УДК 519.683.5

Теоретические исследования производительности CLB-деревьев
Theoretical research of the performance of CLB-Trees

А. Б. Веретенников
A. B. Veretennikov

Уральский государственный университет им. А. М. Горького
Ural State University

Полнотекстовый поиск, CLB-дерево, поисковые системы, инвертированные файлы.

Рассматриваются задачи поиска в большом объеме текстов и их решение с использованием ранее предложенной автором новой структуры данных – CLB-дерева. Приведены теоретические оценки производительности и их обоснование.

Full text search, CLB-Tree, search engines, inverted files.

The problems of search in the large text arrays are considered and solved using the new data structure proposed earlier by author, CLB-Tree. Theoretical estimations of efficiency and proofs are given.

Введение

Для решения задач поиска слов или фраз часто используются инвертированные файлы [1].

В [2-5] автором данной работы была описана новая структура данных – CLB-дерево, предназначенная для поиска в большом объеме электронных документов, написанных на естественном языке.

CLB-дерево не уступает инвертированным файлам и их аналогам по скорости поиска, но CLB-дерево гораздо быстрее создается и быстрее обновляется. Чтобы добавить данные в уже существующий инвертированный файл требуется переписать весь индекс. В отличие от этого, для добавления данных в CLB-дерево требуется внести изменения в индекс, по объему сравнимые с объемом только новых данных, а текущий размер индекса практически не влияет на скорость его обновления.

В [2-4] автором были даны: описание структуры данных, алгоритмов работы с ней и результаты экспериментов. Также были сформулированы несколько теоретических оценок, без подробных доказательств. В [5] были представлены метод, позволяющий значительно ускорить обновление

CLB-дерева, и результаты экспериментов многократного добавления в CLB-дерево данных различного размера.

Подробные результаты экспериментов создания и обновления CLB-дерева включены в [5, 6].

Результаты исследований были включены автором в [7] и легли в основу разрабатываемого поискового движка CLB Search [8, 9].

В данной работе автор подводит некоторый итог по данной теме приводя ряд важных теоретических оценок и их обоснование.

Автор следует той терминологии, которая была дана в [4]. Данная работа является по сути продолжением [4] и автор постоянно опирается на описанные в [4] алгоритмы.

Оценка количества дисковых операций при создании индекса.

Для создания CLB-дерева используется морфологический анализатор. Для каждой словоформы, известной анализатору, возвращается одна или несколько базовых форм. Словоформы, хранящиеся в словаре анализатора, составляют более 90% всех словоформ в типичных документах. Автор использует морфологические анализаторы [10-12].

Под известными словоформами будем понимать словоформы, входящие в словарь анализатора, остальные назовем неизвестными. Будем считать, что неизвестная словоформа является своей базовой формой. Таким образом, каждой словоформе сопоставляется набор ее базовых форм.

Под вставкой слова в CLB-дерево автор понимает добавление информации о вхождении данного слова в тексте в CLB-дерево, эта информация имеет определенную структуру и далее называется записью о вхождении. Запись о вхождении может включать, например, идентификатор документа и позицию слова в документе. Размер записи о вхождении обычно 2-3 байта, при применении некоторых методов кодирования. Пусть $Max1$ – максимальный размер записи о вхождении, далее будем считать это число константой.

Создаваемый индекс состоит из 3-х основных частей: файл с кластерами, B-дерево [13] и файл с таблицей известных слов. Кластер – это блок фиксированного размера. Каждой базовой форме слова соответствует цепочка кластеров, в которой сохранена информация о вхождениях данной формы слова в документах. Цепочка кластеров описывается структурой данных, далее называемой указателем. B-дерево сохраняет в себе неизвестные слова и указатели для их цепочек кластеров, файл с таблицей известных слов хранит в себе указатели для базовых форм известных слов.

Введем обозначения:

Пусть d – доля известных слов в текстах, R – текущее количество слов в CLB-дереве.

K_FORMS – количество базовых форм в словаре (для текущего словаря 200 тыс.).

K – размер кластера, H – размер страницы B-дерева.

Следует учесть, что в CLB-дерево мы добавляем информацию о вхождениях для каждой базовой формы слова. Некоторые слова имеют несколько базовых форм. Если у нас было изначально N слов в тексте, то суммарное количество их базовых форм равно $N' = N \times KF$, где $KF > 1$. Для используемых словарей русского языка $KF \approx 0.25$, для конкретного языка данный коэффициент является константой и поэтому, по мнению автора, не влияет на асимптотику.

