УДК 519.683.5

Эффективный полнотекстовый поиск с учетом близости слов при помощи трехкомпонентных ключей. Efficient full-text proximity search by means of three component keys.

А. Б. Веретенников
A. B. Veretennikov

Уральский федеральный университет.
Институт естественных наук и математики.
Ural Federal University.
Institute of Natural Sciences and Mathematics.

Полнотекстовый поиск, поисковые системы, инвертированные файлы, дополнительные индексы, поиск с учетом близости слов.

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. Разработан новый вид индекса с трехкомпонентными ключами. Приведены алгоритмы создания индекса и результаты экспериментов поиска.

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

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. A new three component key based index has described. Results of experiments are given.

Введение

Задача 1. Задача полнотекстового поиска в общем виде заключается в поиске документов, которые содержат заданные слова. Результат поиска – это список документов и информация о том, где в них располагаются слова запроса. Если слова запроса располагаются в документе вблизи, в идеале, в виде фразы, то такие документы в списке результатов показываются в начале, как более релевантные, в отличие от документов, в которых те же слова находятся на значительном расстоянии друг от друга, например, в разных разделах документа.

Если есть два документа, в которых искомые слова располагаются друг от друга на, примерно, одинаковом расстоянии, то применяется отдельная функция релевантности, для того, чтобы определить, какой из них более важен. Задача 1 требует для своего решения сохранения в индексе информации о каждом вхождении каждого слова. Поэтому, время выполнения запроса зависит от **суммарного объема текстов** (например, в байтах).

Для решения данной задачи применяются инвертированные файлы [1] и их аналоги [2, 3].

Задача 2. Задача полнотекстового поиска с учетом близости слов, предполагает, что мы ищем только те документы, в которых слова запроса располагаются близко друг к другу. Предполагается, что близко расположенные слова связаны друг с другом по смыслу, являются частью некого высказывания или фразы.

Задача 3. Задача полнотекстового поиска без учета расстояния заключается в поиске документов, в которых искомые слова могут находиться где угодно. Предполагается, что эти слова не являются частью некой фразы или высказывания, но могут быть связаны по тематике или предметной области. Задача 3, в отличие от Задачи 1, для конкретного слова требует сохранения информации лишь о первом его вхождении в документе. Что соответствует информации о том, есть ли данное слово в документе или нет. Поэтому, время выполнения запроса зависит от **количества документов**. В случае, если документы достаточно велики, (например, это книги, или статьи, объемом от 10-20 страниц) то запрос выполняется на порядок быстрее, чем в случае задачи 1.

Дополнительные индексы

В текущем исследовании рассматриваются способы оптимизации решения Задачи 2. Предложены ряд методов, которые позволяют искать в условиях Задачи 2, в десятки раз быстрее, чем в общем случае, то есть в условиях Задачи 1.

Как было сказано, в определенных случаях, если документы имеют достаточно большой размер, Задача 3 выполняется существенно быстрее Задачи 1. Кроме того, если все слова запроса являются часто встречающимися, то поиск в условиях Задачи 3 выполнять не требуется, так как по отдельности такие слова, как правило, не несут существенного смысла.

Актуальность применения дополнительных индексов будем обосновывать следующим **основным** утверждением.

Задача 1 может быть сведена к **Задаче 2** и **Задаче 3**. Вначале мы можем выполнить поиск в условиях Задачи 2. Затем, если релевантных результатов не найдено, то выполняется поиск в условиях Задачи 3.

Если поиск в условиях Задач 2 и 3 выполняется существенно быстрее, чем в условиях Задачи 1, то мы можем утверждать, что за счет разработанной методологии поиск в общем случае может осуществляться существенно быстрее.

Для эффективного решения Задачи 2 можно создать дополнительные индексы. Различные слова встречаются в текстах с разной частотой [4]. Чем чаще встречается слово, тем дольше осуществляется поиск. Поэтому дополнительные индексы создаются для часто встречающихся слов. В [5, 6] были рассмотрены несколько стратегий использования дополнительных индексов. В [7] даны алгоритмы создания изучаемых дополнительных ин-

дексов. В [8] рассмотрены детально алгоритмы поиска.

