УДК 519.683.5

# Использование дополнительных индексов для более быстрого полнотекстового поиска фраз, включающих часто встречающиеся слова
# Using 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 sciences (IMCS).*

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

Рассматриваются задачи поиска фраз и наборов слов в большом объеме текстов с помощью дополнительных индексов. Показано, что применение дополнительных индексов может более чем в десять раз снизить максимальное время выполнения поискового запроса.

*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.

## Введение

Для решения задач поиска слов или фраз часто используются инвертированные файлы или их аналоги [1-6]. Инвертированный файл представляет собой набор записей вида (ID,P), ID – идентификатор документа, P – позиция, например порядковый номер, слова в документе. Все записи, соответствующие одному слову, хранятся последовательно для их быстрого чтения при поиске. Слова в документах встречаются с различной частотой. Такой важный параметр, как максимальное время выполнения поискового запроса, определяется наиболее часто встречаемыми словами. В связи с этим возникает задача ускорить поиск фраз, включающих часто используемые слова. Это можно сделать, если создать дополнительные индексы.

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. Рассматриваются задачи точного поиска фразы и поиска с учетом расстояния, т. е. мы ищем те 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.

Данная работа является продолжением [7], где все слова были разделены 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 назвать 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**».

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 200 тыс. для русского языка, для используемого словаря).

Если слово не входит в словарь анализатора, то, будем считать, что его базовая форма совпадает с самим словом.

Разделение слов на три группы при использовании анализатора применяется не к исходным словоформам, а к базовым формам слов. Т. е. есть три типа в смысле частоты встречаемости базовых форм слов: стоп базовые формы, часто используемые базовые формы и остальные.

Наличие нескольких базовых форм часто приводит к усложнению программы и в определенных случаях имеет смысл описывать алгоритм без углубления в соответствующие технические детали.

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

**Используемые обозначения**

* – умножение.
<< – побитовый сдвиг влево, применяется для целых чисел без знака.
| – побитовое или, применяется для целых чисел без знака.
$|X|$ – модуль числа $X$.
<= – меньше или равно, сравнение двух чисел.
!= – не равно.

*Record.Field* – обращение к полю *Field* записи *Record*. Запись – стандартный тип данных, переменная которого включает в себя несколько именованных полей.

**Структура индекса**

В данной работе автор при описании основных идей абстрагируется от возможной реализации индекса. Но, при описании технических деталей реализации автор опирается на описанную в [7] структуру индекса.

Будем называть для базовой формы слова потоком список записей о ее вхождениях в документах (ID,P), эти записи хранятся в индексе последовательно. Поток описывается небольшой структурой – дескриптором, в которой сохранена информация о месторасположении потока в файле индекса.

**Индекс для поиска фраз из стоп слов**

В данном случае мы рассматриваем запрос, в котором все слова являются стоп словами. Для такого запроса мы можем использовать дополнительные индексы для стоп слов, описанные в [7-12]. Такой индекс хранит в себе информацию о вхождении для любых стоп слов *w*, *v*, которые находятся рядом друг с другом. Однако, как показано в [7] количество обрабатываемых записей может достигать 3-30 миллионов при индексе, созданном для 30 Гб. текста. Проблема в том, что если фраза содержит *n* стоп слов (3, 4 и более), то требуется рассмотреть $O(n^2)$ потоков для пар слов.

Это приводит к тому, что требуется создать индекс, включающих информацию о вхождении для всех фраз, состоящих из стоп слов.

Введем параметры *MinLength* < *MaxLength*, и будем строить индексы для фраз длины *L*, где *MinLength* <= *L* <= *MaxLength*.

Например, если мы в тексте имеем 10 стоп слов, идущих подряд, что имеем 9 фраз из 2-х слов, 8 фраз из 3-х слов, 7 из 4-х и т.д.

Для создания индекса используется B-дерево [13], ключ в котором – список номеров стоп слов, значение – ссылка на инвертированный индекс, в котором сохранена информация о вхождении соответствующих фраз. Будем искать без учета порядка, поэтому список отсортирован по возрастанию.

