Динамическое хеширование

Предыдущие методы хеширования - статические, т.е. сначала выделяется некая хеш-таблица, под ее размер подбираются константы для хеш-функции.

Это не подходит для задач, в которых размер БД меняется часто и значительно.

По мере роста БД можно:

- пользоваться изначальной хеш-функцией, теряя производительность из-за роста коллизий;
- выбрать хеш-функцию «с запасом» - неоправданные потери дискового пространства;
- периодически менять функцию, пересчитывать все адреса - отнимает очень много ресурсов и выводит из строя базу на некоторое время.

Техника, позволяющая динамически менять размер хеш-структуры

Хеш-функция генерирует т. наз. псевдоключ (“pseudokey”), использующийся лишь частично для доступа к элементу: генерируется дост. длинная битовая последовательность, которая д. б. достаточна для адресации всех потенциально возможных элементов. При статическом хешировании для этого потребовалась бы очень большая таблица (которая обычно хранится в ОП для ускорения доступа), здесь размер занятой памяти прямо пропорционален количеству элементов в базе данных.

Каждая запись в таблице хранится не отдельно, а в каком-то блоке (“bucket”). Эти блоки совпадают с физическими блоками на устройстве хранения данных. Если в блоке нет больше места, чтобы вместить запись, то блок делится на два, а на его место ставится указатель на два новых блока.

Задача состоит в том, чтобы построить бинарное дерево, на концах ветвей кот. были бы указатели на блоки, а навигация осуществлялась бы на основе псевдоключа. Узлы дерева м. б. 2-х видов: узлы, показывающие др. узлы или узлы, показывающие блоки.

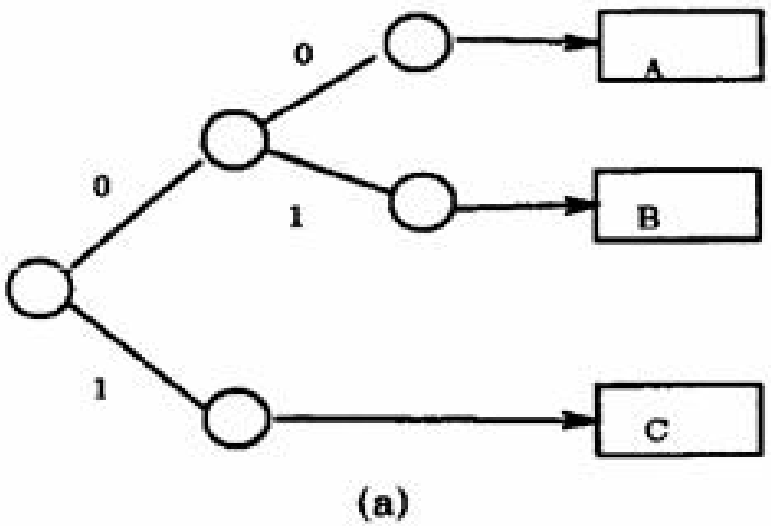
Например, пусть узел имеет такой вид, если он показывает на блок:

- Zero Null
- Bucket Указатель
- One Null

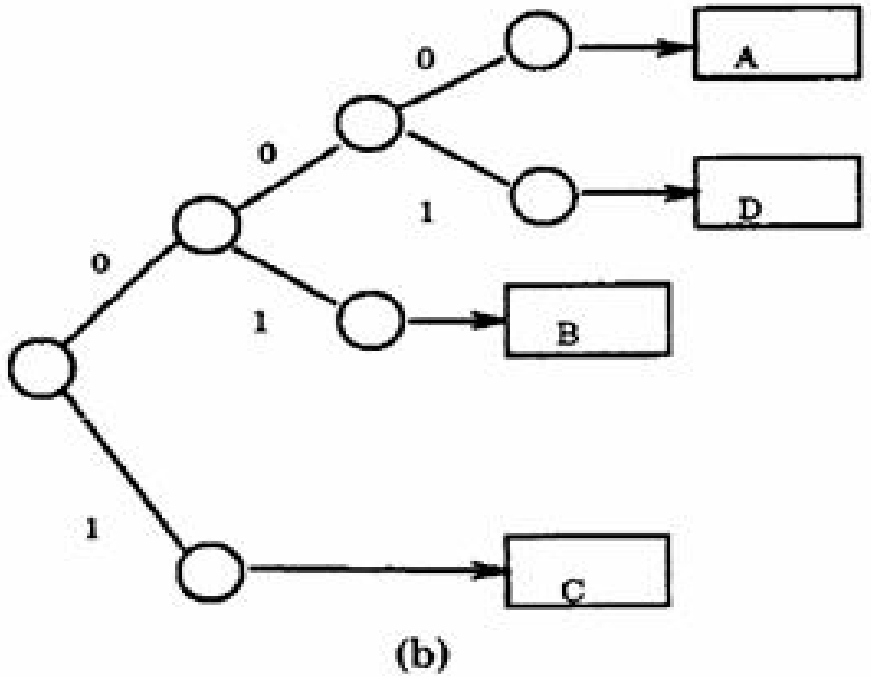
Если он будет показывать на два других узла (a, b), то он будет иметь такой вид:

