Создание легко обновляемых текстовых индексов

Веретенников А. Б.

Задача

- Поиск слов и фраз в большой текстовой коллекции

Инвертированные файлы

- Часто используются для поиска
- Сложно добавлять новые данные

Инвертированные файлы

Для каждой словоформы сохраняется информация о том, в каких документах и где в документах она встречается

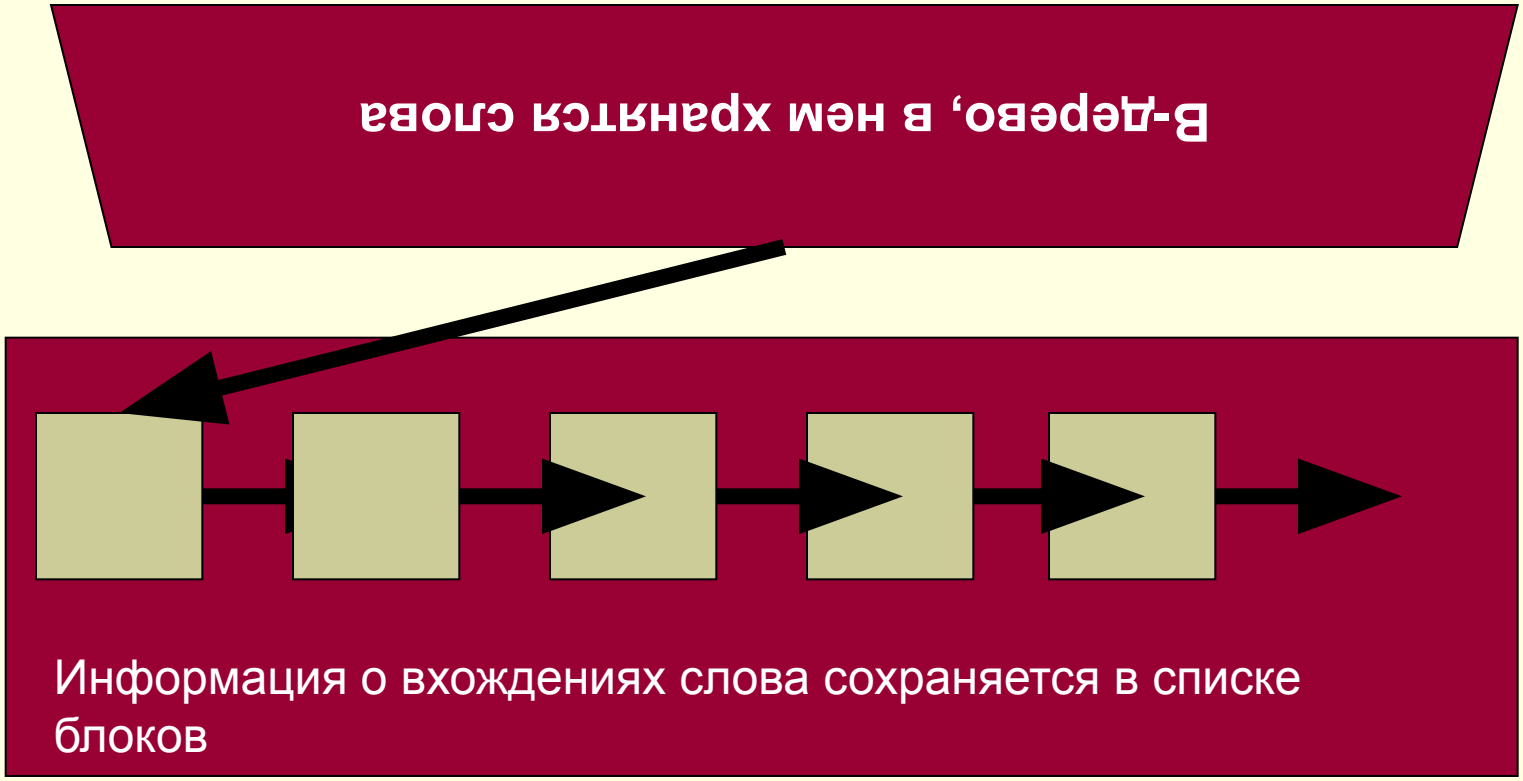
Пример информации о вхождении

1) Номер (ID) файла
2) Позиция словоформы в файле (порядковый номер словоформы, номер предложения, и т. д.)