Виды слов

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

1) «Стоп слова»: и, в, или. Встречаются очень часто и могут, в некоторых поисковых системах, 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».

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, включая стоп 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.

Термины слово и словоформа здесь и далее считаем взаимозаменяемыми. 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. То есть имеем три типа лемм в смысле частоты встречаемости: стоп леммы, часто используемые леммы и остальные.

Слово, что входит 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.

Виды запросов

Для поисковых запросов, включающих разные виды слов, могут применяться разные виды дополнительных индексов и алгоритмы. В [8] рассмотрены следующие подзадачи:

1) Все леммы запроса – стоп леммы.
2) По крайней мере, у одного слова запроса, все леммы часто используемые, остальные леммы запроса часто используемые или обычные.
3) Все остальные виды запросов, при этом в запросе могут присутствовать все виды лемм.

Результаты, представленные в [8], говорят о том, что средняя скорость выполнения запросов, в случае наличия в составе запроса стоп лемм, может быть увеличена в 200 раз и более. В случае наличия в составе запроса часто используемых лемм, но отсутствия стоп лемм, в 50 раз и более.

Частотный список лемм

FL – список известных лемм, отсортированный в порядке убывания частоты их появления в текстах. Введем параметр WsCount, например, WsCount =700. Будем также называть порядковый номер леммы в FL списке – FL-номером, нумерация с нуля.

Пусть первые WsCount элементов списка FL, то есть, первые WsCount самые часто встречаемые леммы, это стоп леммы. Далее, следующие 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.

Запросы, состоящие из стоп лемм

Отдельно следует рассмотреть случай, когда все леммы запроса – стоп леммы. Алгоритм в [8] в данном случае имеет ограничение, находятся только те тексты, где между словами запроса нет других слов, хотя, при этом, поиск осуществляется без учета порядка. В данной работе предлагается расширенное решение данной подзадачи, когда в искомых документах между искомыми словами могут быть и другие слова.

Рассмотрим запрос «who are you who», и, допустим, есть один документ, в котором есть фрагмент «The Who – Who are you», и второй документ, в котором есть фрагмент «Who are you by Who». Во втором случае в тексте посреди искомых слов запроса есть слово «by».

Все леммы слов who, are, you, who, являются стоп леммами и методы из [8] позволят найти только первый документ. В то время как доработанный метод, представленный в данной работе, позволит найти оба документа. Примечание. Who – название исполнителя произведения «Who are you».

Данное ограничение в [8] относится только к таким запросам, в которых все леммы – стоп леммы. Рассмотрим другой пример, запрос «time and a word yes», и, допустим, есть один документ, в котором есть фрагмент «Yes – Time and a word», и второй документ, в котором есть фрагмент «Time and a word by Yes». В данном случае, и методы [8], и новый метод, позволят найти оба документа, так как слово «Yes» является часто используемым (при WsCount = 700 и текущем словаре).

Заметим, что описанная ситуация с запросом «who are you who» встречается очень редко, поэтому ограничением метода [8] во многих случаях можно пренебречь. Однако, хотелось бы иметь метод, который позволяет преодолеть данное ограничение, что является целью данной работы.

Структура инвертированного файла

Инвертированный файл или инвертированный индекс – это ассоциативный массив. В качестве ключа может выступать, например, слово, базо-

вая форма слова, набор слов, набор базовых форм слова. Например, в [9, 10] ключ – пара слов или даже фраза.

Значение ключа – это список записей вида (ID, P), где ID – это идентификатор документа, а P – позиция ключа в документе. В качестве позиции ключа P используем порядковый номер слова в документе. Запись (ID, P) будем называть записью о вхождении слова в документе или словопозицией. Идентификатор документа ID будем считать целым числом, например, номером документа.

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

Инвертированный файл состоит из двух компонентов:

1) Файл данных. Хранит списки словопозиций. Словопозиции для одного ключа должны храниться преимущественно последовательно для их быстрого чтения при поиске.
2) Словарь. Хранит в себе ключи и для каждого ключа дескриптор – структура, которая содержит информацию о том, где в файле данных располагаются словопозиции этого ключа.

