УДК 004.62

ПРОБЛЕМЫ РЕАЛИЗАЦИИ ПОИСКОВОГО ИНДЕКСА ДЛЯ СИСТЕМЫ АНТИПЛАГИАТА

© Е.С. Чиркин

Ключевые слова: антиплагиат; поиск; поисковый индекс.

Описаны некоторые проблемы построения и хранения поискового индекса для систем автоматизированной проверки документов на наличие заимствований, приведен ряд эффективных решений.

ВВЕДЕНИЕ

Любая автоматизированная система по поиску заимствований (система антиплагиата) представляет собой специализированную поисковую систему полнотекстового поиска.

В общем случае процедуру поиска можно представить как поиск фрагментов проверяемого документа в базе, содержащей все остальные документы. Результат подобного пересечения документов и будет ответом – насколько данный документ состоит из других, т. е. фактически вычисляется, насколько данный документ состоит из фрагментов других документов.

Для ускорения поиска, что вообще его делает возможным, каждый документ предварительно обрабатывается – индексируется. Индексирование подразумевает преобразование документа в текст, удаление лишних символов, разбиение на слова (или фразы), построение индекса по ним – сохранение слова и его позиции в документе таким образом, чтобы поиск осуществлялся быстрее. Некоторые этапы построение индекса преследуют своей целью минимизацию его объема.

Рассмотрим значимые составляющие этапов подготовки документа к индексированию, их проблемы и возможные решения, а также решения в модуле антиплагиата информационно-поисковой системы «ИТ-специалист».

ФОРМАТ ДОКУМЕНТА

Документы, в которых должен осуществляться поиск, обычно представлены в наиболее распространенных форматах программных пакетов Microsoft Office, Open Office.Org/Libre Office – т. н. «форматы файлов» .doc, .docx, .rtf, .odt, реже – .pdf и .djvu. Помимо этого база всех текстов пополняется страницами из сети Интернет, что означает необходимость поддержки HTML-формата. Не исключено, что документы будут находиться в архивах форматов 7-Zip, ZIP или RAR, возможно многократное вложение. Не исключено существование внедренных в документ документов (причем, бывает, они не извлекаемы обычными способами через программные интерфейсы). Данное многообразие документов представляет сложность по организации логического каталога для хранения метаинформации о них.

Решение: удобно считать, что любой документ является неким контейнером, содержащим другие контейнеры. Из любого контейнера либо может быть извлечен текст, либо нет. При формировании результатов поиска ссылка выставляется на текст и все документы, содержащие данный текст.

ОСНОВНАЯ ЧАСТЬ

Извлечение текста. Извлечение текста тесно связано с форматом файла документа – например, «формат».docx представляет собой ZIP-архив нескольких файлов, главный из которых – XML-файл специальной структуры, который содержит основной текст документа. Файлы в форматах PDF и DjVu могут содержать текстовый слой, а могут не содержать. Существует известная особенность редактирования .docx-файлов – например, в файле, созданном в более старой версии программного пакета Microsoft Word, отредактированного в более новой и вновь открытого в прежней, могут исчезать отдельные затронутые редактированием в новой версии пробелы между словами. Данная особенность проявляется нерегулярно и спонтанно, однако построение индекса для такого файла будет бессмысленным ввиду повреждения его оригинального содержимого при подобных манипуляциях.

Определенную сложность представляет извлечение текста из HTML-документов. Связано это с тем, что с точки зрения потребления ресурсов выгодно извлекать текст из скачанных страниц в краулере (в самом поисковом роботе, скачивающем по определенному плану страницы из сети Интернет), в то же время: а) разбор страницы занимает слишком большой промежуток времени, чтобы считать эту операцию быстрой; б) нередки случаи, требующие именно детального анализа страницы (ошибка указания кодировки, ошибочный синтаксис HTML-кода, разбор форматирования и другие случаи). При этом бывает, что HTML-страницы не содержат ошибок в форматировании и не представляют сложности по анализу и извлечению из них любой информации. Совместно эти две проблемы порождают проблему извлечения текста – вполне возможно, что один или несколько методов извлекут ошибочный или неполный текст, извлечение окончится неудачей, текст будет извлечен не в ожидаемой форме (например, часто документы – файлы формата RTF, созданные в Open

Office.Org, открываются в неправильной кодировке в Microsoft Word).

