О МЕТОДЕ ОПТИМИЗАЦИИ СОЗДАНИЯ CLB-ДЕРЕВА

Веретенников А.Б.[1]
e-mail: alexander@veretennikov.ru

Для поиска в больших массивах текстовых документов применяются инвертированные файлы [1] и их аналоги. После создания инвертированного файла его сложно обновлять, как правило для добавления данных в индекс его требуется полностью переписать.

В [2] автором была предложена идея новой структуры данных, CLB-дерева. Одним из основных достоинств структуры данных является возможность быстрого добавления новых данных.

В [3, 6] были представлены эффективные алгоритмы создания CLB-дерева и описана структура CLB-дерева и ее основные возможности. В [6, 7] были представлены результаты экспериментов создания CLB-дерева, поиска в нем и была показана эффективность CLB-дерева в сравнении с другими структурами данных. При создании CLB-дерева используется морфологический анализатор. В данной работе описывается ряд важных оптимизаций, позволивших существенно ускорить обновление индекса, и представлены результаты экспериментов. Следует сказать, что название структуры данных, CLB-дерево, в основном обусловлено исторически.

Слова, входящие в словарь морфологического анализатора, составляют 80-90% от общего количества слов в обычных тестах (например, художественная литература, новости). Словарь анализатора позволяет по словоформе получить набор ее базовых форм. Для большинства словоформ существует только одна базовая форма, однако для многих словоформ существует несколько базовых форм. Если словоформа не входит в словарь анализатора – ее базовой формой считаем саму словоформу.

В CLB дереве для каждой базовой формы слова создается цепочка связанных блоков, в которых сохраняется информация обо всех вхождениях данной базовой формы в текстах, например набор пар, состоящих из идентификатора документа и позиции слова в документе. Блок назовем кластером, обозначим его размер как K. При

[1] http://cs.usu.edu.ru/home/abv/

этом, в зависимости от объема информации для данной базовой формы, применяются следующие подходы, подробно описанные в [3]:

P1 Объем информации больше или равен размеру кластера.

Зафиксируем некоторое число c и соответствующее ему число $m = 2^c$. Организуем хранение данных таким образом, чтобы список блоков был разбит на группы, и все блоки, входящие в заданную группу были расположены в файле последовательно. Например, у нас есть 25 блоков и $m = 8$. Разбиваем 25 блоков на группы следующих размеров 8, 8, 8, 1. Т. е. весь список блоков разделяется на группы по m блоков, за исключением последней группы, в которой может быть меньше чем m блоков, т. к. длина списка может не делится нацело на m.

P2 Объем информации меньше размера кластера.

Для эффективного заполнения блоков введем новый тип блоков. Пусть некоторые блоки разделяются на части равного размера. Информация для базовой формы сохраняется в одной из частей блока. Если блок делится на части, то в нем же сохраняется информация о том, какие из его частей заполнены, а какие свободны, $PARTS\_TABLE\_SIZE$ – размер таблицы, в которой сохраняется эта информация. Вводим параметр – минимальный размер части, например 4 байта. Пусть c – такое число степени 2, что $(K - PARTS\_TABLE\_SIZE)/c$ – меньше или равно 4 байта. Это кластеры с максимально возможным количеством частей. Кроме них используются кластеры разделенные на $c/2$ частей, $c/4$, ..., 2 частей. Когда данные слова перестают помещаться в кластер разделенный на x частей, они переносятся в кластер разделенный на $x/2$ частей. Как только суммарный объем информации для заданного слова становится больше, чем размер части у кластера, разделенного на 2 части, информация перемещается в новый пустой блок и далее применяется P1.

Для поддержки быстрого обновления индекса предлагается:

O1 Последние кластеры цепочек кластеров для каждой базовой формы должны хранится в одном месте.

O2 В одной цепочке кластеров можно сохранять информацию о нескольких базовых формах. Предложено в [6] для слов, не входящих в словарь анализатора, применим для всех слов.

O3 Оптимизация кеширования.

Для обеспечения O1 мы введем новый тип кластеров – назовем их кластерами первого уровня. Для их использования введем список номеров свободных кластеров $FIRST\_FREE$. Процедура получения кластера первого уровня для его последующего использования:

1 Если в списке $FIRST\_FREE$ есть элемент, удаляем его из списка и возвращаем его в качестве результата.

2 Если список $FIRST\_FREE$ пуст – выделяем в конце файла блок заданного размера BS (в экспериментах было взято 8 Мб). В нем BS/K элементов. Все выделенные кластеры помещаем в $FIRST\_FREE$ и переходим на шаг 1.

