УДК 519.683.5

О поиске фраз и наборов слов в полнотекстовом индексе
About phrases search in full-text index

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.

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.

Введение

Для решения задач поиска слов или фраз часто используются инвертированные файлы или их аналоги [1-4]. 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.

Разделим все слова 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.
3) Остальные.

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.

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.

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 далее структуру индекса.

Будем называть для базовой формы слова потоком все записи 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.

Для слов, входящих в словарь анализатора дескрипторы потоков хранятся в таблице в оперативной памяти. Для слов, не входящих в словарь анализатора, используется B-дерево [5]. Ключами в B-дереве являются базовые формы слов, а значениями – дескрипторы потоков.

Таким образом, мы имеем два подвида индекса.

В первом из них используется таблица из заданного количества пронумерованных дескрипторов, назовем его *индексом первого типа*.

Во втором используется B-дерево, назовем его *индексом второго типа*.

Определим, что такое *поток, соответствующий x*. В случае если это поток в индексе первого типа, то это поток, дескриптор которого имеет номер *x*. Если же это поток в индексе второго типа, то это поток, дескриптор которого сохранен в B-дереве с ключом *x*.

Обработка стоп слов

При индексации нескольких десятков Гб. текстов, количество вхождений для такого слова может достигать 50-100 миллионов. При обработке стоп слов возможны следующие варианты:

1) Не включаем в индекс.
2) Используем методы сжатия списка вхождений для данного слова (эффективные, т. к. эти слова встречаются часто). См, например, [6,7].
3) Используем дополнительные индексы.

Создание дополнительных индексов для стоп слов

В [8-11] автором данной работы кратко описан вариант решения данной задачи. Данный подход может применяться для поиска фраз, включающих стоп слова, т. е. рассматривается точный поиск, в найденном фрагменте текста между искомыми словами фразы не будет других слов.

Для каждого вхождения слова *x* в документе (ID,P) будем сохранять еще информацию о находящихся рядом стоп словах. Нам требуется сохранить массив номеров стоп слов, т. е. массив чисел. Кроме этого требуется сохранить информацию о том, до или после *x* находилось конкретное стоп слово. Эта информация занимает 1 бит.

В случае использования морфологического анализатора, все стоп слова находятся в словаре анализатора. В этом случае мы работаем не со списком стоп слов, а со списком базовых форм стоп слов, существенной разницы для алгоритма здесь нет.

Отсортируем список стоп слов в порядке снижения частоты нахождения в текстах, и присвоим каждому стоп слову номер (начиная с 0). Часто с данными номерами удобнее работать, чем с идентификаторами базовой формы слова в словаре морфологического анализатора. Самые часто используемые стоп слова имеют самый малый номер. Всего стоп слов обычно берется примерно 100-200.

Немного меняется схема кодирования записи (ID,P) в индексе, чтобы сохранить вместе с (ID,P) еще 1 бит – в котором мы сохраняем информацию о том, были ли вообще стоп слова рядом. Это легко сделать независимо от способа кодирования (ID,P), достаточно P сдвинуть влево на 1 бит, и сохранить в нулевом бите нужную информацию.

Далее при кодировании каждого числа информации о стоп словах также у нас 1 бит хранит в себе информацию о том последнее это число или нет.

Таким образом, имеем массив номеров стоп слов $[y(1),...,y(n)]$, где $y(i)$ = ((номер стоп слова) << 1) | (до или после x было соответствующее стоп слово)). Возможная схема их кодирования описана далее.

Кроме этого имеем индекс первого типа, содержащий $C*C$ потоков, для сохранения информации о парах стоп слов, располагающихся рядом.

Пусть есть стоп слова с номерами $w1$ и $w2$. Вычисляем ($w1$ << 16) | $w2$ и получаем требуемый номер потока. Т. е. в потоке ($w1$ << 16) | $w2$ хранятся записи о вхождениях (ID,P), которые соответствуют вхождениям в документах стоп слова с номером $w1$, когда следом за ним было стоп слово с номером $w2$.

Если мы ищем фразу, содержащую стоп слово w, то вместо потока, содержащего информацию обо всех вхождениях данного стоп слова в текстах, будем использовать:

1) Для каждого слова фразы v, не являющегося стоп словом, при чтении с диска соответствующих ему записей о вхождении в текстах, формируем список записей о вхождении в документах для слова w, когда оно было рядом с v.

2) Для каждого слова фразы u, являющегося стоп словом, используем дополнительные индексы стоп слов, и получаем список записей о вхождении в документах для слова w, когда оно было рядом с u.

Кодирование информации о располагающихся рядом с данным словом стоп словах.

Целью предложенного способа кодирования является минимизация объема закодированной информации и минимальное использование битовых операций. Можно использовать, например, кодирование по алгоритму Хаффмана, но это потребует большего количества операций, чем предлагаемый алгоритм.

