Создание легко обновляемых текстовых индексов

© А.Б. Веретенников

Уральский государственный университет им. А.М. Горького
abvmt2@mail.ru, abvmt@e1.ru

Аннотация

Этот документ описывает структуру данных, предназначенную для создания текстовых индексов, которые легко обновлять. Также дано описание системы, в которой реализованы алгоритмы работы с данной структурой данных.

1 Введение

Для создания текстовых индексов для электронных коллекций, состоящих из естественных текстов, например художественная литература, сайты в сети Internet, в основном используются инвертированные файлы [6] и их аналоги. К сожалению, инвертированные файлы сложны в обновлении. Чтобы добавить новые данные в инвертированный файл часто требуется переписать весь индекс.

В текстовые коллекции часто добавляются новые данные, и требуется быстро обновлять индекс. Автором была разработана новая структура данных – CLB индекс, для создания индекса, в который можно легко добавлять новые данные.

Начальная идея CLB индекса описана в [1]. В этой работе была описана структура данных CLB дерево, разработанная автором данной статьи.

CLB дерево обладает рядом недостатков, которые были устранены с помощью новых алгоритмов, которые описаны в данном тексте.

2 Подержка морфологии

CLB индекс предполагает возможность поиска в текстах с поддержкой морфологии языка. Более того, поддержка морфологии языка позволяет сделать ряд оптимизаций.

3 Идея индекса

CLB дерево представляет собой B-дерево [3, 4, 5], содержащее в себе слова проиндексированных текстов. Вместе с каждым словом в B-дереве сохраняется ссылка на список связанных блоков, в котором сохраняется информация обо всех вхождениях данного слова в текстах.

Например, для каждого вхождения каждого слова сохраняется номер или идентификатор соответствующего документа и позиция слова в этом документе.

Т. е. у нас есть файл, состоящий из блоков заданного размера. Несколько блоков объединяется в список: в каждом блоке из данного списка сохраняется ссылка на следующий блок.

При создании индекса, когда слово встречается первый раз, мы помещаем его в B-дерево и создаем список, состоящий из одного блока, в который записываем информацию о вхождении данного слова в тексте.

Если слово встречается повторно, информация о следующем вхождении добавляется в данный блок. Если свободное место в блоке заканчивается, мы создаем новый пустой блок (увеличивая файл в размере) и прописываем на него ссылку в текущем блоке, и далее информация добавляется в новый блок. Таким образом, мы достигаем возможности легкого обновления индекса.

В [1] были даны теоретические оценки для количества дисковых операций при создании индекса и поиске с его помощью.

Однако, данный подход обладает рядом недостатков. В частности, блоки располагаются не последовательно, что замедляет скорость их чтения при поиске. Данная проблема была решена за счет разработки более эффективных алгоритмов, описание которых дано в разделе 4.1.

Кроме того, некоторые слова редко встречаются в текстах. Соответственно, размер информации о вхождениях для таких слов меньше чем размер одного блока. Следовательно, при описанной организации индекса у нас возникают частично заполненные блоки.

Например, пусть некоторое слово встречается всего один раз. Информация об одном вхождении занимает несколько байт, например, номер документа 4 байта и позиция слова в документе – еще 4 байта, итого 8. При определенных оптимизациях информация об одном вхождении занимает 1–3 байта. Для хранения информации о вхождении данного слова выделяется блок в индексе, размер блока обычно 4–16 кб. Т. е. место на диске расходуется неэффективно.

Труды 10-й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» – RCDL'2008, Дубна, Россия, 2008.

Данная проблема также была решена, описание дано в разделе 4.3.

4 Построение индекса

4.1 Организация данных для эффективного чтения

Зафиксируем некоторое число c и соответствующее ему число $m = 2^c$. Цель алгоритма заключается в том, чтобы организовать хранение данных таким образом, чтобы список блоков был разбит на группы, и все блоки, входящие в заданную группу были расположены в файле последовательно.

Например, если у нас есть 25 блоков и $m = 8$. Разбиваем 25 блоков на группы следующих размеров 8, 8, 8, 1.

Т. е. весь список блоков разделяется на группы по m блоков, за исключением последней группы, в которой может быть меньше чем m блоков, т. к. длина списка может не делится нацело на m.

При проведении экспериментов размеры блоков брались 4–16 килобайт. Число m было выбрано таким образом, чтобы суммарная длина группы последовательно располагающихся блоков была не меньше 8 мегабайт.