Всего имеем *MaxLength* - *MinLength* + 1 индексов.

Заметим, что для стоп слов рассматривается только точный поиск, в найденном фрагменте текста между искомыми словами фразы не будет других слов. Это приемлемо, т. к.:

1) Вследствие того, что эти слова очень часто встречаются, практически для каждой их комбинации есть такое вхождение в текстах.
2) Существенная часть подобных запросов может быть составными терминами или часто используемыми фразами, в которых состав и порядок слов зафиксирован.
3) Подобный запрос может быть получен путем копирования фразы из уже существующего текста, для поиска других документов, включающих подобную фразу.

**Алгоритм создания индекса**

Опишем алгоритм в случае использования морфологического анализатора.

Предполагаем, что у нас есть стоп-список, состоящий из всех стоп базовых форм.

Создадим очередь *Queue*, элементами которой будут записи *QueueItem* из 3-х полей (ID, P, *Forms*, *Index*, *Next*), где *Forms* – список номеров базовых форм слов, *Index* – вспомогательная переменная, *Next* – следующий элемент очереди.

Когда читаем файлы, для каждого вхождения слова (ID,P), формируем список *Forms* базовых форм слова, которые входят в стоп-список.

Если *Forms* не пуст, то добавляем в конец *Queue* запись (ID,P,*Forms*). Далее если длина *Queue* больше *MaxLength*, удаляем первый элемент. Вызываем функцию *Process*(Начало очереди, 1).

Если *Forms* пуст, то пока Queue не пуста: вызываем функцию *Process*(Начало очереди, 1), затем удаляем первый элемент очереди.

Далее опишем функцию *Process*(*Item*, *L*), где *Item* это запись *QueueItem*, *L* – длина фрагмента.

Цикл по списку *Item.Forms*, текущий номер в списке сохраняем в *Item.Index*. На каждой итерации цикла выполняем следующие шаги:

1) *Process*(*Item.Next*, *L*+1),
2) Если *MinLength* <= *L* <= *MaxLength*,выполняем следующие шаги, иначе выходим из функции.
3) Формируем список *WordIDs* форм, в который включаем последовательно *Forms*[*Index*] идя от начала очереди и обрабатывая *L* элементов очереди (*Current* = начало очереди; *L* раз повторяем: добавляем *Current.Forms*[*Current.Index*] в *WordIDs*, *Current* = *Current.Next*).
4) Заменяем в *WordIDs* все номера базовых форм слова на соответствующий номер в стоп-списке.
5) Сортируем *WordIDs* по возрастанию, кодируем по алгоритму Хаффмана для уменьшения размера.

6) Сохраняем в индексе запись (ID,P) используя *WordIDs* как ключ.

**Оптимизация выполнения поисковых запросов с использованием расширенных индексов**

Как описано в [7] расширенный индекс (*w*,*v*) это список вхождений слова *w*, когда в тексте меньше чем на расстоянии *ProcessingDistance* от *w* присутствовало слово *v*. Параметр *ProcessingDistance* может быть задан различным в зависимости от частоты встречаемости слова *w* в текстах. Мы считаем, если расстояние между словами меньше *ProcessingDistance*, то слова связаны по смыслу друг с другом, иначе нет.

Следует отметить, что если для обоих слов *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.

**Расширение хранения информации о стоп словах в индексе**

Кроме индекса стоп слов и индекса для часто используемых слов также используем **основной индекс**.

В основном индексе храним все вхождения часто используемых слов и обычных слов.

В [7] описано, что мы можем в основном индексе для каждого слова, кроме его вхождения, хранить информацию о находящихся непосредственно рядом стоп словах. Этот подход можно расширить, сохраняя информацию о располагающихся на расстоянии не более *MaxDistance* от текущего слова стоп словах. Эксперименты показывают, что увеличение размера индекса при этом приемлемое.

К примеру, возьмем *MaxDistance = 5*.

В результате в основном индексе храним все вхождения часто используемых слов и обычных слов, для каждого вхождения храним информацию о располагающихся рядом стоп словах.

