УДК 519.683.5

Создание дополнительных индексов для более быстрого полнотекстового поиска фраз, включающих часто встречающиеся слова

Creating additional indexes for fast full-text searching phrases that contains frequently used words

А. Б. Веретенников
A. B. Veretennikov

Уральский федеральный университет.
Институт математики и компьютерных наук (ИМКН).
Ural Federal University.
Institute of Mathematics and Computer Science (IMCS).

Полнотекстовый поиск, поисковые системы, инвертированные файлы, дополнительные индексы.

Рассматриваются задачи поиска фраз и наборов слов в большом объеме текстов с помощью дополнительных индексов. Применение дополнительных индексов может более чем в десять раз снизить максимальное время выполнения поискового запроса. Приведен алгоритм создания дополнительных индексов для часто используемых слов.

Full-text search, search engines, inverted files, additional indexes.

The problems of searching phrases in the large text arrays are considered and solved using additional indexes. With additional indexes we can perform search query more than ten times faster than with standard inverted files. Algorithms for creating additional indexes for frequently used words are described.

Введение

Для решения задач полнотекстового поиска, то есть поиска слов или фраз в текстах используются инвертированные файлы и их аналоги [1-4], которые содержат в себе записи о вхождениях слов в обрабатываемых текстах.

Слова в документах встречаются с различной частотой. Такой важный параметр, как максимальное время выполнения поискового запроса, определяется наиболее часто встречаемыми словами. Скорость поиска можно повысить за счет создания дополнительных индексов для часто используемых слов [5]. В [6] рассмотрен способ применения дополнительных индексов, который позволяет более чем в десять раз

снизить максимальное время выполнения поискового запроса. В данной работе мы рассматриваем алгоритм создания расширенных индексов, применение которых описано в [6]. Мы продолжаем использовать терминологию и обозначения из [5] и [6].

Виды слов

Согласно [5], все слова мы разделяем на три группы:

1) «Стоп слова»: и, в, или. Встречаются очень часто и могут вообще не включаться в индекс, поэтому их и можно называть стоп словами. Например, предлоги. Далее будем называть данные слова стоп словами, даже если в каком-то виде включаем информацию о них в индекс.
2) Часто используемые слова. Встречаются часто, но несут в себе существенный смысл и должны включаться в индекс.
3) Остальные, будем называть их «обычные слова».

Морфологический анализатор

При создании индекса автор использует морфологический анализатор. Для каждой словоформы, входящей в словарь анализатора он возвращает список номеров базовых форм слов. Номер базовой формы это число в диапазоне от 0 до WordsCount – 1, где WordsCount – число различных базовых форм (около 260 тыс. для используемого словаря). Если слово не входит в словарь анализатора, то, будем считать, что его базовая форма совпадает с самим словом.

Базовые формы слова также называются леммами, а сам процесс получения набора лемм по словоформе – лемматизацией.

С учетом морфологического анализа разделение слов на три группы при использовании анализатора применяется не к исходным словоформам, а к леммам. Т. е. есть три типа лемм в смысле частоты встречаемости: стоп леммы, часто используемые леммы и остальные.

В данной работе предполагаем, что в поисковом запросе могут встречаться любые слова, независимо от их частоты встречаемости.

Если отсортировать все леммы в порядке убывания частоты их появления в текстах, то скажем 100-400 самых часто встречаемых из них – стоп леммы, далее 500-5000 – часто используемые леммы. Количество стоп и часто используемых лемм зависит от числа поддерживаемых анализатором языков, в данной работе их два, русский и английский.

Слово, входящее в словарь анализатора, назовем известным словом, а его леммы известными леммами. Слова и леммы, не входящие в словарь анализатора назовем неизвестными словами и леммами, соответственно.

Структура инвертированного файла

Инвертированный файл представляет собой набор записей вида (ID,P), ID – идентификатор документа, P – позиция, например порядковый номер, слова в документе. Запись (ID,P) будем называть записью о вхождении слова в документе или словопозицией. Все записи, соответствующие одной лемме, хранятся последовательно для их быстрого чтения при поиске. Идентификатор документа ID будем считать целым числом, например, номером документа.

