

6. Текстовые базы данных

6.1. Поиск без использования индекса

6.2. Инвертированный индекс

6.3. Битово-матричное индексирование

6.4. Сигнатурное индексирование

6.5. S-деревья

6.6. Расширенный информационный поиск

6.7. Латентное семантическое индексирование

Текстовые базы данных

- Содержат текстовую информацию/текстовые документы
- Наиболее распространенные на сегодняшний день
- «Зрелый» уровень развития; «первые шаги» в 60-70-х годах
- Существует условное разбиение:
 - Структурированные (structured) бд: форматированные записи, объект задается структурой
 - Неструктурированные (unstructured) бд: например, полные тексты статей
 - Полуструктурированные (semi-structured) данные: (веб-страницы, XML-документы)
- «Зрелый» уровень развития; «первые шаги» в 60-70-х годах
- Области применения:
 - Электронный документооборот (автоматизация офиса)
 - Цифровые библиотеки
 - Юридические и патентные системы
 - Электронные словари/энциклопедии/справочники
 - Электронные газеты/журналы/книги
 - Библиотеки исходников программного обеспечения (source program libraries)

Текстовые базы данных

- **Документ?**

В зависимости от контекста, книга, одна глава, один параграф, одна статья, одни тезисы к статье, письмо, исходный код какой-нибудь программы, поле типа VARCHAR или TEXT в бд и т.д.

- Задача: поиск в коллекции документов по их содержанию - ассоциативный поиск (associative search)
- Используются ключевые слова (keywords) или термины (terms)
- Поисковый запрос: совокупность ключевых слов, объединенная булевыми (или логическими) выражениями AND, OR, NOT
- Индекс: механизм для нахождения заданного термина в тексте

Поиск без использования индекса

- Последовательное полнотекстовое сканирование текста
- Достоинства:
 - Не требуется дополнительное место для индекса
 - Быстрые изменения/обновления (не нужно вносить изменения в индекс)
 - Нахождение частично совпадающих строк относительно простое (при помощи wildcard-символов: таких как *, ? и т.д.)
 - Возможно приближенное сравнение (approximate matching) (путем задания порогового значения для расстояния Левенштейна (Levenshtein distance или edit distance))

Поиск без использования индекса

Алгоритмы¹ для нахождения точного соответствия (exact matching) в общей памяти:

- *Knuth-Morris-Pratt*: алгоритм линейной сложности ($O(n)$ time); основан на предварительной обработке строки поиска (строка поиска – образец который ищется в тексте)
- *Boyer-Moore*: поиск происходит быстрее для более длинных строк; на практике скорость работы сублинейна; основан на предварительной обработке строки поиска
- *Aho-Corasick*: одновременный поиск нескольких терминов; основан на теории конечных автоматов (finite automata)
- *Rabin-Karp*: используется специальная хеш-функция – хеш-значения последовательно вычисляются для всех подстрок текста и потом сравниваются с хеш-значением строки поиска

¹ Более подробно описаны здесь:

- Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. *Алгоритмы. Построение и анализ*, 2-ое издание, Вильямс, 2005
- http://en.wikipedia.org/wiki/String_searching_algorithm (с текстами исходников)

Инвертированный индекс

- Традиционный способ, широко используется в поисковых машинах
- **Инвертированный?**
Каждый документ - это список терминов (index terms), а инвертированный индекс каждому термину ставит в соответствие список документов в которых встречается данный термин; т.е. получаем обратное (инвертированное) соответствие – у каждого термина есть список документов
- Лексикон: термины, используемые в документе(-ах)
- Правила построения лексикона:
 - Все символы переводятся в нижний регистр (case folding);
 - Удаление суффиксов, в индекс идут корневые формы слов (stemming)
 - В индекс не включаются так называемые стоп-слова: (например, для английского языка не включаются the, is, as, that, и ряд других)

Инвертированный индекс

«Разрешение» индекса:

- Степени детализации индекса (granularity of the index) — «разрешение» индекса: разрешение с которым индекс определяет местоположение термина в документе
- Три степени детализации индекса:
 - Крупноструктурный индекс (coarse-grained index): каждый термин указывает на блок(-и) текста (а блок включает в себя несколько документов);
 - Среднеструктурный индекс (moderate-grained): указывает на документы, содержащие термин (список id-номеров документов);
 - Мелкоструктурный индекс (fine-grained): может указывать на предложение, в котором есть данный термин, или порядковый номер термина внутри документа или даже точную байтовую позицию термина.