- Zero Адрес **a**
- Bucket Null
- One Адрес **b**

- Вначале имеется указатель на динамически выделенный пустой блок. При добавлении элемента вычисляется псевдоключ, и его биты поочередно используются для определения местоположения блока.
- Например, элементы с псевдоключами 00… будут помещены в блок **A**, а 01… - в блок **B**. Когда **A** будет переполнен, он будет разбит т. о., что элементы 000… и 001… будут размещены в разных блоках (**A** и **D**).



(a)



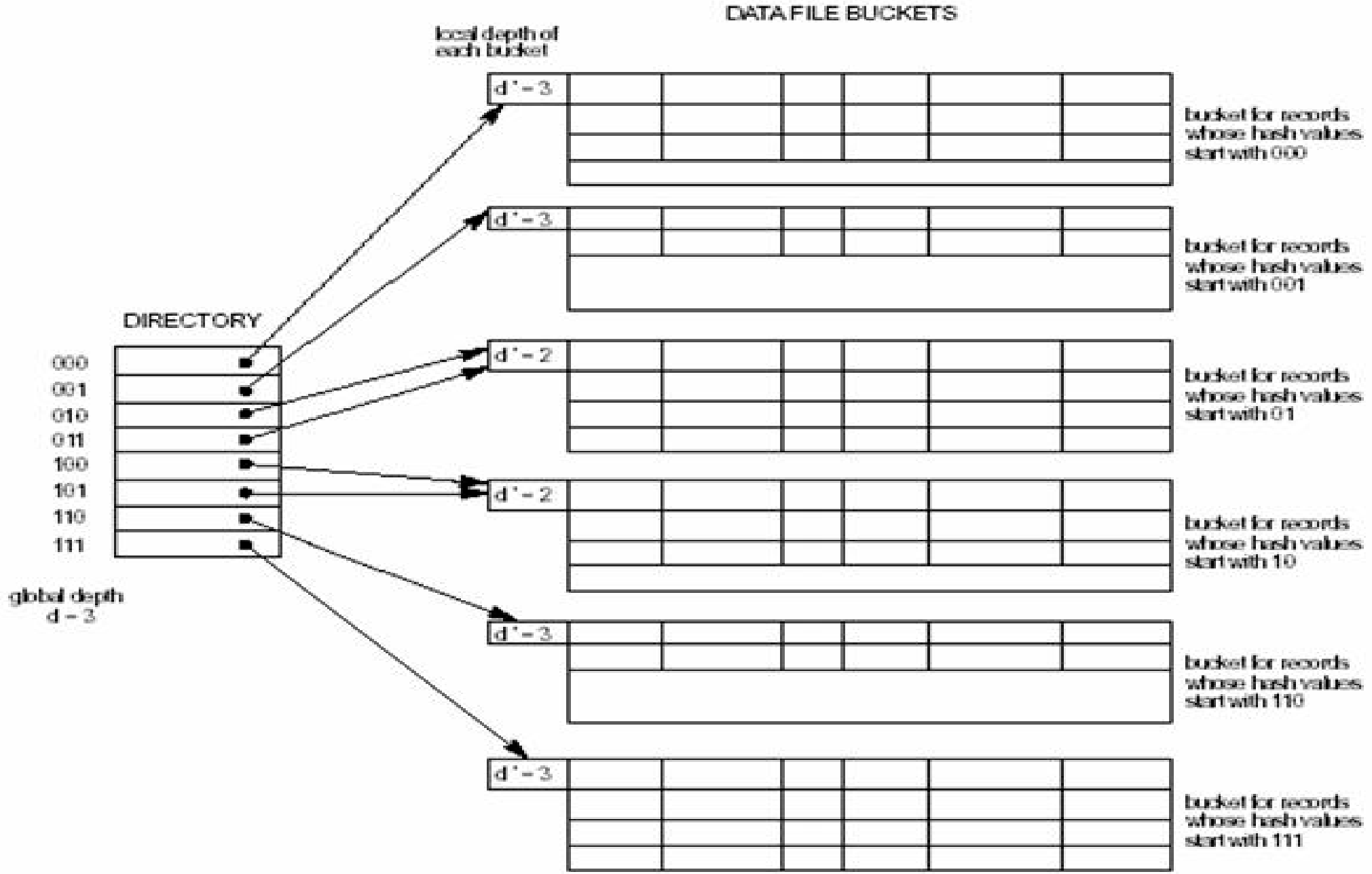
(b)