При создании индекса основным критерием для оценки производительности является число обращений к внешней памяти. Заметим также, что при создании инвертированных файлов большая нагрузка ложится и на процессор, т. к. как правило требуется сортировка данных, вычислительная сложность которой обычно $n \cdot \log_2 n$, где n – количество сортируемых записей. При создании CLB-дерева подобная сортировка не требуется, т. е. CLB-дерево – менее требовательная структура данных с точки зрения затрат ресурсов центрального процессора.

Теорема 1. Вставка N записей о вхождении в CLB-дерево потребует $O(d \cdot N/K + (1 - d) \cdot N \cdot (1 + \log_H((1 - d) \cdot (R + N))))$ обращений к внешней памяти.

Формулировка данного утверждения была дана в [4]. Доказательство теоремы следует из описанного в [4] алгоритма создания индекса.

Первое слагаемое определяется вставкой записей известных слов, второе – вставкой записей неизвестных слов. Значение $h = O(\log_H((1 - d) \cdot (R + N)))$ – это высота B-дерева. Вставка каждого неизвестного слова требует $O(1)$ обращений к внешней памяти для записи в цепочку кластеров и $O(h)$ для записи в B-дерево. Для вставки известных слов B-дерево не используется, а также организован специальный кеш. Эта теорема показывает, что в CLB-дерево можно легко добавлять новые данные.

Для создания индекса в него добавляются записи о вхождении вначале для известных слов, затем для неизвестных слов. Подробно алгоритм описан в [4].

Процесс добавления записи о вхождении слова включает в себя

1. Получение указателя на цепочку кластеров для данной базовой формы слова.
2. Добавление данных в цепочку кластеров.
3. Сохранение обновленного указателя.

Для неизвестных слов пункты 1 и 3 требуют O(h) обращений к внешней памяти, для известных слов не требуют обращений к внешней памяти.

Рассмотрим более подробно пункт 2.

Случай А. Пока суммарный размер информации о вхождениях достаточно мал, данные сохраняются в указателе.

Указатель – некоторая структура фиксированного размера, имеющая ряд полей. Часть из полей может использоваться для организации цепочек кластеров, например номер первого кластера цепочки, номер последнего кластера цепочки, в случае Б и В. Одновременно, те же поля могут использоваться для хранения данных в них в случае А. Структура указателя более подробно описана в [4]. Случай А следует рассматривать как оптимизацию, при отказе от которой конечная оценка количества дисковых операций не изменится. Поскольку здесь изменяется только указатель, то дисковых операций, кроме тех, которые уже учтены в пунктах 1 и 3, не производится.

Случай Б. Пока суммарный размер информации достаточно мал, данные сохраняются в кластерах, разделенных на части.

Выберем число *PARTS*, являющееся степенью 2 и введем $\log_2 PARTS$ списков кластеров. Список с номером i хранит в себе кластеры, разбитые на 2^{i+1} частей (номер 0 соответствует двум частям, $\log_2 PARTS - 1$ соответствует *PARTS* частям). В конце кластера выделяем область, назовем ее таблицей частей, для хранения информации о том, какие части кластера заполнены. Размер этой области равен = *<количество частей в кластере> /8* байт, с округлением в большую сторону (чтобы хранить по одному биту для каждой части*)*. Т. е. для кластеров, разделенных на 2 части требуется не более 1 байта, для кластеров, разделенных на *PARTS* частей – не более *PARTS/8* + 1. Остальная область кластера делится на равные части. При этом для заданной базовой формы слова используется только одна часть кластера, остальные части могут использоваться для других базовых форм других слов. Как только данные перестают помещаться в часть кластера, разбитого на две части (т. е. их размер больше чем (K – 1) / 2), мы переходим к случаю В.

Заметим, что в [4] размер таблицы частей в кластере всегда был постоянным, в не зависимости от количества частей в кластере, т. е. *PARTS/8*, с округлением в верхнюю сторону. Описанная выше оптимизация очевидна.

Случай В. Данные сохраняются в цепочке из нескольких кластеров. Как только последний кластер полностью заполняется, в цепочку добавляется новый кластер. Для предотвращения фрагментации вводится число *BLOCK_L*. Если количество кластеров в цепочке C является степенью 2, и меньше чем *BLOCK_L*, то при добавлении нового кластера в цепочку осуществляется выделение на диске области памяти из $2 \times C$ последова-

тельно расположенных кластеров и копирование текущих данных в первую половину новой области, т. е. *C* операций записи. Остальные кластеры считаются зарезервированными для данной цепочки кластеров.

