Создание набора инструментов для поиска электронных книг в Интернете. Серверная часть, обеспечивающая хранение информации о книгах, поиск по ней и возможность модифицировать её.

Иваницкий Андрей

30 мая 2010 г.

Реферат

Мотивацией для данного проекта послужило желание облегчить работу пользователей с электронными книгами. Ресурсы, которые сейчас есть в Интернете, позволяют пользователю последовательно сначала находить нужную книгу, затем самостоятельно скачивать и искать программу, которая может работать с файлом в заданном формате.

Основная идея проекта — избавить пользователя от лишних и трудоемких действий, оставив ему самую приятную часть — непосредственно чтение книги. Таким образом, проект призван объединить все стадии подготовки к чтению (поиск, скачивание, открытие файла) в одну.

В этом проекте реализована внутренняя часть веб-сервера. С помощью Sphinx реализован мощный быстрый поиск по информации, хранящейся в базе на сервере. Удовлетворены все требования, предъявленные к поисковому механизму. Реализован гибкий, расширяемый протокол взаимодейсвтия с анализатором.

Содержание

1 Введение

В последние годы влияние Интернета на жизнь человека становится все сильнее. Это обусловлено тем, что Интернет предоставляет гораздо больше удобства и возможностей для доступа к информации. Не оказалась в стороне от Интернета и такая важная часть работы и досуга, как чтение книг.

1.1 Описание проблемы

Двадцать лет назад для того, чтобы найти книгу, человек шел в библиотеку или в книжный магазин. Если он искал определённую книгу — достаточно было назвать фамилию автора и название книги, а порой даже одного названия было достаточно. Если же выбор книги ещё не был сделан, всегда можно было получить помощь от сотрудника магазина или библиотеки. Сегодня ситуация с библиотеками и книжными магазинами изменилась несильно, но наравне с бумажными книгами появилась возможность читать книги в электронном виде.

Большое количество книг оцифровано и хранится в сети Интернет, многие из них находятся в свободном доступе. С одной стороны, это должно упрощать процесс получения книг, т. к. теперь они стали доступны из любого места, где есть возможность выхода в Интернет. С другой стороны, перед пользователем встают новые проблемы. Во-первых, теперь ему необходимо самому искать книгу в сети, где количество информации растет с каждым днем. Во-вторых, после того, как он нашел нужную книгу, её нужно скачать, а затем найти программу, которая позволяет читать файл в найденном формате.

На настоящий момент в Интернете существует множество электронных библиотек, поиск по которым весьма затруднен из-за того, что не существует системы, которая могла бы объединить всю информацию с этих библиотек. Следовательно, пользователям приходится либо искать в каждой из электронных библиотек в отдельности, либо пользоваться

существующими поисковыми системами. Есть еще одна проблема, связанная с наличием большого количества электронных библиотек. Она состоит в том, что единый формат для предоставления информации о книгах появился только недавно, поэтому каждая библиотека предоставляет свой формат.

У существующих поисковых систем в свою очередь есть одна очень важная особенность: они обладают ограниченными возможностями в проведении специализированного поиска в сети. Поэтому в результате поиска пользователь получает всю информацию, которая соответствует поисковому запросу и дальше уже начинается работа самого пользователя над тем, чтобы эту информацию профильтровать и извлечь оттуда именно то, что ему нужно.

Таким образом, выявляются две важные проблемы, затрудняющие поиск книг. Первая проблема — различные библиотеки имеют разные интерфейсы, вторая — существует множество форматов, в которых может быть представлена электронная книга.

В данной работе была разработана система, упрощающая задачу поиска, чтения и управления электронными книгами.

1.2 Обзор существующих решений

Все эти проблемы не новы и попытки их решить уже предпринимались.

Например, широко известная поисковая система Google предоставляет свой сервис для поиска книг — Google books [1]. Этот сервис выполняет полнотекстовый поиск по книгам, которые Google сканирует и сохраняет в своей базе данных. В качестве результатов поиска выдается большое количество информации о самой книге, о различных изданиях этой книги и ссылки на ресурсы, где пользователь может приобрести книгу. Благодаря полнотекстовому поиску по содержанию, есть возможность найти книгу, имея сильно ограниченное количество информации о ней. Этот сервис не решает ни проблему унифицированного доступа к

информации о книгах, ни проблему различных форматов книг.