Расширяемое хеширование (extendible hashing)

Этот метод также предусматривает изменение размеров блоков по мере роста БД, но это компенсируется оптимальным использованием места, т.к. за один раз разбивается не более одного блока, накладные расходы достаточно малы.

Вместо бинарного дерева - список, элементы кот. ссылаются на блоки. Сами элементы адресуются по некоторому количеству ***i*** битов псевдоключа. При поиске берется ***i*** битов псевдоключа и через список (directory) находится адрес искомого блока. При добавлении элементов сначала выполняется процедура, аналогичная поиску. Если блок неполон, добавляется запись в него и в БД. Если заполнен, он разбивается на 2, записи перераспределяются по описанному выше алгоритму. В этом случае возможно увеличение числа бит, необходимых для адресации. Размер списка удваивается и каждому вновь созданному элементу присваивается указатель, который содержит его родитель. Т. о., возможно, когда несколько элементов показывают на один и тот же блок. (за одну операцию вставки пересчитываются значения не более, чем одного блока).

Удаление производится по такому же алгоритму, только наоборот. Блоки, соответственно, могут быть склеены, а список – уменьшен в два раза.


DATA FILE BUCKETS
local depth of each bucket
d' = 3
bucket for records whose hash values start with 000
d' = 3
bucket for records whose hash values start with 001
DIRECTORY
000
001
010
011
100
101
110
111
d' = 2
bucket for records whose hash values start with 01
d' = 2
bucket for records whose hash values start with 10
d' = 3
bucket for records whose hash values start with 110
d' = 3
bucket for records whose hash values start with 111
global depth
d = 3

Рассмотрим это на примере

Хеш-таблица = каталог (directory), а каждая ячейка будет указывать на **емкость** (*bucket*) которая имеет определенную **вместимость** (capacity).

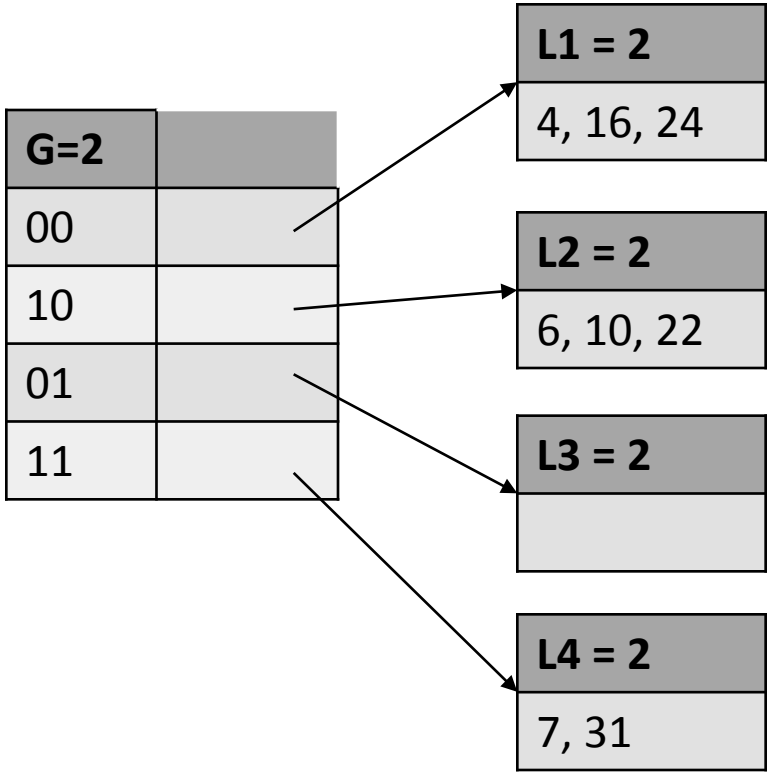
Глобальная глубина показывает сколько младших бит будут использоваться для того чтобы определить в какую емкость следует заносить значения. А из разницы локальной глубины и глобальной глубины можно понять сколько ячеек каталога ссылаются на емкость