Если в дальнейшем *C* кратно *BLOCK_L* – то при добавлении в цепочку нового кластера сразу выделяется область из *BLOCK_L* кластеров, запись осуществляется в первый кластер новой области, остальные кластеры области резервируются для данной цепочки.

Если *C* – другое число, то у мы используем зарезервированные ранее для данной цепочки кластеры. Описанный метод подробно описан в [2, 4].

Рассмотрим процесс добавления данных известных слов. При добавлении в индекс известного слова вначале осуществляется получение указателя на цепочку кластеров. Этот указатель хранится в таблице в оперативной памяти и его получение не требует обращений к внешней памяти. При записи известных слов следует учитывать, что для каждой базовой формы известного слова последний кластер цепочки хранится в оперативной памяти.

Рассмотрим **случай А**. В данном случае не требуется обращений к внешней памяти при записи данных об известных словах, т. к. данные указатели целиком хранятся в таблице в оперативной памяти.

Рассмотрим **случай Б** . Последние кластеры цепочек хранятся в оперативной памяти. Это означает, что для случая Б кластер хранится в оперативной памяти как первый и последний кластер для цепочки кластеров по крайней мере для одной базовой формы слова. Таким образом, данные кластеры хранятся в оперативной памяти на протяжении всего процесса добавления данных и сохраняются только после завершения этого процесса. Количество кластеров, разделенных на части не более *K_FORMS*, т. к. по крайней мере данные об одной базовой форме слова должны быть сохранены в каждом таком кластере. Следовательно операций записи для таких кластеров не более *K_FORMS*.

Рассмотрим **случай В**. Последний кластер каждой цепочки хранится в оперативной памяти и сохраняется во внешнюю память в двух подслучаях: во первых, если данный кластер заполнен и мы добавляем в цепочку новый кластер, во вторых, если мы завершили добавление данных и сохраняем все хранящиеся в оперативной памяти кластеры во внешнюю память.

Количество операций во втором подслучае не больше *K_FORMS*. Для первого подслучая количество операций $(N \times Max1) / K$, где $N \times Max1$ – максимальный объем добавляемых данных.

Также следует учесть, что если количество кластеров в цепочке *C* является степенью 2, и меньше чем *BLOCK_L*, то при добавлении нового кластера в цепочку осуществляется выделение на области памяти из $2 \times C$ последовательно расположенных кластеров и копирование текущих данных

в первую половину новой области, т. е. C операций записи, как это описано в [4]. Пусть у нас в текущем блоке R кластеров. Количество операций перезаписи составляло $1 + 2 + 4 + 8 + ... + R / 2 < R$. Таким образом, процедуры переноса данных в новую область увеличивают общее число обращений к внешней памяти не более чем в два раза. Если количество кластеров в цепочке больше чем *BLOCK_L*, то подобные переносы данных больше не производятся.

Без ограничения общности будем считать, что *K_FORMS* – это константа, значительно меньшая чем число N, как это всегда бывает в реальных приложениях. Соответственно, число обращений к внешней памяти при записи известных слов составляет $O(d\cdot(K\_FORMS + N/K)) = O(d\cdot(N/K))$, т. к. *K_FORMS* можно пренебречь.

Рассмотрим добавление неизвестных слов в индекс. При получении указателя на цепочку кластеров для неизвестного слова требуется найти данное слово в B-дереве. Это требует $O(h)$ обращений к внешней памяти. Процедура сохранения обновленного указателя в B-дереве требует также $O(h)$ обращений к внешней памяти. При добавлении данных об одном вхождении неизвестного слова требуется одна операция записи во внешнюю память. Если мы учитываем переносы данных, как описано в пункте B, то они увеличивают количество операций не более чем в 2 раза. Соответственно получаем оценку $O((1 - d)\cdot N\cdot(1+ \log_H((1 - d)\cdot(R+N))))$.

Теорема доказана.

Следует отметить, что на практике мы можем получить более лучшие результаты при добавлении неизвестных слов в индекс используя стандартные схемы организации кэша.

Оценка количества дисковых операций при поиске.

Теорема 2. Поиск набора слов из N-элементов в CLB-дереве потребует $O(N\cdot(\log_H((1 - d)\cdot R) + occ/K))$ обращений к внешней памяти (здесь occ – количество вхождений данных слов в текстах).