Задача

Нужно сделать индекс, который бы позволял легко добавлять в него новые данные

CLB-дерево


Информация о вхождениях слова сохраняется в списке блоков
B-дерево, в нем хранятся слова

Морфология

Морфологический анализатор

Для каждой словоформы из словаря выдается набор базовых форм.

Базовых форм ~ 200 тысяч.

Словоформ ~ 4 млн.

Кэширование

Храним в B-дереве не словоформы, а базовые формы. Можем хранить в памяти последний блок для каждой базовой формы.

Плюсы

Можно быстро добавлять новые данные. Информация о новых вхождениях слова добавляется в последний блок списка. Когда он заполняется - создается новый блок.

Минусы

- 1) Фрагментация – блоки могут располагаться в разных местах
- 2) Неэффективное использование дисковой памяти, блоки могут быть слабо заполнены
- 3) Требует много памяти для использования большого размера блока (200 000 x <Размер блока>).

Проблема фрагментации

Пусть в списке блоков k блоков.

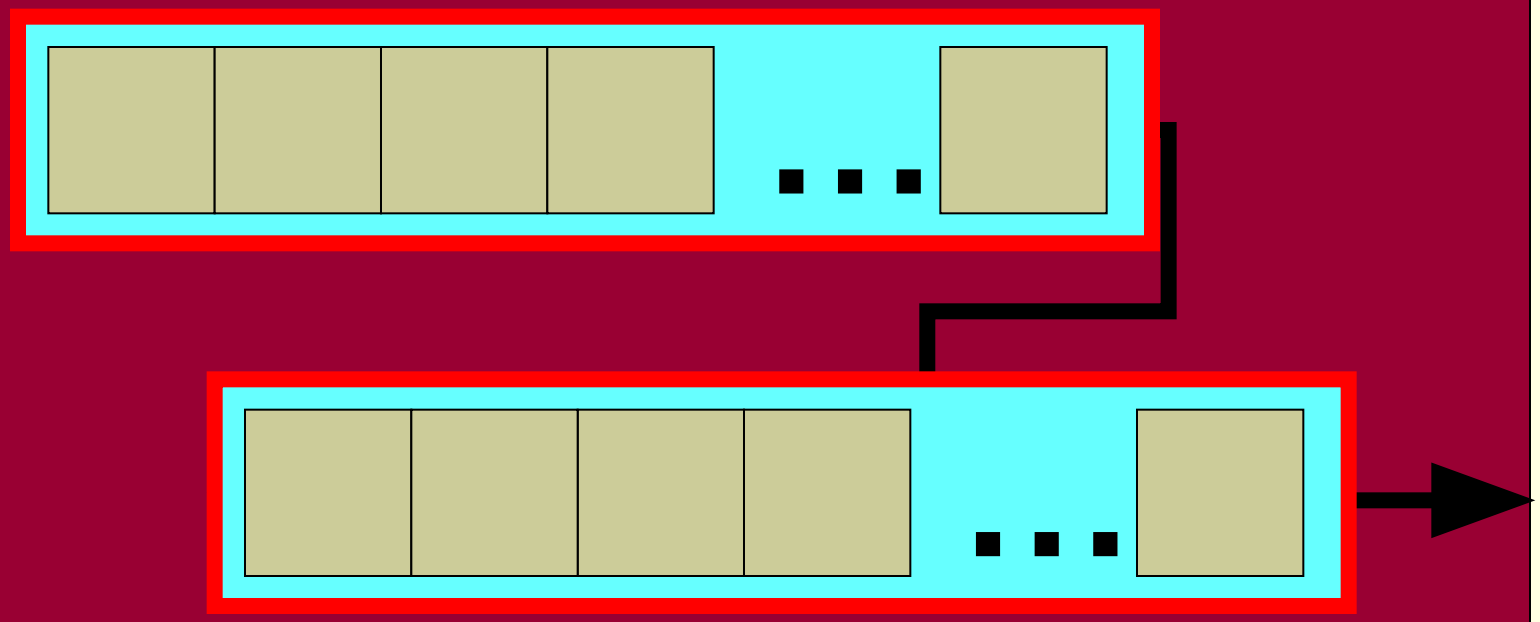
Выберем число $m = 2^c$

Разделим весь список блоков на группы, размером m блоков в каждой, за исключением последней.