В результате скорость считывания данных из списка блоков практически совпадает со скоростью последовательного чтения данных.

4.2 Алгоритм

Далее кратко описан алгоритм записи списка блоков подобным образом.

Вначале у нас есть только один блок. Далее, пусть у нас есть k заполненных подряд расположенных блоков B_1, …, B_k, в частности последний блок также заполнен, и нам требуется взять где-то новый блок.

Рассмотрим случай, когда $k = 2^x$, где x – некоторое целое число, меньшее c (число c было введено в начале раздела 4.1).

Мы ищем $2k$ подряд располагающихся блоков N_1, … N_{2k}. Затем копируем информацию из старых k блоков в первую половину новых блоков, т. е. в блоки N_1, … N_k соответственно. Старые блоки B_1, …, B_k помечаются как свободные. Запись далее осуществляется в блок N_k+1. Остальные блоки N_{k+2}, …, N_{2k}, помечаются как зарезервированные.

Как мы ищем $2k$ подряд располагающихся блоков? Поиск осуществляется либо среди блоков, обозначенных как свободные, или мы добавляем блоки в конец файла, увеличивая его размер.

Если $x = c$, то это означает, что у нас есть группа подряд располагающихся блоков длины m. В этом случае мы начинаем формировать новую группу блоков.

В остальных случаях, т. е. если k не является степенью 2, мы осуществляем запись в блоки, которые были ранее зарезервированы описанным выше способом.

4.3 Эффективное заполнение блоков

Для эффективного заполнения блоков введем новый тип блоков. Пусть некоторые блоки разделяются на части равного размера. При этом, некоторым словам может соответствовать не список блоков, в крайнем случае состоящий из одного блока, а только часть блока. Если блок делится на части, то в нем же сохраняется информация о том, какие из его частей заполнены, а какие свободны. Разработан эффективный алгоритм управления такими блоками.

Как только суммарный объем информации для заданного слова становится достаточно большим, информация перемещается в новый пустой блок обычного типа.

5 Кэширование

5.1 Кэширование для слов, входящий в словарь морфологического анализатора

Очевидно, что если мы будем каждый раз при добавлении информации о вхождения слова записывать данные в файл, мы будем создавать индекс очень долго.

Основная оптимизация осуществляется за счет использования морфологии языка. Автором использовался морфологический анализатор [2].

Слова, входящие в состав данного анализатора составляют 80-90% от общего количества слов в обычных тестах (например, художественная литература, новости). В словарь анализатора входит 3,5 млн словоформ русского языка, образованных от 205 тыс. базовых форм.

Словарь анализатора позволяет по словоформе получить набор ее базовых форм. Для большинства словоформ существует только одна базовая форма, однако для многих словоформ существует несколько базовых форм. В данной работе автор отождествляет термины «слово» и «словоформа». Где это необходимо используется термин «базовая форма слова».

При сохранении данных о вхождении слова в тексте, слово сначала приводится в своей базовой форме. Если базовых форм несколько, то информация добавляется для всех базовых форм.

Соответственно для каждой базовой формы слова существует список блоков, для хранения информации о вхождении данной базовой формы в текстах.

При определенном размере блока мы можем сохранять в оперативной памяти последний блок для каждого из этих списков блоков. Этот блок сохраняется на диск, только когда он становится полностью заполненным.

Данный подход можно применять только для слов, входящих в словарь морфологического анализатора, т. к. мы знаем количество базовых форм и можем заранее создать кэш требуемого размера.

Слов, не входящих в словарь анализатора может быть гораздо больше, например, в одной из используемых автором текстовых тестовых коллекций бы-

ло 65 млн различных слов, не входящих в словарь морфологического анализатора. О том, как обрабатывать такие слова, описано в разделах 5.3, 5.4.

5.2 Оптимизация кэширования

Метод, описанный в разделе 5.1, задает ограничения на размеры блоков.

Например, если у нас есть 200 000 базовых форм слов и размер блока 4 кб., то общий размер кэша равен 800 мб. Соответственно мы не можем значительно увеличивать размер блока.

Чтобы снять эти ограничения все базовые формы слов разделяются на группы. Для каждой группы базовых форм слов создается временный файл.

Вначале при чтении файлов запись информации о вхождениях слов осуществляется во временные файлы. Затем независимо обрабатывается каждый временный файл. При этом требуется организовывать кэш только для базовых форм слов, входящих в соответствующую группу.