Расстояние между словами

Рассмотрим пример: «скажи мне, кто твой самый близкий друг».

Поскольку у каждого слова есть номер, мы можем говорить, например, что в тексте на расстоянии 2 друг от друга были слова «скажи» и «кто».

Поскольку у каждого слова может быть одна или несколько лемм, мы можем сказать, что в тексте на расстоянии 2 друг от друга были леммы «сказать» и «кто». Или, что на расстоянии 1 друг от друга были леммы «сказать» и «я». В данном примере у каждого слова ровно одна лемма, соответственно: [сказать], [я], [кто], [твой], [самый], [близкий], [друг]. Или, [58], [4], [91], [236], [100], [425], [170], если мы заменим леммы на их номера в FL-списке.

Рассмотрим пример, когда слово имеет несколько лемм: «она живет у нас уже». Здесь все словоформы имеют одну лемму, кроме «уже», которая имеет леммы «уж, уже, узкий». Можно сказать, что лемма «узкий» располагается в тексте на расстоянии 4 от леммы «она».

Расширенный индекс стоп лемм.

Расширенный индекс стоп лемм (f,s,t) – это список словопозиций стоп леммы f, когда в тексте не более чем на расстоянии MaxDistance от f располагались стоп леммы s и t. Значения f, s и t являются номерами соответствующих стоп лемм в FL списке. MaxDistance – заданный параметр, например, 5.

Заметим, что если мы храним расширенный индекс (f,s,t), то сохранять расширенный индекс, с другим порядком компонент, например, (f,t,s) или (s,t,f) – избыточно. Данные (f,t,s), (s,t,f) и других комбинаций порядка f, s, t можно восстановить по данным (f,s,t). Поэтому будем создавать для

трех стоп лемм f, s, t, только такой расширенный индекс (f,s,t), что $f \le s \le t$. Что означает, что f – встречается в тексте не реже, чем s, а s встречается в тексте не реже, чем t.

В составе словопозиции сохраняем расстояние от s до f и от t до f. Нужно также указывать для s и t, была ли соответствующая лемма в тексте до или после f. Если s была после f в тексте, сохраняем положительное число, иначе отрицательное. Аналогично для t.

Рассмотрим небольшой текст: скажи мне, кто твой самый близкий друг.

Примеры словопозиций:

Ключ (я, самый, твой) –> (4, 100, 236): словопозиция (1, 3, 2), где 1 – номер леммы «я» в тексте, начиная с нуля, 3 – расстояние между леммами «самый» и «я», 2 – расстояние между «твой» и «я».

Ключ (я, сказать, друг) –> (4, 58, 170): словопозиция (1, –1, 5). Расстояние (–1) между «сказать» и «я», т. к. лемма «сказать» в тексте присутствует до леммы «я», 5 – расстояние между «я» и «друг».

Словарь индексов стоп лемм

Поскольку число разных ключей (f,s,t) ограничено, имеет смысл в качестве словаря использовать массив, хранящийся в оперативной памяти. Введем переменную C = WsCount.

Пусть KeySet – множество троек целых чисел (f,s,t), с условиями:

1) $f \le s \le t$,
2) $0 \le f < C,\ 0 \le s < C,\ 0 \le t < C$,

Должна быть определена биекция множества KeySet на множество всех целых чисел из полуинтервала $[0, X)$, где $X = |\,\text{KeySet}\,| = \binom{C+2}{3}$ – количество элементов в массиве словаря.

Используем стандартные формулы:

$\binom{n}{k}$ – число сочетаний из n по k, $\binom{n+k-1}{k}$ – число сочетаний с повторениями из n по k.

Об алгоритме создания расширенных индексов стоп лемм

Будем применять алгоритм, подобный алгоритму создания индекса двухкомпонентных ключей из [7], за исключением того, что будет использоваться один временный файл (в смысле определения «временного файла» из [7], раздел «Создание индекса»).

