Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

СЕТЕВОЕ ПРОГРАММНОЕ СРЕДСТВО   
«ЭЛЕКТРОННАЯ БИБЛИОТЕКА»

БГУИР КР I-40 01 01 628 ПЗ

Студент: гр.351006 Шульга Е.С.

Руководитель: Трус В.В.

Минск 2015

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc421290240)

[1 ПОСТАНОВКА ЗАДАЧИ 6](#_Toc421290241)

[2 обзор аналогов 7](#_Toc421290242)

[2.1 MyHomeLib 7](#_Toc421290243)

[2.2 MyRuLib 8](#_Toc421290244)

[3 Описание форматов данных 9](#_Toc421290245)

[3.1 Хранение файлов книг 9](#_Toc421290246)

[3.2 Метаданные коллекций книг 11](#_Toc421290247)

[3.3 Список жанров 14](#_Toc421290248)

[4 15](#_Toc421290249)

[5 разработка программного приложения 16](#_Toc421290250)

[5.1 Разработка структур данных 16](#_Toc421290251)

[5.2 Модели данных алгоритмов 20](#_Toc421290252)

[5.3 Реализация алгоритмов 21](#_Toc421290253)

[6 Руководство по использованию приложения 24](#_Toc421290254)

[6.1 Сжатие файлов 24](#_Toc421290255)

[6.2 Распаковка файлов 28](#_Toc421290256)

[6.3 Исследование алгоритмов 30](#_Toc421290257)

[6.4 Просмотр статистики 30](#_Toc421290258)

[7 Исследование алгоритмов 32](#_Toc421290259)

[Заключение 34](#_Toc421290260)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 35](#_Toc421290261)

[ПРИЛОЖЕНИЕ А (обязательное) Исходные коды функций 36](#_Toc421290262)

[ПРИЛОЖЕНИЕ Б (обязательное) Схемы алгоритмов 44](#_Toc421290263)

# ВВЕДЕНИЕ

Долгое время единственным источником распространения и сохранения информации являлась обычная печатная книга. Однако в связи с бурным развитием технологий, в настоящее время возникло такое явление как информационный взрыв[[1]](#footnote-1)). Нарастающее число публикаций поставило перед библиотеками вопрос хранения столь большого массива информации. Появление электронной публикации позволяет не печатать информацию в виде бумажных книг, а хранить её на гораздо более ёмких электронных носителях в электронных библиотеках [1].

В свою очередь развитие Интернета способствует широкому распространению информации. Электронные библиотеки имеют возможность предоставления доступа к книгам дистанционно.

Также электронные книги решают и другие задачи, такие как упрощение получения книги читателем, уменьшение стоимости издания и распространения книги и т.д. [2]

Согласно некоторым источникам, первая в истории электронная книга была набрана на компьютере в 1971 году Майклом Хартом. В дальнейшем коллекция электронных материалов продолжала пополняться. Была создана первая общедоступная электронная библиотека, в настоящее время известная как проект «Гутенберг» [3].

Одной из крупнейших электронных библиотек русскоязычного сегмента Интернета является Либрусек [4]. Принцип его работы основан на википодобной архитектуре, то есть наполнением сайта, исправлением ошибок занимаются пользователи. Это обеспечивает высокую степень актуальности и обновляемости библиотеки.

В настоящее время пользователи библиотеки начали заниматься независимыми проектами, связанными с Либрусеком. Например, появились обновляемые копии базы книг, программы для работы с ними вне сети Интернет [4]. Однако, такие программы для своей работы требуют наличия у пользователя всей базы книг, что приводит к нерациональному использованию дискового пространства.

В данной курсовой работе предлагается решение данной проблемы путем размещения базы книг на одном компьютере в сети и организации доступа к этой базе с других компьютеров; предлагается реализация сетевой электронной библиотеки, которая может быть использована в локальных сетях как зеркало сайта Либрусек.

# ПОСТАНОВКА ЗАДАЧИ

Входными данными для работы являются упакованные файлы книг, список метаданных книг в специальном формате и список жанров книг. Необходимо изучить форматы хранения этих данных и организовать простой и понятный сетевой интерфейс доступа к ним.

В рамках курсового проекта необходимо спроектировать и реализовать программное средство, включающее в себя клиентское и серверное приложение.

Серверное приложение должно быть реализовано как веб-сервис, осуществляющий импорт входных данных, организующий их хранение и предоставляющий программный интерфейс для удаленного доступа к этим данным без права их модификации: поиск записей книг по шаблону, получение списка найденных книг, получение файла книги. Также необходимо обеспечить возможность получения информации о запросах.

Реализовать графический интерфейс клиентского приложения, предоставляющее возможность формирования шаблона для поиска, отображения полученного списка книг, возможность выбора книги из списка и ее скачивания. Также необходимо реализовать функцию автообнаружения сервера, если клиент и сервер находятся в одной сети.

Обе части приложения должны иметь возможность простого развертывания на компьютерах пользователей.

Для закрепления знаний, полученных на учебных занятиях в рамках курсов КСиС, ООТПиСП, ВебТех и ТРПО, языком программирования был выбран язык C#, для проектирования клиент-серверной архитектуры была выбрана технология WCF, для разработки графического интерфейса была выбрана технология WPF.

# обзор аналогов

В связи с тем, что большинство людей предпочитает не иметь большие коллекции книг на диске, программ для организации домашних электронных библиотек существует не так много. Рассмотрим наиболее распространенные.

## MyHomeLib

**2.1.1** Данное программное средство является наиболее используемым при организации домашней библиотеки в странах СНГ. Данное приложение может использоваться и для каталогизации собственной коллекции книг пользователя, и как клиент для работы с копией библиотеки Либрусек (и других библиотек, использующих такие же форматы служебных файлов).

**2.1.2** Главная функциональная возможность данного приложения – поиск по автору, названию, жанру и другим параметрам книг в коллекции. Пример окна поиска приведен на рисунке 2.1.

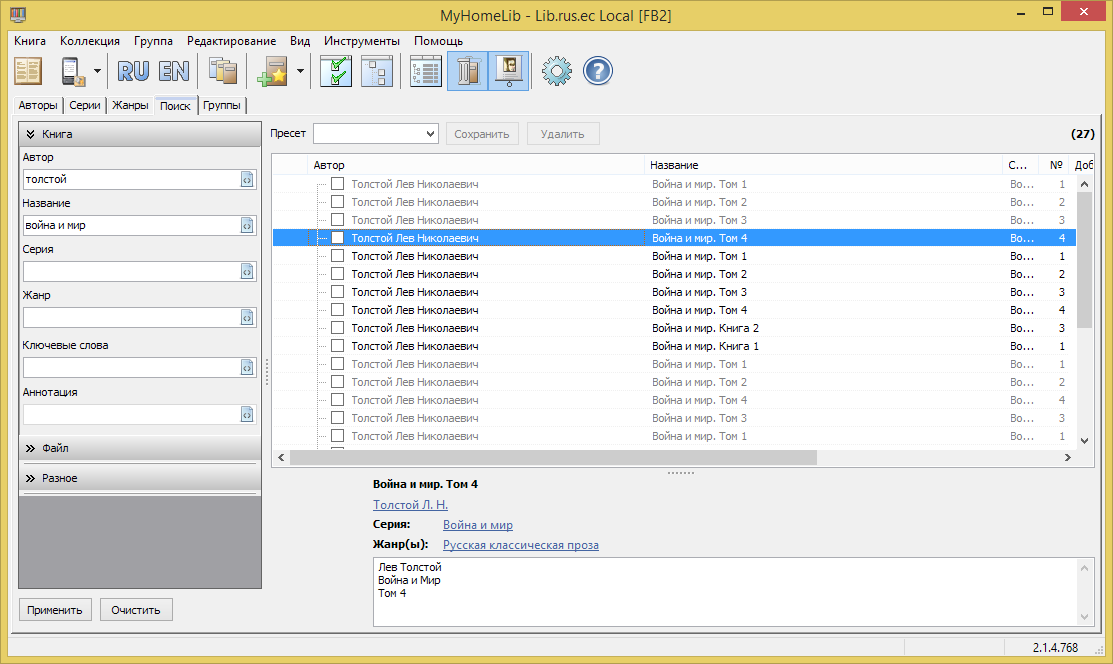


Рисунок 2.1 – Поиск книг в программном средстве MyHomeLib

**2.1.3** После нахождения требуемой книги программное средство осуществляет его экспорт в заранее определенную папку на диске. Возможность чтения книг появляется после экспорта с помощью сторонних приложений.