Пример

- У нас есть 25 блоков и $m = 8$.
- Разбиваем 25 блоков на группы следующих размеров 8, 8, 8, 1.

Проблема фрагментации



Информация о вхождениях слова сохраняется в списке блоков

Алгоритм

- Пусть есть k заполненных подряд расположенных блоков B_1, …, B_k, в частности последний блок также заполнен, и нам требуется взять где-то новый блок.

$k = 2^x, x < c$

- ищем $2k$ подряд располагающихся блоков $N_1, \ldots N_{2k}$.
- Затем копируем информацию из старых k блоков в первую половину новых блоков, т. е. в блоки $N_1, \ldots N_k$ соответственно.
- $B_1, \ldots, B_k$ помечаются как свободные.
- Запись далее осуществляется в N_{k+1}.
- $N_{k+2}, \ldots, N_{2k}$, помечаются как зарезервированные

$$k = 2^x, x = c$$

- Заканчиваем текущую группу блоков, в ней уже есть $m = 2^c$ блоков.
- Начинаем формировать новую группу блоков.

Остальные случаи

- Используем зарезервированные ранее блоки (в случае $k = 2^x$, $x < c$)

Эффективное использование дисковой памяти

Эффективное использование памяти

Все базовые формы разделяются на *n* групп. Используем *n* временных файлов. Вначале читаем документы, записываем информацию о вхождениях для *i*-й группы в *i*-й временный файл.

При создании индекса обрабатываем отдельную группу. Кэш используется только для одной группы.

Сравнение с существующими разработками

- Общий объем 35,2 гб, 191 074 файла
- Все файлы были в кодировке Windows-1251 (CP1251).
- Язык документов – русский.
- Все файлы представляли собой обычный текст.

Описание конфигурации оборудования

- 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: 4 гб, DDR2 800.
- Жесткий диск: Seagate Barracuda 7200.10, 7200 RPM, кэш 16 мб., объем 750 гб.
- FSB 1066 MHz.

Создание индекса

- Создание инвертированного файла: время 9 часов, размер 40 гб.
- Создание CLB индекса: время 3 часа, 32 мин., размер 24 гб.

Для CLB индекса использовался размер блока 16 КБ.

Добавление в индекс одного файла среднего размера

- Время добавления одного документа 1,2 мб. для CLB индекса: 9 мин.
- Время добавления одного документа 1,2 мб. в инвертированный файл: 57 мин.

Добавление в индекс одного файла малого размера

- Время добавления одного документа размером 534 байта для CLB индекса: 22 с.
- Время добавления одного документа размером 534 байта в инвертированный файл: 57 мин (т. е. такое же, как при размере файла 1,2 мб).

Время поиска

- Время поиска в инвертированном файле и CLB-индексе практически совпадают.

Выводы

- Проведенные эксперименты показывают высокую эффективность CLB индекса при добавлении в него данных небольшого размера.

Сравнение с существующими разработками

- Процессор: Intel Pentium 4, 3.0 GHz, кэш: L1 Data – 16 кб, L1 trace – 12 Kuops, L2 - 2048 кб.
- Оперативная память: 4 гб, DDR2 533.
- Жесткий диск: Seagate Barracuda 7200.8, 7200 RPM, кэш 8 мб., объем 200 гб.
- FSB: 800 MHz.

Создание CLB индекса

- Размер индекса 26,2 гб.
- Время создания 5 часов 49 мин.
- Использовался размер блока 16 КБ.

SearchInform Desktop (http://www.searchinform.com)

- Размер индекса 16,15 гб.
- Время создания 9 часов.

Архивариус 3000 http://www.likasoft.com/

- Размер индекса 24,83 гб.
- Время создания 6 часов 46 мин.