Например, пусть у нас есть 200 000 базовых форм слов. Разделим этот набор на группы из 5000 базовых форм слов в каждой.

В результате, при обработке конкретной группы требуется сохранять в памяти только 5000 блоков, что требует значительно меньше памяти и позволяет увеличить размер блока и соответственно уменьшить количество операций записи.

Автор обычно использует блоки, размер которых равен 16 кб. В результате проведенных испытаний, не выявилось значительного повышения производительности при увеличении размера блока до 32–64 кб. Следует отметить, что имеет смысл выбирать размер блока кратным размеру кластера файловой системы.

5.3 Кэширование для слов, не входящих в словарь морфологического анализатора

Подход, описанный в разделе 5.1, мы не можем использовать для слов, не входящих в словарь анализатора. В данном случае можно реализовать некие стандартные стратегии кэширования, например, при необходимости добавления нового блока в кэш выгружать из кэша блок, запись в который наиболее давно не производилась.

5.4 Оптимизированная обработка слов, не входящих в словарь морфологического анализатора

Автором также был реализован подход описанный далее. Слова, не входящие в состав морфологического анализатора, как правило, встречаются очень редко. Соответственно, суммарный объем информации об их вхождениях достаточно мал.

Предлагается сохранять информацию о вхождениях для нескольких слов в одном списке блоков. При этом, при записи информации о вхождении, вместе с ней сохраняется тег, определяющий, к какому слову она относится.

Данный метод значительно сокращает количество операций записи при создании индекса, при этом эффективность поиска снижается незначительно.

Например, пусть в одном списке блоков сохраняется информация о вхождениях слов: *ааа*, *ббб*, *ввв*. Присвоим этим словам номера 1, 2, 3. Если мы ищем слово *ааа*, то мы считываем весь этот список блоков. При чтении из списка блоков информации о каждом вхождении считываем также и тег, соответствующий данному вхождению. Рассматриваем только те записи, у которых тег равен 1.

При создании индекса контролируется, сколько раз уже встретилось слово. Если оказывается, что слово встречается слишком часто, то для него создается новый список блоков, в который переносится информация о вхождениях данного слова, далее в данном списке блоков будет сохранена информация только об этом слове.

Автором был проведен эксперимент. Было зафиксировано число 4096, и для слов, число вхождений которых превышало данное число, создавались отдельные списки блоков.

Для тех слов, число вхождений которых меньше или равно 4096, в один список блоков помещалась информация сразу о 32 различных словах.

Обрабатывалось 35 гб. текстов, содержащих 69 млн. различных слов.

В результате оказалось, что максимальное количество записей в списке блоков, для списков, содержащих несколько слов, было 19 431.

Данное число показывает, что эффективность не пострадала, т. к. объем указанных списков блоков не превышает нескольких десятков килобайт (т. к. одна запись занимает примерно 2–4 байта).

Слов, которые встречаются больше чем 4096 раз, оказалось всего 19 593.

7 Обработка стоп слов

Как правило, часто встречающиеся слова, например предлоги, не добавляются в индекс, т. к. считается, что они не несут в себе нового смысла. Подобные слова называются стоп словами и составляют до 30-40% текста. Кроме того, отказ от индексирования таких слов позволяет значительно увеличить производительность.

Примеры стоп слов
"и", "в", "он", "не", "я", "что", "на"

Частота встречаемости данных слов в типичных текстах

3.025%,2.298%,1.767%,1.609%,1.432%,1.284%

Если все-таки потребуется искать фразу, содержащую слово "и", то мы столкнемся с большими трудностями. Слово "и" встречается в 3% случаях, соответственно объем данных, соответствующих слову "и" очень большой.

Для ускорения подобного поиска реализован следующий опциональный вариант: вместе с вхождением каждого слова сохранять также информацию о местонахождении рядом с ним стоп слов.

Это позволяет осуществлять эффективный поиск точный фраз, содержащих стоп слова.

8 Поиск

За счет алгоритма создания индекса, описанного в разделе 4, по своим возможностям и эффективности CLB индекс не уступает инвертированным файлам.

Реализованы различные виды поиска, в частности поиск слов с учетом расстояния между ними и точный поиск фраз. Поиск может быть осуществлен с учетом или без учета порядка слов.

Были проведены эксперименты, показывающие, что скорость чтения информации о вхождениях одного слова из индекса, совпадает со скоростью линейного чтения с диска.