Инвертированный индекс

- Плюсы и минусы крупноструктурного индекса:
 - + Маленький размер
 - + Легко поддерживать (обновлять, вносить изменения)
 - Требуется значительного полнотекстового сканирования
 - Нерелевантен при поиске нескольких терминов одновременно (например, искомые слова могут находиться в одном и том же блоке, но необязательно в одном и том же документе)
- Плюсы и минусы мелкоструктурного индекса:
 - + Позволяет находить слова, расположенные рядом (proximity queries)
 - Большой размер индекса
 - Тяжело поддерживать
- Если запросов в которых ищут близко расположенные слова немного, то используют среднеструктурный индекс

Инвертированный индекс

Сжатие:

- Инвертированные файлы - ориентировочно 50-100% от размера суммарного текста
- Лучшее сжатие, если:
 - Индекс без потери общности можно задать в виде целых чисел; например, лексикон сортируется в алфавитном порядке, и затем каждому термину из лексикона ставится в соответствие число - номер по списку; указатели на документы - id-номера документов
 - Храним неполные id-номера документов; например, <8: 3, 5, 19, ...> , где 8 – это номер термина в лексиконе, а 3, 5, 19 – id-номера документов, можно упростить до <8: 3, 2, 14, ...>
 - Располагаем термины лексикона в алфавитном порядке, тогда общие префиксы расположенных рядом терминов можно представлять более компактно
 - Для мелкоструктурного индекса: вместо сохранения полных указателей на термины в документе, можно хранить интервалы между последовательно встречающимися терминами

Инвертированный индекс

- Составные запросы:
 - **AND**: выполняется путем извлечения для каждого термина списка документов и нахождения пересечения
 - **OR**: аналогично, но возвращаем объединение списков
 - **NOT**: обычно сочетается с оператором AND, затем вычисляется логическая разность между списками
- Структуры данных для инвертированного индекса:
 - В+-дерево, где термины являются ключами, а списки указателей листьями
 - Организация в виде хеша, обычно динамическая версия:
 - Линейный хеш (linear hash)
 - Расширяемый хеш (extendible hash)
 - Бор-структура или трай-структура («trie-structure» от **retrieval**, «бор» от **выборка**: на каждый узел приходится только один символ, термин находится движением по бору (дереву) с вершины до соответствующего листа
- Список документов находится в структуре данных либо локально, либо (предпочтительно) от нее обособлен

Инвертированный индекс

Алгоритм построения индекса:

1. Для каждого документа в коллекции, сформировать список терминов, где для каждого термина есть список указателей на месторасположение данного термина(-ов) в документе. Итог - большой последовательный файл S.
2. Отсортируем S, с помощью например mergesort (сортировка слиянием).
3. Объединим последовательные элементы, представляющие один и тот же индекс-термин.
4. На основе выбранной структуры данных (B+-дерево, хеш, ...) сформировать индекс. При этом пусть листья содержат указатели на хранящиеся обособлено списки id-номеров документов (и указателей на месторасположение терминов в документах, если строится мелкоструктурный индекс). Сами же списки хранятся как записи переменной длины.

Резюме:

- Очень эффективный и популярный метод извлечения данных
- Несмотря на минусы (большой размер и сложность поддержки индекса) широко используется для коллекций статичных документов

Битово-матричное индексирование

- Еще один способ представления инвертированного индекса: в матричной форме
- Крупноструктурная или среднеструктурная степень детализации
- Битовая матрица – матрица, в которой число строк соответствует числу терминов в индексе (размер лексикона), число столбцов = число документов в коллекции. $\langle i, j \rangle$ равен 1, если термин i встречается в документе j , иначе $\langle i, j \rangle$ равен 0.
- Эффективно для запросов с логическими выражениями: AND, OR, NOT могут быть реализованы в «железе» (например, с параллельной обработкой по 64 бита)
- Проблема: индекс занимает много места (размер матрицы = число терминов*число документов, матрица разреженная (нолей много больше чем единиц))
- Комбинация индексов: инвертированный индекс для не часто встречающихся терминов и бинарная матрица для более часто встречающихся терминов
- Сжатие индекса: кодирование с переменной длиной строки (или метод RLE); заменяет последовательности нулей на их количество

