具体的な講義の項目

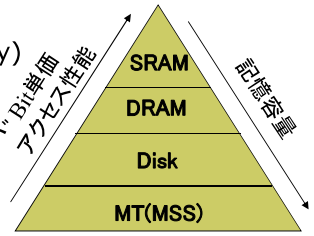
- イントロダクション
- データベースのモデル
- 関係データベース（関係代数、関係論理）
- 正規化
- 問い合わせ言語
- トランザクション処理
- データベースの内部構造
- データベースの問い合わせ処理
- その他（オブジェクト指向DB、アクティブDB等々）

本日の内容

- データの格納
  - 内部スキーマ
  - 磁気ディスク
  - ページの格納
  - レコードの構成
- 索引
  - Btree
  - ハッシング
  - スーパーインポーズドコード

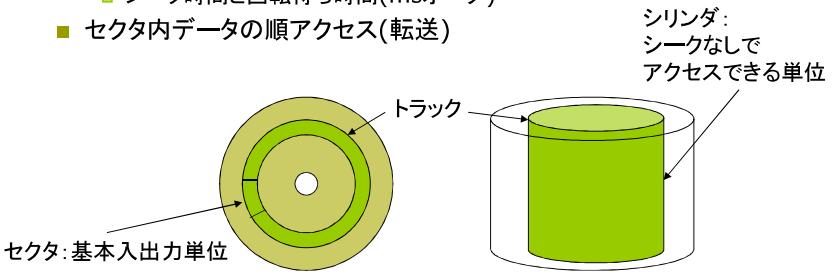
データの格納

- 内部スキーマ
  - システムに依存して変る
- 記憶素子
  - 不揮発性(Non-Volatile)
    - システムが停止（電源を遮断）しても内容を記憶
    - Flash Memory、FeRAM（強誘電体メモリ）、MRAM、PRAM（相変化メモリ）
      → SCM (Storage Class Memory)
  - 記憶容量とコスト(bit 単価)
    - 記憶容量とコストおよび性能のピラミッド



磁気ディスク

- 不揮発性で記憶容量が大きい
- bit 単価が低い
- 磁気テープ(MT)に比べて性能面で優れる
  - セクタ単位のランダムアクセス
    - シーク時間と回転待ち時間(msオーダ)
  - セクタ内データの順アクセス(転送)



最初の磁気ディスク

- RAMAC
  - Random Access Method of Accounting and Control
  - IBM製 （1957年）
  - サイズ
    - 直径24インチ（61センチ）
    - アルミ板50枚
  - 容量
    - 5 MB
  - 記憶密度
    - 0.2 Kb/inch^2

一般的なディスクの特性

- 回転速度：3,000 rpm ～ 15,000 rpm
  - (平均待時間: 10ms ～ 2 ms)
- シーク時間: 0.3 ms (隣接)
  ～ 5 ms (最内←→最外)
- セクタのサイズ: 512B
- ディスクのサイズ: 3.5, 2.5, 1 inch
  - (昔は 14、10.5、8、5.25)
- シリンダ数：6,000 ～ 10,000
- ディスク枚数: 1 ～ 9