Пусть мы обработали один набор документов и создали индекс. Затем мы получили еще один набор документов. Требуется получить индекс, включающий данные первоначальных и новых документов. Операцию построения нового индекса с использованием первоначального индекса назовем обновлением индекса. Существует два способа создания инвертированных индексов: внешняя сортировка и легко обновляемые индексы [7].

В первом случае мы читаем документы, формируем и сохраняем в файле данных пары (ключ, словопозиция) в порядке их встречаемости. Далее файл данных сортируется по ключу. Затем в словаре для каждого ключа сохраняется адрес расположения его значений в файле данных, а сам ключ из файла данных устраняется. Процесс обновления индекса заключается в построении нового индекса на основании нового набора документов и последующем слиянии предыдущего и нового индекса.

Во втором случае словопозиции ключа хранятся в наборе блоков, которые могут располагаться в разных местах файла данных. Для каждого ключа в словаре хранится информация об используемых блоках. При добавлении новой словопозиции в индекс, в список блоков ключа могут быть добавлены новые блоки. Обновление индекса осуществляется также как и первоначальное создание индекса, то есть последовательно добавляются словопозиции для конкретного ключа в его набор блоков.

В [7] и в данной работе используется второй вариант. Применяются блоки одинакового размера, которые называем кластерами. Для эффективной работы алгоритма создания индекса предполагается, что не менее одного блока для каждого ключа хранится в оперативной памяти в момент обновления индекса.

При большом числе ключей множество ключей разбивается на группы. Группы ключей обрабатываются последовательно. За счет этого в памяти требуется хранить блоки только одной группы, а не всего множества ключей. При завершении обработки одной группы ключей ее кэш сохраняется на диск и очищается, и обработка последующей группы ключей осуществляется с новым кэшем.

Например, если ключ – это лемма слова. Мы разбиваем WordsCount лемм на некоторое число групп. Затем записываем в индекс словопозиции, соответствующие первой группе ключей. После чего кэш сохраняется на диск и очищается. Далее записываем в индекс словопозиций второй группы ключей, и так далее.

Количество индексов трехкомпонентных ключей

Будем создавать m индексов, где m – вычисляемый параметр. Каждый индекс будет хранить данные подмножества ключей, при этом разделение на подмножества осуществляется на основании первой компоненты ключа. Множества ключей индексов определяются диапазоном допустимых значений первой компоненты ключа.

Ключи конкретного индекса будут делиться на необходимое количество групп, при этом разделение на группы осуществляется на основании второй компоненты ключа. Группа ключей определяется диапазоном допустимых значений второй компоненты ключа.

Введем обозначения для диапазона целых чисел.

Целое число x принадлежит диапазону $[a, b]$, если $a \le x \le b$.

Целое число x принадлежит диапазону $[a, b)$, если $a \le x < b$.

Под индексом $[a,b]$ будем понимать индекс, в словаре которого хранятся ключи, у которых первая компонента принадлежит диапазону $[a,b]$.

Создание и обновление индекса

Осуществляется чтение документов, формируется временный файл FList, представляющий собой список всех словопозиций стоп лемм, упорядоченный по словопозиции. Данный список хранится в оперативной памяти. То есть, мы читаем документы подряд, и для каждого вхождения стоп леммы формируем словопозицию и помещаем ее в конец списка FList.

Размер списка ограничен. Как только размер списка достигает заданного предела, осуществляется обновление индексов. Далее список очищается, и чтение документов продолжается.

Обновление индексов в случае трехкомпонентных ключей

Для m индексов запускается m программных потоков (одновременно может быть запущено менее m потоков, например, $g < m$, а (g+1)-й будет запущен, когда один из g потоков завершит работу и т.д.). Один поток соответствует одному индексу.