Словопозиция также может включать в себя дополнительные поля или данные. В качестве примера можно рассмотреть, информацию о располагающихся рядом с текущим словом стоп-словах [5, 7, 8], которую мы будем называть далее NSW-записью (от Near Stop Words).

Будем считать, что словопозиция A меньше словопозиции B, если A.ID < B.ID или (A.ID = B.ID и A.P < B.P).

Мы также можем определить расстояние между словопозициями A и B, как |B.P – A.P|, если B.ID = A.ID, иначе "максимальное число" (используем целые числа с фиксированным числом бит, например, 64 битное число, максимальное число в этом случае определяется как самое большое число, которое можно закодировать).

Виды поисковых запросов

В данной работе рассматриваются задачи поиска фраз или наборов слов в текстах, поисковый запрос это несколько слов, а результат запроса это список документов с указанием позиций в них, где встречаются заданные слова или фраза. Могут быть рассмотрены задачи:

1) Поиск без учета расстояния, то есть поиск документов, содержащих заданный набор слов.
2) Точный поиск фразы.
3) Поиск с учетом расстояния, ищем документы, где искомые слова располагаются как можно ближе друг к другу. Данная задача требует сохранения информации в индексе о каждом вхождении каждого слова в документах.
4) Различные дополнительные условия поиска с применением операций «и», «или», «отрицание», например, поиск документа содержащего слово А или Б, поиск документа, содержащего слово В, но не содержащего слово Г.

Структура индекса

Индекс обычно состоит из двух компонентов:

1) Инвертированный индекс. Содержит словопозиции.

2) Таблица дескрипторов. Содержит для каждой леммы дескриптор, содержащий информацию о том, где в инвертированном индексе содержатся словопозиции, которые соответствуют данной лемме. Таблица хранится в оперативной памяти или в ассоциативном массиве, хранящемся во внешней памяти, например, B-дереве.

Обновлением индекса называется процесс добавления в уже существующий индекс новых данных, например, есть индекс построенный по набору текстов 10 Гб., и мы индексируем еще 10 Гб. новых текстов.

В рамках теоретического дискурса сложилось два способа формирования индексов:

Способ 1. Внешняя сортировка слиянием.

Записываем в файл словопозиции, затем сортируем его таким образом, чтобы словопозиции, соответствующие одной лемме, шли подряд.

При обновлении индекса создается новый индекс и осуществляется процесс слияния предыдущего и нового индекса.

Поскольку операция слияния индексов весьма затратная, в некоторых случаях организуют несколько индексов. Но, при наличии нескольких индексов мы увеличиваем число операций ввода-вывода в разных местах диска при поиске. Поэтому число индексов требуется ограничивать неким числом. При каждом добавлении новых данных в набор индексов создаем новый индекс. Когда этих индексов становится слишком много, выбираются некоторые из них и сливаются. См., например, в [3].

Способ 2. Легко обновляемые индексы.

Словопозиции хранятся в наборе блоков, которые могут располагаться в разных местах инвертированного индекса. Процесс создания индекса заключается в последовательном добавлении в индекс словопозиции, в процессе чего набор блоков может меняться, например, могут добавляться новые блоки, и обновлении дескриптора при необходимости.

Обновление индекса осуществляется аналогично созданию индекса, т. е. заключается в последовательном добавлении в существующий индекс новых записей. В качестве примера можно рассмотреть [4].

При создании данного индекса требуется больше операций ввода-вывода. Это может быть решено различными способами организации кэш памяти и стратегиями формирования набора блоков. Также необходимо, чтобы блоки одной леммы располагались преимущественно подряд для снижения числа операций ввода-вывода при поиске. Алгоритмы, решающие данные проблемы рассмотрены автором в [7-10]. Преимущество данного способа в отсутствии процедуры слияния при обновлении индекса.

Рассматриваемый нами индекс относится ко второму типу индексов.

Кроме обычного индекса, который для каждой леммы содержит список ее вхождений в документах, можно создавать дополнительные индексы для стоп-лемм и часто используемых лемм.

Далее мы будем рассматривать создание индекса для часто используемых лемм, использование которого рассматривалось в [6]. Данный индекс в [6] был обозначен как "расширенный индекс ".

Создание индекса

При создании индекса [8] применятся двух этапный или двух уровневый процесс.