ちょっと考えてみよう（9－1）

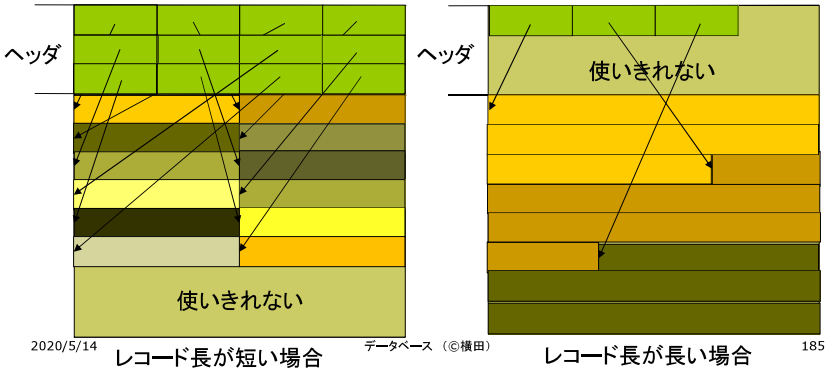
- 平均回転待ち時間2ms、平均シーク時間1msで、データ転送性能が10MB/s (512Bのデータに要する転送時間が0.05ms）の性能のHDDを想定します。
  - 512B毎にトラックが異なるセクタに4096Bのデータが蓄積されていた場合のアクセス時間を算出してみましょう。
  - 4096Bのデータを連続したセクタにデータが蓄積されていた場合のアクセス時間を算出してみましょう。

ページへの格納

- ページ：論理的なアクセスの固まり
  - 普通 4KB [8 セクタ] 程度
    - (場合によって 512B ～ 1MB)
- ページ内のアクセス
  - スロット番号からオフセットを得る
    - 相対レコード番号：Tuple ID (TID)= Page# + Slot#
  - ページ内の有効利用
    - ヘッダ情報とデータを逆から詰める
      - でないとレコードのサイズが変った時に有効利用不可