**2.1.4** Основными недостатками данного программного средства является отсутствие возможности работы по сети и медленное выполнение следующих операций: запуск программы, поиск книг, отображение списков.

## MyRuLib

**2.2.1** Данное программное средство является кроссплатформенным аналогом MyHomeLib. Так же может использоваться для организации и управления домашней библиотекой и для доступа к копии библиотеки Либрусек.

**2.2.2** MyRuLib так же предоставляет возможность поиска по следующим критериям: автор, название. Пример работы программы (поиска книг) представлен на рисунке 2.2.

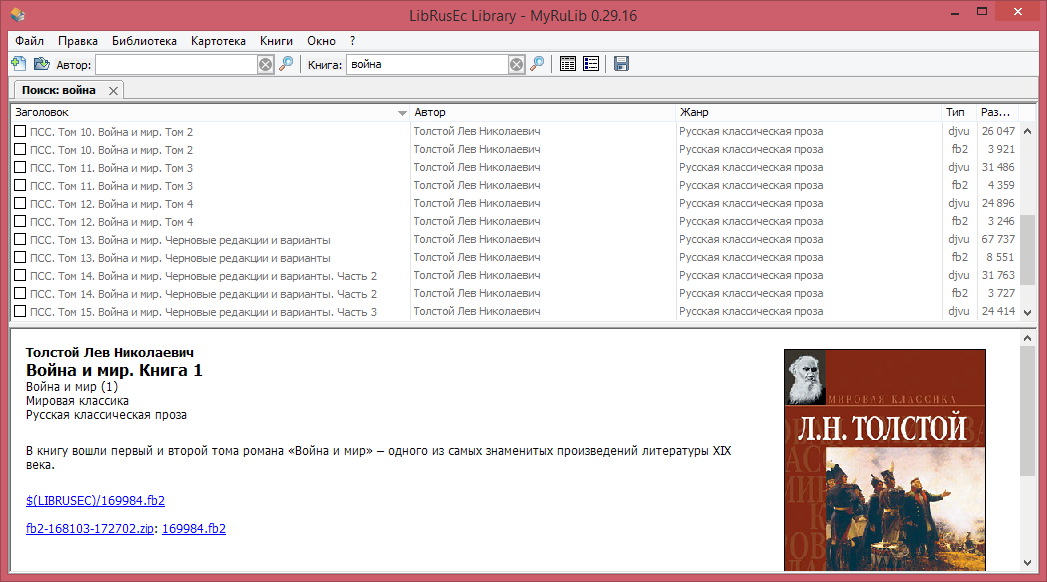


Рисунок 2.2 – Поиск книг в программном средстве MyRuLib

**2.2.3** Данному программному средству присущи следующие недостатки:

* Долгое время импорта книг. Это связано с тем, что MyRuLib не использует файлы метаданных, предоставляемые сетевыми электронными библиотеками.
* Невозможность поиска одновременно и по автору, и по названию (отсутствие возможности задания шаблона для поиска).
* Отсутствие возможности выбора папки для экспорта и неудобно реализованная возможность сохранения книг на диск. Вместо этого при выборе книг предлагается сразу запустить приложение для чтения (такие приложения могут быть не установлены у пользователя).

# Описание форматов данных

В настоящее время люди, занимающиеся распространением электронных книг, организовали унифицированную форму распространения книг из нескольких электронных библиотек. Кроме Либрусека, в таком же формате распространяются копии веб-сайта Флибуста [6] и некотороые другие. Кроме того, и для Либрусека, и для Флибусты существует несколько вариантов распространения: *fb2* копии, содержащие только книги формата .fb2, и *usr* копии, которые могут содержать книги различных форматов: doc, rtf, txt, pdf, epub и т.д.

Перед началом проектирования и разработки необходимо описать структуру входных данных.

## Хранение файлов книг

**3.1.1** В связи с большим размером коллекций целесообразно оптимизировать способы их хранения. Принято использовать упаковку книг в zip архивы (пример папки хранилища книг можно увидеть на рисунке 2.1).

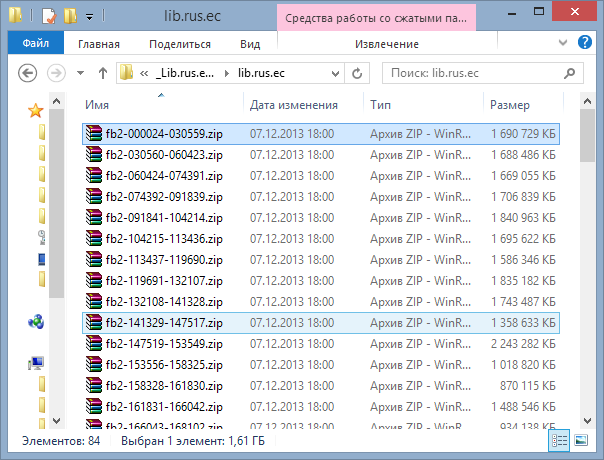


Рисунок 2.1 – Пример папки-хранилища книг

**3.1.2** Можно увидеть следующее правило формирования имен архивов:

<Тип-архива>-<ID-первой-книги>-<ID-последней-книги>.zip

Причем <Тип-архива> может быть *fb2* или *usr* (для коллекций разных форматов). Две другие части имени используются для упрощения управлением коллекцией и для уникальной идентификации каждого архива. Кроме того, как будет сказано ниже, имена архивов и имена файлов метаданных, их описывающих, должны совпадать.

**3.1.3** На рисунке 2.2 можно увидеть содержание одного из архивов.

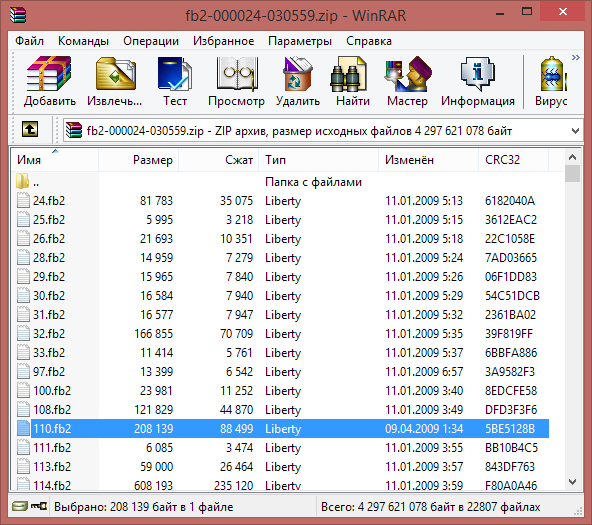


Рисунок 2.2 – Пример содержания архива из хранилища книг

Файлы, представленные на рисунке 2, непосредственно являются книгами. Можно просто распаковывать архивы и выбирать необходимые книги. Однако это неудобно, так как имена файлов книг ничего не говорят об авторе, или названии, или содержании книги – вместо этого используется числовой ключ (при создании пользовательских коллекций имена могут быть произвольными, однако это повлияет на файлы метаданных).

Для коллекций библиотек Либрусека и Флибусты данные ключи используются для уникальной идентификации каждой книги (утверждается что они будут уникальными).

Для организации доступа к книгам из этих архивов и разрабатывается данная курсовая работа.

## Метаданные коллекций книг

**3.2.1** Для описания коллекций книг был разработан специальный формат файлов. В настоящее время его используют для создания коллекций копий веб-сайтов электронных библиотек. Также его целесообразно использовать при каталогизации обширных собственных библиотек. Файлы этого формата имеют расширение inpx.

По своей структуре файлы данного формата представляют собой zip-архив, содержащий файлы с расширением inp. В качестве примера на рисунке 2.3 приведена структура inpx файла библиотеки Либрусек.