Мы имеем массив чисел $[y(1),...,y(n)]$.

При их кодировании вначале сортируем массив по возрастанию, далее считаем, что массив отсортирован, затем переходим к массиву

$WS = [z(1) = y(1), z(2) = y(2) – y(1), \ldots, z(n) = y(n) – y(n – 1)]$.

При кодировании нового массива будем хранить число в одном или нескольких байт. Рассмотрим два варианта кодирования элемента $z = z(i)$ из WS.

Вариант 1. Пусть всего стоп слов <= 127.

Если $z < 127$, то кодируем z в виде одного байта со значением z.

Иначе, если $z >= 127$, то кодируем его в виде двух байт, первый из них 255, второй $z – 127$.

В последнем байте закодированного значения z старший бит содержит информацию о том, является ли z последним элементом *WS*.

Таким образом, если $z = 126$ и не является последним элементом в *WS*, то оно будет закодировано как 254 (т. е. старший бит установили в 1). Т. е. мы резервируем возможное значение первого байта 255 как признак того, что z закодировано в виде двух байт.

Максимальное значение $z = ((126 << 1) | 1) = 253$ кодируется в виде байт 255 и 126. Если оно располагается после x, то в виде 255, 254.

Вариант 2. Пусть всего стоп слов больше 127. В этом случае нам потребуется до 3 байт для кодирования z.

Если $z < 126$, то кодируем z в виде одного байта со значением z.

Иначе, если $z <= 126 + 127$, то кодируем его в виде двух байт, 254 и $z – 126$ (Максимальное значение последнего байта должно быть <= 127, т. к. старший бит будет использован).

Иначе в виде 3-х байтов, в первом из них 255, в остальных двух хранится $z – (126 + 127)$. Используется процессоры Intel с Little-endian кодированием чисел.

В последнем байте закодированного значения z старший бит содержит информацию о том, является ли z последним элементом *WS*.

Таким образом, максимальное значение z будет равным $((1 << 15) – 1) + 126 + 127$.

Здесь мы резервируем два значения первого байта 255 и 254 как признаки того, сколько байт занимает z в закодированном виде.

Для большего значения количества стоп слов можно продолжить по аналогии, если большее количество имеет смысл.

Самые часто используемые стоп слова имеют самые малые номера и информация о них занимает 1 байт.

Создание дополнительных индексов для часто используемых слов

Эти слова нельзя исключать из индекса, т. к. они имеют смысл, но их достаточно много. Если отсортировать все базовые формы слов в порядке убывания частоты их появления в текстах, то скажем 100-200 самых часто встречаемых из них – соответствуют стоп словам, далее 500-5000 – словам, которые автор поместил в группу часто используемых слов. Для 100 Гб. текстов, такое слово может встретиться до 1-5 млн. раз.

В [12] автором данной работы предложено создавать дополнительные индексы.

Можно предположить, что если пользователем введена фраза, то слова в ней связаны по смыслу и следует искать документ, где они располагаются рядом. При этом, чем слова в документе ближе друг к другу, тем данный документ более релевантный по отношению к поисковому запросу.

Введем параметр *ProcessingDistance* и будем считать, что если расстояние между словами меньше *ProcessingDistance*, то слова связаны друг с другом, иначе нет.

Сведем задачу поиска к следующим двум задачам:

1) Ищем с учетом расстояния документы, в которых слова располагаются близко друг к другу.

2) Ищем без учета расстояния (данный поиск требует хранения в индексе только первого вхождения в документе искомого слова).

Можно также отметить, что современные поисковые системы выводят фрагмент текста, включающего найденные слова, его длина не более 20-30 слов.

Разделим список часто используемых слов на наборы. Дополнительный индекс строится для набора слов *G*. Для каждого вхождения (ID,P) каждого слова *x* индексируемых текстов, такого, что есть вхождение (ID,P2) слова *w* с номером *y* из *G*, |P – P2| < *ProcessingDistance*, сохраняем (ID,P2,*y*) в индекс.

Далее подробнее с технической точки зрения.

Для набора *G* мы имеем два дополнительных индекса, один первого типа, другой второго.

Запись *R* = (ID,P2,*y*) сохраняется в потоки, соответствующие слову *x*.

Если *x* входит в словарь морфологического анализатора, то *R* сохраняется в первом индексе, в потоках, соответствующих номерам базовых форм *x*.

Иначе, если *x* не входит в словарь морфологического анализатора, во втором индексе, в потоке, соответствующем *x*.

В дополнительном индексе мы для любого слова *x* и часто используемого слова *w* можем быстро найти записи о вхождениях слова *w*, когда оно было рядом с *x*.

