2InputStream stream = **new** BZip2InputStream(inputStream))  {  Parser parser = Parser.Create(stream);  number\_of\_articles\_found = **0**;  **await** Task.Run(() =>  {  RunTimer();  pageCount = **0**;  **foreach** (**var** page **in** parser.ReadPages())  {  **if** (cancel)  {  cancel = **false**;  \_timer.Stop();  **break**;  }  pageCount++;  **foreach** (**string** keyWord **in** KeyWords)  {  **if** (page.Title.ToLower().Contains(keyWord) && page.Text != "" && !page.Text.ToLower().Contains("#redirect") && !page.Text.ToLower().Contains("#перенаправление"))  {  findedPages.Add(page.Title, page.Text);  number\_of\_articles\_found++;  Dispatcher.BeginInvoke(() => { UpdateStatus(); });  }  }  **if** (pageCount % **100** == **0**)  {  Dispatcher.BeginInvoke(() => { UpdateStatus(); });  }  }  });  UpdateStatus();  \_timer.Stop();  }  }  **catch** (OperationCanceledException) { }  **catch** (Exception ex)  {  MessageBox.Show(ex.Message);  }  **finally**  {  btnOpenFile.IsEnabled = **true**;  btnDownloadFile1.IsEnabled = **true**;  btnOpenFile.Visibility = Visibility.Visible;  btnDownloadFile1.Visibility = Visibility.Visible;  progressBar.Visibility = Visibility.Hidden;  **if** (!ifCanceled && !exit)  {  **await** **SaveConfig**();  }  MessageBox.Show("Поиск завершен: найдено " + number\_of\_articles\_found + " из " + pageCount + " статей." + "\nВремя: " + timer.Text, "Завершено");  }  } |

Поиск статей начинается сразу и идет по мере получения данных, поэтому поиск идет медленнее, т. к. скорость замедляется за счет скорости скачивания и последовательного извлечения данных.

Поэтому был реализован «оффлайн» метод парсинга, который пропускает задачу скачивания (вам требуется скачать дамп вручную).

Методы идентичны друг другу за исключением оптимизации:

|  |  |  |
| --- | --- | --- |
| **private** **async** Task **ReadDumpFromFile**(Stream inputStream)  {  btnOpenFile.IsEnabled = **false**;  btnDownloadFile1.IsEnabled = **false**;  progressBar.Visibility = Visibility.Visible;  ifCanceled = **false**;  **try**  {  **string**[] tempKeyWordsStringA = keyWords.Text.Split(' ');  **for** (**int** i = **0**; i < tempKeyWordsStringA.Length; i++)  {  KeyWords.Add(tempKeyWordsStringA[i]);  }  **using** (BZip2InputStream stream = **new** BZip2InputStream(inputStream))  {  Parser parser = Parser.Create(stream);  number\_of\_articles\_found = **0**;  **await** Task.Run(() =>  {  RunTimer();  pageCount = **0**;  Parallel.ForEach(parser.ReadPages(), (page, state) =>  {  **if** (cancel)  {  cancel = **false**;  \_timer.Stop();  ifCanceled = **true**;  state.Break();  }  pageCount++;  **foreach** (**string** keyWord **in** KeyWords)  {  **if** (page.Title.ToLower().Contains(keyWord) && page.Text != "" && !page.Text.ToLower().Contains("#redirect") && !page.Text.ToLower().Contains("#перенаправление"))  {  findedPages.Add(page.Title, page.Text);  number\_of\_articles\_found++;  Dispatcher.BeginInvoke(() => { UpdateStatus(); });  }  }  **if** (pageCount % **100** == **0**)  {  Dispatcher.BeginInvoke(() => { UpdateStatus(); });  }  });  });  UpdateStatus();  \_timer.Stop();  }  }  **catch** (OperationCanceledException) { }  **catch** (Exception ex)  {  MessageBox.Show(ex.Message);  }  **finally**  {  btnOpenFile.IsEnabled = **true**;  btnDownloadFile1.IsEnabled = **true**;  btnOpenFile.Visibility = Visibility.Visible;  btnDownloadFile1.Visibility = Visibility.Visible;  progressBar.Visibility = Visibility.Hidden;  **if** (!ifCanceled && !exit)  {  **await** **SaveConfig**();  }  MessageBox.Show("Поиск завершен: найдено " + number\_of\_articles\_found + " из " + pageCount + " статей." + "\nВремя: " + timer.Text, "Завершено");  }  }    Ускорение поиска происходит за счет параллельного извлечения данных из xml-файла:   |  | | --- | | Parallel.ForEach(parser.ReadPages(), (page, state) =>  {  … |   Найденные статьи сохраняются в текстовом формате, в указанное пользователем расположение:   |  | | --- | | **private** **void** **WriteToFile**()  {  **try**  {  **string** line = "";  **string** fileName = "";  **while** (**true**)  {  **if** (findedPages.Count > **0**)  {  **foreach** (KeyValuePair<**string**, **string**> fp **in** findedPages.ToArray())  {  fileName = (**string**)fp.Key;  line = (**string**)fp.Value;  **if** (line != "" && line != " ")  {  **try**  {  fileName = Regex.Replace(fileName, @"[/\\\|:""']", " ");  **if** (!File.Exists(path + @"\" + fileName + ".txt"))  {  **using** (sw = **new** StreamWriter(path + @"\" + fileName + ".txt"))  {  sw.Write(line);  sw.Close();  }  }  }  **catch** (Exception e)  {  MessageBox.Show("Exception: " + e.Message);  }  //numberOfArticleWrited++;  }  }  }  }  }  **catch** (Exception e)  {  MessageBox.Show("Exception: " + e.Message);  }  } | |