В каждом потоке осуществляется цикл по группам ключей индекса. В итерации цикла осуществляется последовательное чтение FList и для:

- каждой словопозиции P1 стоп леммы f, входящей в допустимый диапазон значений первой компоненты ключа, определенный для текущего индекса,
- каждой словопозиции P2 стоп леммы s, входящей в допустимый диапазон значений второй компоненты ключа, определенный для текущей группы ключей, с условием $s \geq f$,
- каждой словопозиции P3 стоп леммы t, с условием $t \geq s$,

если расстояние между P1 и P2, а также расстояние между P1 и P3, по модулю $\leq \mathrm{MaxDistance}$, то формируем словопозицию P для ключа (f,s,t), на основании P1, с включением расстояний между f, s, и f, t. Словопозицию P сохраняем в индекс.

Дополнительные условия на функцию отображения трехкомпонентных ключей на отрезок.

Текущий алгоритм построения индекса предполагает, что одновременно мы можем помещать в индекс ограниченное число ключей. А именно, для каждого ключа в момент добавления данных в индекс, необходимо хранить минимум один кластер в оперативной памяти.

Кроме того, требуется распараллелить создание индекса, поэтому множество KeySet требуется разделить на некоторое число непересекающихся подмножеств, данные каждого из которых обрабатывать независимо.

Будем разделять множество KeySet по первой компоненте элемента множества. Подсчитаем, сколько всего различных ключей при фиксированном f.

Если мы рассматриваем тройки (f,x,y), при условии $x \geq f$, $y \geq x$, то x

меняется в пределах от f до максимального значения $C-1$, т. е. $f \le x \le C-1$, всего будет $p=(C-f)$ разных значений x. Для каждого такого значения x мы имеем $(C-x)$ значений y. Сколько всего различных ключей, при фиксированном f, определяет функция $Q(f)$, см. формулу (1):

$$Q(f)=\sum_{x=f}^{C-1}(C-x)=\binom{C-f+1}{2}= \quad (1)$$

$$(p(p+1))/2=((C-f)(C-f+1))/2.$$

Один индекс Ind_k, $0 \le k < m$, будет содержать в себе все ключи (f,s,t), где $a_k \le f \le b_k$, значения s, t – все возможные допустимые, то есть $f \le s \le C-1, s \le t \le C-1$.

Границы диапазонов a_k и b_k следует выбрать таким образом, чтобы времена построения индексов примерно совпадали, для наилучшего распараллеливания создания индексов, $a_0=0$, $a_{k+1}=b_k+1$, для $0 \le k < m-1$, последний $b_{m-1}=C-1$, всего индексов m. То есть, диапазон $[0,C)$ разбиваем на множество подряд идущих диапазонов.

Пусть некий индекс содержит n ключей, где первая компонента ключа принимает значения в диапазоне $[a,b]$. Зная количество ключей для каждого значения первого компонента ключа, мы можем сопоставить каждому ключу номер ячейки массива словаря. Ключам вида (a,x,y) сопоставим первые $Q(a)$ элементов массива словаря. Ключам вида $(a+1,x,y)$ сопоставим следующие $Q(a+1)$ элементов массива и так далее. Для каждого значения f, $a \le f \le b$, ключам вида (f,x,y) будет соответствовать диапазон номеров ячеек массива словаря, обозначим его $[S_f, E_f]$.

Количество ключей при фиксированной первой компоненте ключа и второй компоненте ключа, принадлежащей заданному диапазону

Следующий вопрос, как отобразить множество пар (f,x,y) при фиксированном f, $f \le x$, $x \le y$, на диапазон номеров ячеек массива, соответствующий f. Функция $L(v,w)$, формула 2, определяет, сколько всего имеем ключей при фиксированном f, и при x принадлежащем диапазону $[v,w]$, $w \ge v \ge f$.

$$L(v,w)=\sum_{x=v}^{w}(C-x)=$$

$$(C-v)+(C-v)-1+\ldots+(C-v)-(w-v)= \quad (2)$$

$$(C-v)(w-v+1)-(0+1+\ldots+(w-v))=$$

$$(C-v)(w-v+1)-((w-v)(w-v+1))/2.$$

Отображение ключей (f,x,y) на диапазон $[S_f,E_f]$.