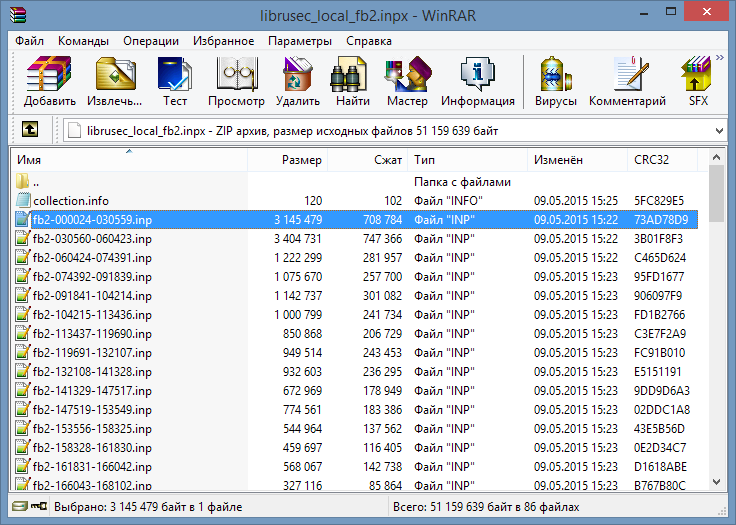


Рисунок 2.3 – Пример содержания inpx файла

В качестве первого файла можно увидеть файл collection.info. Он является необязательным и содержит комментарий, относящийся к данной коллекции.

**3.2.2** Как было сказано в пункте 2.1, имена .inp файлов должны совпадать с именами архивов, к которым они относятся.

Данные файлы являются текстовыми в формате UTF-8. Они построчно содержат информацию обо всех книгах, содержащихся в коллекции. Пример такого файла приведен на рисунке 2.4.



Рисунок 2.4 – Пример inp файла

Кроме того, утверждается, что количество строк такого файла и число книг в соответствующем архиве совпадает, то есть каждая книга имеет свою запись в файле метаданных (безошибочность достигается тем, что создание inpx файла производится автоматически, а не вручную).

**3.2.3** Каждая строка содержит определенным образом расположенные метаданные. Некоторые из этих полей необязательны, то есть они могут содержать пустую строку (тем не менее они должны присутствовать), некоторые поля могут содержать несколько значений. Поля разделяются символом с кодом 04 (EOT – End of Transmission). Названия и описания полей приведены в таблице 2.1.

**3.2.4** В формате .inp файла реализована возможность наличия у книги нескольких (не менее одного) авторов. Признаком конца имени автора является символ с кодом 58 (':'), разделителем фамилии, имени и отчества служит символ с кодом 44 (','):

<Фамилия1>,<Имя1>,<Отчество1>:<Фамилия2>,<Имя2>,<Отчество2>:

Поля фамилии, имени и отчества не являются обязательными и могут быть пропущены. Наличие разделителей полей обязательно.

Таблица 2.1 – Поля записи книги в файле метаданных

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | Условное обозначение | Название | Примечание |
| 0 | AUTHOR | Полное имя автора | См. 2.2.4 |
| 1 | GENRE | Жанр | См. 2.2.5 |
| 2 | TITLE | Название |  |
| 3 | SERIES | Название серии | Может быть не заполнено |
| 4 | SERNO | Номер в серии | Может быть не заполнено |
| 5 | FILE | Имя файла в архиве | Для Либрусека совпадает с полем LIBID |
| 6 | SIZE | Размер файла | Размер файла книги в байтах |
| 7 | LIBID | ID книги | Совпадает с FILE; уникально идентифицирует каждую книгу |
| 8 | DEL | Индикатор удаления файла | См. 2.2.6 |
| 9 | EXT | Расширение файла | Для fb2 коллекций – всегда fb2 |
| 10 | DATE | Дата добавления | Дата добавления в базу Либрусека; может быть не заполнено |
| 11 | LANG | Язык книги | Может быть не заполнено |
| 12 | LIBRATE | Внешний рейтинг | Рейтинг книги на сайте Либрусек; обычно не актуально и не заполнено |
| 13 | KEYWORDS | Ключевые слова | Как правило не заполнены |

**3.2.5** Поле жанров, как и поле авторов, может содержать несколько значений; признаком окончания жанра также служит символ с кодом 58 (':'). Однако, это поле принимает одно из предопределенных значений, причем эти значения инициализируются из специального файла жанров. Список жанров формируется библиотекой Либрусек. Подробнее про инициализацию списка жанров см. пункт 2.3.

**3.2.6** Индикатор удаления файла является служебной информацией. Для большинства книг он установлен в 0 (то есть книга не удалена). Если же индикатор установлен в 1, то это может означать, что с книгой что-то произошло: файл был добавлен, а затем удален модераторами, или имеется другая версия книги с существенно лучшим качеством. Тем не менее, если книга уже попала в базу книг, то она не удаляется, а помечается этим флагом.

## Список жанров

**3.3.1** В большие сетевые библиотеки пользователи постоянно добавляют новые книги, и необходимо обеспечить их категоризацию по жанрам. Может так получиться, что новая книга не принадлежит ни одному из заранее определенных жанров, может понадобится расширение списка.

Список жанров предоставляется электронной библиотекой, поэтому при создании копии какой-либо библиотеки рекомендуется использовать актуальный список жанров именно этой библиотеки.

Рассмотрим формат файла списка жанров библиотеки Либрусек.

**3.3.2** Файлы списка жанров имеют расширение .glst. Файлы этого формата являются текстовыми; в них построчно размещена информация о каждом жанре. Формат каждой строки:

<Номер-группы-жанра>.<Номер-жанра>.<Номер-поджанра> <Имя-жанра>;<Описание-жанра>

**3.3.3** Существует возможность упорядочивания по трёхуровневой иерархической структуре: группа жанра, жанр, поджанр. Фактически при этом признак, который устанавливается каждой книге, – это поджанр. Тогда группировка по жанрам и группам жанров является условной, облегчающую присвоение записям книг данного признака.

Номер группы жанра, номер жанра, номер поджанра разделены между собой символом с кодом 46 ('.'); после номера поджанра следует символ с кодом 32 (' ').

**3.3.4** Поле имени жанра содержит сокращенное название жанра. Обычно оно записывается с использованием букв латинского алфавита и знаков подчеркивания, например: detective, det\_espionage, vers\_libre и т.д. Данное поле служит для упрощения идентификации каждого жанра.

**3.3.5** За полем имени жанра и до конца строки следует поля описания жанра. В этом поле принято размещать название поджанра в удобном для чтения пользователем виде. Описание поджанров на сайте Либрусек принято заполнять на русском языке, например: Советская классическая проза, Киносценарии, Европейская старинная литература и т.д.

**3.3.6** Формат файлов предусматривает также размещение комментариев. Комментарием является текст, ограниченный слева символом с кодом 35 ('#') и справа концом строки. Комментарии могут быть полезны при группировке поджанров по жанрам и группам. Пример комментария:

#---------- 0.2 Детективы и Триллеры -----------------------

# проектирование программного средства

## Проектирование архитектуры

## Проектирование сервера

## Проектирование клиента

# разработка программного приложения

# Руководство по использованию приложения

При запуске приложения отображается начальное окно, приведенное на рисунке 1.

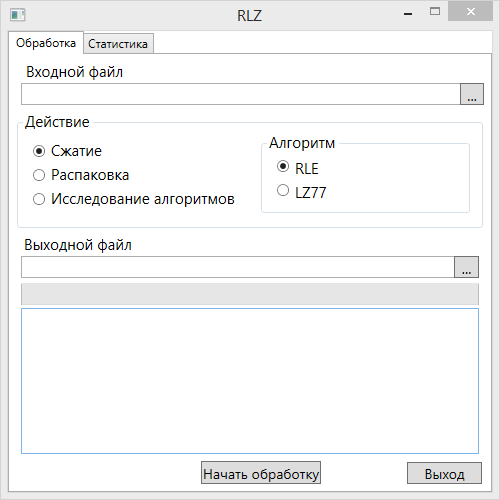


Рисунок 1 – Начальное окно приложения

С помощью данного программного средства можно выполнять следующие действия:

* выполнять сжатие любого файла одним из алгоритмов;
* выполнять распаковку ранее сжатого файла;
* проводить исследование алгоритмов на каком-либо файле;
* просматривать собранную статистику по исследованиям.

## Сжатие файлов

Для начала необходимо выбрать режим «Действие» – «Сжатие».

Затем необходимо выбрать, каким алгоритмом будет осуществляться сжатие входного файла: «RLE» или «LZ77».

Затем необходимо выбрать входной файл на диске. Для этого нужно нажать кнопку «…», расположенную справа от поля «Входной файл». При этом открывается окно, представленное на рисунке 2.