Все леммы разделяются на группы, для известных лемм размер группы определен параметром, например, 1000, т. е. весь набор известных лемм делится на части по 1000 штук в каждой. Для неизвестных лемм используется хеширование, число групп задается параметром, например, 32.

На первом этапе создаются временные файлы, один временный файл соответствует одной группе лемм и содержит словопозиции для лемм данной группы.

Получается какое-то число файлов для групп известных лемм, назовем из KW-файлами и еще сколько-то файлов для групп неизвестных лемм, UW-файлы.

KW (Known Words) и UW (Unknown Words) файлы будем также называть O-файлами, Ordinary, или обычными временными файлами.

Во временных файлах вместе со словопозицией также содержится информация о лемме вхождения, для известных лемм это номер леммы в ее группе, для неизвестных, само слово. Вместе со словопозицией может сохраняться дополнительная информация, например, о располагающихся рядом стоп формах. Словопозиции во временных файлах сохраняются в порядке их встречаемости в текстах.

На втором этапе последовательно обрабатываются временные файлы конкретной группы, и данные из них добавляются в индекс (цикл по временным файлам).

При создании расширенных индексов также используются временные файлы. В отличие от создания индекса без расширенных индексов, где временный файл на втором этапе требуется прочитать ровно один раз последовательно, при создании расширенных индексов мы читаем временные файлы несколько раз.

Оптимальный подход, в данном случае, заключается в хранении временных файлов в оперативной памяти. Возможность легкого обновления индекса позволяет нам разбить индексируемый набор текстов на части и последовательно добавлять эти части в индекс.

Мы можем определить, какой объем оперативной памяти будем использовать для хранения временных файлов, например 20 Гб. Затем, когда вся память заполнена, мы переносим данные в индекс в рамках второго этапа. Затем очищаем временные файлы и продолжаем процесс.

Параметры создания расширенного индекса

Далее приведем список параметров с примером значения:
WS_COUNT = 700. Количество стоп лемм.
ADVANCED_INDEXES_WORDS = 2100. Количество часто используемых лемм.
ADVANCED_INDEXES_DISTANCE_MAP=5,500,6,500,7,500.
В [6] расширенный индекс (w,v) это список вхождений слова w, когда в тексте не более чем на расстоянии ProcessingDistance от w присутствовало слово v. С учетом морфологического анализа под w и v мы понимаем леммы слов. Лемма w является часто используемой, лемма v – произвольная.

Параметр ProcessingDistance может быть задан различным в зависимости от частоты встречаемости леммы w в текстах. Мы считаем, если расстояние между словами меньше или равно ProcessingDistance, то слова связаны по смыслу друг с другом, иначе нет.

Следует отметить, что если для обеих лемм w и v существует расширенный индекс, т. е. существуют индексы (w,v) и (v,w), то достаточно создать один из них, например, (w,v), но для каждого вхождения слова w сохранять также расстояние до v. Это позволяет уменьшить объем индекса. В данной работе мы будем определять параметр ProcessingDistance на основании данного параметра. В рассмотренном примере он означает, что для первых 500 часто используемых слов используется ProcessingDistance = 5, для следующих 500-от, значение 6, для последующих 500-от, значение 7.
ADVANCED_INDEXES_DISTANCE = 7. Определяет значение ProcessingDistance для тех часто используемых лемм, номера которых выходят за пределы значений, определяемых параметром ADVANCED_INDEXES_DISTANCE_MAP. Таким образом, мы определяем ProcessingDistance для каждой часто используемой леммы.
ADVANCED_INDEXES_DESC_SIZE = 50. Набор часто используемых лемм разделим на группы, назовем их группами часто используемых слов (ГЧИС). Количество элементов в ГЧИС определяем данным параметром. Каждой ГЧИС будет соответствовать отдельный набор файлов индекса, таким образом, мы имеем индекс ГЧИС. В индексе ГЧИС будут сохраняться расширенные индексы (w,*) где w входит в эту ГЧИС, а * – произвольная лемма. Каждая ГЧИС может обрабатываться в отдельном потоке. Индексы разных ГЧИС могут храниться на разных дисках для

оптимизации ввода-вывода. Кроме разделения на группы фиксированного размера может применяться разделение с учетом суммарной частоты встречаемости лемм группы.