При поиске, если в искомом наборе слов есть часто используемое слово *w*, то вместо того, чтобы извлечь список всех его вхождений в текстах, заменяем его на списки вхождений слова *w*, когда оно было рядом с другими словами поискового запроса.

Разделять часто используемые слова на наборы в данном случае имеет смысл исходя из суммарной частоты вхождения всех слов набора в текстах. Эту суммарную частоту в данном случае можно ограничить некоторым параметром. Т. к. часто используемые слова отсортированы в порядке

убывания частоты появления слова в текстах, то в первых наборах будет небольшое количество слов, например, 3-5, а в последних до 10-20.

Имеет смысл также ограничить количество элементов каждого набора, чтобы для хранения *y* в записи (ID,P2,*y*) требовалось не более одного байта.

Недостатки и пути их решения

1) Большой размер индекса (примерно в 3-7 раз больше размера индексируемых текстов).

2) Требуется осуществить больше обращений к диску при поиске (т. е. читаем суммарно меньше, но самих операций больше). Последнее не имеет значения при использования SSD). Для обычных HDD проблема становится менее существенной с ростом размера индекса.

Второй вариант дополнительных индексов для часто используемых слов

Предыдущий вариант обладает определенным недостатком, в частности нельзя разделить список часто используемых слов на наборы достаточно большого размера. А также то, что в дополнительных индексах в одном потоке для произвольного слова *x* хранится информация сразу о нескольких часто встречающихся словах. С одной стороны это уменьшает число потоков, что при определенной организации индекса снижает число дисковых операций, требуемых для построения индекса, с другой стороны снижает скорость поиска. Далее предлагается второй вариант.

Введем параметр *GroupSize* (Примерно 100-200). Разделим все часто используемые слова на наборы размером *GroupSize*.

Для каждого вхождения (ID,P) каждого слова *x* индексируемых текстов, такого что есть вхождение (ID,P2) слова *w* с номером *y* из *G*, |P – P2| < *ProcessingDistance*, сохраняем (ID,P2) в индекс.

Далее подробнее с технической точки зрения.

Для набора *G* мы имеем два дополнительных индекса. Один первого типа, другой второго. Запись *R*=(ID,P2) сохраняется в одном из них.

Для слов, входящих в словарь морфологического анализатора, *R* сохраняется в индексе первого типа, в потоки соответствующие $N(x)(i) + WordsCount * y$, где $N(x)(i)$ – номер *i*-й базовой формы *x*.

Для слов, не входящих в словарь анализатора, *R* сохраняется в индексе второго типа в потоке *ToString*(*y*)+*x*, где *ToString(y)* – строка содержащая число *y*, *x* – само слово, «+» – операция конкатенации строк. Т. е. мы добавляем к исходной базовой форме *x* префикс, в котором кодируем номер *w* в *G*.

В данном случае мы не храним в самих потоках слов номера часто используемых слов в их наборе *G*, вместо этого мы увеличиваем число потоков слов.

Недостатки:

1) Увеличение используемой оперативной памяти при поиске, (до нескольких Гб., в проведенных экспериментах для индекса, построенного для 30 Гб. текстов). В связи с этим 32-я версия реализация алгоритма в этом случае уже не использовалась. Основные затраты на оперативную память требуются для хранения в памяти таблиц дескрипторов потоков первого типа. При использовании SSD данные таблицы можно не кешировать в оперативной памяти.

2) Увеличение количества дисковых операций при создании дополнительных индексов.

Преимущества:

1) При поиске мы читаем только то, что нужно.

2) Можем часто используемые слова разделять на группы большего размера, за счет чего ускоряется создание индекса.

3) Уменьшение суммарного размера индекса, т. к. в потоках не храним номер часто используемого слова в его наборе.

Оптимальное значение *ProcessingDistance.*

Важно выбрать такое значение *ProcessingDistance*, чтобы размер индекса и скорость поиска были приемлемы и результаты поиска были релевантными.

Чем меньше *ProcessingDistance*, тем результаты поиска могут быть менее релевантными.

Также имеет смысл использовать различное значение *ProcessingDistance* при обработке разных часто используемых слов. Например, для первых 50 часто используемых слов, частота встречаемости которых самая большая, определим *ProcessingDistance* = 3, для следующих 200 слов, *ProcessingDistance* = 5, для остальных 10.

Это имеет смысл, т. к. чем чаще слово встречается, тем оно несет в себе меньше смысла, а используется скорее как связь между другими словами текста.

Эксперименты, окружение и исходные данные

Эксперименты осуществлялись используя 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 Enterprise x64 Edition Service Pack 2.

Все эксперименты были проведены, используя коллекцию текстов размером 30 Гб. Данные документы представляли собой обычный текст. По стилю – в основном художественная литература, журналы. Всего около 220 тыс. документов.