Проект eBdb (electronics books data base) [2] — это поисковая система, которая обходит интернет и сохраняет у себя ссылки на те страницы сторонних ресурсов, которые содержат ссылки на книги. В результате поиска выдается список ссылок на страницы, содержащие книги. Для того, чтобы получить книгу пользователь должен перейти по ссылке, и, возможно, зарегистрироваться на ресурсе. Это решает проблему поиска книг, находящихся в свободном доступе, но остаются другие задачи, которые пользователь вынужден выполнять самостоятельно (скачивание книг и поиск подходящей программы для просмотра). Эта поисковая система тоже не решает проблемы унифицированного доступа и проблему различных форматов книг.

Есть большие системы, решающие проблему унифицированного доступа и различных форматов книг, а именно: Amazon Kindle [3], Sony Reader [4] и подобные. Это программно-аппаратные платформы для чтения электронных книг. Они предоставляют устройство, которое имеет доступ к определенному хранилищу. Пользователь может подключить свое устройство к Интернету, найти нужную книгу в этом хранилище и купить её. Дальше устройство само скачает книгу и откроет её. Из минусов у таких систем то, что эти системы платные и зависимые от устройства. Количество доступных книг ограничено теми книгами, которые хранят/продают хранилища, к которым они подключаются. Но это не единственные минусы таких систем. Так, к примеру, у Amazon Kindle за каждый загруженный текст (вне зависимости от источника загрузки) требуется заплатить компании Amazon от 10 центов. Еще Amazon контролирует информацию, содержащуюся на устройствах, находящихся у пользователей, и по своему усмотрению удаляет её (в том числе книги, приобретённые непосредственно у Amazon).

Существует открытая технология, решающая проблему унифицированного доступа, — OPDS (The Open Publication Distribution System) [5]. OPDS — это новый активно развивающийся стандарт, который построен на базе расширяемого языка разметки Atom. Этот стандарт был специ-

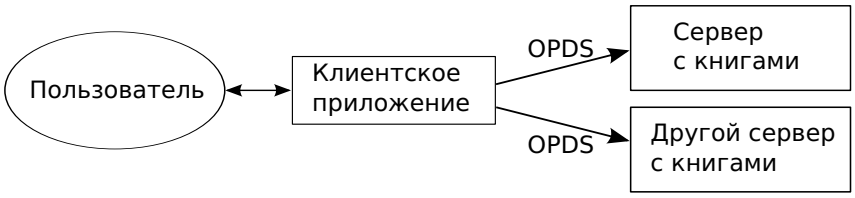

Рис. 1. Схема взаимодействия с использованием формата OPDS

ально разработан для предоставления информации об электронных документах. В нем учтены особенности такого рода информации: наличие аннотации, автора, изображения обложки и пр.

Основная идея существования такого стандарта в том, что если многие сайты будут распостранять свою информацию о книгах в таком формате, то клиентские программы могут работать с ними единообразно (см. рис. 1). Несмотря на то, что стандарт появился недавно, уже существуют сайты и клиетские программы, использующие этот протокол.

Одним из самых крупных сайтов, предоставляющих информацию по протоколу OPDS, является BookServer [6]. Этот некоммерческий проект является частью проекта Internet Archive [7] и является универсальной и открытой системой распостранения электронных книг. BookServer — это архитектура, объединяющая различные форматы книг, конвертируя их при необходимости в нужный формат. Система обеспечивает каталогизацию книг, имеющихся в магазинах, библиотеках или в открытом доступе. Электронный текст можно прочитать на любом конечном устройстве, будь то нетбук, смартфон или специализированное устройство для чтения, наподобие Kindle. Хотя сайтом уже можно пользоваться, он еще находится на стадии разработки, с чем, вероятно, связано отсутствие расширенного поиска по книгам, фильтрации по языкам/жанрам. Также минусом этого проекта для русскоговорящих пользователей является невозможность поиска с использованием русских символов.

1.3 Описание системы в целом

Проект состоит из серверной и клиентской частей.

Серверная часть собирает в сети Интернет информацию об электронных книгах и представляет собранную информацию для обычных пользователей в виде веб-интерфейса и для клиентских программ — в формате OPDS.

Клиентская часть представляет собой программу для удобной работы с сервером, предоставляющим по запросам информацию в формате OPDS. Клиент должен работать как с «родным» сервером, так и с другими серверами, поддерживающими OPDS, например, feedbooks [8]. Пользователь устанавливает у себя на машине одну из клиентских программ. При поиске книги программа обращается к одному из серверов, поддерживающих протокол OPDS. Сервер обрабатывает запрос и возвращает данные в в нужном формате.

Клиентских программ написано 2:

1. На C++ (с использованием библиотеки Qt);
2. На Java.