Объект-лента или итератор записей о вхождении

Введем понятие объекта-ленты или итератора словопозиций. Данный объект имеет операцию, возвращающую следующую запись. К примеру, если у нас есть временный файл, то можно сопоставить с ним объект-ленту, с помощью которого можно прочитать друг за другом все записи из этого файла. Записи, которые выдает объект-лента, должны быть упорядочены, то есть следующая запись должна быть не меньше предыдущей. В случае одного временного файла это выполняется, так как записи в него сохраняются по порядку.

Также можно сопоставить объект-ленту с несколькими файлами. В этом случае при получении следующей записи нужно анализировать записи в каждом из файлов и выдавать минимальную запись, затем смещаться к следующей записи в ее файле. Для этого можно использовать бинарную кучу [11].

При создании индекса каждая ГЧИС обрабатывается независимо.

При этом создаются два вида итераторов.

Итератор ГЧИС должен перебирать все записи, которые соответствуют леммам обрабатываемой ГЧИС.

Можно рассмотреть два варианта реализации данного итератора.

1) Запись лемм ГЧИС в отдельном файле. Введем дополнительные временные файлы, назовем их AI-временными файлами. Каждой ГЧИС соответствует один AI-временный файл. Вносим изменение в первый этап. Когда словопозиция записывается во временный файл, определяем, является ли лемма часто используемой. Если да, то определяем ее AI-временный файл и записываем словопозицию также и в этот файл. Кроме самой словопозиции записываем номер леммы в ГЧИС файла. Хотя мы тратим больше памяти для хранения дублирующей информации, это дает нам преимущества в скорости работы и именно этот способ мы используем.
2) Использование обычных временных файлов. В этом случае требуется организовать итератор, который одновременно читает несколько обычных временных файлов и при этом фильтровать читаемые записи, передавая из итератора только те, которые входят в текущую ГЧИС.

Введем также дополнительное поле Skip для словопозиции, которую возвращает итератор ГЧИС. Это поле по умолчанию имеет значение 0, оно предназначено для того, чтобы мы могли исключать запись в индекс определенных записей, смысл будет пояснен далее.

Мы также предполагаем, что в словопозициях итератора ГЧИС может сохраняться информация о располагающихся рядом стоп-словах, т. е NSW-записи. Эта информация может сохраняться в индексе и использоваться при поиске, но ее использование выходит за рамки текущей статьи.

Второй итератор предназначен для чтения O-временного файла.

Процесс создания индексов

1) Чтение текстов и запись временных файлов.
2) Запись обычных индексов.
3) Запись расширенных индексов.
4) Запись индексов стоп слов (см. [5])
5) Запись индексов фраз стоп слов (см. [5]).

Предварительная обработка перед записью расширенных индексов.

Поскольку запись расширенных индексов осуществляется после записи обычных индексов, мы можем переписать обычные временные файлы, которые относятся к известным леммам. Для этого для каждого временного файла известных лемм создается новый временный файл. Затем мы переписываем друг за другом записи из исходного файла в новый. При этом память исходных временных файлов освобождается. Смысл этого процесса заключается в том, что в исходных файлах словопозиции могут содержать NSW-записи, которые не нужны при построении расширенных индексов в случае обычных временных файлов. Алгоритм, описанный далее, не зависит о того, осуществляем ли мы эту предварительную обработку или нет.

Данная предварительная обработка применяется, только если мы используем AI-временные файлы, так как для итератора ГЧИС требуется сохранение NSW-записей в словопозициях.

Создание расширенных индексов заключатся в независимой обработке каждой ГЧИС.

Обработка группы часто используемых слов

Для каждой ГЧИС мы запускаем два потока. Число одновременно работающих потоков ограничено доступным объемом оперативной памяти и размером кэша создания индекса.