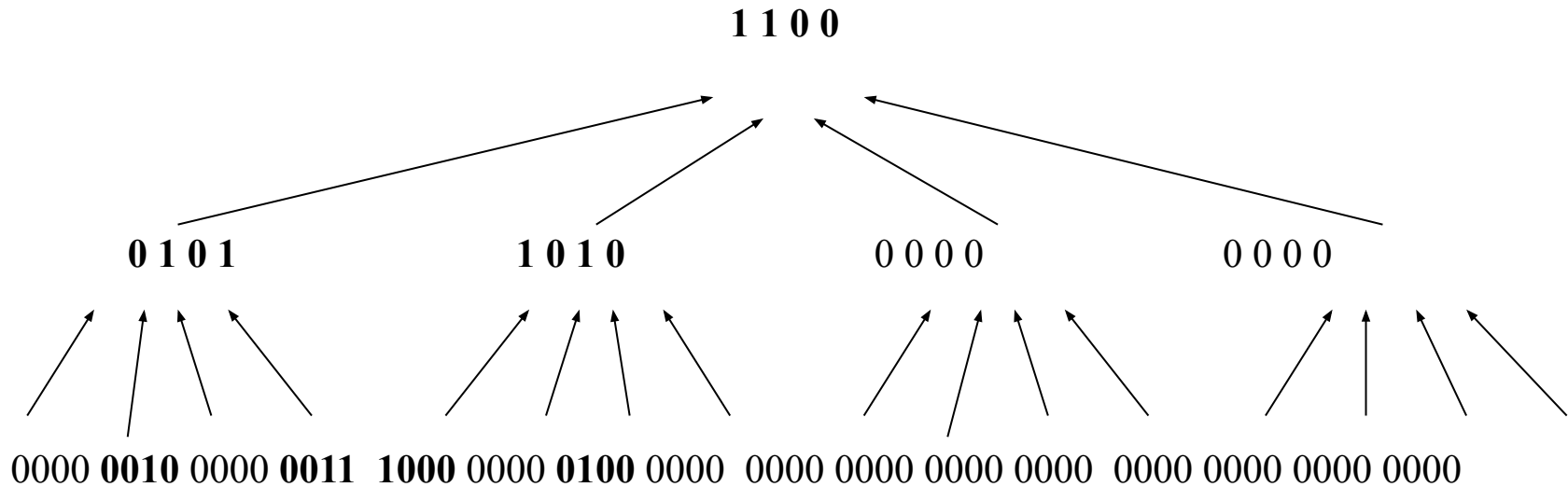
Битово-матричное индексирование

Иерархическое сжатие битовых строк:

- Разбиваем строку на блоки символов одинакового размера
- Для каждого блока применим к битам данного блока операцию дизъюнкции (OR - операция логического сложения), результат даст нам бит следующего уровня; операция OR примененная к блоку со одними нулями даст ноль, если в блоке имеется хоть одна единица, то получим единицу
- Рекурсивно повторяем процесс на всех уровнях иерархии

Каждый бит легко определяется построением соответствующего пути от вершины дерева

Битово-матричное индексирование



- Матрица обычно разреженная, большинство листьев - блоки нулей
- Сжатие происходит за счет того, что блоки нулей можно не хранить совсем

Сигнатурное индексирование

- Вероятностная техника
- Сигнатура – битовый массив, генерируемый путем хеширования терминов в массив индексов (о генерации ниже); длина сигнатуры обычно как минимум несколько сотен бит
- Сигнатуры хранятся в отдельном (сигнатурном) файле; его размер **меньше** чем размер всей коллекции документов
- Сигнатурный файл - фильтрующий механизм, уменьшающий количество данных, которые просматриваются во время поиска
- Возможен поиск по частичному совпадению
- Поисковый запрос рассматривается как самостоятельный документ (состоящий из терминов, заданных в запросе); у запроса есть своя сигнатура; при поиске сигнатура запроса Q сравнивается с сигнатурами документов D_j ; если единичные биты Q есть в сигнатуре документа D_j , то D_j является возможным результатом (candidate result)
- Сигнатура приблизительное описание документа; документы нерелевантные запросу (их называют false drops) должны быть устранены