При фиксированных f и x для y имеем $(C-x)$ значений. Первым $(C-f)=L(f,f)$ значениям диапазона $[S_f,E_f]$ сопоставим ключи вида (f,f,y), следующим $(C-(f+1))$ значениям диапазона $[S_f,E_f]$ сопоставим ключи $(f,f+1,y)$ и т. д. Числовые значения ключей (f,x,y), при фиксированных f и x, будут начинаться с локального внутри $[S_f,E_f]$ номера, определяемого функцией $N(f,x)$, см. формулу 3.

$$N(f,x)=\begin{cases}0, \text{при } x=f,\\ L(f,x-1), \text{при } x>f.\end{cases} \quad (3)$$

При фиксированном f, если мы рассмотрим множество H ключей (f,x,y), где x входит в диапазон $[s,e]$, то есть $s\le x\le e,\ x\le y,\ f\le s\le e$, то числовые значения ключей H внутри $[S_f,E_f]$ будут начинаться с локального внутри $[S_f,E_f]$ номера $N(f,s)$ и последнее будет равно $(N(f,e+1)-1)$.

Функция отображения трехкомпонентного ключа для индекса

Для Ind_k функция отображения ключа на диапазон значений $\left[0,\sum_{x=a_k}^{b_k}Q(x)\right]$ определяется по формуле 4:

$$Z(f,s,t)=\left(\sum_{x=a_k}^{f-1}Q(x)\right)+N(f,s)+(t-s) \quad (4)$$

При этом, на практике для вычисления ее значения первое слагаемое рассчитывается заранее для всех возможных f.

Параметры и переменные

MaxGroupsPerIndex – максимальное число групп ключей для одного индекса, задаем параметром, порядка 30-100.

MaxGroupSize – максимальный размер группы ключей.

$Freq(j)$ – число вхождений леммы с индексом j в FL списке в эталонном наборе текстов. В эталонном наборе текстов подсчитано количество встречаемости каждой леммы, и далее построен FL список путем упорядочения известных лемм по убыванию количества встречаемости.

$Avg=\left(\sum_{x=0}^{C-1}Freq(x)\right)/C$, среднее количество встречаемости стоп леммы.

Алгоритм формирования набора групп для индекса

Пусть есть индекс, определенный диапазоном значений первой компоненты ключа $[a,b]$. Группа ключей определяется диапазоном допустимых значений второй компоненты ключа.

Вначале создаем пустой диапазон, затем в цикле увеличиваем размер диапазона текущей группы. Если размер группы с увеличенным диапазоном превышает MaxGroupSize, текущую группу фиксируем, формируем новый пустой диапазон, начинающийся сразу после диапазона текущей группы, начинаем формировать следующую группу.

Более подробно, начинаем строить группу, начало s и конец диапазона e равны a. Пробуем расширить диапазон. Если у группы, определяемой диапазоном $[s,e+1]$ количество ключей $\le$ MaxGroupSize, то диапазон расширяем, увеличиваем $e=e+1$. Иначе, фиксируем группу $[s,e]$, затем присваиваем $s=e+1, e=e+1,$ и начинаем строить новую группу.

Количество ключей группы [s, e] для индекса [a, b] определяется по формуле 5:

$$\sum_{f=a}^{b} N(f,\max(f,e+1)) - N(f,\max(f,s)) \quad (5)$$

Выполняем эти действия, пока s < C.

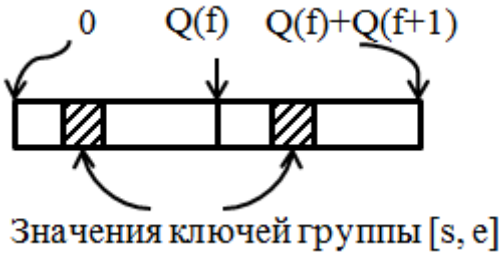

Рис 1. Пример отображения ключей группы на массив словаря.

На рис 1. показан пример отображения ключей некой группы ключей $[s,e]$ на массив дескрипторов словаря некого индекса, $[f,f+1]$, диапазон изменения первой компоненты ключа включает в себя два значения.