Здесь $h=O(\log_H((1 - d)\cdot R))$ – высота B-дерева и $O(N\cdot\log_H((1 - d)\cdot R))$ – столько обращений к внешней памяти требуется для чтения информации из B-дерева для неизвестных слов, которые могут быть среди искомых слов. Это слагаемое мало и на практике его можно не учитывать. Основное время занимает $O(occ/K)$ обращений к внешней памяти, для чтения кластеров, содержащих информацию обо всех вхождениях словоформы в текстах.

При этом при чтении данных можно использовать не блоки размера K, а блоки размера $M = BLOCK\_L \cdot K$ – это число соответствует размеру областей подряд располагающихся кластеров в цепочке кластеров. Тогда для

оценки поиска мы получим, что поиск потребует $O(N \cdot (h + occ/M))$ дисковых операций. Где M – на самом деле, как указано в [4], является легко настраиваемым параметром, равным обычно нескольким десяткам мегабайт.

Т. е, как описано в пункте В, кластеры по сути практически располагаются последовательно. Следовательно, поиск с помощью CLB-дерева имеет такую же эффективность, как и с помощью инвертированных файлов.

Оценка объема индекса.

Теорема 3. Размер файла с кластерами для CLB-дерева не более чем $4 \cdot S \cdot d + 4.5 \cdot S \cdot (1 - d) + P$, где S – суммарный объем сохраненных в индексе данных, а P — некоторая константа, не зависящая от S.

Заметим, что в прилагаемом далее доказательстве в случаях освобождения используемых кластеров они далее не используются. Однако на практике кластер, освобожденный в рамках построения одной цепочки, почти всегда используется для другой цепочки, в результате размер файла менее $2 \cdot S$.

Доказательство

Данная теорема позволяет оценить максимальный размер файла с кластерами для CLB-дерева.

Оценим сколько занимают кластеры для известных слов.

Рассмотрим **случай А**. Кластеры не используются.

Рассмотрим **случай Б**. Количество кластеров разделенных на части для известных слов не более *K_FORMS*.

Рассмотрим **случай В**. Если цепочка кластеров состоит из одного кластера, то он заполнен как минимум на половину, т. к. иначе использовались бы кластеры, разделенные на части. Размер таблицы частей в кластере разделенном на две части равен одному байту, т. е. размер части в таком кластере $(K - 1) / 2$. Соответственно, поскольку данные не помещаются в одну часть такого кластера, то их размер больше или равен $K / 2$.

Рассмотрим процесс переноса данных из блока в блок. Если количество кластеров в цепочке C является степенью 2, и меньше чем *BLOCK_L*, то при добавлении нового кластера в цепочку осуществляется выделение на диске области памяти, равной $2 \times C$ и копирование текущих данных в первую половину новой области. В итоге половина кластеров в блоке автоматически становится полностью заполненной. Если равно *BLOCK_L*, то мы выделяем для данной цепочки сразу *BLOCK_L* кластеров, однако в данном случае у нас уже есть *BLOCK_L* заполненных кластеров, т. е. половина кластеров цепочки заполнена.

Следовательно, в худшем случае размер области, используемой для хранения цепочек кластеров известных слов, не более чем в два раза больше суммарного размера добавленных данных.

Оценим объем кластеров, которые были освобождены при переносе данных, которые осуществляются если $C < BLOCK\_L$, C – степень *2*. Это число равно $1 + 2 + \ldots + C \leq 2 \times C$. Т. е. в худшем случае кроме кластеров, входящих в цепочку кластеров, мы имеем еще и освобожденные кластеры и суммарный размер не более чем в 4 раза больше размера данных.

Рассмотрим хранение данных неизвестных слов.

Рассмотрим **случай А**. Кластеры не используются.

Рассмотрим **случай Б**. В худшем случае у нас каждое неизвестное слово встретится один раз, при этом информация об его вхождении займет часть в кластере разделенном на части. За счет выбора параметра – максимального количества частей в кластере, мы можем сделать размер данной части сколь угодно малым. Например 1 байт. Размер таблицы частей при этом равен $K/8$.

Данные переносятся из одного кластера в другой, разделенный на части большего размера, если не помещаются в одну часть текущего кластера. Следовательно, если часть кластера используется, то она заполнена более чем на половину, иначе данные хранились бы в кластере, разделенном на более мелкие части. Новый кластер разделенный на части добавляется, только если все кластеры с такими размерами частей заполнены.

В кластерах разделенных на части хранятся данные для некоторых базовых форм слов. Рассмотрим одну базовую форму. Информация о ее вхождениях хранится в одной части некоторого кластера.