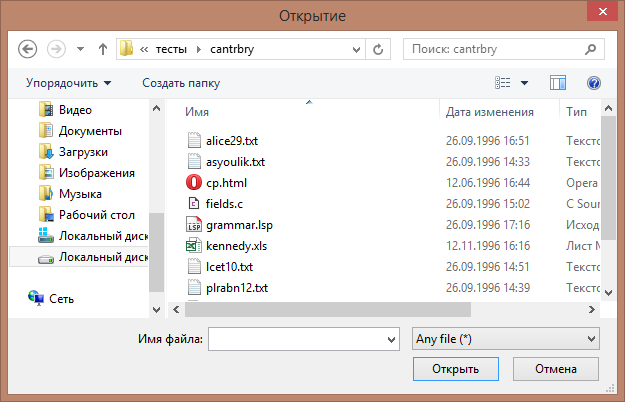


Рисунок 2 – Окно открытия файла при сжатии

При этом автоматически заполняются поля путей входного и выходного файла, причем предлагается сохранить сжатый файл в той же директории (но с добавленным расширением «.rlz»). Однако пользователь может сохранить выходной файл в любое другое место на диске. Для этого необходимо нажать кнопку «…», расположенную справа от поля пути выходного файла. При этом открывается окно, представленное на рисунке 3.

Затем пользователь, нажав клавишу «Enter» на клавиатуре или кнопку «Начать обработку» на форме, может запустить обработку входного файла; если входной или выходной файл не были выбраны, то появится соответствующее сообщения и обработка запущена не будет. При этом пользователь увидит состояние окна, подобное тому, что представлено на рисунке 4. Процесс обработки визуализируется с помощью индикатора выполнения, постепенно заполняющегося по мере обработки.

Пользователь может остановить обработку. Для этого необходимо нажать кнопку «Остановить». Состояние окна при принудительной остановке обработки приведено на рисунке 5.

Если же сжатие будет завершена успешно, то будут выведены характеристики процесса сжатия, и пользователь увидит состояние окна, подобное до того, что приведено на рисунке 6.

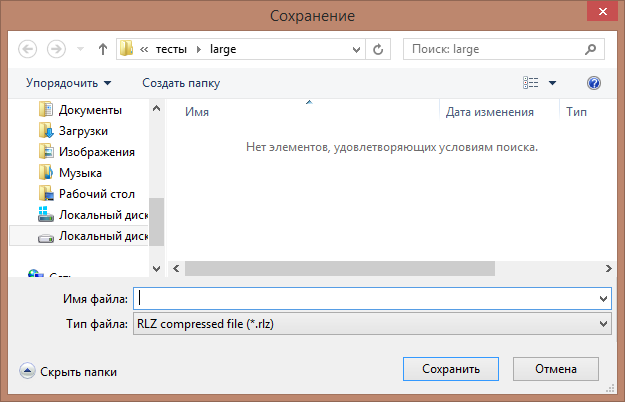


Рисунок 3 – Окно сохранения файла при сжатии

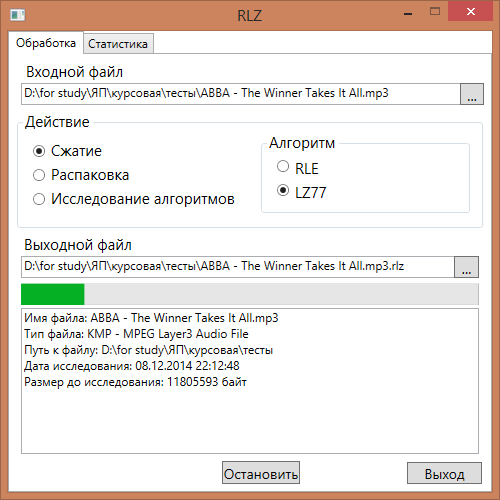


Рисунок 4 – Состояние окна после начала обработки

|  |  |
| --- | --- |
| Рисунок 5 – Состояние окна при принудительной  остановке обработки | Рисунок 6 – Состояние окна при успешном завершении сжатия |

## Распаковка файлов

Для начала необходимо выбрать режим «Действие» - «Распаковка».

Так как выбор алгоритма при распаковке происходит автоматически, то выбор алгоритма становится невозможным.

Затем необходимо выбрать входной файл на диске. Для этого необходимо нажать кнопку «…», расположенную справа от поля пути входного файла. При этом открывается окно, представленное на рисунке 7.

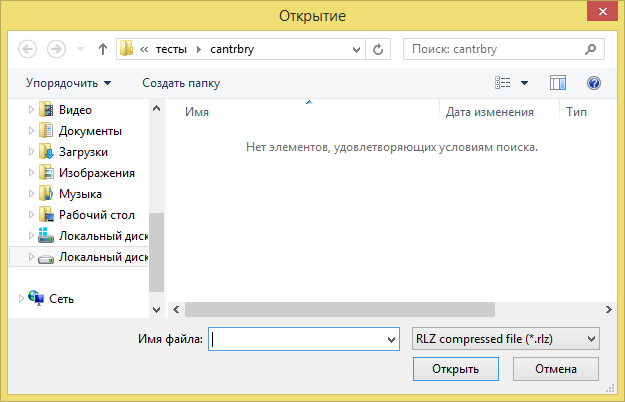


Рисунок 7 – Окно открытия файла при распаковке

После выбора пользователем входного файла, автоматически заполняются поля путей входного и выходного файлов, причем предлагается сохранить распакованный файл в той же директории, что и сжатый. Однако пользователь может сам выбрать директорию для сохранения. Для этого необходимо нажать кнопку «…», расположенную справа от поля пути выходного файла. При этом открывается окно, представленное на рисунке 8.

Затем пользователь, нажав клавишу «Enter» на клавиатуре или кнопку «Начать обработку» на форме, может запустить обработку входного файла; При этом пользователь увидит состояние окна, подобное тому, что представлено на рисунке 4.

Пользователь может остановить обработку. Для этого необходимо нажать кнопку «Остановить». Состояние окна при принудительной остановке обработки приведено на рисунке 5.

Если же сжатие будет завершена успешно, то будут выведены характеристики процесса сжатия, и пользователь увидит состояние окна, подобное до того, что приведено на рисунке 9.

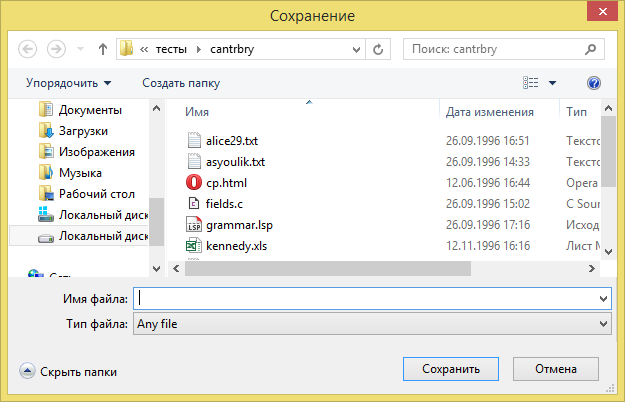


Рисунок 8 – Окно сохранение файла при распаковке

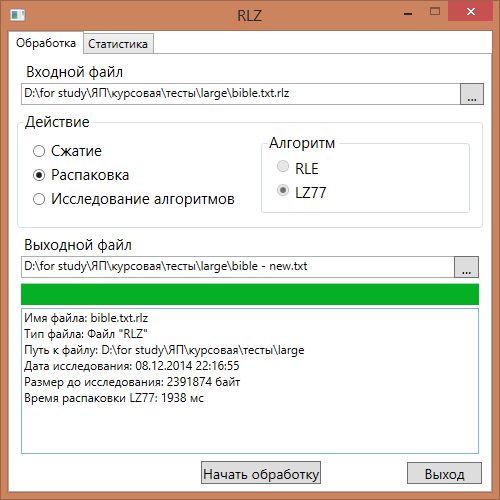


Рисунок 9 – Состояние окна после завершения распаковки

## Исследование алгоритмов

Для начала необходимо выбрать режим «Действие» - «Исследование».

Так как при исследовании последовательно запускаются все алгоритмы, то выбор алгоритма становится невозможным.

Затем необходимо выбрать входной файл на диске. Для этого необходимо нажать кнопку «…», расположенную справа от поля пути входного файла. При этом открывается окно, представленное на рисунке 2.

При этом автоматически заполняются поля путей входного и выходного файла, причем предлагается сохранить временный файл в той же директории (но с добавленным расширением «.rlz»). Однако пользователь самостоятельно выбрать место для хранения временного файла.. Для этого необходимо нажать кнопку «…», расположенную справа от поля пути выходного файла. При этом открывается окно, представленное на рисунке 3.