Количество ссылающихся ячеек $K=2^{G-L}$ где G — глобальная глубина, L — локальная глубина.

Пусть есть хеш-таблица, где содержатся числа:

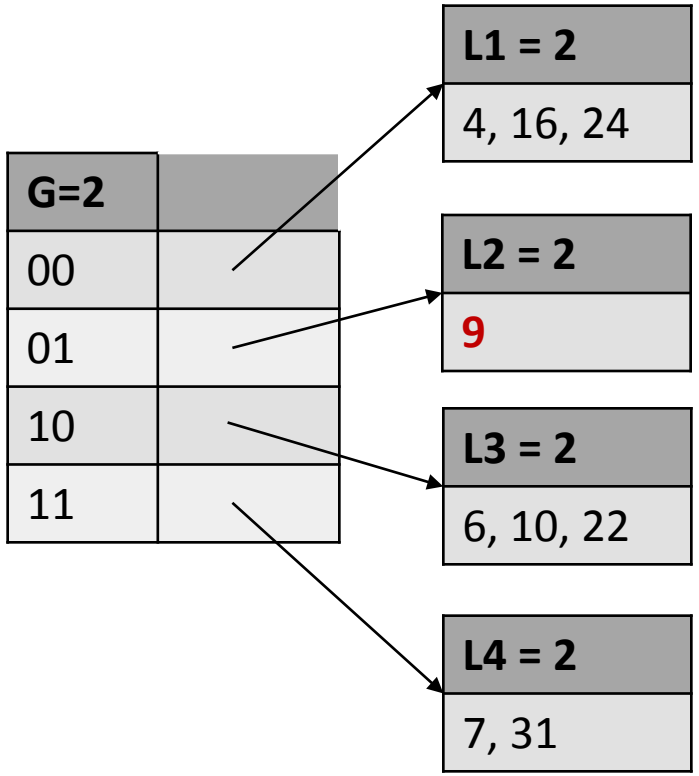
4	100
6	110
7	111
10	1010
16	10000
22	10110
24	11000
31	11111

Глобальная глубина таблицы G=2,
локальные глубины
емкостей L1,Ll2, L3, L4 (=2), вместимость емкостей = 3