Google Desktop

- Размер индекса ~ 5 гб
- Время создания 31 час 25 минуты

Выводы

- Эксперименты показывают высокую скорость создания CLB индекса.

Эксперименты

- Общий объем 86 гб, 400 049 файла
- Все файлы были в кодировке Windows-1251 (CP1251).
- Язык документов – русский.
- Все файлы представляли собой обычный текст.

Описание конфигурации оборудования

- 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: 4 гб, DDR2 800.
- Жесткий диск: Seagate Barracuda 7200.10, 7200 RPM, кэш 16 мб., объем 750 гб.
- FSB 1066 MHz.

Создание CLB индекса

- Размер индекса 56,5 гб.
- Время создания 4 часа 28 минут.
- Использовался размер блока 64 КБ.

Инвертированные файлы

- Размер индекса 117,7 гб.
- Время создания 20 часов 6 минут.

Архивариус 3000 http://www.likasoft.com/

- Размер индекса 62,65 гб.
- Время создания 6 часов 10 минут.

Инструментарий

- Автором разработана библиотека для создания индексов и поиска в текстах, в которой реализована описанная структура данных и алгоритмы.

Форматы файлов

- Библиотека может индексировать файлы в различных форматах, например RTF, PDF, CHM, HTML, DJVU и кодировках, например UNICODE, UTF8, CP1251, ASCII, KOI8.

Архивы

- Поддерживается обработка архивов форматов ZIP, CAB, RAR, 7Z, ARJ, TAR, и др.

Архитектура

- Библиотека реализована в виде COM сервера для операционных систем Windows
- Написана на C++.

Архитектура

- Ядро, осуществляет создание индекса и поиск.
- Модуль поддержки морфологии
- Модуль распознавания кодировки. При распознавании кодировки также учитывается морфология.

Форматы файлов

- Модуль поддержки форматов файлов. Поддержка форматов файлов и архивов реализована с помощью подключаемых дополнительных модулей, которые могут быть реализованы в виде динамических библиотек или написаны на Java. Модуль поддержки форматов файлов реализован в виде отдельного процесса для повышения надежности системы.

Архитектура

- Модуль атрибутов документов, для сохранения описания документов.
- Модуль репозитария, для сохранения текстов документов. Создается для того, чтобы при поиске можно было быстро получать фрагмент текста, содержащий найденную фразу.

Архитектура

- Модуль COM осуществляет доступ к остальным модулям извне с помощью COM, что позволяет использовать библиотеку в различных языках программирования.

Системные требования

- Реализованные алгоритмы достаточно нетребовательные к ресурсам компьютера. Для создания индекса достаточно иметь 300–400 мегабайт свободной оперативной памяти.
- Автором проводились эксперименты по созданию индексов на машине с оперативной памятью размером 512 мб.

SSD

- Эффективность описанных в данном алгоритмов значительно возрастет с применением дисков SSD (Solid-state drive), за счет более быстрого чтения блоков малого размера. При этом эффективность таких структур данных как инвертированные файлы возрастет менее, т. к. для добавления в инвертированный файл информации его все равно придется практически переписать целиком.

Литература

- Веретенников А. Б., Лукач Ю. С. Еще один способ индексации больших массивов текстов // Известия Уральского государственного университета. Сер. «Компьютерные науки». – 2006, №43. – С. 103–122.
- Лукач Ю. С. Быстрый морфологический анализ флективных языков // Междунар. алгебраическая конф. : 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.
- Bayer, R., McCreight, E. Organization and maintenance of large ordered indexes. Acta Informatica 1, 3 (1972), 173-189.
- Ferragina, P., Grossi, R. The string B-tree: a new data structure for string search in external memory and its applications. Journal of the ACM, 46, 2 (1999), 236-280.
- Ferragina P., Grossi R. An experimental study of SB-trees. 7th ACM-SIAM symposium on Discrete Algorithms, 1996.
- Prywes, N. S., Gray, H. J. The organization of a Multilist-type associative memory. IEEE Trans. on Communication and Electronics, 68 (1963), 488-492.