УДК 519.683.5

# Эффективный полнотекстовый поиск с использованием дополнительных индексов часто встречающихся слов
# Efficient full-text search by means of additional indexes of 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. Применяются дополнительные индексы, за счет которых время выполнения поискового запроса может быть снижено более чем в десять раз. Приведены алгоритмы поиска с использованием дополнительных индексов и результаты экспериментов.

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

Searches for phrases and word sets in large text arrays by means of additional indexes are considered. Their use may reduce the query processing time by an order of magnitude in comparison with standard indexes. Several search strategies described. Results of experiments are given.

## Введение

Задачи поиска слов или фраз в текстах, когда результат поискового запроса – это информация о документах и местах в них, где встречается искомый поисковый запрос, решаются с помощью инвертированных файлов [1, 2] и их аналогов [3].

Поисковый запрос – это набор слов. Рассматриваем задачу поиска с учетом расстояния, в документах искомые слова должны располагаться как можно ближе друг к другу. Эта задача требует сохранения в индексе информации о каждом вхождении в документе каждого слова.

Слова в документах встречаются с разной частотой. Считается, что распределение частот слов соответствует закону Ципфа [4]. Слова, которые часто используются, могут встречаться в тысячи и более раз чаще, чем редко используемые слова.

Максимальное время выполнения поискового запроса, определяется наиболее часто встречаемыми словами. С помощью создания дополнительных индексов для часто встречаемых слов объем обрабатываемых данных при выполнении поискового запроса можно существенно снизить. В [5, 6] были рассмотрены несколько стратегий использования дополнительных индексов. В [7] даны алгоритмы создания изучаемых дополнительных индексов. В данной работе предлагается несколько алгоритмов поиска и результаты углубленного исследования производительности поиска.

**Виды слов**

Разделим слова на группы, на основании их частоты встречаемости. Для разных групп слов определим разные методы их обработки. В [5] мы определили три группы слов:

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».

Далее учитываем все виды слов при поиске, то есть, для любого слова информация в каком-то виде включается в индекс. В поисковом запросе учитываются все 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.

**Используемые обозначения**

* – умножение.

(F1, F2, F3) – запись, состоящая из полей F1, F2, F3.

L = [(F1,F2,F3)] – список L состоит из записей вида (F1,F2,F3).

**Структура инвертированного файла**

Инвертированный файл это ассоциативный массив, ключ в котором – словоформа или ее лемма, а значение – это список записей вида (ID, P), где ID – это идентификатор документа, а P – позиция слова в документе. В качестве позиции слова P мы используем 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.

Число обрабатываемых словопозиций при выполнении поискового запроса можно существенно снизить с использованием дополнительных индексов, соответственно, запросы будут выполняться быстрее.

Словопозиция также может включать в себя дополнительные поля или данные. Например, будем использовать информацию о располагающихся близко с текущим словом стоп словах [6], которую мы будем называть далее NSW-записью (от Near Stop Words). Данная запись для словопозиции T включает в себя информацию о стоп леммах слов, которые находятся на расстоянии от T не более чем MaxDistance (параметр, например, 5).

Будем считать, что словопозиция A меньше словопозиции B, если A.ID < B.ID или (A.ID = B.ID и A.P < B.P). Словопозиции одной леммы хранятся в индексе упорядоченные по возрастанию.

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).

**Поиск фраз в текстах, существующие методы**

В [8, 9] приводятся алгоритмы создания дополнительных индексов для более быстрого поиска фраз. Авторы предлагают две оптимизации:

1) Вводится частичный индекс фраз, в котором ключами являются часто используемые фразы (набор которых определяется часто используемыми запросами пользователей). Этот индекс, по сути, является кешем часто используемых запросов.

2) Вводится индекс пар слов (nextword – индекс). Для пары слов хранится информация об их вхождениях в текстах. При этом фиксируется порядок слов в паре и слова находятся непосредственно рядом друг с другом.

В [9] дается обоснование необходимости учета стоп слов при поиске. Простое устранение стоп слов из индекса и поиска может привести к непредсказуемым результатам, если стоп слово имеет для конкретного случая особый смысл (например, см. пример далее). При этом на поиск накладываются условия:

1) Учет порядка слов фразы.
2) Отсутствие «лишних» слов в составе фразы в искомых текстах.