## Экспериментальное исследование программы парсинга википедии

Экспериментальное исследование проводилось на ПК с данными характеристиками:

Процессор: AMD Ryzen 5 3600 6-Core Processor 4.40 GHz

Кэш-память: 32 MB

Оперативная память: 16,0 ГБ

Тип системы: Windows 10, 64-разрядная операционная система, процессор x64

Программа запускалась несколько раз с ключевым словом «автомобиль».

Данные тестирования занесены в таблицы 2.2.1.1, 2.2.2.1, которые состоят из:

1) Номер тестирование

2) Время на поиск статей

3) Количество найденных статей

4) Всего статей

### Тестирование онлайн-метода

|  |  |  |  |
| --- | --- | --- | --- |
| № | Время (hh:mm:ss) | Найдено | Всего |
| 1 | 40:51 (2 451c) | 975 | 5 000 497 |
| 2 | 1:03:06 (3 786c) | 975 | 5 000 497 |
| 3 | 21:35 (1 295c) | 975 | 5 000 497 |
| 4 | 21:51 (1 311c) | 975 | 5 000 497 |
| Средний | 36:50 (2 210,75c) | 975 | 5 000 497 |

Таблица 2.2.1.1

### Тестирование оффлайн-метода

Время, затраченное на скачивание дампа вручную – 17 минут.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Время (hh:mm:ss) | Найдено | Всего |
| 1 | 13:21 (801с) | 975 | 5 000 497 |
| 2 | 13:52 (832с) | 975 | 5 000 497 |
| 3 | 13:23 (803с) | 975 | 5 000 497 |
| 4 | 13:28 (808с) | 975 | 5 000 497 |
| Средний | 13:31 (811с) | 975 | 5 000 497 |

Таблица 2.2.2.1

Общее время с учетом скачивания дампа – 30 минут 32 секунды.

### Выводы по разделу

Как видно из таблиц, поиск статей, по ключевым словам, «онлайн» методом уступает в скорости «оффлайн» методу, но вам не нужно скачивать дамп вручную, а также хранить его для запуска программы.

# Заключение

Разработанное приложение имеет вид WPF приложения, интерфейс интуитивно понятный, а каждый активный элемент имеет подсказку.

В ходе разработки программы парсинга википедии были получены знания по языку разметки XAML, а также навыки работы с C# и википедией.

# Библиографический список

1. bzip.org [Электронный ресурс] – URL: http://www.bzip.org/ (дата обращения: 13.12.2021).
2. Xml-ql: A query language for xml. Daniela Florescu.
3. R. Richards, Pro PHP XML and Web Services, DOI 10.1007/978-1-4302-0139-7\_9, © 2006 by Robert Richards
4. XML in .NET: .NET Framework XML Classes and C# Offer Simple, Scalable Data Manipulation. Aaron Skonnard.
5. ИНФОРМАЦИОННЫЕ РЕСУРСЫ И СИСТЕМЫ В ЭКОНОМИКЕ, НАУКЕ И ОБРАЗОВАНИИ: cборник статей IX Международной научно-практической конференции. – Пенза: Приволжский Дом знаний, 2019. – 216 с. ISBN 978-5-8356-1723-4
6. dumps.wikimedia.org [Электронный ресурс] – URL: <https://dumps.wikimedia.org/> (дата обращения: 13.12.2021).