Если слово встречается часто, то для хранения информации о стоп словах может использоваться отдельный поток, т. в этом случае для хранения информации о вхождениях можем иметь до 3-х потоков:

1) Хранение ID документа + первое вхождение в этом документе + количество вхождений для этого документа.
2) Остальные вхождения.
3) Информация о стоп словах.

Это позволяет не считывать информацию о стоп словах, если в конкретном запросе она не нужна.

Ранее в [7] использовали потоки 1) и 2), это позволяет использовать один и тот же индекс для поиска с учетом расстояния и без, причем поиск без учета расстояния существенно быстрее, т. к. в этом случае используется только поток 1) и записей обрабатывается на порядок меньше. Для редко используемых слов все данные можно хранить в одном потоке, что снижает количество операций ввода вывода при создании индекса.

**Выполнение запросов**

Далее для различных видов запросов рассмотрим, как они могут быть выполнены. В примерах будем перечислять те потоки, содержащие списки вхождений слов, которые будут задействованы при поиске и каким образом.

**Все слова фразы являются стоп словами**

Используем индекс стоп слов, формируем ключ как отсортированный список ID стоп слов (форм) и по нему извлекаем все вхождения из индекса.

Пример 1: кроме того у них.
Пример 2: что будет если.

**Все слова фразы часто используемые**

В данном случае мы рассматриваем запрос, в котором нет стоп слов и все слова часто используемые.

Вначале рассмотрим случай, когда все слова запроса являются часто используемыми, т. е. для каждого из них имеется расширенный индекс.

Выберем слово $w_i$ запроса, которое в текстах встречается реже всего. Теперь для поиска фразы достаточно рассмотреть $n - 1$ расширенных индексов, а именно для каждого слова $w_k$, $k$ != $i$, индекс ($w_k$, $w_i$ ).

Пример: границами служат реки.
Слово «граница» встречается в текстах реже всего.

1) (река, граница) – поиск в расширенном индексе.
2) (служить, граница) – поиск в расширенном индексе.
3) граница, служить, река – основной индекс, первое вхождение в документе.

Здесь и далее, вначале мы ищем с учетом расстояния, в примере это пункты 1 и 2, далее, если ничего не нашли, ищем без учета расстояния, в

примере – пункт 3. Поиск без учета расстояния требует индекса, в котором для слов сохраняется только первое вхождение в документе. Как отмечено в [7] это на порядок снижает количество записей о вхождении.

**Не все слова фразы часто используемые, нет стоп слов**

Теперь рассмотрим случай, когда, по крайней мере, одно слово фразы не является ни стоп словом, ни часто используемым словом. Пусть в запросе $m$ таких слов. Выберем из них слово $w_i$, которое в текстах встречается реже всего. Назовем его «**основное слово запроса**». Теперь для поиска фразы достаточно рассмотреть $n - m$ расширенных индексов, а именно для каждого слова $w_k$, $k$ != $i$, для $w_k$ существует расширенный индекс, рассмотрим индекс ($w_k$, $w_i$ ). Также для слов, для которых не существует расширенного индекса, используем обычный индекс.

При извлечении записей из основного индекса информация о располагающихся рядом стоп словах при возможности, т. е. если хранится в отдельном потоке, пропускается.

Пример: двести сорок миль

Здесь слово «двести» не входит в список часто используемых слов, словоформа «сорок» имеет две базовые формы. Основное слово запроса «миль».

1) (двести, миля) – поиск в расширенном индексе.
2) (сорока, миля), (сорок, миля) – поиск в расширенном индексе.
3) двести, миля, сорока, сорок – основной индекс, первое вхождение в документе.

**Во фразе есть все виды слов.**

Действуем как в предыдущем случае. Но для выбранного основного слова запроса $w_i$, мы будем считывать все его вхождения, и обрабатывать информацию о находящихся рядом стоп словах, чтобы учесть входящие во фразу стоп слова. В данном случае основным словом запроса может также быть и часто используемое слово и в этом случае мы считываем из основного индекса все его вхождения.