Мы хотим добавить в этот каталог числа
9, 20, 26

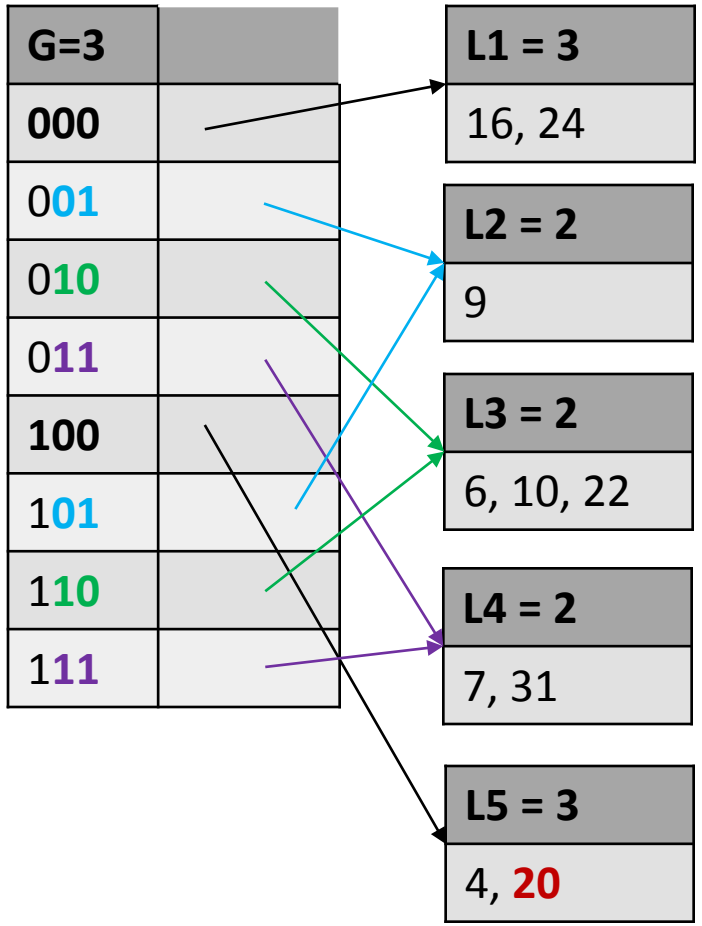
Добавляем 9 (1001)



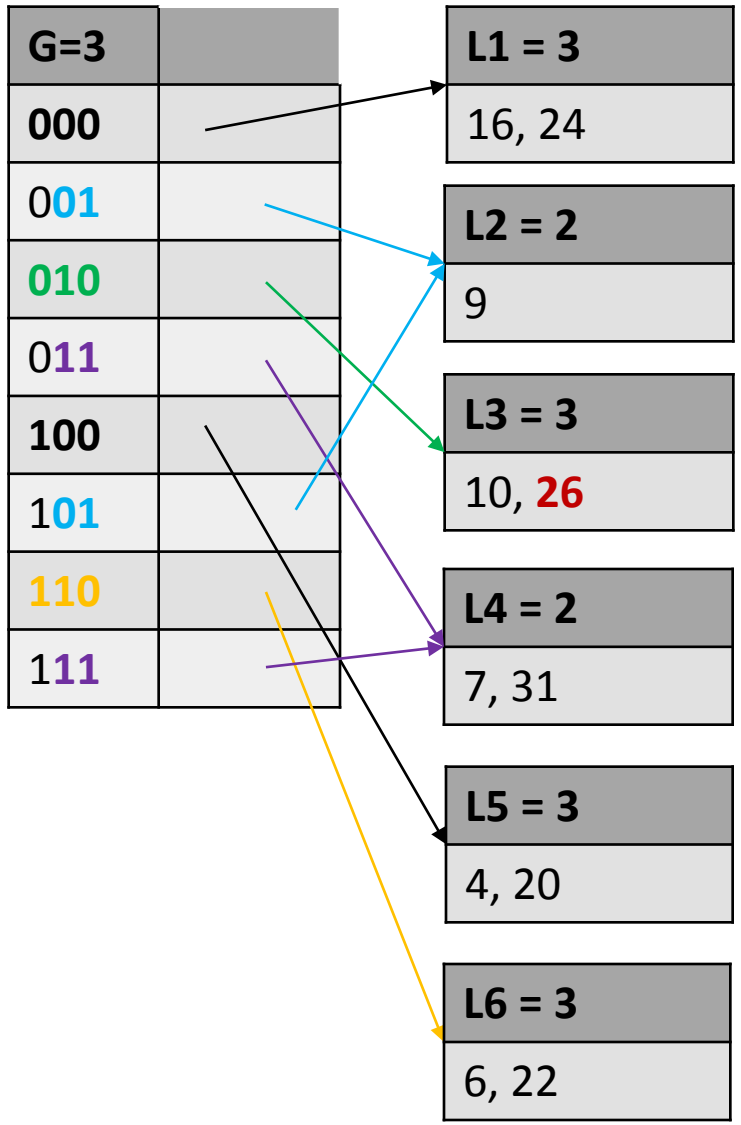
Добавляем 20 (10100)

Емкость 00 заполнена, L1=G

Удваиваем к-во ячеек каталога G=3



Далее на вход поступает 26 (11**010**)



Последние 3 бита
соответствуют емкости #3
Она заполнена.
При этом L3<G
Значит делим емкость на 2,
увеличив локальную глубину,
Перехешируем значения
емкости, распределив
ее по новым емкостям.