# Приложения

## Приложение А. Листинг программы

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows;

using System.IO;

using System.Threading;

using System.Windows.Threading;

using WikiDumpParser;

using ICSharpCode.SharpZipLib.BZip2;

using System.Net;

using Microsoft.Win32;

using System.Diagnostics;

using System.Text.RegularExpressions;

using System.Text.Json;

namespace Demo

{

//https://dumps.wikimedia.org/ruwiki/20211201/ruwiki-20211201-pages-articles-multistream.xml.bz2

//https://dumps.wikimedia.org/ruwiki/latest/ruwiki-latest-pages-articles.xml.bz2

//https://dumps.wikimedia.org/ruwiki/latest/ruwiki-latest-pages-articles-multistream.xml.bz2

public class Config

{

//public DateTimeOffset Date { get; set; }

public double count\_pages { get; set; }

}

public partial class MainWindow : Window

{

private readonly string URL = "https://dumps.wikimedia.org/ruwiki/latest/ruwiki-latest-pages-articles.xml.bz2";

DispatcherTimer \_timer; string ConfigFileName = "config.json";

TimeSpan \_time;

string path; int pageCount = 0; int number\_of\_articles\_found = 0; bool cancel = false;

Dictionary<string, string> findedPages = new Dictionary<string, string>();

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

StreamWriter sw;

bool configLoaded = false; bool exit = false; bool ifCanceled = false;

Config config = new Config();

public MainWindow()

{

InitializeComponent();

}

private void WriteToFile()

{

try

{

string line = "";

string fileName = "";

while (true)

{

if (findedPages.Count > 0)

{

foreach (KeyValuePair<string, string> fp in findedPages.ToArray())

{

fileName = (string)fp.Key;

line = (string)fp.Value;

if (line != "" && line != " ")

{

try

{

fileName = Regex.Replace(fileName, @"[/\\\|:""']", " ");

if (!File.Exists(path + @"\" + fileName + ".txt"))

{

using (sw = new StreamWriter(path + @"\" + fileName + ".txt"))

{

sw.Write(line);

sw.Close();

}

}

}

catch (Exception e)

{

MessageBox.Show("Exception: " + e.Message);

}

//numberOfArticleWrited++;

}

}

}

}

}

catch (Exception e)

{

MessageBox.Show("Exception: " + e.Message);

}

}

private void RunTimer()

{

\_time = TimeSpan.FromSeconds(0);

\_timer = new DispatcherTimer(new TimeSpan(0, 0, 1), DispatcherPriority.Normal, delegate

{

timer.Text = \_time.ToString("c");

\_time = \_time.Add(TimeSpan.FromSeconds(+1));

}, Application.Current.Dispatcher);

\_timer.Start();

}

private async Task ReadDumpFromFile(Stream inputStream)

{

btnOpenFile.IsEnabled = false;

btnDownloadFile1.IsEnabled = false;

progressBar.Visibility = Visibility.Visible;

ifCanceled = false;

try

{

string[] tempKeyWordsStringA = keyWords.Text.Split(' ');

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

{

KeyWords.Add(tempKeyWordsStringA[i]);

}

using (BZip2InputStream stream = new BZip2InputStream(inputStream))

{

Parser parser = Parser.Create(stream);

number\_of\_articles\_found = 0;

await Task.Run(() =>

{

RunTimer();

pageCount = 0;

Parallel.ForEach(parser.ReadPages(), (page, state) =>

{

if (cancel)

{

cancel = false;

\_timer.Stop();

ifCanceled = true;

state.Break();

}

pageCount++;

foreach (string keyWord in KeyWords)

{

if (page.Title.ToLower().Contains(keyWord) && page.Text != "" && !page.Text.ToLower().Contains("#redirect") && !page.Text.ToLower().Contains("#перенаправление"))

{

findedPages.Add(page.Title, page.Text);

number\_of\_articles\_found++;

Dispatcher.BeginInvoke(() => { UpdateStatus(); });

}

}

if (pageCount % 100 == 0)

{

Dispatcher.BeginInvoke(() => { UpdateStatus(); });

}

});

});

UpdateStatus();

\_timer.Stop();

}

}

catch (OperationCanceledException) { }

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

finally

{

btnOpenFile.IsEnabled = true;

btnDownloadFile1.IsEnabled = true;

btnOpenFile.Visibility = Visibility.Visible;

btnDownloadFile1.Visibility = Visibility.Visible;

progressBar.Visibility = Visibility.Hidden;

if (!ifCanceled && !exit)