9 Эксперименты по созданию индекса

9.1 Эксперимент 1

Обрабатывались файлы, сжатые в архивы формата RAR. В основном файлы имели формат txt, html.

Суммарный размер файлов 16,3 гб (сжатый размер).
Суммарный размер файлов 42,5 гб (несжатый размер).
Суммарный размер текста 36,5 гб.
Всего файлов 300 тысяч.
Доля словоформ, входящих в словарь морфологического анализатора 87%.
Различных словоформ, не входящих в словарь морфологического анализатора – 65 млн.
Время создания индекса 7 часов, из них 4 часа – чтение файлов, 3 часа – создание индекса.
Объем индекса 34 гб.

Эксперимент проводился на следующей конфигурации:

Процессор: Intel Core 2 Duo E6700, 2.66 GHz, кэш: L1 Data – 2 x 32 кб, L1 inst. 2 x 32 кб, L2 – 4096 кб.
Оперативная память: 4 гб, DDR2 800.
Жесткий диск: Seagate Barracuda 7200.10, 7200 RPM, кэш 16 мб., объем 500 гб.
FSB 1066 MHz.

9.2 Эксперимент 2

Обрабатывались не сжатые файлы форматов txt, html.

Суммарный размер файлов 75 гб.
Суммарный размер текста 70 гб.
Всего файлов 600 тысяч.
Доля словоформ, входящих в словарь морфологического анализатора 93%.
Время создания индекса 10 часов, из них 4 часа – чтение файлов, 6 часов – создание индекса.
Объем индекса 50 гб.

Эксперимент проводился на следующей конфигурации:

Процессор: Intel Pentium 4, 3.0 GHz, кэш: L1 Data – 16 кб, L1 trace – 12 Kuops, L2 – 2048 кб.
Оперативная память: 2048 гб, DDR2 533.
Жесткий диск: Seagate Barracuda 7200.8, 7200 RPM, кэш 8 мб., объем 200 гб.
FSB: 800 MHz.

Последний эксперимент был повторен, при этом запись индекса была осуществлена в 2 приема. Вначале был создан индекс на основании 40 гб текста, затем в него были добавлены остальные данные. Суммарное время в этом случае составило 12 часов.

Оба эксперимента были осуществлены без отбрасывания стоп слов.

10 Сравнение с инвертированными файлами

10.1 Описание тестовых данных

Было произведено сравнение по скорости создания индекса. Обрабатывались текстовые документы, общий объем 35,2 гб, 191 074 файла. Все файлы были в кодировке Windows-1251 (CP1251). Язык документов – русский. Все файлы представляли собой обычный текст.

10.2 Описание конфигурации оборудования

Эксперименты проводились на следующей конфигурации:

Процессор: Intel Core 2 Duo E6700, 2.66 GHz, кэш: L1 Data – 2 x 32 кб, L1 inst. 2 x 32 кб, L2 – 4096 кб.
Оперативная память: 4 гб, DDR2 800.
Жесткий диск: Seagate Barracuda 7200.10, 7200 RPM, кэш 16 мб., объем 750 гб.
FSB 1066 MHz.

10.3 Создание индекса

Создание инвертированного файла: время 9 часов, размер 40 гб.

Создание CLB индекса: время 3 часа, 32 мин., размер 24 гб.

10.4 Добавление в индекс одного файла среднего размера

Был проведен замер скорости добавления в индекс одного документа, размер документа 1,2 мб.

Время добавления одного документа 1,2 мб. для CLB индекса: 9 мин.

Время добавления одного документа 1,2 мб. в инвертированный файл: 57 мин.

10.5 Добавление в индекс одного файла малого размера

Был проведен замер скорости добавления в индекс одного документа, размер документа 534 байта.

Время добавления одного документа размером 534 байта для CLB индекса: 22 с.

Время добавления одного документа размером 534 байта в инвертированный файл: 57 мин (т. е. такое же, как при размере файла 1,2 мб).

10.6 Поиск

Время поиска в инвертированном файле и CLB-индексе практически совпадают.

10.7 Выводы

Проведенные эксперименты показывают высокую эффективность CLB индекса при добавлении в него данных небольшого размера.

11 Сравнение с существующими разработками

11.1 Описание тестовых данных

Было произведено сравнение по скорости создания индекса и размеру индекса. Использовались те же документы, которые указаны в разделе 10.