То есть ищутся только те тексты, в которых фраза встречается точно так, как она введена пользователем. Например, если в тексте есть фраза «Time and a world by Yes», то при поиске: «Time and a world Yes», «Yes time and a world», данный текст не будет найден методами [8, 9], по причине наличия «by» в тексте (для обоих запросов) и несовпадении порядка слов (для второго запроса). Примечание. Yes – название группы, «Time and a world» – название произведения.

В [10, 11] дано развитие идей nextword – индекса. Идет переход от пар слов к последовательностям из нескольких слов. Однако указанные недостатки [8, 9] остаются. Таким образом, методы из [8-11] ограничены в применении.

**Отличие предлагаемых методов от других методов поиска фраз.**

Существенное отличие настоящей работы в том, что мы ищем без учета порядка слов, и поиск осуществляется с учетом того, что в текстах посередине фразы может еще что-то быть. Поэтому мы и говорим о поиске «наборов слов» и фраз. Потенциальная область применения рассматриваемых методов существенно шире, чем у [8-11].

На поисковый запрос не накладывается никаких дополнительных условий. Поиск с использованием дополнительных индексов по своим параметрам идентичен поиску с использованием обычных индексов, но выполняется существенно быстрее. За исключением двух ограничений. Во-первых, слова фразы в искомых текстах должны быть все-таки вблизи, то есть число «лишних» слов между ними должно быть ограничено (допустимое количество настраивается параметром). Во вторых, для запросов, которые состоят только из стоп лемм, не допускаем в тексте лишних слов во фразе, но ищем без учета порядка.

**Используемые виды индекса**

Перечислим используемые индексы и их параметры. Для параметра приводим значения, используемые в приведенных в конце статьи экспериментах.

WS_COUNT=700. Определяет число стоп лемм. Стоп леммы – это первые 700 лемм, в списке FL всех известных лемм, отсортированных по частоте встречаемости в текстах по убыванию. Список всех известных лемм

определен морфологическим анализатором, включающим леммы русского и английского языков.
AI_WORDS=2100. Определяет число часто используемых лемм. Следующие 2100 лемм в списке FL, после стоп лемм.
**Обычный индекс с NSW-записями**. Для каждой леммы, за исключением стоп лемм, хранит список всех ее словопозиций. Словопозиция содержит NSW-запись. MaxDistance = 5. NSW-запись для словопозиции T включает информацию о стоп леммах слов, располагающихся на расстоянии не более MaxDistance от T в документе. Для каждой стоп леммы S сохраняется ее номер в FL списке, информация о том, до или после T она находится и расстояние от S до T. Данная информация эффективно кодируется, с учетом того, что кодируемые числа небольшие (расстояния не более MaxDistance, номера стоп лемм в FL также малы).

Рассмотрим алгоритм кодирования расстояний в NSW-записи. Пусть мы имеем последовательность из MaxDistance*2 + 1 слов. Текущее слово – в середине. Каждое слово имеет список лемм. В этом списке лемм есть подсписок стоп лемм. Вместо сохранения расстояния для каждой стоп леммы можно организовать битовый поток и сделать следующее:

Обрабатываем MaxDistance слов. Для каждого слова:

1) Для каждой стоп леммы пишем в битовый поток бит 0.
2) Пишем в битовый поток бит 1.

Этот алгоритм применяем вначале для тех MaxDistance слов, которые находятся до текущего слова, а затем для тех, которые после него. В результате при MaxDistance = 5 информация о расстояниях будет занимать около 2-х байт. Для кодирования номеров стоп лемм в FL списке можно использовать алгоритм из [5] или алгоритм Хаффмана.

Кроме того, для каждой стоп леммы храним в индексе список первых словопозиций в документах. Это используется, только если запрос представляет собой ровно одно слово, лемма которого – стоп лемма.
**Расширенный индекс**. Для каждой часто используемой леммы w и каждой часто используемой или обычной леммы v хранит расширенный индекс (w,v). 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, не стоп лемма.

Словопозиция в расширенном индексе также включает в себя:

1) Расстояние от w до v.
2) До или после w находится v в тексте.

Словопозиций для случаев, когда w и v располагаются близко, меньше, чем суммарное число словопозиций для w и v по отдельности, что позволит ускорить выполнение поискового запроса переходом к новому, составному, виду ключа, в случае если w и v есть в составе запроса.