Пусть размер данной части X. Для простоты будем считать, что если мы переносим данные из одной части в часть более большего размера, то старую часть больше не используем – это худший случай. Для данной базовой формы часть размера X заполнена на половину. Объем свободного места в данной части не более $X / 2$. При этом ранее были использованы части размером $X / 2$, $X / 4$, …, $X / 8$, …, 1, и все они теперь свободны. Т. е. объем использованных частей в худшем случае равен $X / 2 + X / 4 + X / 8 + \ldots + 1 \leq X$. Суммарный объем в худшем случае соответственно $2X$ – в 4 раза больше добавленных данных. Данный расчет не учитывает таблицу частей, которая в худшем случае равна 1/8 остальной части, при учете этого получаем оценку $4S + (4S)/8 = 4.5S$. Случай, когда ранее данные для базовой формы хранились в кластере, разделенном на части, но были перенесены в целый кластер, рассматривается аналогично.

Случай В рассматривается для неизвестных слов аналогично рассмотренному случаю В для известных слов.

Доказательство завершено.

Таким образом мы имеем теоретические оценки затрат ресурсов на добавление данных в CLB-дерево, поиск данных в CLB-дереве и хранение CLB-дерева.

Примечания

Название структуры CLB-дерево обусловлено исторически [14, 15] тем, что при создании CLB-дерева для используется B-дерево, которое является по сути ассоциативным массивом. B-дерево было выбрано в связи с желанием не зависеть от объемов оперативной памяти Автор предполагает, что при определенных размерах исходных данных вместо B-дерева можно использовать другие структуры данных, такие как хеш-таблицы и АВЛ-деревья, которые обычно хранятся в оперативной памяти.

Одним из вопросов для повышения производительности является уменьшение количества неизвестных слов. В описанном процессе индекса в качестве неизвестных слов берутся все слова, не входящие в словарь морфологического анализатора. При этом обрабатываемые документы могут быть в различных форматах, файл может быть как простым текстом, так и включать в себя дополнительную информацию, такую как информация о форматировании, бинарные заголовки и т. д. Если формат файла известен, то эта информация может быть просто пропущена. Если нет, то она может быть воспринята как набор неизвестных слов и также включена в индекс. Исключение этой информации выходит за рамки данной работы. Также уменьшить количество неизвестных слов можно за счет автоматического включения часто используемых неизвестных слов в словарь морфологического анализатора, выделяя некоторую базовую форму слова каким либо образом.

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. 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.

3. Веретенников А. Б. Эффективное создание текстовых индексов. Проблемы теоретической и прикладной математики: Труды 39-й Всероссийской молодежной конференции. Екатеринбург: УрО РАН, 2008. с. 348-350.

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

5. 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.

6. Веретенников, А. Б. Сравнение эффективности CLB-дерева в 32-битных и 64-битных архитектурах. Материалы межвузовской научной конференции по проблемам информатики «СПИСОК 2009». Екатеринбург. 2009. с. 7-13.

7. 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.

8. Веретенников А. Б. О платформе для электронной текстовой библиотеки. Материалы конференции «Математическое моделирование, численные методы и комплексы программ». Екатеринбург: Изд-во Урал. Ун-та, 2010. с. 30-34.

9. CLB Search, www.veretennikov.org.

10. Лукач Ю. С. Быстрый морфологический анализ флективных языков. Международная алгебраическая конференция: К 100-летию со дня рождения П. Г. Конторовича и 70-летию Л. Н. Шеврина. Тез. докл. Екатеринбург. Изд-во Урал. ун-та, 2005, с. 182-183.

11. Сокирко А. В. Семантические словари в автоматической обработке текста (по материалам системы ДИАЛИНГ). Диссертация на соискание ученой степени кандидата технических наук. Москва, 2001.

12. Автоматическая обработка текста, www.aot.ru.

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

14. Веретенников А. Б., Лукач Ю. С. CLB-деревья: новый способ индексации больших массивов текстов. Международная алгебраическая конференция: К 100-летию со дня рождения П. Г. Конторовича и 70-летию Л. Н. Шеврина. Тез. докл. Екатеринбург: Изд-во Урал. ун-та, 2005, с. 173-175.

15. Веретенников А. Б., Лукач Ю. С. Еще один способ индексации больших массивов текстов. Известия Уральского государственного университета. Серия «Компьютерные науки», 2006. №43. с. 103-122.