Сигнатурное индексирование

- Достоинства сигнатурных файлов:
 - Меньший размер (в сравнении с мелко-структурным инвертированным индексом); типичный размер – 10-20% от общего размера коллекции документов
 - Более удобны чем многомерные индексы (о них будет рассказано позже)
- Два основных метода генерации сигнатур:
 - **Метод сигнатур слов** (word signature method): каждому термину хеш-функция ставит в соответствие определенную битовую комбинацию (bit pattern) заданной длины; получившиеся композиции соединяют вместе, что дает сигнатуру документа; метод сохраняет информацию о последовательности с которой термины встречаются в документе; в тоже время, дает более высокую вероятность появления нерелевантных запросу документов
 - **Суперпозиционное кодирование** (superimposed coding): каждому термину ставится в соответствие битовая комбинация такой же длины, что и полная сигнатура; получившиеся комбинации логически складывают (выполняют OR), получая итоговую сигнатуру; метод по умолчанию далее
- Как уменьшить число false drops?
 - Количество нолей и единиц в сигнатуре должно быть примерно одинаковым (вес сигнатуры - количество битовых единиц в ней)
 - Ноли и единицы должны быть однородно распределены по сигнатуре

Сигнатурное индексирование

Пример:

- Есть три документа со следующими терминами:
 - D_1 : база данных, объект, программирование, схема
 - D_2 : алгоритм, компьютер, программирование
 - D_3 : алгоритм, структура данных, программирование
- Длина сигнатуры 8 бит, хеш-функция возвращает позицию бита с единицей (остальные семь бит – 0)
 - hash(“алгоритм”) = 3 (т.е. 00100000);
 - hash(“компьютер”) = 1 (10000000);
 - hash(“база данных”) = 7 (00000010);
 - hash(“структура данных”) = 5;
 - hash(“объект”) = 6;
 - hash(“программирование”) = 4;
 - hash(“схема”) = 1.
- Сигнатура D_1 – 10010110, сигнатура D_2 – 10110000, D_3 – 00111000
- Запрос: найти документы с «алгоритм» и «программирование»
Запрос Q задается сигнатурой 00110000; единичные биты Q есть у D_2 и D_3 (технически, выполняется побитовая операция AND); для получения окончательного результата нужно проверить D_2 и D_3 на наличие «алгоритм» и «программирование»

00000010
00000100
00010000
10000000
<u>10010110</u>

Сигнатурное индексирование

Организация сигнатурного файла:

Основные принципы:

- Неизбыточный размер файла
- Минимизация операций ввода/вывода при выполнении запросов
- Простая поддержка (изменение, удаление, добавление) индекса

(1) Сингулярный файл:

- Составлен из списка сигнатур для всех документов из коллекции
- Выполнение запроса: генерирование сигнатуры запроса; ее сравнение (выполнение побитовой операции AND) с каждой сигнатурой, имеющейся в сигнатурном файле (т.е. приходится сканировать весь сигнатурный файл); найденные документы (возможные результаты) затем просматриваются полностью, чтобы исключить false drops
- Скорость выполнения запросов невелика
- Замечание: для физического представления сигнатурного файла В-дерева не подходят

Сигнатурное индексирование

(2) Транспонированный сингулярный файл:

- Сингулярный файл – матрица (таблица) в которой число строк равно числу сигнатур (= числу документов), число столбцов длине сигнатуры. Обеспечивается прямой доступ к любому из столбцов матрицы (столбец матрицы также называют битовым срезом, англ. bit-slice).
- Выполнение запроса: рассматриваются только те столбцы (битовые срезы), что соответствуют позициям единиц в сигнатуре запроса (и доступ требуется только к этим столбцам). Затем, эти столбцы логически перемножаются (AND) и единицы в получившемся столбце укажут на документы (см.след.слайд).
- Поддержка трудоемка (например, добавление нового документа в индекс может вызвать изменения во всех столбцах). В целом, время добавление нового термина в индекс в случае транспонированного сигнатурного файла сравнимо со временем вставки нового термина в файл инвертированного индекса.