Отдельно следует рассмотреть случай, когда обе леммы w и v являются часто используемыми. В этом случае достаточно создать один расширенный индекс, например, (w,v). Если есть расширенный индекс (w,v), который представляет собой список словопозиций [(ID, P, Delta, DeltaFlag)], (ID, P) – словопозиция леммы w в текстах, где близко с w есть лемма v, Delta – расстояние между словопозициями w и v, DeltaFlag определяет до или после w находится v, (–1) или 1, соответственно. То, мы можем считать, что также есть логический расширенный индекс (v,w), список словопозиций [(ID, PV, Delta, (–1)*DeltaFlag)], где PV = P + Delta * DeltaFlag. Физически требуется хранить только расширенный индекс (w,v), но читать из индекса можно (w,v) и (v,w).

В зависимости от частоты встречаемости леммы w параметр ProcessingDistance определяется следующими параметрами, см. [7], в данной работе считаем, что он принимает значение от 5 до 7.

AI_DISTANCE=7.

AI_DISTANCE_MAP=5,500,6,500,7,500.

Для первых 500 часто используемых лемм используется ProcessingDistance = 5, для следующих 500, значение 6, для последующих 500, значение 7, для остальных также 7.

**Индекс последовательностей стоп лемм**. Для каждой последовательности стоп лемм, длины от 2 до FW_MAX_LENGTH, хранит словопозиции вхождения этой последовательности в текстах, [6]. Данный индекс используется, например, если в запросе все леммы – стоп леммы. Ключ в индексе – последовательность номеров стоп лемм в FL списке, которая отсортирована по возрастанию, для поиска без учета порядка.

FW_MAX_LENGTH=5. Определяет максимальную длину последовательности. Хранятся данные по последовательностям, длины от 2 по 5.

**Выполнение поискового запроса**

Поисковый запрос это строка текста, его выполнение включает в себя:

1) Выделение отдельных слов в запросе и учет логических операторов (или, и).
2) Морфологический анализ слов.
3) Формирование запроса в структурированном виде.
4) Учет разных видов лемм в запросе, преобразование его в набор запросов, при необходимости.
5) Выполнение запроса.
6) Сортировка результатов с учетом функций релевантности и длин найденных фрагментов. Например, предположим, что чем короче найденный фрагмент, тем он лучше и должен быть раньше в результатах поиска. Результаты одинаковой длины сортируем по релевантности.

**Поисковый запрос в структурированном виде**

Секцией будем называть объект, который содержит в себе лемму и список словопозиций. Поля секции:

**Lemma.** Лемма.

**Postings.** Список словопозиций, упорядочен по возрастанию.

**Value**. Определяет текущий элемент списка словопозиций. Вначале это первый элемент списка, затем мы можем последовательно брать следующий элемент списка.

Введем понятие «клетка» поискового запроса. Клеткой будем называть список секций. Поисковый запрос в структурированном виде это список клеток. Рассмотрим два примера.

Морфологический анализатор может возвращать для словоформы не одну, а несколько лемм. Этот набор лемм преобразуется в клетку. К примеру, «Мал золотник, да дорог». С использованием текущего словаря получаем: [мал, маленький] [золотник] [да] [дорогой, дорога]. Имеем четыре клетки.

Мы можем допустить использование логических операторов в составе поискового запроса, например, «|» – или. Поисковый запрос «a | b» возвращает документы, в которых встречается «a» или «b». Поисковый запрос «a | b c» возвращает документы, в которых встречается «c» и рядом с ним «a» или «b». В этом случае данный запрос преобразуется в две клетки, первая содержит базовые формы «a» и «b». Вторая, базовые формы «c». Например, «красный | алый цветок»: [красный, алый] [цветок].

Поскольку секция однозначно определяет лемму, мы можем говорить о клетке как о списке лемм.

**Учет разных видов лемм в запросе**

Как отмечено в [6], если в одной клетке присутствуют разные в смысле частоты встречаемости леммы, т. е. к примеру, одна лемма – стоп лемма, другая лемма – часто встречающаяся, то требуется поделить запрос на 2 части и выполнить их по отдельности. Иначе алгоритм поиска существенно усложняется.