Серверная часть проекта состоит из 3 подпроектов:

1. поисковый робот (java);
2. анализатор найденной информации, разбирающий информацию о книгах (java);
3. собственно веб-сервер, включающий базу данных, поиск по ней, представление информации в веб и opds форматах (python, django).

Веб-сервер в свою очередь состоит из двух частей. Первая, внутренняя часть, обеспечивает поиск информации по базе и взаимодействие с анализатором. Вторая, внешняя часть, предоставляет информацию и занимается сбором дополнительной. Информация предоставляется двумя способами: в виде веб-интерфейса для пользователя и по протоколу OPDS для клиентских программ.

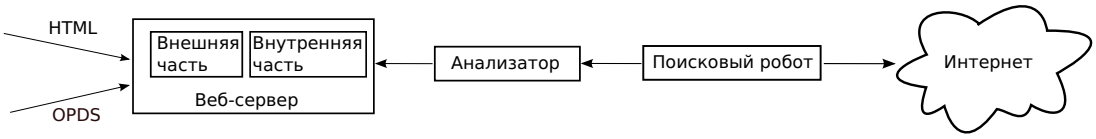

Рис. 2. Схема взаимодействия компонентов системы

1.4 Описание внутренней структуры системы

Поисковый робот обходит Интернет в поисках электронных книг в различных форматах. Если робот нашёл нечто, похожее на книгу, он передаёт ссылку на этот объект и ссылку на страницу, на которой он нашёл ссылку на книгу. После этого анализатор пытается распознать: действительно ли это книга, какое у неё название, авторы. На этом этапе анализатор использует как свой внутренний механизм, так и информацию о книгах, авторах, хранящуюся на сервере. Для этого он использует специальный интерфейс, предлагаемый сервером. Если анализатор распознал в полученом объекте книгу, то далее он пытается понять, существует ли на сервере автор данной книги, существует ли такое произведение. Эта задача нетривиальна, так как в различных файлах одной и той же книги написание названия/имена авторов могут различаться. После упешного распознавания анализатор добаляет информацию о файле и ссылку на него в базу на сервер, используя для этого специальный интерфейс модификации данных, предлагаемый сервером. После всех этих действий информация о книге хранится на сервере и доступна через веб-интерфейс или в OPDS формате.

1.5 Использование веб-фреймворка Django

Для реализации веб-сервера использовалось Django [9].
Django (Джанго) — это свободный фреймворк для веб-приложений на языке Python.

Богатая стандартная библиотека Python, а также большое количество сторонних библиотек для этого языка позволили легко подключить и использовать другие приложения (Sphinx, aspell).

Архитектура Django похожа на «Модель – Представление – Управление» (Model – View – Controller, MVC) [10]. Её использование обеспечило разделение логики работы от представления. Одно из существенных архитектурных отличий этого фреймворка, — это идея строить сайт из одного или нескольких приложений. Это позволяет делать отдельные компоненты сервера слабосвязанными.

Для работы с базой данных Django использует собственный ORM (Object-Relational Mapping), в котором модель данных описывается классами Python, и по ней генерируется схема базы данных. Этот простой, но мощный способ даёт возможность обеспечить надежное хранение данных, а богатый интерфейс манипулирования данными покрывает многие задачи.

2 Постановка задачи

2.1 Поиск по данным

Если существует некоторая большая база с информацией, то очевидно, что для быстрой и удобной работы с ней необходим мощный, быстрый и удобный поиск.

В нашей модели для пользователя есть несколько сущностей: название книги, список её авторов, язык, на котором написана книга, тэги (характеризующие её стиль, жанр, содержание).

Ниже сформулированы требования для функции поиска с точки зрения пользователя:

1. Релевантный поиск как по отдельным сущностям, так и по различным их комбинациям;
2. Фильтрация результатов поиска по некоторым сущностям (язык книги, тэг);
3. Поиск с учётом морфологии языка;
4. Поиск с учётом опечаток в запросе или их исправление;
5. Простой поиск (простой в использовании).

При разработке поискового механизма необходимо: сохранить слабую связанность отдельных компонентов программы, обеспечить возможность замены реализации поиска с минимальными изменениями в остальном коде, учесть возможность расширения и изменения требований к функции поиска.

2.2 Интерфейс модификации данных

Одной из сильных сторон является расширение и улучшение базы информации о книгах. Для этого есть несколько путей: ручное управление информацией (администраторская функциональность), автоматиче-

ская классификация и сбор дополнительной информации со сторонних ресурсов, получение информации от анализатора. Для обеспечения возможности добавлять новую и изменять существующую информацию со стороны анализатора необходимо разработать и реализовать протокол взаимодействия. Этот протокол необходимо сделать полным, гибким и расширяемым.