Сигнатурное индексирование

3-gram Signatures

3-gram		3-gram signature
ate		0001 1000 0000 0000
eri		0100 0001 0000 0000
ery		0000 0100 0000 0100
ina	n-gram	0010 0010 0000 0000
ine	hash	1000 0000 0100 0000
Kat	function	0000 0100 0000 0001
rin		0000 0100 0001 0000
ryn		0010 0000 0000 0100
ter		0000 0000 1000 0100
yne		0000 0010 0001 0000

$F=16$
 $S=2$

Lexicon

no.	word	
n1	Katerina	*
n2	Katerine	
n3	Kateryn	
n4	Kateryne	**

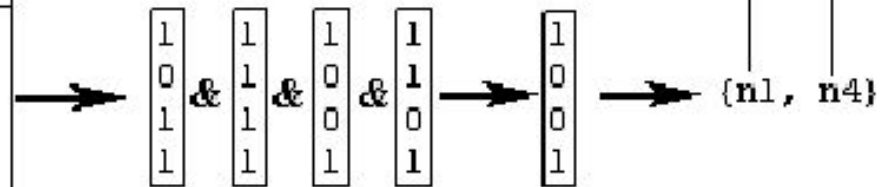
N (no. of words) = 4

*: true match

** : false match

Signature File

no.	superimposed signature
n1	0111 1111 1001 0101
n2	1101 1101 1101 0101
n3	0011 1100 1000 0101
n4	0011 1110 1001 0101



Query: *rina* 0010 0110 0001 0000

Один из найденных сравнением сигнатур документов нерелевантен запросу

Сигнатурное индексирование

(3) Двухуровневые сигнатурные файлы:

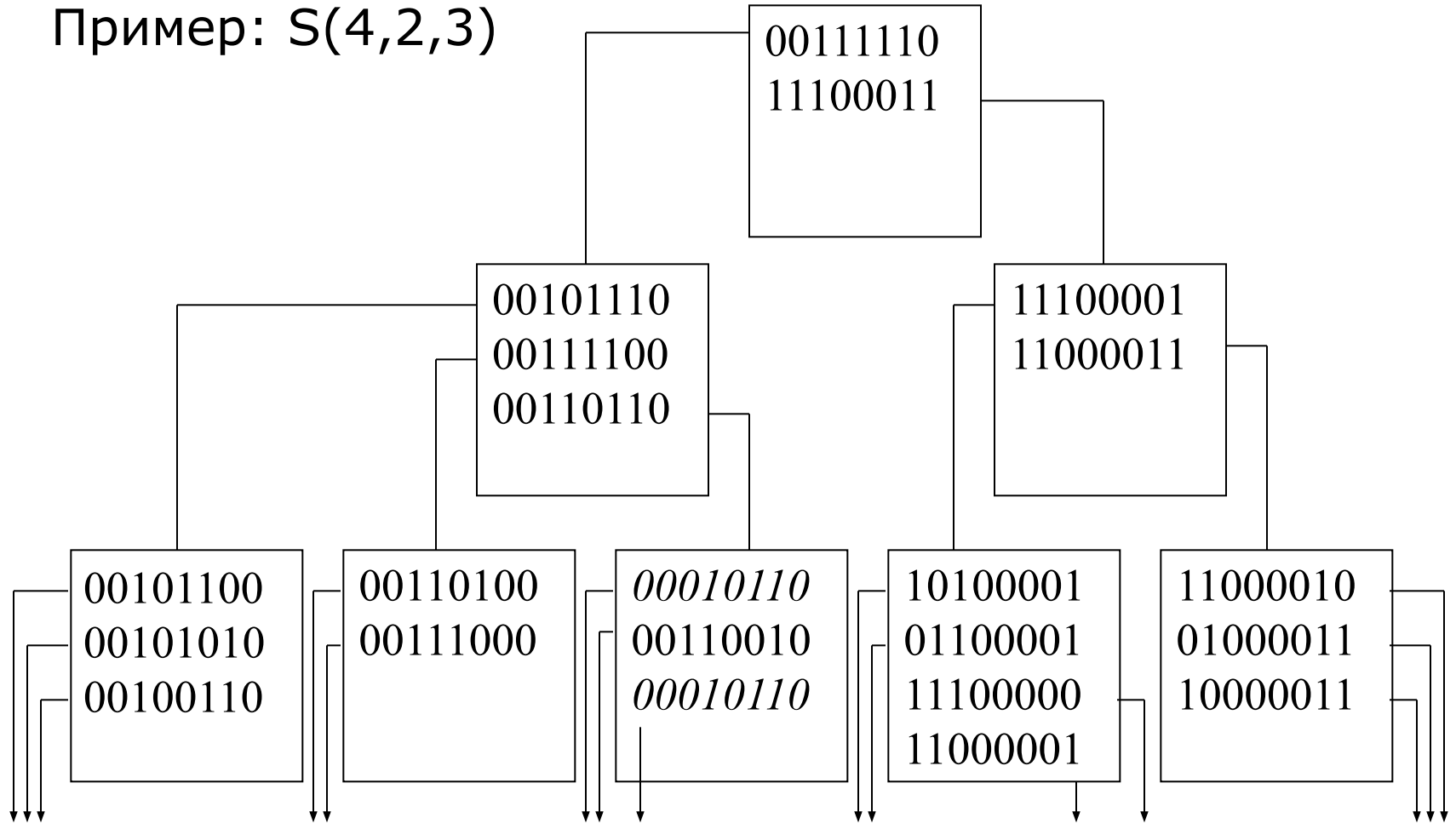
- Два уровня (типа) сигнатур: высшего и низшего уровня
- Сигнатура высшего уровня является фильтром для низшего уровня
- Генерация сигнатуры высшего уровня:
 - а) Разбить документы на группы и генерировать сигнатуры для групп; сигнатуры больше в длину, но суммарный размер меньше
 - б) Генерировать сигнатуры путем логического сложения (OR) сигнатур низшего уровня; недостаток - в силу OR будет расти число единиц и, значит, количество false drops