Пример: записки о Галльской войне

стоп слова в данном запросе «о» и «война».

1) галльский – все вхождения с анализом информации о стоп словах.
2) (галльский, записка) – поиск в расширенном индексе.

Здесь возникает вопрос, следует ли искать без учета расстояния, т. к. будет ли иметь смысл такой поиск для стоп слов. При необходимости можно осуществить и этот поиск.

В следующем примере используются часто используемые слова и стоп слова.

Пример: приготовить все необходимое для похода.
стоп слова в данном запросе «все», «необходимое», «для».
1) поход – все вхождения с анализом информации о стоп словах.
2) (поход, приготовить)

**Выполнение запроса**

Для каждого слова запроса морфологических анализатор дает нам несколько базовых форм слов. Таким образом, для запроса из *n* слов имеем *n* списков базовых форм. В случае если в одном из списков присутствуют разные в смысле частоты встречаемости базовые формы, т. е. к примеру одна форма – стоп форма, другая форма – часто встречающаяся, то требуется поделить запрос на 2 части и выполнить их по отдельности. Иначе алгоритм поиска существенно усложняется.

Более подробно: имеем список *Query*, каждый элемент которого является списком базовых форм слов. Рассмотрим каждый элемент *Quer*y, пусть, *i* – номер текущего рассматриваемого элемента. Если в нем присутствуют *m* разных типов в смысле частоты встречаемости базовых форм, то создаем *m* копий поискового запроса, в каждой из которых *i*-й элемент содержит только один вид базовых форм. Далее для каждого нового запроса действуем аналогично, обрабатывая (*i*+1)-й и последующие элементы. Затем выполняем полученные запросы и объединяем их результаты.

**Эксперименты, окружение и исходные данные**

Все эксперименты были проведены, используя коллекцию текстов размером 45 Гб. Данные документы представляли собой обычный текст. По стилю – в основном художественная литература, журналы. Всего около 130 тыс. документов.

Для экспериментов поиска использовалось следующее окружение:
CPU: Intel(R) Core(TM) i7 CPU 920 @ 2.67GHz.
HDD: Seagate Barracuda 7200.11, 7200 RPM, кэш 32 Мб., 2 Гб, ST32000641AS.
Оперативная память: 24 Гб.

OS: Microsoft Windows 2008 R2 Enterprise.

Параметры тестового индекса: *MinLength* = 2, *MaxLength* = 5, *MaxDistance* = от 5 до 7 в зависимости от частоты встречаемости *слова*.

Стоп слова: самые часто встречающиеся 700 слов. Обычно стараются делать список стоп слов как можно короче, если их не включают в индекс. Поскольку мы их включаем в индекс, мы можем сделать этот список достаточно большим.

Часто используемые слова: 2100.

Данные параметры позволяют искать фразы длиной до 5 слов. При поиске более длинных фраз, включающих стоп слова или часто используемые слова, фразу можно разделить на части и искать каждую часть по отдельности, потом объединить результаты.

**Структура экспериментов поиска**

Проведенные эксперименты поиска заключались в следующем:

1) Выберем некоторый документ в индексе. Документ выбирается автором случайным образом.
2) Фразы для поиска выбираем следующим образом:
   a) Выберем набор подряд располагающихся слов.
   b) Выберем набор подряд располагающихся слов, пропуская каждое второе слово.
3) Произведем поиск каждого выбранного набора слов. При поиске читаем в индексе все записи, которые соответствуют данному слову (т. е. если даже нашли искомый набор слов, все равно читаем все до конца).

Выбирались наборы слов, состоящие из 3, 4, 5 слов.

Если одно из слов запроса имело стоп базовую форму, то осуществлялся поиск только подряд располагающихся слов.

Основной параметр, который измеряем, это количество прочитанных записей о вхождении слов при выполнении одного поискового запроса. Скорость поиска в основном зависит от этого. Выполняя большое количество запросов, измеряем среднее и максимальное значение этого параметра. Также изменяем время выполнения запроса.