{

await SaveConfig();

}

MessageBox.Show("Поиск завершен: найдено " + number\_of\_articles\_found + " из " + pageCount + " статей." + "\nВремя: " + timer.Text, "Завершено");

}

}

private async Task ProcessDump(Stream inputStream)

{

btnOpenFile.IsEnabled = false;

btnDownloadFile1.IsEnabled = false;

progressBar.Visibility = Visibility.Visible;

try

{

string[] tempKeyWordsStringA = keyWords.Text.Split(' ');

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

{

KeyWords.Add(tempKeyWordsStringA[i]);

}

using (BZip2InputStream stream = new BZip2InputStream(inputStream))

{

Parser parser = Parser.Create(stream);

number\_of\_articles\_found = 0;

await Task.Run(() =>

{

RunTimer();

pageCount = 0;

foreach (var page in parser.ReadPages())

{

if (cancel)

{

cancel = false;

\_timer.Stop();

break;

}

pageCount++;

foreach (string keyWord in KeyWords)

{

if (page.Title.ToLower().Contains(keyWord) && page.Text != "" && !page.Text.ToLower().Contains("#redirect") && !page.Text.ToLower().Contains("#перенаправление"))

{

findedPages.Add(page.Title, page.Text);

number\_of\_articles\_found++;

Dispatcher.BeginInvoke(() => { UpdateStatus(); });

}

}

if (pageCount % 100 == 0)

{

Dispatcher.BeginInvoke(() => { UpdateStatus(); });

}

}

});

UpdateStatus();

\_timer.Stop();

}

}

catch (OperationCanceledException) { }

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

finally

{

btnOpenFile.IsEnabled = true;

btnDownloadFile1.IsEnabled = true;

btnOpenFile.Visibility = Visibility.Visible;

btnDownloadFile1.Visibility = Visibility.Visible;

progressBar.Visibility = Visibility.Hidden;

if (!ifCanceled && !exit)

{

await SaveConfig();

}

MessageBox.Show("Поиск завершен: найдено " + number\_of\_articles\_found + " из " + pageCount + " статей." + "\nВремя: " + timer.Text, "Завершено");

}

}

private void UpdateStatus()

{

if (configLoaded && config.count\_pages > pageCount)

{

progressBar.Value = Math.Truncate((100 \* (double)pageCount) / config.count\_pages);

countPages.Text = $"Найдено: {number\_of\_articles\_found} | {progressBar.Value}%";

}

else

{

config.count\_pages = pageCount;

progressBar.Value = 20;

progressBar.IsIndeterminate = true;

countPages.Text = $"Найдено: {number\_of\_articles\_found}";

}

txtProgress.Text = $"Страниц: {pageCount}";

}

private async Task SaveConfig ()

{

try

{

var options = new JsonSerializerOptions

{

WriteIndented = true,

};

config.count\_pages = pageCount;

using FileStream createStream = File.Create(ConfigFileName);

await JsonSerializer.SerializeAsync(createStream, config, options);

}

catch (Exception ex)

{

MessageBox.Show("Ошибка сохранения конфига.\n" + ex.Message, "Error");

}

}

private async Task LoadConfig()

{

try

{

if (File.Exists(ConfigFileName))

{

using FileStream openStream = File.OpenRead(ConfigFileName);

config = await JsonSerializer.DeserializeAsync<Config>(openStream);

configLoaded = true;

}

else

{

MessageBox.Show("Конфиг не найден, будет создан новый.");

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private async void BtnParse\_Click(object sender, RoutedEventArgs e)

{

if (keyWords.Text != "")

{

OpenFileDialog dlg = new OpenFileDialog()

{

Filter = "latest articles bz2|\*.bz2",

};

if (dlg.ShowDialog() != true)

{

return;

}

Thread \_thread = new Thread(WriteToFile);

\_thread.Start();

using (Stream fs = dlg.OpenFile())

{

path2TB.Text = dlg.FileName;

chooseSaveFolderBT.IsEnabled = false;

btnCancel.IsEnabled = true;

path2TB.Visibility = Visibility.Visible;

path2TB.IsEnabled = true;

await ReadDumpFromFile(fs);

}

}

else

{

MessageBox.Show("Вы не ввели ключевые слова. Искать нечего.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);