Таким образом, данные кластеры хранятся в виде групп последовательно расположенных кластеров. Эти кластеры мы будем использовать для хранения в них последних кластеров для каждой цепочки кластеров.

Соответственно, когда мы обновляем индекс, мы быстро считываем эти кластеры в кеш, за счет их последовательного хранения на диске.

Заметим, что все кластеры P2 хранятся в виде кластеров первого уровня, т. к. в они являются последними (и единственными) кластерами цепочки кластеров. В P1 требуется ввести изменение. Кроме указанного разделения цепочки на группы, последний блок всегда хранится отдельно от описанной структуры групп.

Оптимизация кеширования заключается в следующем.

Будем использовать список $CACHE$, в котором хранятся записи, каждая из которых содержит номер кластера и данные кластера.

Помещение кластера в кеш

1. Берем из $CACHE$ последний элемент.

2. Сохраняем данные взятого элемента.

3. Присваиваем элементу новый номер кластера и помещаем его в начало списка.

Как только некий кластер становится заполненным – мы перемещаем его запись в конец списка. При обращении к кластеру в кеше мы поднимаем его по списку на 1 элемент. Таким образом, если размер $CACHE$ – равен количеству различных базовых форм, то для каждой базовой формы в кеше будет хранится последний кластер ее цепочки кластеров.

Ранее размер кеша выбирали исходя из того, чтобы в кеше хранился один кластер для каждой базовой формы слова. Теперь увеличим кеш. За счет увеличения размера кеша, для многих базовых форм в кеше сохраняется несколько последних кластеров. При удалении из кеша кластера одновременно будем сохранять и рядом с ним располагающиеся кластеры. В результате за счет того, что запись данных будет осуществляться последовательно, получим ускорение. При сохранении кеша, когда запись в индекс завершается, кластеры сохраняем в порядке их номера.

Ранее в [7] использовался размер кластера 256 Кб, при уменьшении размера кластера мы получали уменьшение скорости создания индекса. В приведенных далее экспериментах используется размер кластера 16 Кб, при этом скорость создания индекса изменилась не существенно. В приведенных далее экспериментах размер кеша выбирался таким образом, чтобы для каждой базовой формы хранилось в нем 15 последних кластеров. Уменьшение размера кластера позволяет ускорить обновление индекса.

Далее приведены результаты экспериментов. Структура эксперимента:

1. Создание индекса для базового массива документов.
2. Добавление в индекс небольшого файла.
3. Добавление в индекс 1, 2, 3, ..., 9 Мб данных.
4. Добавление в индекс 10, 20, 30, ..., 90 Мб данных.
5. Добавление в индекс 100, 200, 300, ..., 900 Мб данных.
6. Добавление в индекс 1000, 2000, 3000, ..., 10000 Мб данных.

При добавлении в индекс каждой порции данных после добавления сохраняется весь кеш и программа создания индекса завершает работу, затем она снова запускается и добавляет следующую порцию данных. При работе с файлом, содержащим кластеры, кеширование операционной системы отключается.

Таким образом мы проверяем скорость обновления индекса для данных самого разного размера.

Эксперименты проводились на следующей конфигурации:

Процессор: Intel Core 2 Duo E6700, 2.66 GHz, кэш: L1 Data – 2 x 32 Кб, L1 inst. 2 x 32 Кб, L2 - 4096 Кб. Оперативная память: 8 Гб, DDR2 800. Жесткий диск: Seagate Barracuda 7200.11, 7200 RPM, кэш 16 Мб., объем 1 Гб. FSB 1066 MHz. ОС: Microsoft Windows 2008 Enterprise x64 Edition with Service Pack 2.

Далее в таблице, размер – размер текста, добавляемый в индекс, общее время – суммарное время, составленное из время чтения файлов и времени записи индекса. В последних двух колонках указан суммарный объем прочитанных и записанных кластеров. Все обрабатываемые файлы представляли собой обычный текст.