Преимущество подобного подхода:

1) Проверяем, что индекс действительно построен корректно и ищет то что нужно. Т. к. выбираем фразы из уже проиндексированного документа, то мы их точно должны найти. Проверяем, что в результатах поиска присутствует запись, соответствующая тому документу, на основании которого мы выбрали запрос.

2) Искомые фразы достаточно разнообразны и включают в себя большое количество различных слов, значительная часть фраз включает в себя стоп слова и часто используемые слова.

Все запросы выполнялись последовательно в одном потоке программы, т. е. использовалось одно ядро процессора.

**Размер индекса**

| Индекс | Размер |
|---|---|
| Индекс для поиска фраз из стоп слов | 80 Гб. |
| Расширенный индекс, для поиска фраз включающих часто используемые слова | 79 Гб. |
| Основной индекс | 67 Гб. |
| Всего (все индексы, метаданные, сжатые тексты документов) | 259 Гб. |

**Скорость поиска**

Всего было выполнено 45 тыс. запросов, на что потребовалось 1 ч. 38 мин. В запросах встречались все возможные виды слов. Далее показаны результаты.

| Показатель | Среднее значение | Максимальное значение |
|---|---|---|
| Время выполнения запроса | 0.13 с. | 1.31 с. |
| Количество обработанных записей | 274 тыс. | 6 млн. |

Для этого же набора документов был построен индекс программой Sphinx 2.0.6 [14]. Объем индекса 18.7 Гб. Были выполнены те же запросы, что и ранее, далее результаты:

| Показатель | Среднее значение | Максимальное значение |
|---|---|---|
| Время выполнения запроса | 1.01 с. | 17.82 с. |
| Количество обработанных записей | 112 млн. | 505 млн. |

## Результаты

Рассмотрены разработанные автором методы организации дополнительных индексов для стоп слов и часто используемых слов, целью которых является ускорение поиска.

Проведены эксперименты поиска для различных фраз, включающих стоп слова или часто используемые слова. Максимальное время поиска при использовании дополнительных индексов более чем в десять раз меньше, чем при использовании обычных индексов.

Проведены эксперименты по созданию индекса, включающего дополнительные индексы часто используемых слов, при индексации 45 Гб. текстов размер дополнительных индексов при оптимальных параметрах занимает около 250 Гб. Увеличение объема индекса может быть приемлемо, если при этом будет существенно увеличена скорость поиска.

## 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, Justin, and Alistair Moffat. Inverted files for text search engines. ACM Computing Surveys 38(2), 2006, Article 6.
3. Heinz, Steffen, and Justin Zobel. Efficient single-pass index construction for text databases. JASIST 54(8), 2003, pp. 713-729.
4. Tomasic A., Garcia-Molina H., and Shoens K. Incremental updates of inverted lists for text document retrieval. In Proc. ACMSIGMOD Int. Conf. on the Management of Data, Minneapolis, Minnesota, 1994, pp. 289-300.
5. Trotman, Andrew. 2003. Compressing inverted files. IR 6(1):5–19. DOI: dx.doi.org/10.1023/A:1022949613039. 106, 532.
6. Falk Scholer, Hugh E. Williams, John Yiannis, Justin Zobel. Compression of Inverted Indexes For Fast Query Evaluation, School of Computer Science and Information Technology RMIT University, GPO Box 2476V, Melbourne, 3001, Australia, 2001.
7. Веретенников А.Б. О поиске фраз и наборов слов в полнотекстовом индексе// Системы управления и информационные технологии, №2.1(48), 2012. – С. 125-130.
8. 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. с. 149-154.

9. Веретенников А. Б. Эффективная индексация текстовых документов с использованием CLB-деревьев. Системы управления и информационные технологии, 2009, 1.1(35). - с. 134-139.

10. 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.

11. Веретенников А.Б. Об оптимизации поиска фраз и наборов слов в полнотекстовом индексе. Современные проблемы математики. Тезисы Международной (43-й Всероссийской) молодежной школы-конференции. Екатеринбург 2012. с. 203-205.

12. 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. с. 157-163.

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

14. Sphinx. http://sphinxsearch.com.