3 Интерфейс поискового механизма

3.1 Методы

Для обеспечения модульности, гибкости кода реализация поиска и его использование разделено.

Отдельно описан интерфейс поискового механизма. Код, использующий поисковые функции, опирается только на этот интерфейс. За ним скрыта вся логика поиска.

При разработке конкретного поискового механизма необходимо реализовать этот интерфейс. Если в будущем понадобилось изменить или даже заменить поисковый механизм, то эти изменения не затронут остальной код программы.

Интерфейс описан классом SearchEngine. Этот интерфейс содержит всего три метода:

```
author_search(**kwargs)
book_search(**kwargs)
simple_search(query, **kwargs)
```

author_search() осуществляет поиск по авторам, принимает имена авторов как обязательный аргумент и, возможно, дополнительные аргументы (зависит от реализации).

book_search() осуществляет поиск по книгам, принимает название книги как обязательный аргумент, имена авторов как необязательный аргумент и, возможно, дополнительные аргументы.

simple_search(query) осуществляет поиск по книгам, принимает аргумент *запрос*, не специфицирующий сущность, к которой относится. Этот метод обеспечивает *простой поиск*. Позволяет пользователю не указывать сущность запроса (возможен смешанный запрос, например «Пушкин Евгений Онегин»). Допускаются дополнительные аргументы.

Упомянутые выше *дополнительные аргументы* — это аргументы типа тэг, язык книги, фильтрующие результаты поиска.

3.2 Возвращаемое значение

Каждый метод возвращает итерируемую коллекцию, содержащую соответствующие сущности (авторов или книги).

У каждого возвращаемого значения есть атрибут suggestion, содержащий словарь возможных исправлений запроса или None, если таковых не имеется. Пример:

```
suggestion = {
    author: 'Tolstoy',
    title: 'Anna Kerenina',
}
```

4 Реализация поискового механизма

4.1 Выбор инструмента

Теоретически функциональность поиска можно было реализовать, используя встроенные средсва СУБД, например, like для MySQL и другое. Но на практике это неразумно. Так как встроенные средсва достаточно бедны и их функциаональность сильно ограничена. Например, они не предоставляют возможность учёта морфологии языка. Следовательно, пришлось бы самостоятельно реализовывать подобные возможности, что достаточно трудоёмко. И, не зависимо от пути реализации, скорость работы такого механизма, скорее всего была бы невысока.

Естественно, подобная задача важна и широко распространена. Поэтому существуют достаточно много готовых решений, называемых поисковыми системами. Вот неполный список одних из самых популярных решений:

1. The Apache Lucene;
2. Xapian;
3. Sphinx;
4. Яндекс.Сервер.

Все они предоставляют схожую функциональность.

Для реализации интерфейса поискового механизма в этом проекте был выбран Sphinx [11]. Это бесплатная поисковая система с открытым исходным кодом, предназначенная для быстрого поиска текста. Автор проекта — россиянин Андрей Аксёнов.

4.2 Реализация поиска с помощью Sphinx

Процесс осуществления поиска состоит из двух этапов.

Первый — это индексация данных, то есть преобразование входных данных и добавление их в некоторую базу. Затем эта база используется

для полнотекстового поиска информации. Sphinx предоставляет большое число возможных настроек процесса индексации. Второй этап — это непосредственно поиск. Sphinx также предлагает богатые возможности тонкой настройки вида поиска.

Sphinx вводит три понятия для пользовательских настроек поведения поисковой системы: источник данных, индекс, поиск. А также два для административных настроек: индексер, поисковый сервис. Пользовательские настройки позволяют реализовать требуемую функциональность поиска.

4.2.1 Релевантный поиск

Sphinx допускает несколько вариантов сортировки результатов поиска в настройках поиска. Один из них SPH_SORT_RELEVANCE вариант, обеспечивающий сортировку результатов по релевантности.

4.2.2 Фильтрация результатов

У каждой книги существует понятие язык, на ктором она написана. Книга может быть написана только на одном языке (по крайне мере, всегда можно выделить основной язык). Таким образом отношение между сущностью книга и язык «один к одному». То есть язык является атрибутом книги. В Sphinx присутствует возможность фильтрации результатов поиска по атрибутам. Эта необходимо сделать в настройках источника данных. Во-первых, надо указать, откуда брать атрибут (language_id):

```
sql_query = SELECT id, title, language_id FROM book_book
```

Во-вторых, указать, что по этому атрибуту необходимо будет фильтровать рузультаты:

```
sql_attr_uint = language_id
```

Для тэгов ситуация схожа, но отношение между сущностью книга (или автор) и тэг «многие ко многим». Поэтому настройки имеют несколько другой вид.