Затем пользователь, нажав клавишу «Enter» на клавиатуре или кнопку «Начать обработку» на форме, может запустить обработку входного файла; если входной или выходной файл не были выбраны, то появится соответствующее сообщения и обработка запущена не будет. При этом пользователь увидит состояние окна, подобное тому, что представлено на рисунке 4. Процесс обработки визуализируется с помощью индикатора выполнения, постепенно заполняющегося по мере обработки. Так как последовательно запускаются все алгоритмы, то до завершения обработки индикатор заполнится четыре раза.

Пользователь может остановить обработку. Для этого необходимо нажать кнопку «Остановить». Состояние окна при принудительной остановке обработки приведено на рисунке 5.

Если же исследование будет завершена успешно, то будут выведены характеристики процесса сжатия, и пользователь увидит состояние окна, подобное до того, что приведено на рисунке 10.

## Просмотр статистики

Переключив вкладку можно просмотреть статистику, собранную во время предыдущих исследований. Окно, которое видит пользователь, представлено на рисунке 11.

Если таблица не пуста, то ее можно очистить с помощью кнопки «Удалить все записи». Если выбрана какая-либо запись, то ее можно удалить с помощью кнопки «Удалить запись».

|  |  |
| --- | --- |
| Рисунок 10 – Состояние окна после завершения  исследования | Рисунок 11 – Вкладка статистики |

# Исследование алгоритмов

Будем проводить сравнение алгоритмов по следующим характеристикам: коэффициент сжатия, время сжатия, время распаковки.

Как было сказано в пункте 2.3, для исследования алгоритмов будем использовать набор Canterbury Corpus. В таблице 13 приведены результаты исследования файлов, входящих в этот набор.

Таблица 13 – Исследование алгоритмов RLE и LZ77 на наборе файлов   
 Canterbury Corpus

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Имя файла | Размер до обработки, байт | Коэффициент сжатия RLE | Коэффициент сжатия LZ77 | Время сжатия RLE, мс | Время сжатия LZ77, мс | Время распаковки RLE, мс | Время распаковки LZ77, мс |
| alice29.txt | 152089 | 1,91 | 0,69 | 238 | 432 | 234 | 76 |
| asyoulik.txt | 125179 | 1,94 | 0,72 | 199 | 354 | 194 | 69 |
| cp.html | 24603 | 1,88 | 0,61 | 40 | 61 | 38 | 12 |
| fields.c | 11150 | 1,77 | 0,49 | 22 | 26 | 18 | 4 |
| grammar.lsp | 3722 | 1,77 | 0,52 | 7 | 10 | 7 | 2 |
| kennedy.xls | 1029744 | 1,68 | 0,27 | 1597 | 2275 | 1402 | 268 |
| lcet10.txt | 426754 | 1,88 | 0,69 | 650 | 1182 | 643 | 211 |
| plrabn12.txt | 481861 | 1,96 | 0,75 | 728 | 1411 | 747 | 260 |
| ptt5 | 513216 | 0,30 | 0,16 | 747 | 1155 | 127 | 92 |
| sum | 38242 | 1,53 | 0,57 | 61 | 104 | 46 | 16 |
| xargs.1 | 4227 | 1,96 | 0,67 | 10 | 13 | 8 | 2 |

Проанализировав полученные данные, можно установить некоторые особенности алгоритмов RLE и LZ77:

* Неэффективность алгоритма RLE. Практически на всех файлах этот алгоритм показал наихудший результат по качеству сжатия (коэффициент сжатия приблизительно равен двум).
* Однако, высокая эффективность этого алгоритма проявляется на38 файлах с большим количеством серий. Таким файлом и является файл ptt5, который представляет собой необработанное факс-изображение.
* Эффективность алгоритма LZ77. Все файлы данного набора этот алгоритм обработал с коэффициентом сжатия. меньшим единицы. Фактически, алгоритм LZ77 является частным случаем алгоритмом RLE, поэтому его качество сжатия даже на файле ptt5 оказалось выше, чем у алгоритма RLE.
* Время сжатия и распаковки алгоритма RLE оказалось приблизительно равным, что говорит о примерно одинаковой алгоритмической сложности сжатия и распаковки.
* Алгоритм LZ77 отличается несимметричностью сжатия и распаковки по времени: распаковка во всех случаях в несколько раз быстрее чем сжатие. И в самом деле, при сжатии этим алгоритмом большое количество времени тратится на поиск совпадения, когда при распаковке этого производить не нужно.
* Также в полтора-два раза время сжатия алгоритмом LZ77 больше времени сжатия алгоритмом RLE. Так получилось вследствие того, что, опять же, большое время тратится на поиск совпадения, а также из-за использования неоптимальной структуры List.

Результаты исследования свидетельствуют о том, что простейший алгоритм RLE крайне неэффективен, и его приемлемо применять только на немногих типах файлов. Алгоритм LZ77 значительно сложнее в реализации, чем RLE. Но и эффективность сжатия также значительно выше.

# Заключение

В ходе выполнения курсового проекта изучены базовые метода сжатия и распаковки RLE и LZ77. На основании этих методов составлены алгоритмы, которые затем были реализованы в программном приложении, позволяющее производить сжатие файлов, распаковку ранее сжатых файлов, а также проводить сравнение исследуемых алгоритмов с последующим наглядным представлением собранной в результате исследования информации.

Алгоритм LZ77, в основе которого лежит базовый алгоритм RLE, эффективен даже без применения каких-либо оптимизаций и улучшений. Тем не менее, различные модификации этого алгоритма образовывают целое семейство эффективных современных алгоритмов.

В результате выполнения курсового проекта были закреплены навыки программирования на языке C#, использования различных стандартных структур данного языка, создания оконных приложения с использованием технологии WPF.

При дальнейшей разработке будут реализованы следующие возможности:

* возможность приостановить обработку;
* сжатие и распаковка файлов другими алгоритмами семейства LZ, алгоритмами других семейств;
* исследование этих алгоритмов сжатия и распаковки;
* автоматический выбор применяемого алгоритма сжатия в зависимости от типа файла;
* реализация более оптимального алгоритма поиска совпадающей подпоследовательности для алгоритма LZ77.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

|  |  |
| --- | --- |
| [1] Википедия, Сжатие данных [Электронный ресурс]. - Электронные данные. - Режим доступа: http://ru.wikipedia.org/wiki/Сжатие%20данных.. | |
| [2] A Universal Algorithm for Sequential Data Compression / J. Ziv, A. Lempel // IEEE Transactions of Information Theory. - 1977. - Т.IT-23, № 3, pp. 337-343. | |
| [3] Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео: учебно-справочное издание / Д. Ватолин [и др.] - М: ДИАЛОГ-МИФИ, 2003. - 384 с. | |
| [4] Data Compression. The Complete Reference. Fourth Edition. / D. Salomon - London: Springer-Verlag, 2007. | |
| [5] Википедия, Кодирование длин серий, [Электронный ресурс]. - Электронные данные. - Режим доступа: http://ru.wikipedia.org/wiki/RLE. |
| [6] A corpus for the evaluation of lossless compression algorithms / A. Ross, Bell T. - Christchurch: Department of Computer Science, University of Canterbury, | |
| [7] Text Compression / T. C. Bell, J. G. Cleary, I. H. Witten. - Englewood Cliffs: Prentince Hall, 1990. | |
| [8] MSDN, BackgroundWorker - класс [Электронный ресурс]. - Электронные данные. - Режим доступа: http://msdn.microsoft.com/ru-ru/library/system.componentmodel.backgroundworker(v=vs.110).aspx. | |
| [9] Large Corpus [Электронный ресурс]. - Электронные данные. - Режим доступа: http://corpus.canterbury.ac.nz/resources/large.zip. | |
| [10] ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. - Введ. 01.01.1992, М: Изд-во стандартов, 1991. |
| [11] Доманов, А.Т. Стандарт предприятия. Дипломные проекты (работы). Общие требования / А. Т. Доманов, Н. И. Сорока - Минск: БГУИР, 2010. |

# ПРИЛОЖЕНИЕ А (обязательное) Исходные коды функций

class LZ77

{

const int stepToUpateProgressbar = 1000;

const int substringIndexBits = 10;

const int substringLengthBits = 6;

const int maxDictionarySize = 1024; // == 2 ^ 10

const int maxBufferSize = 64; // == 2 ^ 6

static public MainWindow.MetricsOfAlgorythm **Encode**(RLZ.MainWindow.IOFilePaths filePaths, ref System.ComponentModel.BackgroundWorker backgroundWorker)