S-деревья

- Многоуровневые сигнатурные файлы (multi-level signature file)
- Обобщение двухуровневых сигнатурных файлов: сигнатуры в каждом узле получаются логическим сложением (OR) сигнатур, содержащихся в дочерних узлах (см. «двухуровневые сигнатурные файлы», способ б)
- S-деревья:
 - S-деревья и B-деревья имеют несколько общих свойств
 - S-дерево - сбалансированное по высоте дерево с множественным ветвлением (height balanced multiway tree)
 - Каждый узел соответствует странице (page) (блоку данных)
 - У корня дерева (корневого узла) не менее двух и не более K (= емкость страницы) дочерних узлов (если корень не является листом (leaf) дерева)
 - Все узлы за исключением корневого имеют не менее k и не более K дочерних узлов ($1 \leq k \leq K$, например $k=K/2$)
 - Каждый узел (за исключением возможно корневого) содержит не менее k сигнатур
 - Листья дерева содержат указатели на документы
- Отличия S-дерева от B-дерева:
 - Сигнатуры в узлах не упорядочены
 - Одна и та же сигнатура может присутствовать в нескольких местах (сигнатуры разных документов могут совпадать)
 - Запросу вполне может соответствовать несколько путей движения по дереву

S-деревья

Пример: $S(4,2,3)$



Указатели на документы

S-деревья

- Выполнение запросов с помощью S-деревьев:
 - Сгенерировать сигнатуру запроса путем суперпозиционного кодирования терминов
 - Обход дерева начнем с вершины
 - Для каждого узла просматриваем содержащиеся в нем сигнатуры; если 1-биты сигнатуры запроса совпадают с 1-битами сигнатур узла, то переходим к соответствующим дочерним узлам
 - Дойдя до листьев дерева, получим указатели на документы, которые нужно просмотреть для устранения false drops
- Вставка сигнатуры в S-дерево:
 - Основная идея: логическое сложение (OR) всех сигнатур в узле должно иметь как можно меньший вес; тогда достигается наивысшая избирательность (selectivity)
 - Для определения подходящего узла для вставки осмотр начинают с вершины дерева, переходя на наиболее «близкие» дочерние узлы; под наиболее «близким» узлом может пониматься:
 - Узел, сигнатуры которого, имеют наименьшее расстояние Хемминга (Hamming distance) с сигнатурой вставляемого документа, или
 - Узел, имеющий наименьшее увеличение веса (weight increase), когда сигнатуры узла и добавляемую сигнатуру логически складывают (OR)
 - После вставки нужно внести изменения во все выше лежащие узлы

S-деревья

Переполнение узла:

- Узел разбивают на два, стараясь при этом минимизировать весы новых получающихся сигнатур и уменьшить перекрытие между ними
- Используют следующий эвристический алгоритм для разбиения узла S-дерева:
 - (1) Выбирается сигнатура имеющая наибольший вес; она добавляется в группу 1; в группу 2 добавляется та сигнатура, что имеет наибольшее увеличение веса по отношению к сигнатуре из группы 1
 - (2) Каждая из оставшихся сигнатур добавляются в ту группу в которой достигается наименьшее увеличение веса при логическом сложении сигнатур уже добавленных в группу и рассматриваемой сигнатурыВ равнозначных случаях используется следующие правила (по порядку):
 - а) добавить в меньшую группу;
 - б) добавить в группу с меньшим весом;
 - в) добавить в случайно выбранную группу.Если на каком-то шаге в одной из групп K-к сигнатур, добавить остающиеся сигнатуры в другую группу
- Распространение:
 - Как и у B-деревьев, разбиение узлов может быть распространено на выше лежащие узлы вплоть до вершины
 - После разбиения узла все выше лежащие узлы должны быть соответствующим образом изменены
- Обновление документа: индекс обновляется путем последовательного удаления и вставки сигнатуры

Расширенный информационный поиск

Основные принципы:

- Результаты поиска не обязательно точно соответствуют запросу
- Ранжирование (ranking) документов, т.е. все документы из коллекции могут быть расставлены в определенном порядке на основе ранга каждого документа; ранг документа определяется на основе «расстояния» от документа до запроса
- Учитываются семантические связи между терминами

Расширенный информационный поиск

Важные понятия:

- Синонимия (synonymy): различные термины имеют одинаковое или близкое значение
- Полисемия (многозначность) (polysemy): один термин имеет несколько значений
- Вес (weight) термина: указывает на важность термина в документе
- Добротность (качество) поиска:
 - Точность (precision) поиска: отношение количества извлеченных документов, релевантных (удовлетворяющих условиям поиска) поиску, к общему количеству извлеченных документов
 - Полнота (recall) поиска: отношение количества извлеченных документов, релевантных поиску, к общему количеству релевантных документов

Часто происходит объединение этих двух понятий в одно – релевантность; релевантность поискового запроса - степень соответствия искомого (запроса) и найденного (результатов запроса), уместность результата; релевантность - субъективное понятие

Расширенный информационный поиск

- Коллекцию документов можно представить в виде матрицы D (обобщение битово-матричного индекса).
- $D_{t,i}$ = значение веса термина t в документе i (вес термина может задаваться как количество раз, которое термин встречается в документе).
- Столбцы матрицы (векторы) характеризуют документы.
- Запрос – документ: запрос \rightarrow вектор (при построении веса входящих в запрос терминов как правило не учитываются).
- Задача информационного поиска: *Найти k документов, вектора которых наиболее «близки» к вектору запроса.*
- Вопрос: как измерять «расстояние» между документами (между запросом и документом)?
- Первой попытка: меры схожести (similarity), скалярное произведение векторов запроса и документа:

$$\text{similarity}(Q, D_i) = Q \cdot D_i = \sum_{t \in \text{Terms}} Q_t D_{t,i}$$

Расширенный информационный поиск

- Сложности:
 - Если вес термина задается как количество экземпляров данного термина в документе, то широко используемые термины (слова) будут иметь избыточный вес
 - Длинные документы (содержат больше терминов) будут в более привилегированном положении чем короткие
- Закон Ципфа (Zipf's law): *Формулировка закона - если все слова некоторого языка (или достаточно длинного текста) упорядочить по убыванию частоты их использования, то частота n-го слова в таком списке окажется приблизительно обратно пропорциональной его порядковому номеру n. Можно записать: $f * r = c$, где f – частота слова в документе, r – ранг слова, c – константа.*

Следствия: небольшое количество слов встречается очень часто; среднее количество слов встречается со средней частотой; большинство слов встречается очень редко

Расширенный информационный поиск

- Если какое-то слово встречается часто, то это слово хорошо передает смысл документа (бОльший вес)
- Слова, которые часто встречаются во многих документах, не несут существенную информацию
- Правило TF*IDF (TF*IDF rule, TF*IDF – **t**erm **f**requency times **i**nverse **d**ocument **f**requency):

$D_{t,i} = tf * idf = f_{t,i} * \log(1 + N/f_t)$, где $tf = f_{t,i}$ - частота термина t в документе i

tf и idf могут вычисляться по разному – например, частота термина (tf) может быть нормализована

Расширенный информационный поиск

- Геометрические расстояния в векторном пространстве:
 - Евклидово расстояние (Euclidean distance):

$$d(Q, D_i) = \sqrt{\sum_{t \in Terms} (Q_t - D_{t,i})^2}$$

Выделяет длинные документы; степень подобия (similarity) документа и запроса может определяться по мере обратной евклидовому расстоянию