}

}

private void BtnDownloadFile\_Click(object sender, RoutedEventArgs e){}

private void BtnCancel\_Click(object sender, RoutedEventArgs e)

{

cancel = true;

btnCancel.IsEnabled = false;

}

private async void btnDownloadFile1\_Click(object sender, RoutedEventArgs e)

{

if (keyWords.Text != "")

{

btnCancel.IsEnabled = true;

chooseSaveFolderBT.IsEnabled = false;

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(URL);

Thread \_thread = new Thread(WriteToFile);

\_thread.Start();

using (HttpWebResponse response = (HttpWebResponse)request.GetResponse())

{

using (Stream receiveStream = response.GetResponseStream())

{

await ProcessDump(receiveStream);

}

}

}

else

{

MessageBox.Show("Вы не ввели ключевые слова. Искать нечего.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);

}

}

private void ProgressBar\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e) { }

private void Button\_Click(object sender, RoutedEventArgs e){}

private void chooseSaveFolderBT\_Click(object sender, RoutedEventArgs e)

{

var dialog = new Ookii.Dialogs.Wpf.VistaFolderBrowserDialog();

var local\_path\_ = System.IO.Path.GetDirectoryName(Process.GetCurrentProcess().MainModule.FileName);

dialog.SelectedPath = local\_path\_;

if (dialog.ShowDialog(this).GetValueOrDefault())

{

path = dialog.SelectedPath;

pathTB.Text = path;

btnOpenFile.IsEnabled = true;

btnDownloadFile1.IsEnabled = true;

pathTB.Visibility = Visibility.Visible;

}

}

private void WDP\_Closed(object sender, EventArgs e)

{

exit = true;

Environment.Exit(0);

}

private async void WDP\_Loaded(object sender, RoutedEventArgs e)

{

btnOpenFile.ToolTip = "Быстрее.";

btnDownloadFile1.ToolTip = "Медленнее.";

btnOpenFile.Visibility = Visibility.Visible;

progressBar.Visibility = Visibility.Hidden;

btnCancel.IsEnabled = false;

keyWords.ToolTip = "Введите слова через пробел.";

chooseSaveFolderBT.ToolTip = "Выбрать место для сохранения статей.";

btnCancel.ToolTip = "Прекратить поиск.";

path2TB.Visibility = Visibility.Hidden;

pathTB.Visibility = Visibility.Hidden;

await LoadConfig();

}

}

}

using System.Collections.Generic;

using System.IO;

using System.Xml;

using System.Xml.Linq;

using System.Xml.Serialization;

using WikiDumpParser.Models;

namespace WikiDumpParser

{

public class Parser

{

readonly XmlReader xmlReader;

public Siteinfo SiteInfo { get; private set; }

private Parser(Stream stream)

{

xmlReader = XmlReader.Create(stream);

}

private void ReadSiteInfo()

{

xmlReader.ReadToFollowing("siteinfo");

XmlSerializer serializer = new XmlSerializer(typeof(Siteinfo));

SiteInfo = (Siteinfo)serializer.Deserialize(xmlReader);

}

public IEnumerable<Page> ReadPages()

{

XNamespace nss = "http://www.mediawiki.org/xml/export-0.10/";

while (xmlReader.ReadToFollowing("page"))

{

XElement el = XNode.ReadFrom(xmlReader) as XElement;

XElement titleNode = el.Element(nss + "title");

XElement nsNode = el.Element(nss + "ns");

XElement idNode = el.Element(nss + "id");

XElement redirectNode = el.Element(nss + "redirect");

XElement revisionNode = el.Element(nss + "revision");

XElement textNode = revisionNode.Element(nss + "text");

string title = titleNode.Value;

int nsKey = int.Parse(nsNode.Value);

int id = int.Parse(idNode.Value);

string redirectTitle = redirectNode?.Attribute("title").Value;

string text = textNode.Value;

Page page = new Page(id, nsKey, redirectTitle, title, text);

yield return page;

}

}

public static Parser Create(Stream stream)

{

Parser parser = new Parser(stream);

parser.ReadSiteInfo();

return parser;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WikiDumpParser.Models

{

public class Page

{

public Page(int id, int namespaceKey, string redirect, string title, string text)

{

Id = id;

Namespace = namespaceKey;

Redirect = redirect;

Title = title;

Text = text;

}

public int Id { get; }

public int Namespace { get; }

public string Redirect { get; }

public string Title { get; }

public string Text { get; }

public bool IsRedirect

{

get { return Redirect != null; }

}

}

}