Аргументы потока это номер группы часто используемых слов и набор обычных временных файлов. Первый поток запускается на наборе из всех KW-временных файлов. Второй поток на наборе UW-временных файлов. Действия, осуществляемые в отдельном потоке:

Перебираем в цикле каждый из О-временных файлов, которые являются аргументами потока. Для каждого из файлов выполняем:

1) Создаем итератор для О-файла.
2) Создаем итератор для группы часто используемых слов.
3) Читаем записи из обоих итераторов и передаем их в конвейер записи. Записи передаются упорядоченно, то есть последующая запись не меньше предыдущей. Смотрим на текущую запись каждого итератора и выбираем из них наименьшую для передачи в конвейер записи, и переходим в соответствующем итераторе к следующей записи.

Конвейер записи

Конвейер записи это специальный объект, который контролирует запись данных в расширенный индекс.

Он содержит в себе очередь словопозиций. В конвейер помещаются словопозиции, при этом последующая словопозиция должна быть не меньше предыдущей.

В очереди каждая словопозиция содержит также в себе дополнительный признак, переменную Processed, которая равна 0 при помещении записи в очередь. Также на этом этапе словопозиция содержит в себе поле Word, которое определяет лемму словопозиции.

Предполагается, что в очереди конвейера между первой и последней записью расстояние не более ProcessingDistance*2. Перед добавлением новой записи в очередь, пока расстояние между первой записью очереди и новой записью больше ProcessingDistance*2, мы вызываем процедуру удаления первого элемента очереди RemoveFirstItem.

В этой процедуре мы идем от начала очереди и обрабатываем те элементы очереди, которые соответствуют часто используемым леммам текущей ГЧИС, ранее не были обработаны, и расстояние между началом очереди и обрабатываемым элементом не более ProcessingDistance. У обработанных элементов выставляем Processed в 1, чтобы их обработка не происходила более одного раза. После обработки всех элементов очереди, которые удовлетворяли указанным условиям, первый элемент очереди удаляется из начала очереди.

Процедура удаления первого элемента из очереди вызывается нужное число раз, пока не будут достигнуты условия для добавления новой записи в очередь. Только после этого в конец очереди добавляется новый элемент.

Это означает, что в момент обработки записи леммы ГЧИС, в очереди находятся все элементы читаемых временных файлов, слова которых находятся в текстах на расстоянии не более ProcessingDistance от данной записи, раньше или позже ее.

Обработка записи R, функция ProcessItem, означает запись в индекс информации обо всех записях, которые располагаются рядом с R. Пусть

запись R соответствует часто-используемой лемме R.Word. Рассматриваем все записи T из очереди, которые близки к R, то есть |R.P – T.P| <= ProcessingDistance. Для каждой такой записи T записываем в расширенный индекс (R.Word, T.Word) словопозицию R, с информацией о расстоянии между R и T в текстах.

Переменные конвейера:

1) First. Ссылка на первый элемент очереди.
2) NextProcessItem. Ссылка на первый необработанный элемент очереди, с условием Processed = 0, лемма которого соответствует ГЧИС. Эта переменная вводится для оптимизации перебора элементов очереди в функции RemoveFirstItem.
3) FileId. ID текущего документа.

Помещение записи о вхождении в конвейер

Функция Push(A), A – словопозиция.

1) Если A.ID != FileId, то мы завершаем обработку текущего документа, то есть последовательно вызываем функцию удаления первого элемента очереди, RemoveFirstItem, пока все элементы не будут удалены из очереди.
2) Вызываем процедуру проверки очереди CheckItems(A).
3) Словопозиция A помещается в конец очереди, присваиваем FileId = A.ID.

Процедура проверки очереди

Функция CheckItems(A), A – словопозиция.

Пока (A.P – First.P) > ProcessingDistance * 2, осуществляется удаление первого элемента очереди, то есть вызов функции RemoveFirstItem.

Процедура удаления первого элемента очереди

Функция RemoveFirstItem().

Вводим локальную переменную T, которая равна NextProcessItem, если NextProcessItem не NULL, иначе равна First.

Вводим локальную переменную MaxP = First.P + ProcessingDistance.

В цикле осуществляем:

1) Если T.P > MaxP, выход из цикла, присваиваем NextProcessItem = T.
2) Если очередь закончилась, выход из цикла. Присваиваем NextProcessItem = последний элемент очереди.

3) Если T соответствует часто используемому слову текущей ГЧИС, и T.Processed = 0, то осуществляем обработку T, вызываем ProcessItem(T). Присваиваем T.Processed = 1.
4) Присваиваем T = следующий элемент очереди.