Указание источника атрибута, здесь же указание требования фильтрации по этим атрибутам:

```
sql_attr_multi = uint tag_id from query; \
select b.id, bt.tag_id \
from book_book as b \
join book_book_tag as bt on b.id=bt.book_id;
```

4.2.3 Поиск с учётом морфологии

Поиск с учётом морфологии в поисковых системах зачастую реализуется с помощью стемминга. Стемминг [14] — это процесс нахождения основы слова для заданного исходного слова. Основная идея заключается в неразличении слов, находящихся в раздичных словоформах. Очевидно, что такое преобразование необходимо как на стадии индексации данных, так и на стадии поиска; в последнем случае преобразование происходит над поисковым запросом. В Sphinx существует встроенный стемминг для английского и русского языков. Но есть возможность подключить любой другой алгоритм стемминга. Включается данная опция в настройках индекса

```
morphology = stem_enru
```

4.2.4 Поиск с опечатками

При поиске книг у пользователя есть возможность указать или название книги, или имя автора, или оба параметра. В каждом из этих запросов пользователь может допустить опечатку или ошибку. Исправление опечаток в запросе при указании названия книги осуществляется с помощью aspell. Aspell [12] — это свободная программа для исправления орфографии. Для проверки орфографии используется словарь. Проверяемое слово сравнивается со словами, находящимися в словаре. В случае, если проверяемое слово определено как неправильное, aspell может предложить варианты исправления. Так как в основе механизма исправления

слов лежит словарь, то, очевидно, для различных языков такие словари будут различны. Для aspell существуют словари для более 84х языков [13], среди которых есть и русский.

При использовании системы пользователь может искать книги на различных языках. Пользователь может включить фильтрацию по языкам, но может и не указывать язык. В реализации функции исправления запроса был применён следующий приём. Если указан язык на котором написан запрос, то программа использует соответствующий словарь; если такового не имеется в системе, то проверка орфографии не производится. Сообщение об отсутствии требуемого словаря записывается в лог. Таким образом, проанализировав лог, можно понять, какие словари более всего требуются пользователям. Достаточно их установить в систему, и программа в следующие разы будет по необходимости использовать новые словари.

Если язык запроса не указан, то программа пытается автоматически распознать язык. Пока это реализовано только для русского языка, в остальных случаях используется словарь анлийского языка.

Метод проверки запросов по словарю хорошо работает для обычной лексики. Но с именами нарицательными ситуация несколько хуже.

Если в русском языке чаще можно правильно записать фамилию или имя на слух, то в английском языке это почти всегда непросто. Поэтому для осуществления поиска по авторам с опечатками в запросе применяется совершенно другая идея. Для решения задачи поиска имён по звучанию используется алгоритм сравнения двух строк по их звучанию. Основная идея таких алгоритмов заключается в сопоставлении слову некоторого ключа, характеризующего его звучание, а не написание. Подобных алгоритмов существует несколько видов: Soundex [15], Metaphone [16], Double Metaphone.

Sphinx имеет встроенную поддержку как и Soundex, так и Metaphone. Подобная опция устанавливается в настройках индекса

```
morphology = soundex
```

Но при таком решении возникает небольшая проблема: в результатах поиска автор, имя которого полностью совпадает с поисковым запросом, может оказаться не на первом месте. А на первом месте может оказаться автор, имя которого звучит также как и запрос.

Дабы решить эту проблему была применена следующая идея. Сначала авторы ищутся в индексе, с отключенной морфологией, после — в индексе с включённой морфологией. Затем эти результаты объединяются таким образом, что вначале идут авторы из первого результата, а после — из второго.

4.2.5 Проблема диакритических знаков

Как и в названиях книг, так и в именах их авторов могут встречаться символы с различными надстрочными, подстрочными знаками, называемыми *диакритическими знаками*. Естественно, пользователь может указать подобные символы в поисковом запросе. Есть мысль, что если в запросе есть диакритические знаки, то искать хочется с учётом оных. С одной стороны, это так: пример тому слово «marché» (фр. «рынок») и «marche» (фр. «ходит»).

Но тут возникает проблема. Пользователь может указать в слове (или запросе) только часть необходимых диакритических знаков, тогда, скорее всего, он получит результат хуже, чем при запросе без диакритических знаков.