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.

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.

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

Дополнительные индексы стоп слов

Были созданы три индекса с разными настройками:

1) Проиндексированы все словоформы.
2) Созданы дополнительные индексы для стоп слов, стоп слова не индексированы.

3) Стоп слова не индексированы, дополнительные индексы не созданы.

Далее в таблице показаны размеры полученных индексов для каждого случая:

	1	2	3
Размер индекса	17 Гб.	24,5 Гб. (из них 5,5 Гб. – индексы для пар стоп слов, располагающихся рядом)	12 Гб.

Это показывает определенное увеличение размера индекса при создании дополнительных индексов стоп слов. В частности для 2-го индекса, имеем, 5,5Гб. – размер индексов для пар стоп слов, размер остальных индексов 19 Гб. Сравнивая с 12 Гб., для третьего индекса, получаем, что информация о располагающихся рядом стоп словах для всех остальных слов, занимает 7 Гб.

Также были проведены эксперименты поиска для сравнения первого и второго вариантов, далее в таблице, которые показывают существенное снижение объема (до 10-20 раз) чтения данных с диска при поиске. Большие значения в таблице вызваны тем, что значительное количество искомых фраз включало в себя стоп слова.

	1	2
Среднее количество обработанных записей о вхождении на 1 запрос	66 млн.	3 млн.
Максимальное количество обработанных записей о вхождении на 1 запрос	328 млн.	32 млн.

Дополнительные индексы часто используемых слов

При создании индекса использовались морфологические словари для английского и русского языков. Параметр *ProcessingDistance* был взят такой, как описано в разделе «Оптимальное значение *ProcessingDistance*», т. е. максимальное значение 10.

Были созданы следующие индексы.

1) Создан индекс без дополнительных индексов.

2) Первый вариант дополнительных индексов для часто используемых слов. Размер списка часто используемых базовых форм слов: 700. Размер наборов, на которые поделены часто используемые слова: примерно 10-20.

3) Второй вариант дополнительных индексов для часто используемых слов. Размер списка часто используемых базовых форм слов: 700. Размер наборов, на которые поделены часто используемые слова: 250.

4) Второй вариант дополнительных индексов для часто используемых слов. Размер списка часто используемых базовых форм слов: 5000. Размер наборов, на которые поделены часто используемые слова: 250.

Далее в таблице показано, сколько дискового пространства потребовалось для хранения дополнительных индексов. Также показано время создания всех индексов (обычных и дополнительных).

	2	3	4
Размер дополнительных индексов	117 Гб.	45 Гб.	104 Гб.
Время создания индекса	11 часов.	8 часов.	12 часов.

Таким образом, первый вариант дополнительных индексов существенно менее эффективен, его преимущество только в меньшем количестве дисковых операций при создании индекса и требованиям к оперативной памяти.

Далее результаты экспериментов поиска.

Эксперименты поиска производились без учета стоп слов, т. е. если в наборе слов (которые мы получаем, как описано в разделе «Структура экспериментов поиска») включается стоп слово, данный набор слов просто не ищем.

Эксперименты поиска проводятся вначале для обычного индекса, затем для индекса, включающего дополнительные индексы часто встречающихся слов. Рассмотрим только второй вариант дополнительных индексов.

	1	3	4
Среднее количество обработанных записей о вхождении на 1 запрос	2,3 млн.	1,1 млн.	640 тыс.
Максимальное количество обработанных записей о вхождении на 1 запрос	10 млн.	4,8 млн.	2,4 млн.

Если рассмотреть последний вариант индекса, то данные эксперименты показывают, что число обрабатываемых записей о вхождениях при выполнении поисковых запросов при использовании дополнительных индексов снижается в 4 раза.

Результаты

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

Проведены эксперименты поиска для фраз, включающих стоп слова: количество записей о вхождениях, извлекаемых при поиске, снижается до 10 раз.

Проведены эксперименты поиска для фраз, включающих часто используемые слова: количество записей о вхождениях, извлекаемых при поиске, снижается в 4 раза.

Проведены эксперименты по созданию индекса, включающего дополнительные индексы часто используемых слов, при индексации 30 Гб. текстов размер дополнительных индексов при оптимальных параметрах занимает около 100 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. Bayer R., McCreight E. Organization and maintenance of large ordered indexes. Acta Informatica, 1972, 1, 3, 173-189.
6. Trotman, Andrew. 2003. Compressing inverted files. IR 6(1):5–19. DOI: dx.doi.org/10.1023/A:1022949613039. 106, 532.
7. 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.
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. О методе оптимизации создания CLB-дерева. Проблемы теоретической и прикладной математики: Тезисы 41-й Всероссийской молодежной конференции. Екатеринбург: УрО РАН, 2010. с. 429-435.

11. 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.

12. 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.