{

MainWindow.MetricsOfAlgorythm LZ77EncodindMetrics = new MainWindow.MetricsOfAlgorythm();

BinaryReader inputFile = new BinaryReader(File.Open(filePaths.input, FileMode.Open));

BinaryWriter outputFile = new BinaryWriter(File.Open(filePaths.output, FileMode.Create));

outputFile.Write((byte)RLZ.MainWindow.algorytm.lz77);

List<byte> slidingWindow = new List<byte>(inputFile.ReadBytes(maxBufferSize));

int bufferSize = slidingWindow.Count;

int bufferStart = 0;

int dictionarySize = 0;

while (bufferSize != 0)

{

int bufferCounter = bufferStart;

int dictionaryCounter = 0;

int maxEqualLength = 0;

int maxDictionaryEqualStart = dictionaryCounter;

if (dictionarySize != 0)

{

while (dictionaryCounter != bufferStart)

{

int equalLength = 0;

int dictionaryEqualStart = dictionaryCounter;

while ((bufferCounter < slidingWindow.Count - 1) && (slidingWindow[dictionaryCounter] == slidingWindow[bufferCounter]))

{

equalLength++;

bufferCounter++;

dictionaryCounter++;

}

if (equalLength > maxEqualLength)

{

maxEqualLength = equalLength;

maxDictionaryEqualStart = dictionaryEqualStart;

}

dictionaryCounter = dictionaryEqualStart + 1;

bufferCounter = bufferStart;

}

}

outputFile.Write((byte)((maxDictionaryEqualStart) >> (substringIndexBits - 8)));

outputFile.Write((byte)(((maxDictionaryEqualStart) << substringLengthBits) + maxEqualLength));

outputFile.Write(slidingWindow[bufferStart + maxEqualLength]);

//восстановление плавающего окна

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

{

if (dictionarySize == maxDictionarySize)

{

slidingWindow.RemoveAt(0);

}

else

{

dictionarySize++;

bufferStart++;

}

if (inputFile.BaseStream.Position != inputFile.BaseStream.Length)

{

slidingWindow.Add(inputFile.ReadByte());

}

else

{

bufferSize--;

}

}

if (inputFile.BaseStream.Position % stepToUpateProgressbar == 0)

{

backgroundWorker.ReportProgress((int)inputFile.BaseStream.Position);

if (backgroundWorker.CancellationPending)

{

break;

}

}

}

LZ77EncodindMetrics.FinalizeMetrics(outputFile.BaseStream.Length);

inputFile.Close();

outputFile.Close();

if (backgroundWorker.CancellationPending)

{

File.Delete(filePaths.output);

}

return LZ77EncodindMetrics;

}

static public MainWindow.MetricsOfAlgorythm **Decode**(RLZ.MainWindow.IOFilePaths filePaths, ref System.ComponentModel.BackgroundWorker backgroundWorker)

{

MainWindow.MetricsOfAlgorythm LZ77DecodingMetrics = new MainWindow.MetricsOfAlgorythm();

BinaryReader inputFile = new BinaryReader(File.Open(filePaths.input, FileMode.Open));

BinaryWriter outputFile = new BinaryWriter(File.Open(filePaths.output, FileMode.Create));

inputFile.BaseStream.Seek(1, SeekOrigin.Begin);

List<byte> slidingWindow = new List<byte>();

int equalLength;

int dictionaryEqualStart;

int dictionarySize = 0;

byte nextSymbol;

while (inputFile.BaseStream.Position != inputFile.BaseStream.Length)

{

dictionaryEqualStart = inputFile.ReadByte();

equalLength = inputFile.ReadByte();

dictionaryEqualStart = (dictionaryEqualStart << (substringIndexBits - 8)) + (equalLength >> substringLengthBits);

equalLength = equalLength & (maxBufferSize - 1);

nextSymbol = inputFile.ReadByte();

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

{

slidingWindow.Add(slidingWindow[dictionaryEqualStart]);

dictionaryEqualStart++;

dictionarySize++;

}

slidingWindow.Add((byte)nextSymbol);

dictionarySize++;

while (dictionarySize > maxDictionarySize)

{

outputFile.Write(slidingWindow[0]);

slidingWindow.RemoveAt(0);

dictionarySize--;

}

if (inputFile.BaseStream.Position % stepToUpateProgressbar == 0)

{

backgroundWorker.ReportProgress((int)inputFile.BaseStream.Position);

}

if (backgroundWorker.CancellationPending)

{

break;

}

}

while (dictionarySize > 0)

{

outputFile.Write(slidingWindow[0]);

slidingWindow.RemoveAt(0);

dictionarySize--;

}

LZ77DecodingMetrics.FinalizeMetrics(outputFile.BaseStream.Length);

inputFile.Close();

outputFile.Close();

if (backgroundWorker.CancellationPending)

{

File.Delete(filePaths.output);

}

return LZ77DecodingMetrics;

}

}

static class RLE

{

const int stepToUpateProgressbar = 10000;

static public MainWindow.MetricsOfAlgorythm **Encode**(RLZ.MainWindow.IOFilePaths filePaths, ref System.ComponentModel.BackgroundWorker backgroundWorker)

{

MainWindow.MetricsOfAlgorythm RLECompressionMetrics = new MainWindow.MetricsOfAlgorythm();

BinaryReader inputFile = new BinaryReader(File.Open(filePaths.input, FileMode.Open));

BinaryWriter outputFile = new BinaryWriter(File.Open(filePaths.output, FileMode.Create));

outputFile.Write((byte)RLZ.MainWindow.algorytm.rle);

byte currentByte, nextByte;

int currentLength = 0;

currentByte = inputFile.ReadByte();

while (inputFile.BaseStream.Position != inputFile.BaseStream.Length)

{

currentLength = 0;

nextByte = inputFile.ReadByte();

while ((nextByte == currentByte) && (currentLength < 256) && (inputFile.BaseStream.Position != inputFile.BaseStream.Length))

{

currentLength++;

nextByte = inputFile.ReadByte();

}

outputFile.Write((byte)currentLength);

outputFile.Write(currentByte);

currentByte = nextByte;

if (inputFile.BaseStream.Position % stepToUpateProgressbar == 0)

{

backgroundWorker.ReportProgress((int)inputFile.BaseStream.Position);

}

if (backgroundWorker.CancellationPending)

{

break;

}

}

outputFile.Write((byte)currentLength);

outputFile.Write(currentByte);

RLECompressionMetrics.FinalizeMetrics(outputFile.BaseStream.Length);

inputFile.Close();

outputFile.Close();

if (backgroundWorker.CancellationPending)

{

File.Delete(filePaths.output);

}

return RLECompressionMetrics;

}

static public MainWindow.MetricsOfAlgorythm **Decode**(RLZ.MainWindow.IOFilePaths filePaths, ref System.ComponentModel.BackgroundWorker backgroundWorker)

{

MainWindow.MetricsOfAlgorythm RLEDecompressionMetrics = new MainWindow.MetricsOfAlgorythm();

BinaryReader inputFile = new BinaryReader(File.Open(filePaths.input, FileMode.Open));

BinaryWriter outputFile = new BinaryWriter(File.Open(filePaths.output, FileMode.Create));

inputFile.BaseStream.Seek(1, SeekOrigin.Begin);

int currentLength, i;

byte currentByte;

while (inputFile.BaseStream.Position != inputFile.BaseStream.Length)

{

currentLength = inputFile.ReadByte();

currentByte = inputFile.ReadByte();

for (i = 0; i <= currentLength; i++)

{

outputFile.Write(currentByte);

}

if ((inputFile.BaseStream.Position - 1) % stepToUpateProgressbar == 0)

{

backgroundWorker.ReportProgress((int)inputFile.BaseStream.Position);

}

if (backgroundWorker.CancellationPending)

{

break;

}

}

RLEDecompressionMetrics.FinalizeMetrics(outputFile.BaseStream.Length);

inputFile.Close();

outputFile.Close();

if (backgroundWorker.CancellationPending)

{

File.Delete(filePaths.output);

}

return RLEDecompressionMetrics;