После выполнения цикла удаляем первый элемент из очереди. Если очередь пуста, присваиваем NextProcessItem = NULL.

Обработка элемента очереди

Функция ProcessItem(R), R – элемент очереди.

Вводим локальную переменную T, которая равна First.

Вводим локальную переменную MaxP = R.P + ProcessingDistance.

В цикле осуществляем:

1) Если T.P > MaxP выход из цикла.
2) Если R не равен T, вызываем функцию записи данных WriteNear(R, T).
3) Присваиваем T = следующий элемент очереди.

Запись данных

Функция WriteNear(R, T), R – элемент очереди, соответствующий ГЧИС, T – другой элемент очереди.

На входе функции записи данных мы имеем:

1) словопозиция R = (ID, P) часто используемой леммы w = R.Word,
2) y = номер w в ГЧИС,
3) словопозиция T другой леммы v = T.Word,
4) расстояние между словами в текстах, равное |R.P – T.P|,
5) информация о том, расположена ли словопозиция T ранее R в текстах или после R.

Если R.Word и T.Word обе являются часто используемыми леммами, то для каждой из них определяем номер в упорядоченном, по частоте вхождения в текстах, в порядке убывания, списке часто используемых лемм. Осуществляем запись, только если R.Word является более часто используемой леммой, чем лемма T.Word, то есть ее номер меньше.

Также, если T.Skip = 1, то запись не осуществляется.

Определяем ключ для таблицы дескрипторов. Если v – известная лемма, то дескрипторы хранятся в таблице, иначе в B-дереве.

Как описано в [5] мы должны сохранить в индексе запись R. При этом мы также должны сохранить y, по которому однозначно определяется w. В [5] рассмотрены два варианта.

Если мы храним информацию об y в части ключа, то в случае известной леммы ключ строиться на основании v и y. Нам потребуется таблица, число возможных дескрипторов в которой равно (число базовых лемм в словаре анализатора * число часто используемых лемм), т. е. около 260 000

* 2100 = 546 000 000 в рассматриваемых примерах. Заметим, что во время создания индекса нам требуется хранить в памяти только ту часть таблицы, которая соответствует обрабатываемым ГЧИС. При среднем размере ГЧИС в 50 элементов и запуске в 8 одновременно работающих потоков, мы имеем примерно в 5 раз меньше затрат по памяти. При необходимости, за счет уменьшения размера ГЧИС мы можем минимизировать эти затраты.

В случае неизвестной формы мы можем кодировать y как строку и присоединять к v для формирования ключа.

Альтернативный вариант, не кодировать y как часть ключа, а сохранять y в словопозиции, при ее помещении в индекс. Однако это увеличивает число записей, которые нужно будет прочитать при поиске, так как словопозиции для нескольких разных лемм будут считываться, а затем фильтроваться, одновременно. Поэтому это способ менее удобен. В экспериментах поиска в [6] и в данной работе применялся первый способ.

После формирования ключа словопозиция помещается в индекс. Логика записи описана в [7-10].

Устранение дубликатов при работе конвейера

Пусть ГЧИС содержит леммы А и Б, и где-то в текстах А и Б расположены близко, соответственно, словопозиции R и T, пусть А является более часто встречающейся леммой, чем Б.

Поскольку создание расширенных индексов предполагает цикл по O-временным файлам, и на каждой итерации цикла мы полностью перебираем все значения итератора ГЧИС, то в конвейер R и T попадают на каждой итерации цикла. Соответственно, требуется, чтобы запись в расширенный индекс (R.Word, T.Word) словопозиции R осуществлялась только один раз, а не на каждой итерации цикла.

Поэтому для записей итератора ГЧИС мы выставляем флаг Skip = 1, если их лемма Word не входит в состав текущей группы слов O-файла.

По построению алгоритма все леммы ГЧИС являются известными. Для любой леммы из любой ГЧИС существует ровно одна группа известных слов, и соответственно один KW-файл, в который входят словопозиции этой леммы.