Решение: ввиду многообразия и сложности форматов документов, а также неполной совместимости файлов одного формата, созданных различными программами, следует считать, что текст из контейнера может быть извлечен разными способами. Например, текст из файла формата .doc можно извлечь: Microsoft Word, Libre Office, Open Office.Org. Из HTML – Microsoft Word, Libre Office, Open Office.Org, через COM-объект Internet Explorer, библиотеками движков браузеров Gecko и Web Kit, простым парсером.

Маскировка текста. Извлеченный из полноценного документа текст почти всегда содержит нетекстовые элементы. Например, символы «мягкого» переноса, добавляемые текстовым редактором при каждом переносе слова в абзаце, у которого не запрещены автоматические переносы. Возможно использование различных видов пробелов, разрывов слов, строк.

Решение: данные символы относятся к категории «невидимых», или символов форматирования, и должны быть либо удалены, либо заменены более простыми, эквивалентными им по смыслу. Например: всевозможные разрывы строк – на символы переноса строки, все виды пробелов – на обыкновенный пробел, «мягкие» переносы – удаляются, буква «ё» заменяется на «е» и другие преобразования.

Ошибки в написании слов. Если качество текста научных работ, публикации в периодических и непериодических изданиях контролируются редакторами, корректорами, научными руководителями, то страницы в сети Интернет изобилуют синтаксическими ошибками. Причем ошибки в т. ч. могут содержаться в публикациях в доверенных сетевых источниках и представляющих немалую практическую ценность.

Решение: при построении индекса оказывается достаточно эффективным удалять подряд повторяющиеся символы, а также мягкий и твердый знаки.

Преобразование слов для индексации. Каждое слово сначала приводится к его базовой грамматической форме (этап лемматизации). Потом полученное слово усекается (этап стемматизации) для повышения инвариантности форм слов относительно его суффиксов и окончания. Два этих этапа совместно необходимы для обработки слов, для которых невозможно восстановить базовую форму (например, для неизвестных слов). Например, так делается в Sphinx [1].

Полученный результат служит минимальной индексируемой единицей. На любом из этапов может быть определено, что слово является стоп-словом, и оно исключается из формирования индекса.

Между этапом лемматизации и стемматизации существует этап нормализации слова относительно его синонимов, т. е. слово, если это возможно, заменяет некоторый его синоним, назначенный «базовым». В системе содержится не очень большое количество синонимов, т. к. «базовых» слов (не словосочетаний) для цепочки синонимов в русском языке на самом деле не очень много.

Если все предыдущие этапы подготовки текста к индексации – служебные, то этот этап преследует цель сокращения размера поискового индекса.

Индексация. Этап заключается в уменьшении «длины» слова, а также превращении слова в некий код, пригодный для его упорядочивания (для последующего ускорения поиска). Возможны несколько вариантов индекса. Каждую запись о слове можно представить в виде <docid, wordno, wordid>, где docid – код документа (точнее, текста) в базе данных, wordno – номер слова в тексте, wordid – код слова. Запись существует для каждого слова документа. В идеальном случае docid – это хеш от текста (т. е. не порядковый номер записи в базе данных, а непосредственное идентифицирующее текст значение), wordno можно считать беззнаковым 32-битным целым (достаточно почти для любых случаев), wordid – также MD5-хеш от слова или само слово. Помимо этого для ускорения дополнительно должен быть отсортирован столбец wordid, т. е. данный столбец должен существовать в удвоенном виде – упорядоченным и в составе записи. К сожалению, в таком случае объем даже одной записи оказывается очень большим, в базу данных на доступных носителях объемом в 1 Тб поместится менее 1 млн документов, индексированных документов, не считая накладных расходов на организацию структур хранения.