}

}

# ПРИЛОЖЕНИЕ Б (обязательное) Схемы алгоритмов

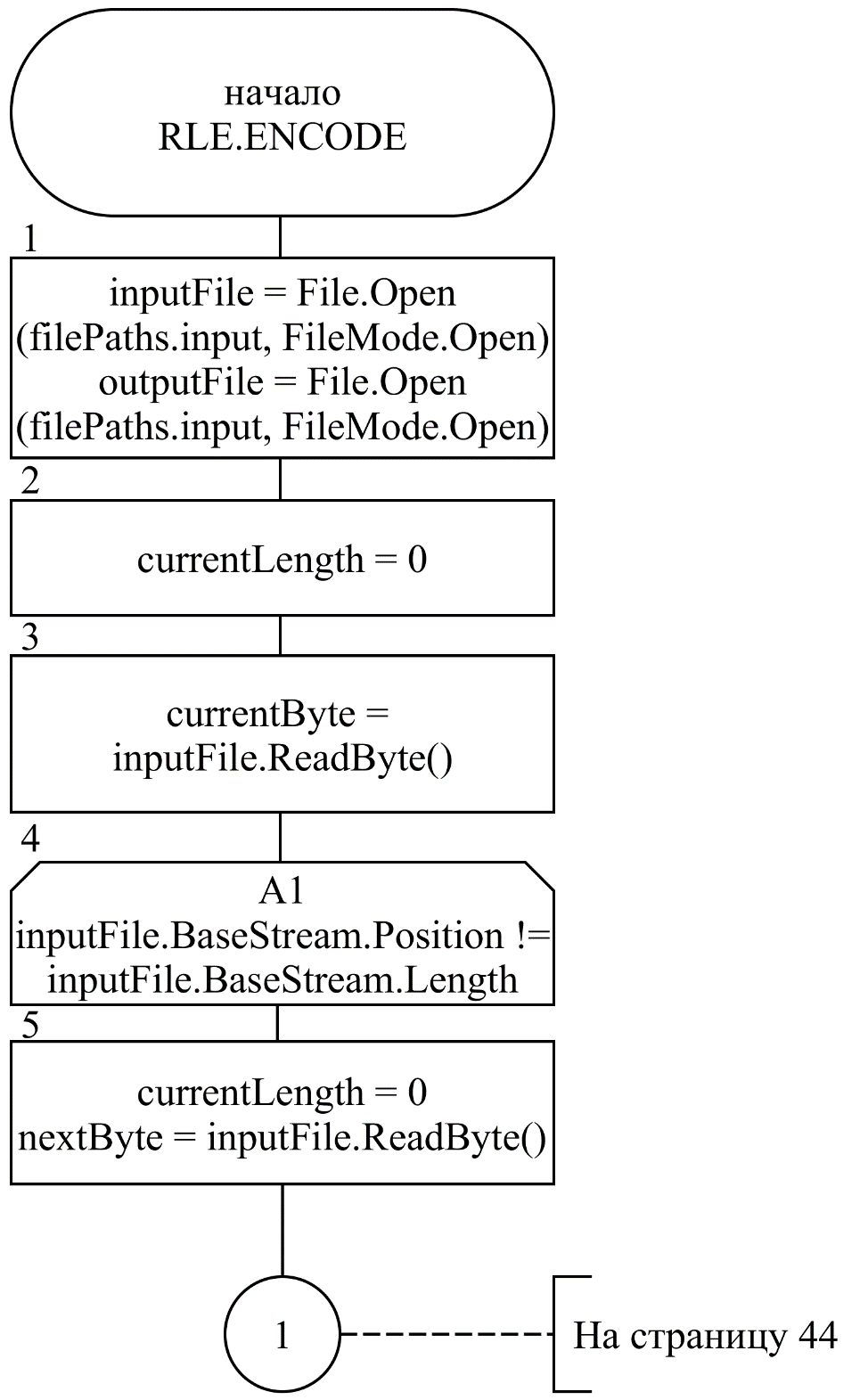


Рисунок Б.1 – Схема алгоритма сжатия RLE

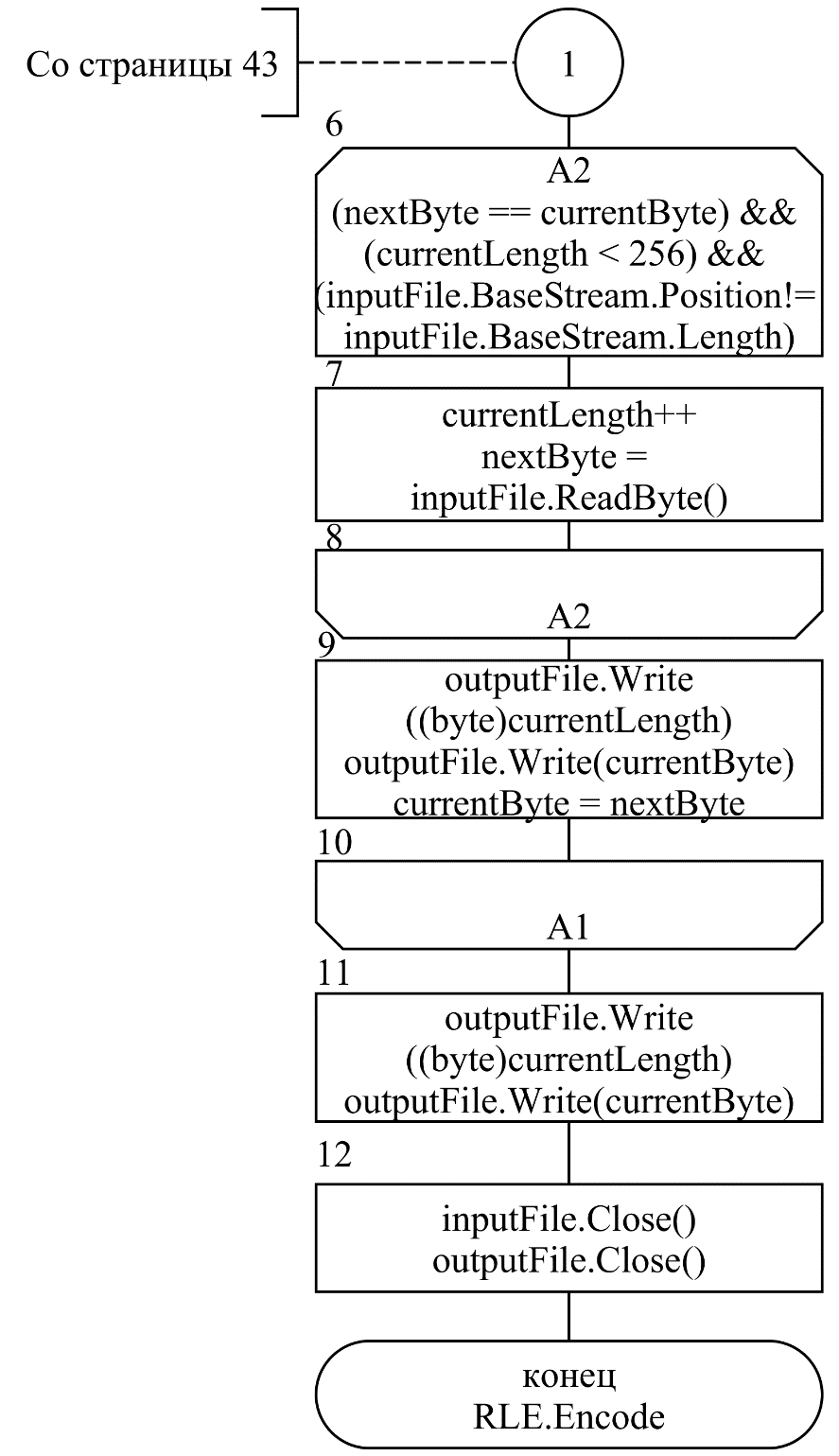


Рисунок Б.2 – Продолжение схемы алгоритма сжатия RLE

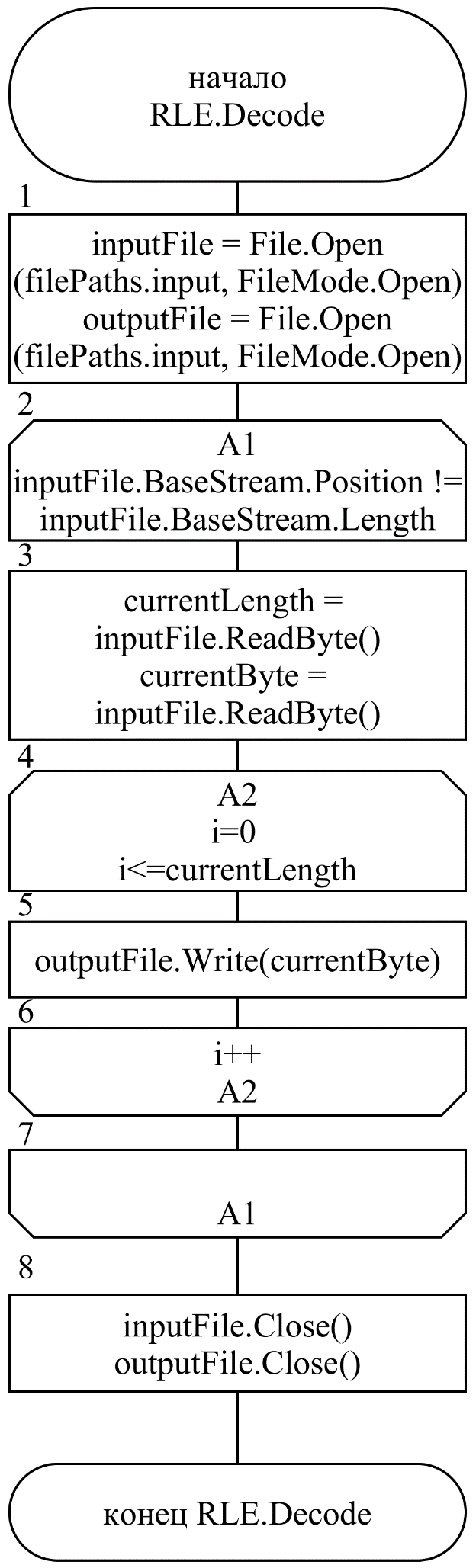


Рисунок Б.3 – Схема алгоритма распаковки RLE