Для такого поведения пользователя есть несколько причин:

1. Просто лень, невнимательность;
2. Отсутствие требуемого символа на клавиатуре (француз ищет книгу на испанском);
3. Пользователь не знает правильное написание фамили автора, но правильно указал диакритический знак в названии книги.

Поэтому было принято решение: не различать буквы с диакритическими знаками и без.

В программе в работе со строками используется Юникод. В Юникоде символы, имеющие дополнительные над- или подстрочные элементы, могут быть представлены в виде построенной по определённым правилам последовательности кодов (составной вариант, composite character) [17] или в виде единого символа (монолитный вариант, precomposed character).

Для игнорирования диакритических знаков в составном варианте достаточно указать в настройках индекса Sphinx «игнорировать модифицирующие символы»

```
ignore_chars = U+0300, U+0301, U+0302, U+0303, ...
```

Для игнорирования диакритики в едином символе используется возможность Sphinx в настройках индекса определять таблицу символов. Она позволяет задавать правила для отображения одних сиволов в другие. Эти правила будут использоваться как для данных при индексации, так и для поисковых запросов при поиске.

```
charset_table = U+00C0->a, U+00C1->a, ...
```

4.2.6 Простой поиск

Для обеспечения быстрой и удобной работы с базой важную роль играет *простой поиск*. Он позволяет пользователю быстро и удобно получить необходимую информацию.

Пользователю предлагается только одно поле для ввода, в которое он может ввести смешанный запрос. В таком запросе он может указать как и название книги, так и автора, так и то и другое. В каждом случае необходимо вернуть корректные результаты. Поисковый запрос пользователя может быть одним из трёх вариантов:

1. название книги, автор;
2. название книги;
3. автор.

Так как от пользователя приходит только смешанный поисковый запрос, то узнать какой это из перечисленных вариантов невозможно. Данная задача была решена следующим образом.

Если поиск среди имён авторов дал плохие результаты, то, видимо, в поисковом запросе нет имени автора. Тогда пользователю возвращаются результаты поиска по названиям книг.

Если поиск среди названий книг не дал результатов, а поиск среди авторов оказался большим, то скорее всего в поисковом запросе указано только имя автора. В этом случае в качестве результатов поиска возвращается список книг автора, оказавшимся первым в результатах поиска по именам авторов.

В случае, если результаты поиска и по названиям книг, и по авторам дали хорошие результаты, производится фильтрация книг по именам авторов. Тем самым обеспечивается требуемое поведение в случае, когда в запросе указаны и название книги и её авторы.

5 Интерфейс к анализатору

5.1 Алгоритм взаимодействия

Важной частью функционирования системы в целом является процесс добавления информации на сервер со стороны анализатора. Этот процесс состоит из двух этапов.

Первый этап — это распознавание анализатором названия книги, её авторов и прочей информации о книге. На этом этапе анализатор обращается к серверу с запросами вида: есть ли в базе автор с именем, похожим на заданное, какие книги заданного автора есть в базе.

Второй этап — это добавление информации на сервер. В этот момент анализатор пользуется интерфейсом модификации данных.

В разных файлах одной и той же книги могут быть по разному записаны имена авторов. Для поддержания базы в корректном состоянии необходимо распознавать такие неточности.

К сущностям *автор, книга, файл книги* было добавлено новое понятие *индекс доверия (credit)*. Это понятие характеризует проверенность информации о сущности. Также введено понятие *индекс релевантности (relevance)*. Этот индекс характеризует похожесть двух строк. Эти оба индекса возвращаются при поиске по авторам и по книгам. Анализатор принимает решение на основании значений этих двух индексов.

Если для автора индекс доверия и релевантности оказались больше пороговых значений, то не создаётся новой сущности. В противном случае создаётся новый автор. Если для книги индекс доверия и релевантности оказались больше пороговых значений и авторы этой книги распознались как уже существующие в базе, то новой сущности не создаётся.

5.2 Фаза распознавания информации

В фазе распознавания книги анализатору необходимо знать, какие авторы, книги уже присутствуют в базе на сервере. В результатах по-

иска по названиям книг или по именам авторов необходимо к каждой сущности указывать индекс доверия и релевантности. Индекс доверия хранится в базе данных, поэтому с ним нет больших трудностей. А индекс релевантности необходимо рассчитывать для каждого запроса.