Имеем запрос Query. Рассмотрим каждую клетку Query, пусть, i – номер текущей рассматриваемой клетки. Если в ней присутствуют $m > 1$ разных типов в смысле частоты встречаемости лемм, то создаем m копий поискового запроса, за исключением i-й клетки. В i-ю клетку каждого из новых поисковых запросов помещаем секции только с одним видом лемм, взятые из i-й клетки Query. Далее для каждого нового запроса действуем аналогично, обрабатывая (i+1)-ю и последующие клетки.

**Обработка запроса, состоящего только из стоп лемм**

Если запрос содержит только стоп леммы, то необходимо, чтобы в каждой клетке был ровно один элемент. Например, «Она живет у нас уже»

Здесь все словоформы имеют одну лемму, кроме «уже», которая имеет леммы «уж, уже, узкий». Все леммы – стоп леммы (при текущих настройках). Данный запрос превращается в 3 запроса:

1) [она], [мы], [у], [уже], [жить], 2) [она], [мы], [у], [уж], [жить],
3) [она], [мы], [у], [узкий], [жить].

Для запроса, который состоит только из стоп лемм, используется индекс последовательностей стоп лемм. В вышеуказанном примере каждый из 3-х запросов определяет ключ в этом индексе.

Предложение на развитие. Можно комбинировать индекс последовательностей стоп лемм с идеей хранения NSW-записей. Если последовательность стоп лемм длинная, то в составе ее словопозиции можно сохранить NSW-запись, о находящихся близко от нее, на расстоянии не более MaxDistance, слева или справа, других стоп леммах. Вместо хранения последовательностей стоп лемм длиной 5 или 6, можно сохранять только последовательности длиной 4 с NSW-информацией.

**Поисковый запрос в форме выполнения**

Для выполнения поискового запроса мы должны дополнить информацию в клетке. Введем понятие объекта-ленты или итератора словопозиций. Данный объект имеет операцию Next, возвращающую следующую запись и поле Value, которое содержит текущую запись. Записи, которые выдает итератор, должны быть упорядочены, то есть следующая запись должна быть не меньше предыдущей.

Каждая секция это итератор словопозиций. Реализация итератора включает в себя чтение списка словопозиций из индекса. Вначале итератор соответствует первой записи списка словопозиций. Операция Next позволяет перейти к следующей записи.

Клетка в свою очередь является итератором словопозиций. Итераторы секций клетки образуют упорядоченный список. Итератор с минимальным значением текущей словопозиции находится в начале списка, и его значение есть значение итератора клетки Value. Операция Next для клетки вызывает Next для секции с минимальным значением, после чего список секций переупорядочивается при необходимости, если значение текущей секции становится больше значения другой секции.

**Простой алгоритм поиска**

Определим вначале простой алгоритм поиска, а затем алгоритмы поиска, использующие дополнительные индексы.

**Параметры и переменные**

FragmentDistance = 1024 – определяет рекомендуемую длину фрагмента текста в словах, в котором должен быть заключен результат поиска, если поиск идет с учетом расстояния.

RecordFound – признак того, что в текущем документе найдет результат. Если в документе найден хотя бы один результат, то все результаты, длина которых превосходит FragmentDistance, игнорируются.

**Процедура инициализации поиска**

Последующие алгоритмы будем рассматривать, предполагая, что поле ID текущих словопозиций всех клеток одинаково.

Определим MinID = минимальное значение поля Value.ID среди всех клеток. Если в какой-то клетке значение Value.ID превышает MinID, то все словопозиции документа MinID пропускаются. Процедура повторяется, до тех пор, пока Value.ID у всех клеток не будет одинаковым. После чего список клеток сортируем по значению Value.P.

**Простая процедура поиска**

Все клетки образуют упорядоченный список, по возрастанию значения Value.P, начало S, конец E. Текущее значение Value всех клеток имеет одинаковое поле ID. Выполняем в цикле:

1) Присваиваем I = E.Value.P – S.Value.P
2) Found = I < FragmentDistance или RecordFound = FALSE.
3) Если Found = TRUE выполняем:
   a. Добавляем в результат поиска запись (S.Value.ID, S.Value.P, I).
   b. Присваиваем RecordFound = TRUE.
4) Переходим к следующей записи в S.
   a. Выполняем S.Next.
   b. Проверяем условие упорядоченности списка, R = следующая клетка за S в списке, если R.Value.P < S.Value.P тогда список требуется переупорядочить: определяем клетку T, T.Value.P >= S.Value.P, если такой нет, то E = S, иначе, вставляем S перед T; присваиваем S = R.
   c. Если S.ID поменялся, значение клетки имеет другой ID, то осуществляется процедура инициализации поиска.