Массив дескрипторов разбит на два диапазона. Первый соответствует значению f первой компоненты ключа, он начинается с 0 и заканчивается $(Q(f)-1)$, второй соответствует значению f+1, начинается с $Q(f)$ и заканчивается $(Q(f)+Q(f+1)-1)$. На каждом из этих диапазонов группе ключей $[s,e]$ выделен вложенный диапазон значений.

Алгоритм формирования набора индексов стоп лемм

Вначале определяем максимальный размер группы ключей MaxGroupSize, исходя из условия, что в момент помещения в индекс словопозиций группы ключей мы храним в памяти не менее K кластеров, K > 0, на один ключ. Заданный размер кэша (например, 2 Гб.) делим на (размер кластера $\times$ K) и получаем максимальный размер группы ключей.

Определим: MaxKeys равным MaxGroupSize $\times$ MaxGroupsPerIndex.

Для каждого индекса нужно определить диапазон значений первой компоненты ключа. Диапазон определяется началом диапазона Start и количеством значений в диапазоне Count. Алгоритм похож на расчет групп ключей для индекса, вначале мы инициализируем начальный диапазон, затем в цикле пытаемся его расширить. Если расширение диапазона невозможно, текущий индекс фиксируем и начинаем определять следующий.

Присваиваем Start = 0, Count = 0, TotalKeys = 0, ATotalKeys = 0.

Далее выполняем в цикле, перебираем f от 0 до $C-1$ включительно:

1) $\text{Keys} = Q(f)$.
2) $\text{AKeys} = (\text{Keys} \times \text{Freq}(f) \times 0{,}3)/\text{Avg}$.
3) Вычисляем для индекса $[\text{Start}, \text{Start} + \text{Count} + 1)$ список групп ключей и их количество: TotalGroupsN.
4) Присваиваем $\text{TotalKeysN} = \text{TotalKeys} + \text{Keys}$.
5) Присваиваем $\text{ATotalKeysN} = \text{ATotalKeys} + \text{AKeys}$.
6) Если $\max(\text{TotalKeysN}, \text{ATotalKeysN})$ превышает MaxKeys, или TotalGroupsN превышает MaxGroupsPerIndex, то: фиксируем индекс $[\text{Start}, \text{Start} + \text{Count})$, затем начинаем определять следующий индекс, присваиваем: Start = f, Count = 0, TotalKeys = 0, ATotalKeys = 0.
7) Увеличиваем Count на 1, присваиваем $\text{TotalKeys} = \text{TotalKeys} + \text{Keys}$, $\text{ATotalKeys} = \text{ATotalKeys} + \text{AKeys}$.

По окончании цикла, определяем последний индекс на основании текущего диапазона $[\text{Start}, \text{Start} + \text{Count})$.

Поскольку для тех ключей, первая компонента которых встречается чаще, количество словопозиций будет больше, мы вычисляем значение AKeys, которое превышает Keys для самых часто встречающихся лемм, чтобы получить первые индексы с меньшим числом ключей.

Пример: при значении WsCount = C = 150, MaxGroupsPerIndex = 10, была получена следующая конфигурация индексов:

$0:[0,4]->[0,54][55,149]$,

$1:[5,15]->[5,32][33,60][61,104][105,149]$,

$2:[16,52]->[16,37][38,47][48,56][57,66]$

[67,77][78,90][91,107][108,143][144,149],

$3:[53,149]->[53,80][81,94][95,107]$

[108,121][122,149].

Имеем 4 индекса, в нулевом диапазон значений первой компоненты ключа [0, 4], в первом [5, 15], во втором [16, 52], в последнем [53, 149]. Для каждого индекса перечислены группы ключей, заданные диапазоном значений второй компоненты ключа. Рассмотрим для примера, в индексе 0, определенным диапазоном [0, 4], первая группа ключей [0, 54], включает в себя ключи: $(0, x, y)$, $0 \le x \le 54$, $x \le y$; $(1, x, y)$, $1 \le x \le 54$, $x \le y$;

$(2, x, y)$, $2 \le x \le 54$, $x \le y$; ...; $(4, x, y)$, $4 \le x \le 54$, $x \le y$.