Размер	Общее время, ч.:м.:с.	Время записи индекса	Чтение кластеров	Запись кластеров
87.08 Гб	04:23:28	01:51:10	7.14 Гб	57.45 Гб
1.41 Кб	00:01:42	00:01:36	4.71 Гб	4.46 Мб
1.08 Мб	00:02:25	00:02:19	4.71 Гб	320.51 Мб
2.17 Мб	00:02:35	00:02:29	4.71 Гб	457.46 Мб
3.09 Мб	00:02:47	00:02:40	4.71 Гб	574.32 Мб
4.03 Мб	00:02:51	00:02:44	4.71 Гб	667.21 Мб
5.03 Мб	00:02:57	00:02:50	4.71 Гб	728.78 Мб
6.25 Мб	00:03:02	00:02:55	4.71 Гб	818.37 Мб
7.31 Мб	00:03:07	00:03:00	4.71 Гб	846.29 Мб
8.21 Мб	00:03:06	00:02:58	4.71 Гб	858.79 Мб
9.19 Мб	00:03:13	00:03:05	4.71 Гб	912.27 Мб
10.02 Мб	00:03:14	00:03:06	4.71 Гб	947.84 Мб
10.02 Мб	00:03:13	00:03:05	4.71 Гб	948.30 Мб

20.03 Мб	00:03:31	00:03:22	4.71 Гб	1.14 Гб
30.37 Мб	00:03:42	00:03:33	4.72 Гб	1.26 Гб
40.19 Мб	00:04:02	00:03:52	4.72 Гб	1.34 Гб
50.60 Мб	00:04:07	00:03:55	4.74 Гб	1.39 Гб
60.17 Мб	00:04:16	00:04:04	4.73 Гб	1.41 Гб
70.26 Мб	00:04:16	00:04:03	4.74 Гб	1.42 Гб
80.46 Мб	00:04:25	00:04:11	4.74 Гб	1.45 Гб
90.01 Мб	00:04:32	00:04:17	4.75 Гб	1.49 Гб
100.09 Мб	00:04:31	00:04:15	4.76 Гб	1.47 Гб
200.82 Мб	00:10:43	00:10:15	4.85 Гб	1.77 Гб
300.06 Мб	00:11:21	00:10:43	4.85 Гб	1.80 Гб
401.03 Мб	00:11:53	00:11:10	4.90 Гб	1.97 Гб
500.51 Мб	00:12:02	00:11:15	4.92 Гб	2.06 Гб
600.45 Мб	00:12:38	00:11:41	5.03 Гб	2.21 Гб
700.62 Мб	00:12:54	00:11:47	5.19 Гб	2.22 Гб
800.02 Мб	00:13:43	00:12:26	5.25 Гб	2.33 Гб
900.16 Мб	00:14:02	00:12:33	5.28 Гб	2.41 Гб
1000.23 Мб	00:14:33	00:12:55	5.35 Гб	2.56 Гб
1.95 Гб	00:19:08	00:15:50	5.70 Гб	3.64 Гб
2.92 Гб	00:23:25	00:18:34	6.02 Гб	4.69 Гб
3.90 Гб	00:27:52	00:21:14	6.41 Гб	5.75 Гб
4.88 Гб	00:31:53	00:23:22	6.76 Гб	6.87 Гб
5.85 Гб	00:36:25	00:26:29	7.32 Гб	7.54 Гб
6.83 Гб	00:40:02	00:28:31	7.70 Гб	8.53 Гб
7.81 Гб	00:44:44	00:31:33	8.18 Гб	9.69 Гб
8.78 Гб	00:48:56	00:34:04	8.29 Гб	10.17 Гб
9.76 Гб	00:52:03	00:35:47	8.89 Гб	11.29 Гб

Результаты экспериментов показывают высокую скорость обновления индекса для различных объемов данных.

В текущей реализации используется B-дерево, в котором сохраняются слова, не входящие в словарь анализатора. Выбрано B-дерево, т. к. нельзя заранее знать сколько различных слов, не входящих в словарь анализатора, в текстах. При этом, реализация B-дерева неэффективна, и сейчас, с точки зрения производительности, B-дерево является узким местом. За счет оптимизации этого компонента приведенные результаты экспериментов могут быть улучшены. Приведенные результаты получены с использованием O2, также

были проведены аналогичные эксперименты без использования O2, при этом скорость обновления индекса уменьшилась, но незначительно, т. е. O1 и O3 имеют большее значение.

Список литературы

[1]. *Prywes, N. S., Gray, H. J.* The organization of a Multilist-type associative memory. IEEE Trans. on Communication and Electronics, 68 (1963), 488-492.

[2]. *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.

[3]. *Веретенников А.Б.* Эффективная индексация текстовых документов с использованием CLB-деревьев. // Системы управления и информационные технологии, 2009, 1.1(35). - С. 134–139.

[4]. *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.

[5]. *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.

[6]. *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.

[7]. *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.