**Алгоритм поиска для случая часто используемых слов**

Запрос это несколько слов. В данном разделе мы рассмотрим одну из подзадач поиска, а именно, обработку поискового запроса, в котором:

1) У слов запроса нет стоп лемм.
2) Хотя бы для одного слова запроса все леммы часто используемые.

Смысл алгоритма: выберем некое слово s в запросе, такое, что все леммы s часто используемые, назовем его основным словом запроса. Для каждого другого слова запроса t рассмотрим расширенные индексы (s, t). Расширенный индекс (s, t) содержит словопозиции слова s, когда t было близко с s. Рассмотрим последовательно каждую позицию слова s в текстах, и проверим, находились ли близко с s все из требуемых слов запроса.

То есть, если есть словопозиция R = (ID1, P1) слова s, то для каждого другого слова запроса t расширенный индекс (s, t) должен содержать словопозицию R. Рассмотрим данный подход с учетом того, что словоформа может иметь несколько лемм.

**Выбор основного индекса**

Выберем индекс M, который соответствует клетке запроса, с минимальной суммарной частотой встречаемости лемм, среди таких клеток, у которых все леммы часто используемые. Для каждой секции w клетки M, и для каждой секции v остальных клеток существует

расширенный индекс (w, v). Этот расширенный индекс определит список словопозиций Postings для секции v.

Будем считать, что для v есть список словопозиций Postings = (P, Delta, DeltaFlag), где:

- P – позиция (номер) слова w в документе,
- Delta – расстояние от w до v,
- DeltaFlag – определяет, до или после w находится v в тексте.

Данный список упорядочен по полю P, по возрастанию.

В данном списке сейчас нет ID документа, потому что алгоритм применяется в рамках обработки одного документа. После обработки одного документа списки словопозиций очищаются, и заполняются данными следующего документа, затем поиск повторяется для него.

**Вспомогательные переменные**

Введем набор массивов, количество элементов в которых равно L = («Число клеток запроса» – 1). Одна ячейка массива соответствует одной клетке запроса, не совпадающей с основной клеткой M.

**Flags**. Битовый массив.

**Positions, PositionFlags**. Массивы расстояний. Пусть ячейка массива соответствует клетке T. В тексте есть лемма w клетки M и лемма v клетки T. В ячейке Positions хранится расстояние от v до w, а в PositionFlags – до или после w находится v в тексте.

**Lemmas**. Каждый элемент массива содержит список идентификаторов лемм клетки запроса.

Дополнительные переменные:

**Current**. Текущая позиция.

**Marked**. Счетчик тех клеток запроса V, леммы которых были найдены, т. е. выполнялось V.Value.P = Current.

**Алгоритм**

Формируем временную клетку S, в которую помещаем все секции клеток, не совпадающих с M.

Выполняем в цикле:

1) Если S.Value.P не равно Current то:
   a. Если Marked = L, то мы нашли искомое. Сохраняем позицию Current в результатах поиска. Массивы Positions, PositionFlags определяют позиции начала и конца фрагмента текста, содержащего слова запроса, минимальное по i значение Current + Positions[i]* PositionFlags[i] определяет начало фрагмента, максимальное – конец.
   b. Очищаем Flags (все ячейки теперь равны 0).
   c. Присваиваем Marked = 0.
   d. Current = S.Value.P.
2) Помечаем S.Lemma. Выполняем одно из двух.

a. Если есть индекс i, такой что Lemmas[i] содержит S.Lemma и Flags[i] = 0, то меняем Flags[i] на 1 и сохраняем: S.Value.Delta в Positions[i], PositionFlags[i] = S.Value.DeltaFlag, Marked = Marked + 1.
b. Если есть индекс i, такой что Lemmas[i] содержит S.Lemma и Flags[i] = 1 и Positions[i] > S.Value.Delta, меняем: Positions[i] = S.Value.Delta, PositionFlags[i] = S.Value.DeltaFlag.