Обработка поискового запроса

Обработка запроса осуществляется в соответствии с [8] за исключением того, что индекс последовательностей стоп лемм заменен на индекс трехкомпонентных ключей.

Предварительные результаты экспериментов

Эксперименты поиска проводились в соответствии с [8]. Виды индексов для экспериментов поиска:

Idx1. Обычный индекс. Содержит для каждой леммы, включая стоп леммы, список всех ее словопозиций. Не содержит NSW-записей (см. [8]).

Idx2. Индекс с дополнительными индексами, структура дана в [8], в разделе «Используемые виды индекса». Стоп лемм 700 (524 ru, 176 en), часто используемых лемм 2100 (1648 ru, 452 en).

Idx3. Определен в [8], в текущих экспериментах не используется.

Idx4. Индекс с дополнительными индексами, все параметры, как для Idx2, но индекс последовательностей стоп лемм заменен на индекс трехкомпонентных ключей (эти индексы применяются для выполнения запросов, состоящих только из стоп лемм). Стоп лемм 700. MaxGroupsPerIndex = 30.

Цель экспериментов поиска: проверить, как замена индекса последовательностей стоп лемм на индекс трехкомпонентных ключей повлияет на производительность.

Размеры файлов словопозиций: размер файлов словопозиций трехкомпонентного индекса для Idx4, 622 Гб. Это существенно превышает размеры индекса последовательностей стоп лемм для Idx2, 95,4 Гб. (см. [8]).

Эксперимент 1

Выполнено 4500 запросов. Исходные документы, выполненные запросы и другие параметры, соответствуют эксперименту 1 из [8], за исключением замены Idx3 на Idx4.

Среднее число обработанных словопозиций на один запрос поиска:
Idx1: 171 млн., Idx2: 733 тыс., Idx4: 818 тыс.

Эксперимент 2.

Запросы включали в себя только слова со стоп леммами. Всего выполнено 330 запросов. Эти 330 запросов являются подмножеством запросов из Эксперимента 1. Среднее число обработанных словопозиций:
Idx1: 209 млн., Idx2: 53 тыс., Idx4: 1,1 млн.

Нас в первую очередь интересует сравнение обычного индекса Idx1 с другими индексами. Для Idx4 среднее число обработанных при поиске словопозиций меньше примерно в 190 раз, по сравнению с Idx1, что повлияло на время эксперимента, 3 час 17 мин, для Idx1 и 2 мин 44 сек для Idx4. Для Idx2 число словопозиций еще меньше, но Idx2, в случае, если все леммы запроса являются стоп леммами, позволяет искать только такие документы, где слова запроса располагаются подряд.

Заключение

Разработана структура данных и реализованы индексы с трехкомпонентным ключом. Разработана схема отображения трехкомпонентного ключа в ячейку массива словаря ключей. Создан тестовый индекс. Осуществлены эксперименты поиска. Результаты экспериментов позволяют утверждать, что разработанный вариант индекса решает задачу поиска с учетом близости для запросов, состоящих из любых лемм, включая стоп леммы, более эффективно, чем обычные индексы.

Список литературы

1) Zobel J., Moffat A.; Inverted files for text search engines, ACM Computing Surveys, 2006, 38(2), Article 6.
2) 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.
3) Brown E. W., Callan J. P., Croft W. B.; Fast incremental indexing for full-text information retrieval, In Proceedings of the International Conference on Very Large Databases, 1994, 192–202, Santiago, Chile.
4) George Kingsley Zipf; Relative frequency as a determinant of phonetic change. Harvard Studies in Classical Philology, 1929, Vol 40, 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) Веретенников А.Б. Эффективный полнотекстовый поиск с использованием дополнительных индексов часто встречающихся слов, Системы управления и информационные технологии, 2016, №4(66), 52–60.
9) 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, 215–221.
10) Williams H. E., Zobel J., Bahle D.; Fast phrase querying with combined indexes, ACM TOIS, 2004, №4(22), 573–594.