Проблема решается следующим образом:

1) прямой индекс не является главным и не хранится или вообще не строится;

2) вместо прямого индекса используется инвертированный (при прямом индексе хранится информация – какие слова есть в документе, при инвертированном – какие документы содержат каждое конкретное слово);

3) в качестве wordid используется не уникальный идентификатор слова, а, например, контрольная сумма от слова (например, CRC32 или даже CRC24) (это дает ложноположительные срабатывания при поиске, но устраняется дополнительным анализом или через прямой индекс);

4) индекс хранится в сжатом виде. Обычно используются не обычные алгоритмы побитового сжатия без потерь, а т. н. вычисление дельт (примечание: не путать вычисление дельт с дельта-индексом) (было: 13, 5, 9, 21; стало: 5, 4, 4, 8, т. е. после сортировки: 5, 9, 13, 21, после вычисления дельт: 5, 4 = (9 – 5), 4 = (13 – 5), 8 = (21 – 13)) либо т. н. кодирование переменной ширины (как в Lucene [2]), когда в двоичном представлении числа первые несколько битов хранят длину в байтах двоичного представления числа либо старший бит каждого байта резервируется для признака продолжения двоичного представления числа, связанного с предыдущим байтом, либо любой другой алгоритм.

Хранение индекса. Суть проблемы хранения индекса заключается в том, что при обыкновенном хранении записей прямого индекса в любой реляционной СУБД значение, хранящее docid, повторяется и фактически нигде и никогда не используется, служа основой для индекса по колонке в СУБД. Инвертированный индекс не решает данную проблему, однако делает возможным хранение его в сжатом виде. Оптимально хранить инвертированный индекс в странично организованной памяти или в структурах типа B+ деревьев. В нашей работе было использовано компромиссное решение, не основанное на реляционных СУБД: хранение упорядоченных блоков записей, а также метаинформации. Размер блока в каждом случае, если это возможно, максимален (т. к. чем больше блок – тем быстрее работает двоичный поиск, потому что происходит меньшее количество промежуточных чтений с носителя), в идеальном случае поиск по блоку вырождается, т. к. по метаинформации определяется, что искомого значения в диапазоне значений, хранимых в блоке, нет. Дополнительным достоинством данного типа поиска являет-

ся высочайшая скорость его построения, равная сумме времен упорядочивания блока и записи его на носитель.

ЗАКЛЮЧЕНИЕ

В статье рассмотрены основные этапы построения поискового индекса для систем автоматизированной проверки на заимствования, учета индексируемых документов, приведено оригинальное решение по организации системы хранения поискового индекса.

ЛИТЕРАТУРА

1. Sphinx 2.1.2-release reference manual. Part 3. Indexing. URL: http://sphinxsearch.com/docs/current.html#indexing (accessed: 01.11.2013).

2. Apache Lucene – Index File Formats. URL: http://lucene.apache.org/core/4_5_1/core/org/apache/lucene/codecs/lucene45/package-summary.html#package_description (accessed: 01.11.2013).

БЛАГОДАРНОСТИ: Работа выполнена при финансовой поддержке РФФИ, проект № 12-07-00512.

Поступила в редакцию 20 ноября 2013 г.

Chirkin E.S. SEARCH INDEX FOR SYSTEMS OF ANTI-PLAGIARISM

Some of the problems of building and storing the search index for the systems of automated document checks for the presence of borrowing and some effective solutions are considered..

Kew words: anti-plagiarism; searching; search index.

Чиркин Евгений Сергеевич, Тамбовский государственный университет им. Г.Р. Державина, г. Тамбов, Российская Федерация, программист, старший преподаватель кафедры информатики и информационных технологий, e-mail: chirkin@tsu.tmb.ru

Chirkin Evgeniy Sergeyevich, Tambov State University named after G.R. Derzhavin, Tambov, Russian Federation, Programmer, Senior Lecturer of Informatics and Information Technologies Department, e-mail: chirkin@tsu.tmb.ru