3) Переходим к следующему значению клетки S.

**Алгоритм поиска при наличии стоп слов в запросе.**

Пусть дан запрос длины L. Выберем индекс M, который соответствует клетке запроса, с минимальной суммарной частотой встречаемости лемм, среди таких клеток, в которых нет стоп лемм. То есть в клетке M могут быть либо часто используемые, либо обычные леммы, соответствующие известным или неизвестным словам.

Для клетки M будем извлекать из обычного индекса все словопозиции, при этом для каждой словопозиции будем обрабатывать NSW-запись. Эта запись хранит в себе информацию о располагающихся не более чем на расстоянии MaxDistance от текущего слова стоп леммах. Таким образом, мы учитываем клетки, состоящие из стоп лемм.

Мы можем свести алгоритм поиска к простому алгоритму поиска.
Обрабатываем каждую клетку T.

**Если это клетка M**, то для каждой ее леммы v список словопозиций Postings извлекается из обычного индекса.

**Иначе, если T содержит стоп леммы**, то учитываем это при чтении словопозиций клетки M. Когда читаем словопозиции леммы v клетки M, то извлекаем из ее списка Postings, из NSW-записей данные о стоп леммах. NSW-запись словопозиции X это набор записей (N, Delta, DeltaFlag), где N – номер стоп леммы в частотном списке FL, Delta – расстояние от X до стоп леммы, DeltaFlag – определяет, до или после X располагается в тексте стоп лемма, значения (–1) и 1, соответственно. Если N соответствует стоп лемме q клетки T, то добавляем запись (X.ID, X.P + Delta*DeltaFlag), в список Postings словопозиций ее секции.

**Иначе, если T содержит часто используемые леммы**. Для каждой леммы w клетки T и каждой леммы v клетки M расширенный индекс (w,v) определяет список Postings леммы w.

**Иначе, T содержит обычные леммы**.

1) Если в клетке M находятся часто используемые леммы, то можно поступить как на предыдущем шаге. То есть, для каждой леммы w клетки M, и каждой леммы v клетки T, логический расширенный индекс (v,w) определяет список Postings секции v. Еще раз отметим, что в каждой клетке находятся леммы одного вида, в смысле частоты встречаемости, стоп леммы, часто используемые леммы или обычные. Расширенный индекс (w,v) где w – часто используемая лемма, порождает два логических расширенных индекса (w,v) и (v,w).

2) Если существует клетка Z, отличная от T и M, при этом в Z находятся часто используемые леммы. Если клеток с таким условием несколько, выбираем клетку с минимальной суммарной частотой встречаемости лемм. Для каждой леммы w клетки Z, и каждой леммы v клетки T, логический расширенный индекс (v,w) определяет список Postings секции v.
3) Иначе, мы читаем список словопозиций Postings для лемм клетки T из обычного индекса. При этом NSW-записи могут игнорироваться.

Отметим, что в [6] сказано, что один поток словопозиций может храниться в нескольких потоках данных, например поле ID в одном потоке, поле P в другом, а NSW-записи в третьем. Это позволяет не считывать с диска определенный поток данных, если он не нужен. Как только секции клеток определены, мы можем использовать простой алгоритм поиска.

**Эксперименты поиска**

Было проиндексировано 195 тыс. файлов, объем текста 71,5 Гб., файлы представляли собой обычный текст, однобайтовая кодировка, по стилю, художественная литература, в основном русский язык. Запросы также состоят из слов русского языка.

Методика экспериментов основана на методике из [6]. Проведенные эксперименты поиска заключались в следующем:

1) Выберем случайным образом некоторый документ в индексе.
2) Наборы слов для поиска выбираем из документа 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. Выбранный набор запросов применяем к разным видам индекса. Основной параметр, который измеряем, это количество прочитанных словопозиций при выполнении одного поискового запроса. Скорость поиска в основном зависит от этого. Выполняя большое количество запросов, измеряем среднее значение этого параметра для конкретного вида индекса.

Преимущество подобного подхода:

1) Проверяем, что индекс построен корректно. Так как выбираем запросы из уже проиндексированного документа, то мы их точно должны найти. Проверяем, что в результатах поиска присутствует запись, соответствующая тому документу, из 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.