Для этого была реализована оболочка над поисковым интерфейсом. Сначала производится поиск по заданной сущности с использованием одной из реализаций поискового механизма (в этой работе поисковый механизм основан на Sphinx). Затем для каждой сущности из результатов поиска вычисляется индекс релевантности. В этой задаче необходимо сравнивать похожесть двух строк, состоящих из одного или более слов. Для этого было применено модифицированное расстояние Левенштейна. Алгоритм Левенштейна [18] неплохо работает для названий книг одного автора. Но весьма неудовлетворительно для имён людей.

Расстония Левенштейна между строками «Лев Толстой» и «Толстой Лев Николаевич» будет больше, чем расстояние между «Лев Толстой» и «Александр Толстой». Очевидно, что первая пара имён, скорее всего, соответсвует одному и тому же человеку. Вторая пара — скорее различным людям.

Анализ различных написаний имён авторов показал, что часто в имени автора указываются не все части полного имени. Например, часто опускают отчество (или среднее имя), порой и имя. Также встречается различный порядок написания частей полного имени автора.

Основная идея модифицированного алгоритма Левенштейна — это изменить его поведение на строках, состоящих из нескольких слов.

Алгоритм основывается на следующих предположениях:

1. порядок следования слов в строке не очень важен,
2. цена удаления/добвавления слова из строки скорее всего меньше, чем её длина,
3. расстояние между произвольными парами строк Q и S_1 меньше, чем расстояние между строками Q и S_2, если у Q и S_1 совпало больше слов, чем у Q и S_2.

Алгоритм Расчёт расстояния между строками

Берутся две строки s_1 и s_2 и разбиваются на слова; получается S_1 — набор слов из s_1.

Для каждой пары слов a и b, $a \subset S_1$, $b \subset S_2$ вычисляется расстояние Левенштейна. Таким образом получается *матрица расстояний* M размера $n \times m$.

Затем вычисляется минимальная сумма Σ из $min(n,\ m)$ элементов матрицы M таких, что ни один из элементов не стоит ни в одном столбце, ни на одной строчке с другими элементами. Σ — это сумма цен изменения слов из набора S_1 в слова из набора S_2 без учёта порядка следования слов и их удаления/добавления.

Прибавив к Σ цену добавления/удаления слов $abs(n\text{-}m) * C_{remove}$, где C_{remove} цена удаления/добавления одного слова, получаем минимальное расстояние между набором слов S_1 и S_2.

5.3 Фаза добавления книги

Для анализатора реализован интерфейс, позволяющий добавлять и модифицировать новые сущности.

Запрос состоит из двух секций: define и update.

```
<?xml version="1.0" encoding="UTF-8"?>
<request>
    <define>
        ...
    </define>

    <update>
        ...
    </update>
</request>
```

5.3.1 Секция define

Здесь необходимо описать создаваемые сущности, но не уже существующие в базе. У каждой сущности должен быть атрибут — *уникальный идентификатор ui* (целое положительное число). Это уникальный идентификатор сущности для этого запроса. В следующей секции по этим ui можно обращаться к сущностям.

При определении каждой сущности необходимо указать обязательную информацию:
для author – full_name
для file – link, size, type
для book – title

Можно добавить необязательную информацию: credit, lang, ...

Пример определения автора

```
<author ui="1">
    <full_name> Leo Tolstoy </full_name>
</author>
```

Пример определения файла книги

```
<file ui="2">
    <link>http://example.com</link>
    <type>pdf</type>
    <size>4563214</size>
</file>
```

Пример определения книги

```
<book ui="3">
    <title> Red hat </title>
</book>
```

5.3.2 Секция update

В этой секции возможна модификация данных.

К каждой из трёх сущностей возможен доступ по id, если эта сущность уже существует в базе, либо по ui, если она создается в этом запросе.

Вся указаная информация о сущности будет либо перезаписана, либо добавлена.

Пример изменения имени автора

```
<author id="45">
    <full_name> Alexander Pushkin </full_name>
</author>
```

Пример создания новой книги

```
<book ui="3">
    <authors>
        <author id="343" />
        <author ui="1" />
    </authors>
    <files>
        <file ui="2" />
    </files>
</book>
```

Пример добавления к существующей книге автора (если у этой книги уже существовал автор, то он также останется автором этой книги)

```
<book id="223">
    <authors>
        <author ui="1" />
    </authors>
</book>
```

Для сущности book возможно добавление атрибута reset со значением author или file.

Его наличие с атрибутом author означает, что только ниже перечисленные авторы написали данную книгу. Если у книги существовали до этого авторы, они будут удалены из списка авторов книги.

```
<book id="276" reset="author">
    <authors>
        <author id="343" />
        <author ui="1" />
    </authors>
</book>
```

Если атрибут reset имеет значение file, то поведение аналогично вышеописанному только для файлов книги.