11.2 Описание конфигурации оборудования

Эксперименты проводились на следующей конфигурации:

Процессор: Intel Pentium 4, 3.0 GHz, кэш: L1 Data – 16 кб, L1 trace – 12 Kuops, L2 - 2048 кб.
Оперативная память: 4 гб, DDR2 533.
Жесткий диск: Seagate Barracuda 7200.8, 7200 RPM, кэш 8 мб., объем 200 гб.
FSB: 800 MHz.

11.3 SearchInform Desktop

http://www.searchinform.com

Размер индекса 16,15 гб.
Время создания 9 часов.

11.4 Архивариус 3000

http://www.likasoft.com/

Размер индекса 24,83 гб.
Время создания 6 часов 46 мин.

11.5 Создание CLB индекса

Размер индекса 26,2 гб.
Время создания 5 часов 49 мин.

11.6 Выводы

Эксперименты показывают высокую скорость создания CLB индекса.

12 Библиотека для создания индекса

Автором разработана библиотека для создания индексов и поиска в текстах, в которой реализована описанная структура данных и алгоритмы.

Библиотека может индексировать файлы в различных форматах, например RTF, PDF, CHM, HTML, DJVU и кодировках, например UNICODE, UTF8, CP1251, ASCII, KOI8.

Поддерживается обработка архивов форматов ZIP, CAB, RAR, 7Z, ARJ, TAR, и др.

Библиотека реализована в виде COM сервера для операционных систем Windows.

Состав библиотеки

1. Ядро, осуществляет создание индекса и поиск.
2. Модуль поддержки морфологии.
3. Модуль распознавания кодировки. При распознавании кодировки также учитывается морфология.
4. Модуль поддержки форматов файлов. Поддержка форматов файлов и архивов реализована с помощью подключаемых дополнительных модулей, которые могут быть реализованы в виде динамических библиотек или написаны на Java. Модуль поддержки форматов файлов реализован в виде отдельного процесса для повышения надежности системы.
5. Модуль атрибутов документов, для сохранения описания документов.
6. Модуль репозитария, для сохранения текстов документов. Создается для того, чтобы при поиске можно было быстро получать фрагмент текста, содержащий найденную фразу. Тексты в репозитарии могут сохраняться с использованием различных алгоритмов сжатия.
7. Модуль COM осуществляет доступ к остальным модулям извне с помощью COM, что позволяет использовать библиотеку в различных языках программирования.

Реализованные алгоритмы достаточно нетребовательные к ресурсам компьютера. Для создания индекса достаточно иметь 300–400 мегабайт свободной оперативной памяти.

Автором проводились эксперименты по созданию индексов на машине с оперативной памятью размером 512 мб.

Следует отметить, что эффективность описанных в данном документе алгоритмов значительно возрастет с применением дисков SSD (Solid-state drive), за счет более быстрого чтения блоков малого размера. При этом эффективность таких структур данных как инвертированные файлы возрастет менее, т. к. для добавления в инвертированный файл информации его все равно придется практически переписать целиком.

Литература

[1] Веретенников А. Б., Лукач Ю. С. Еще один способ индексации больших массивов текстов // Известия Уральского государственного университета. Сер. «Компьютерные науки». – 2006, №43. – С. 103–122.

[2] Лукач Ю. С. Быстрый морфологический анализ флективных языков // Междунар. алгебраическая конф. : К 100-летию со дня рождения П. Г. Конторовича и 70-летию Л. Н. Шеврина : Тез. докл. – Екатеринбург : Изд-во Урал. ун-та, 2005. – С. 182–183.

[3] Bayer, R., McCreight, E. Organization and maintenance of large ordered indexes. Acta Informatica 1, 3 (1972), 173-189.

[4] Ferragina, P., Grossi, R. The string B-tree: a new data structure for string search in external memory and its applications. Journal of the ACM, 46, 2 (1999), 236-280.

[5] Ferragina P., Grossi R. An experimental study of SB-trees. 7th ACM-SIAM symposium on Discrete Algorithms, 1996.

[6] Prywes, N. S., Gray, H. J. The organization of a Multilist-type associative memory. IEEE Trans. on Communication and Electronics, 68 (1963), 488-492.

The Effective Creating of Easy Updatable Text Indexes

A.B. Veretennikov

Author introduce a new data structure for fast indexing of large volumes of texts and a software system based on this structure. The latter is more effective for inserting data into indexes than other known data structures, and is comparable with them in the data retrieval.