**Виды индексов для экспериментов поиска**

Idx1. Обычный индекс. Включает для каждой леммы, включая стоп леммы, список всех ее словопозиций. Не содержит NSW-записей.
Idx2. Индекс с дополнительными индексами, структура дана в разделе «Используемые виды индекса». Стоп лемм 700 (524 ru, 176 en), часто используемых лемм 2100 (1648 ru, 452 en).
Idx3. Индекс с дополнительными индексами, число часто используемых лемм 4200 (3375 ru, 825 en), увеличено в 2 раза, по сравнению с Idx2.
Размеры файлов словопозиций:
**Idx1**: 43,4 Гб. **Idx2**: обычный индекс 103 Гб., индекс последовательностей стоп лемм 95,4 Гб., расширенные индексы 142 Гб. **Idx3**: расширенные индексы: 219 Гб., остальные как у Idx2.

**Эксперимент 1**

В запросе могут быть все виды лемм. Если у каждого слова запроса есть стоп лемма, то используется фраза, состоящая только из подряд располагающихся слов. Иначе рассматривается как вариант 2, а, так и 2, б, методики. Выполнено 4500 запросов.
Среднее число обработанных словопозиций:
Idx1: 171 млн., Idx2: 733 тыс., Idx3: 644 тыс.
Можно также отметить, что поиск в Idx1 занял 21,8 часа, а поиск в Idx2, Idx3 занимал 16-14 мин.
В данных запросах 330 запросов включает только стоп леммы. 462 запроса не содержат стоп лемм. За счет того, что большая часть запросов содержит стоп леммы, число обработанных словопозиций велико для Idx1. В данном эксперименте за счет дополнительных индексов среднее число словопозиций, прочитанных при выполнении запросов снизилось в 233 раза для случая Idx2 и в 263 раза для случая Idx3, по сравнению с Idx1. Отметим, что основное значение здесь имеют NSW-записи. Расширенные индексы имеют меньшее значение, что и показала небольшая разница между Idx2 и Idx3.

**Эксперимент 2.**

В запросе нет стоп лемм. Выполнено 5955 запросов.
Среднее число обработанных словопозиций:
Idx1: 1,989 млн., Idx2: 167 тыс., Idx3: 43,8 тыс.
Число словопозиций для всех видов индексов существенно меньше случая, когда в запросе есть стоп леммы. За счет дополнительных индексов среднее число словопозиций, прочитанных при выполнении запросов снизилось в 12 раз для случая Idx2 и в 51,5 раз для случая Idx3, по сравнению с Idx1. То есть, увеличение числа часто используемых лемм в Idx3, по сравнению с Idx2, значительно повлияло на производительность.

**Результаты**

Рассмотрены алгоритмы полнотекстового поиска с использованием дополнительных индексов. Представлены результаты экспериментов поиска. Результаты экспериментов подтверждают эффективность

дополнительных индексов, за счет них число обрабатываемых словопозиций при обработке запроса, может быть снижено от 10 до 200 раз. Результаты экспериментов показывают, что есть потенциал для изменения параметров, таких как MaxDistance = 5 (словопозиция слова может включать в себя информацию о стоп словах, располагающихся на не более MaxDistance от текущего слова).

**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) 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.
4) George Kingsley Zipf. Relative frequency as a determinant of phonetic change. Harvard Studies in Classical Philology, Vol 40, 1929, p. 1–95.
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 дополнительных индексов для более быстрого полнотекстового поиска фраз, включающих часто встречающиеся слова, Системы управления и информационные технологии, 2016, №1(63), 27–33.
8) Bahle D., Williams H.E., Zobel J.; Efficient Phrase Querying with an Auxiliary Index, In Proc. ACM-SIGIR Conf. on Research and Development in Inform. Retrieval, Finland, 2002, p. 215–221.
9) Williams H. E., Zobel J., Bahle D.; Fast phrase querying with combined indexes, ACM TOIS, 2004, №4(22), 573–594.
10) Chang M., Chung Keung Poon; Efficient Phrase Querying with Common Phrase Index, ECIR 2006, LNCS 3936, Springer-Verlag Berlin Heidelberg, 2006, 61–71.
11) Shashank Gugnani, Rajendra Kumar Roul; Triple Indexing: An Efficient Technique for Fast Phrase Query Evaluation, International Journal of Computer Applications, 2014, №13(87), 9–13.