5.3.3 Обработка ошибок

Если при обработке запроса произошла ошибка, то ни одно из изменений не будет применено. В ответе указывается информация об ошибке.

6 Заключение

В результате проделанной работы разработана внутренняя часть веб-сервера. С помощью Sphinx реализован мощный быстрый поиск по информации, хранящейся в базе на сервере. Удовлетворены все требования, предъявленные к поисковому механизму. Реализован гибкий, расширяемый протокол взаимодействия с анализатором.

Продуманная архитектура системы в целом позволяет легко расширять, изменять логику взаимодействия отдельных частей.

Слабая связанность и модульность кода этой работы обеспечивает независимость компонентов, что позволяет добавлять новую функциональность в систему и улучшать уже существующую, открывая тем самым возможности дальнейших исследований в этом направлении.

Проделанная работа является частью большой системы, разрабатываемой в команде. Полная работающая система доступна в Интернете по адресу http://ebooksearch.webfactional.com/. Исходный код всего проекта в svn-репозитории google code — http://code.google.com/p/ebooksearchtool/.

7 Библиографический список

[1] Сервис для полнотекстового поиска по книгам Google Books
http://books.google.com/

[2] База данных электронных книг eBdb
http://ebdb.ru/

[3] Программно-аппаратная платформа для чтения электронных книг Amazon Kindle
http://amazon.com/kindle/

[4] Программно-аппаратная платформа для чтения электронных книг Sony Reader
http://www.learningcenter.sony.us/assets/itpd/reader/

[5] Описание стандарта The Open Publication Distribution System (OPDS)
http://code.google.com/p/openpub/wiki/CatalogSpecDraft

[6] Открытая система распостранения электронных книг BookServer
http://bookserver.archive.org/

[7] Библиотека Internet Archive
http://archive.org/

[8] Сервер предоставляющий доступ к библиотеке в формате OPDS
http://feedbooks.com/

[9] Страница проекта веб-фреймворк Django
http://www.djangoproject.com/

[10] Описание архитектуры Django сайтов на DjangoBook
http://www.djangobook.com/en/2.0/chapter01/

[11] Поисковая система Sphinx
http://sphinx.com/

[12] Aspell
http://aspell.net/

[13] Aspell поддерживаемые языки
http://aspell.net/man-html/Supported.html

[14] Страница про стемминг на английской Wikipedia
http://en.wikipedia.org/wiki/Stemming/

[15] Страница про soundex на английской Wikipedia
http://en.wikipedia.org/wiki/soundex/

[16] Страница про metaphone на английской Wikipedia
http://en.wikipedia.org/wiki/Metaphone/

[17] Список модифицирующих символов из The Unicode Standard, Version 5.2.
http://www.unicode.org/charts/PDF/U0300.pdf/

[18] Страница про расстояние Левенштейна на английской Wikipedia
http://en.wikipedia.org/wiki/Levenshtein_distance/

8 Приложение: Структура дерева исходных кодов

Исходные коды всего проекта находятся в svn-репозитории google code — http://code.google.com/p/ebooksearchtool/.

Дерево исходных кодов этой работы:

/trunk/server/

/trunk/server/generate_sphinx_conf.py — генератор файла настроек Sphinx

/trunk/server/settings.py — файл настроек сервера

/trunk/server/book/

/trunk/server/book/data_modify.py — реализация протокола модификации данных

/trunk/server/book/models.py — описание базы данных (средствами Django)

/trunk/server/book/search.py — реализация протокола поиска по данным для анализатора

/trunk/server/book/update_entity.py — вспомогательные фукции для протокола модификации данных

/trunk/server/book/view.py — view-функции для интерфейса анализатора

/trunk/server/book/search_engine/

/trunk/server/book/search_engine/engine.py — описание интерфейса поискового механизма

/trunk/server/book/search_engine/spell_checker.py — функции для проверки и исправлнения запросов

/trunk/server/book/search_engine/sphinx_engine.py — реализация поискового механизма с помощью Sphinx

/trunk/server/book/tests/ — unit-тесты

/trunk/server/spec/

/trunk/server/spec/distance.py — функция расчёта расстояния между строками

/trunk/server/spec/exception.py — определение иерархии пользовательских исключений

/trunk/server/spec/logger.py — настройка стандартного python логера

/trunk/server/templates/data/ — описание templates для интерфейса анализатора

/trunk/server/templates/sphinx_conf/ — описание templates для формирования файла настроек Sphinx