- Определение угла (косинуса угла):

$$Q \bullet D_i = |Q| \cdot |D_i| \cos \theta, \text{ где } |Q| = \sqrt{\sum_{t \in Terms} Q_t^2}, \text{ соотв. } |D_i|$$

$$\text{степень подобия} - \cos \theta = \frac{Q \bullet D_i}{|Q| \cdot |D_i|}$$

- Вместе с правилом TF*IDF:
$$\cos(Q, D_i) = \frac{\sum_{t \in Q} (f_{t,i} (\log(1 + N / f_t))^2)}{|Q| \cdot |D_i|}$$

Латентное семантическое индексирование

- Матрица, которая ставится в соответствие коллекции документов, обычно разреженная (большинство элементов ноли).
- Альтернатива: латентное семантическое индексирование (LSI)
- LSI позволяет представить термины и документы в «семантическом пространстве», выявляя скрытые (внутренние) семантические связи между терминами и документами.
- Процедура для LSI:
 1. Создать частотную матрицу D для искомой коллекции документов
 2. Вычислить сингулярное разложение (singular value decomposition - SVD) матрицы D ; $D = A \times S \times B$ (см.след.слайд)
 3. Уменьшить размер матриц путем удаления малозначащих строк/столбцов
 4. Сохранить получившиеся матрицы с помощью какого-нибудь метода индексирования
- Выполнение запроса: вектор запроса преобразуется в трансформированный вектор; получившийся вектор сравнивается (определяется степень подобия) с векторами одной из уменьшенных матриц разложения

Латентное семантическое индексирование

Сингулярное разложение:

- Широко используется во многих областях
- Для любой матрицы D существует разложение

$$\mathbf{D} = \mathbf{A} * \mathbf{S} * \mathbf{B}^T:$$

Матрицы A размером $M * R$ и B размером $N * R$ ортогональные, т. е. $A * A^T = I$, $B * B^T = I$; матрица S , сингулярная матрица, размером $R * R$ является диагональной, причем диагональные элементы (они также являются сингулярными числами матрицы A) не возрастают ($s_{j,j} \geq s_{j+1,j+1}$)

- Базовая идея сингулярного разложения – уменьшить размер матрицы S , с R до k ; уменьшив матрицу S до размера $k * k$ (удалив $R - k$ сингулярных значений справа внизу), получим что матрица D может быть представлена в виде произведения матриц A_k ($M * k$), S_k ($k * k$), B_k ($N * k$)

Латентное семантическое индексирование

Сингулярное разложение:

- Эффект: удаление «шума», выявление скрытых (неявных) семантических связей
- Вектор запроса подвергается следующей трансформации: $Q_k = Q^T * A_k * S_k^{-1}$; затем, вычисляется степень подобия между Q_k и векторами матрицы V_k (без доказательства)
- Слабая сторона LSI: метод требует значительных матричных преобразований и вычислений
- LSI подходит для статичных (или редко обновляющихся) коллекций документов

Упражнения

1. Объясните более подробно почему число false drops уменьшается, если ноли и единицы однородно распределены по сигнатуре и вес сигнатуры равен примерно половине ее длины.
2. Объясните почему В-дерево не подходит для физического представления сигнатурного файла.
3. Проверить закон Ципфа для большого текста (>200 000 слов) на русском и английском языках. Для подсчета слов использовать, например, программу QB Text Analyzer.
4. Докажите что при выполнении запросов трансформированный вектор запроса Q_k должен сравниваться с векторами матрицы V_k (см. «Латентное семантическое индексирование»).
5. Пусть документы состоят из следующих терминов:
 D_1 : база данных, объект, программирование, схема
 D_2 : алгоритм, компьютер, программирование
 D_3 : алгоритм, структура данных, программирование
Хеш-функция для терминов такая же как на слайде «Сигнатурное индексирование (пример)».
Следовательно, сигнатура D_1 – 10010110, сигнатура D_2 – 10110000, D_3 – 00111000.
Выполните следующие запросы:
 Q_1 : Найти документы с "компьютер" и "программирование".
 Q_2 : Найти документы с "компьютер" или "алгоритм".
 Q_3 : Найти документы, не содержащие "компьютер".