Примечания по использованию памяти

Поскольку создание индекса запускается в многопоточном режиме и для каждой ГЧИС создаются отдельные файлы индекса и используется независимый кэш, для этого требуется достаточный объем оперативной памяти. Объем требуемой памяти составлен из объема памяти для хранения временных файлов и памяти для кэшей создания индексов. Число

потоков определяется исходя из параметров кэша, требуемого для создания одного индекса, и доступного объема оперативной памяти, в приведенных ниже тестах использовалось 8 потоков.

Эксперимент создания индекса

Было проиндексировано 195 тыс. файлов, объем текста 71,5 Гб., файлы представляли собой обычный текст, однобайтовая кодировка.

Объем памяти для временных файлов: 20 Гб.

Было произведено 2 итерации, то есть вначале сформированы временные файлы объемом 20 Гб., и записаны в индекс, затем были обработаны и записаны в индекс остальные данные. Параметры создания индекса приведены ранее в примерах.

Для экспериментов использовалось следующее окружение:
Intel Xeon X5650 2,67 GHZ (2 процессора), 48 Гб. RAM.
Windows Server 2008 R2 Enterprise, x64 bit.
Индекс создавался на одном жестком диске HGST HUS726060AL, 6 Tb, SATA. В следующей таблице показаны размеры и время создания индексов.

Индекс	**Размер**	**Время создания**
Индекс для поиска фраз из стоп слов. См. [6].	95,4 Гб.	3 час. 20 мин
Расширенный индекс, для поиска фраз включающих часто используемые слова. Алгоритм создания именно этого индекса рассмотрен в данной статье.	151 Гб.	4 час. 7 мин
Основной индекс	103 Гб.	1 час. 49 мин.
Всего (все индексы, метаданные, сжатые тексты документов)	398 Гб.	11 час. 19 мин.

Результаты

Рассмотрены алгоритмы создания расширенных индексов, которые позволяют ускорить поиск в полнотекстовом индексе. Рассмотрены стратегии выделения оперативной памяти для создания расширенных индексов. Представлены результаты эксперимента создания расширенных

индексов. Результаты экспериментов подтверждают возможность создания расширенных индексов за приемлемое время на обычном оборудовании.

Список литературы

1) Prywes N. S., Gray H. J; The organization of a Multilist-type associative memory, IEEE Trans. on Communication and Electronics, 1963, 68, 488-492.
2) Zobel J., Moffat A.; Inverted files for text search engines. ACM Computing Surveys, 2006, 38(2), Article 6.
3) Lester N., Moffat A., Zobel J.; Fast On-Line Index Construction by Geometric Partitioning, In CIKM ’05: Proceedings of the 14th ACM international conference on Information and knowledge management, 2005, 776-783.
4) Tomasic A., Garcia-Molina H., Shoens K.; Incremental updates of inverted lists for text document retrieval, In Proc. ACMSIGMOD Int. Conf. on the Management of Data, Minneapolis, Minnesota, 1994, 289-300.
5) Веретенников А.Б.; О поиске фраз и наборов слов в полнотекстовом индексе, Системы управления и информационные технологии, 2012, №2.1(48), 125-130.
6) Веретенников А.Б.; Использование дополнительных индексов для более быстрого полнотекстового поиска фраз, включающих часто встречающиеся слова, Системы управления и информационные технологии, 2013, №2(52), 61-66.
7) Веретенников А. Б.; Создание легко обновляемых текстовых индексов, Электронные библиотеки: перспективные методы и технологии, электронные коллекции: Труды Десятой Всероссийской научной конференции «RCDL'2008». Дубна: ОИЯИ, 2008, 149-154.
8) Веретенников А.Б.; Эффективная индексация текстовых документов с использованием CLB-деревьев, Системы управления и информационные технологии, 2009, №1.1(35), 134-139.
9) Веретенников А. Б.; Программный комплекс и эффективные методы организации и индексации больших массивов текстов. Диссертация на соискание ученой степени кандидата физико-математических наук, Екатеринбург, 2009.
10) Веретенников А.Б.; Полнотекстовый индекс для часто обновляющихся библиотек, Труды 13-й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2011. Воронеж: Изд.-полигр. центр ВГУ, 2011, 157-163.
11) Williams J. W. J.; Algorithm 232 - Heapsort, Communications of the ACM, 1964, 7(6), 347–348.