Использование

Чаще всего расширяемое хеширование используется в базах данных так как:

- БД большие, при перехешировании всей БД доступа к базе нет на все время пересчета.
- При расширяемом хешировании перехешировать придется только малые группы, что не сильно замедлит работу базы данных.
- расширяемое хеширование хорошо работает в условиях динамически изменяемого набора записей в хранимом файле.

- Т.О., основное достоинство расширяемого хеширования - высокая эффективность, которая не падает при увеличении размера БД, разумное расходование места на устройстве хранения данных, т.к. блоки выделяются только под реально существующие данные, а список указателей на блоки имеет размеры, минимально необходимые для адресации данного кол-ва блоков.
- Расплата: разработчику приходится дополнительно усложнять программный код.

Применение хеширования

- Одно из побочных применений хеширования - создание своего рода слепка, «отпечатока пальца» для сообщения, текстовой строки, области памяти и т. п. Он (слепок) может стремиться как к «уникальности», так и к «похожести» (яркий пример слепка — контрольная сумма CRC).
- В этом качестве применяется в криптографии. Здесь – особенности: скорость вычисления д.б. минимальна, а сложность восстановления максимальна.
- Соответственно необходимо затруднить нахождение другого сообщения с тем же хеш-адресом.
- В российском стандарте цифровой подписи используется хеш-функция (256 бит) стандарта ГОСТ Р 34.11—94.

Хеширование паролей

Например, для шифрования используется 128-битный ключ. Хеширование паролей позволяет запоминать не 128 байт (т.е.256 16-ричных цифр), а некоторое осмысленное выражение, слово или последовательность символов, называющуюся паролем.

Существуют методы, преобразующие произносимую, осмысленную строку произвольной длины – пароль, в указанный ключ заранее заданной длины. Часто для этого используются хеш-функции. В данном случае хеш-функцией (Х-Ф) называется такое математическое или алгоритмическое преобразование заданного блока данных, которое обладает следующими свойствами:

- Х-Ф имеет бесконечную область определения,
- Х-Ф имеет конечную область значений,
- Х-Ф необратима,

Изменение входного потока информации на один бит меняет около половины всех бит выходного потока, то есть результата Х-Ф-ции

Эти свойства позволяют подавать на вход хеш-функции пароли, то есть текстовые строки произвольной длины на любом национальном языке и, ограничив область значений функции диапазоном $0..2^{N-1}$, где N – длина ключа в битах, получать на выходе достаточно равномерно распределенные по области значения блоки информации – ключи.

Еще пример

Пользователь вводит пароль, от него вычисляется хеш, сверяется с тем, что лежит в базе.

Например, хеш-функция MD5 применима к любому тексту и на выходе получает строку фиксированной длины 128 бит. Хеши для возможных паролей вычисляются такими:

MD5("fdgdgdh dfhfgh") = "1a503fd29bc6c64e1ffef9d0266b94e2"
MD5("qwerty") = "a86850deb2742ec3cb41518e26aa2d89"
MD5("Что такое хэш? Просьба написать упрощенно и незамысловато. ") = "7baaa6aab5d700abdd7b2d7d6eb23e9f"

Эта функция просто вычисляется лишь в **одну** сторону. Т. е. посчитать по строке ее хеш легко, а получить из хеша исходную строку - вычислительно сложно и - это соответствие неоднозначно, т.к. могут быть коллизии. Т.О. даже если получить доступ к какой-либо базе данных, паролей на руках не окажется!

- Хеш-функции обычно являются частью механизма электронно-цифровой подписи. Технология ЭЦП является современной заменой обычной бумажной подписи и часто удобнее её.
- Рост известности понятия ЭЦП вполне согласуется с явлением роста электронного документооборота.

- хеш-функции могут использоваться для обнаружения искажений при доставке сообщений по сетевым каналам, в качестве псевдослучайного генератора, для усовершенствования защиты от подмен и подделок документов.
- Хеш-функции могут использоваться в качестве генератора случайных чисел. Правда по скорости они будут уступать обычным.