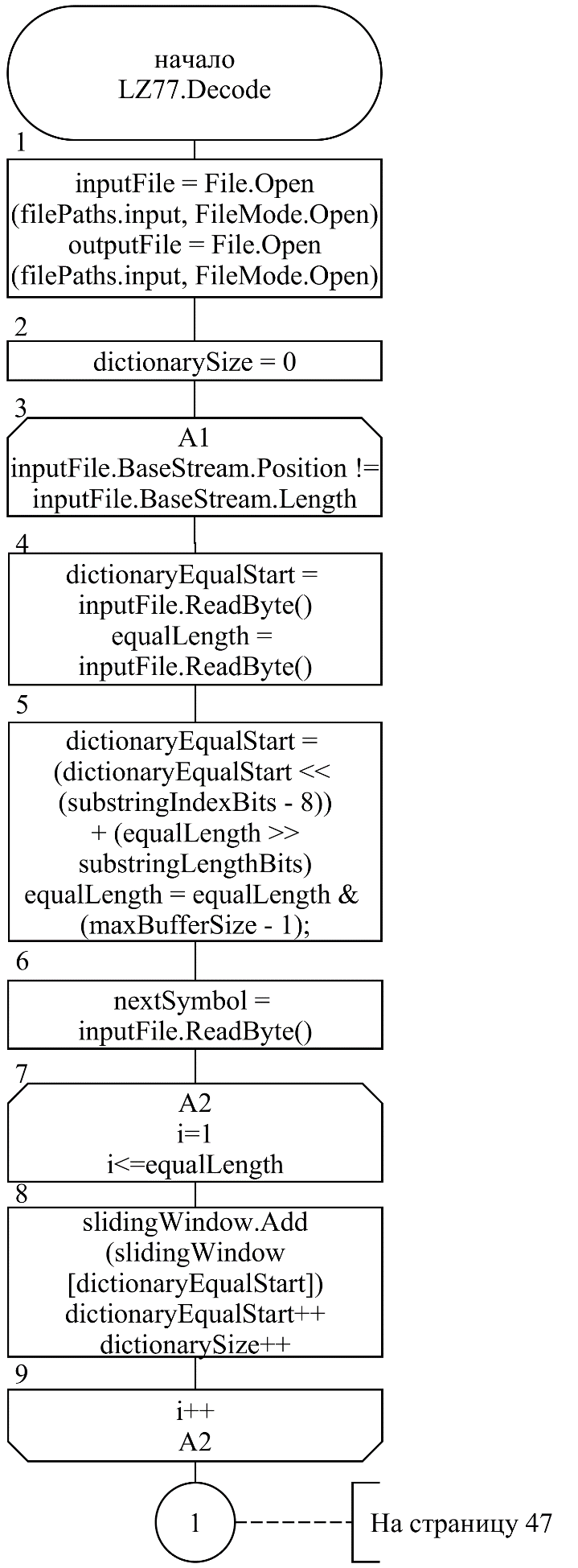


Рисунок Б.4 – Схема алгоритма распаковки LZ77

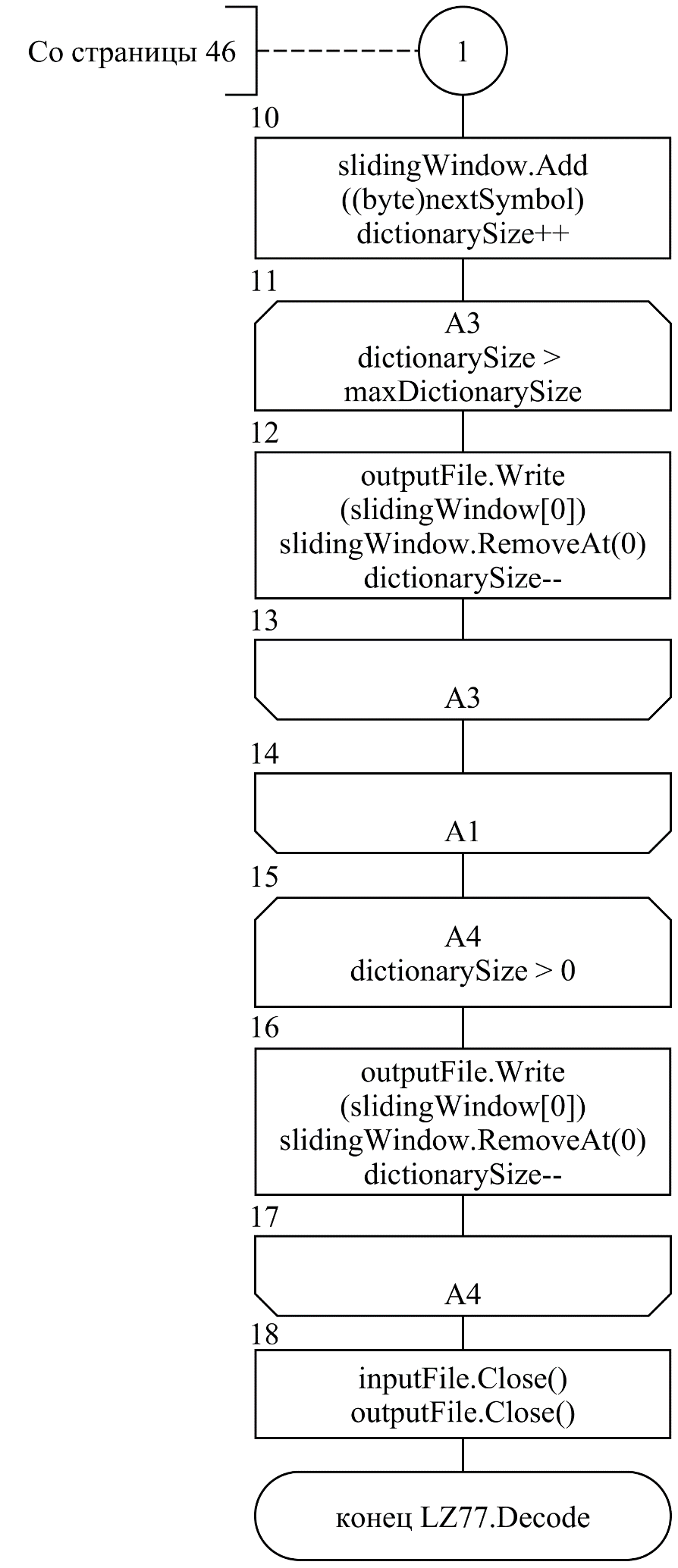


Рисунок Б.5 – Продолжение схемы алгоритма распаковки LZ77

1. 1) Информационный взрыв – постоянное увеличение скорости и объёмов публикаций (объёма информации) в масштабах планеты [5]. [↑](#footnote-ref-1)