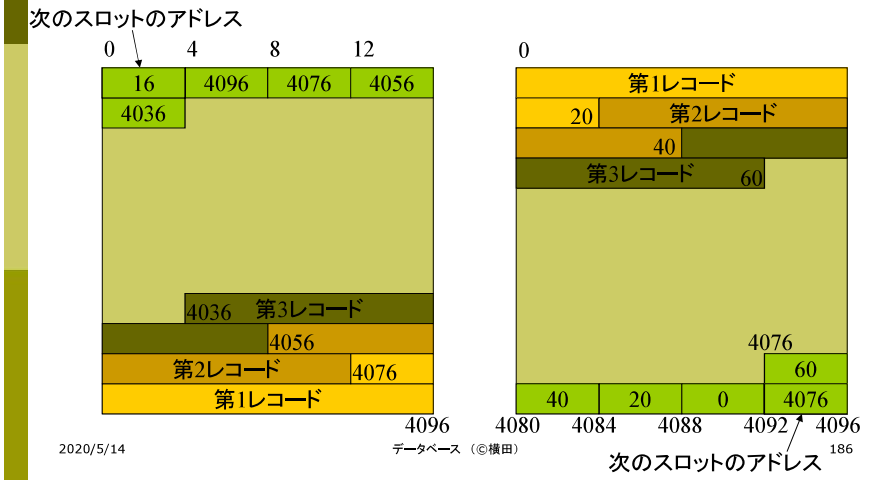
レコードサイズとページへの配置

- レコードはSlot番号でアクセス：ヘッダを持つ
  - ディスクページのサイズは一定
  - 関係データベースのタプル長（レコード長）は変わる



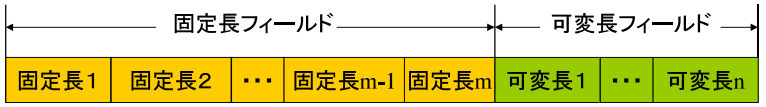
レコード長が短い場合　　レコード長が長い場合

ページの構成

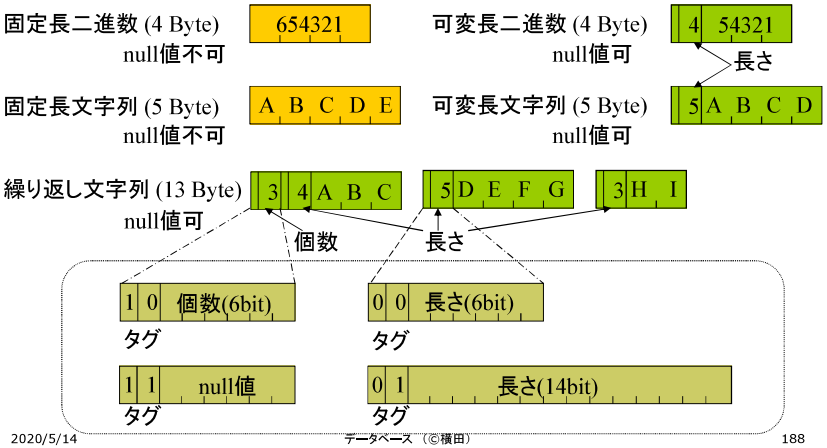


レコードの構成

- 固定長フィールドと可変長フィールドに別けられる
  - 属性の位置と物理的な位置は独立
  - 固定長フィールドの情報（個数、長さ）は別に持つ



データの構造



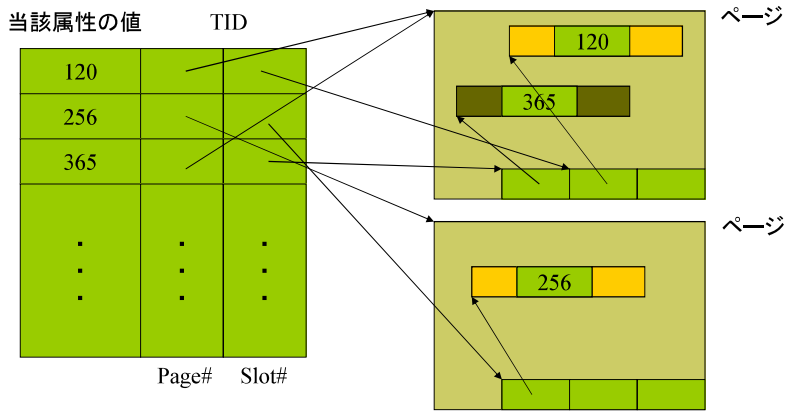
索引（Index）

- 内容から格納場所を知ることよる高速アクセス
  - 書籍の最後にある索引を思い浮かべる
    - キーワードから書かれているページを示す
    - キーワードはソートされている
  - 索引がなければ全ページをアクセスする必要がある
    - 時間がかかる
  - 索引は、内容から TID を示す
    - TID により位置を特定（ページ番号とオフセット）

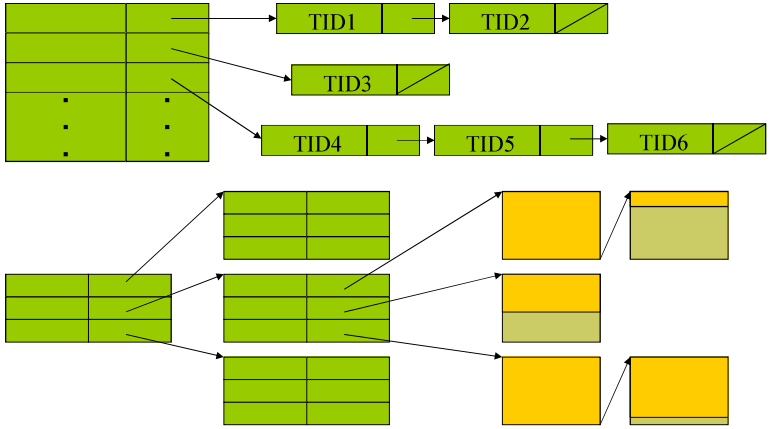
索引の構造

- 転置ファイル(Inverted File)／転置表(Inverted Table)
  - 内容からアドレスと言う意味で逆引き
    - cf. CAM (Content Addressable Memory)
  - 表はソートされ、バイナリサーチ: log(n)
- 転置ファイルもディスクに格納される
  - 転置表自身のサーチコストを減らす
    - 木構造 → バランス木 → Btree (B 氏木)
    - ハッシング
- 多数のキーワードによる検索に対する索引
  - シグニチャードファイル

転置ファイル



転置リストと木構造



Btree

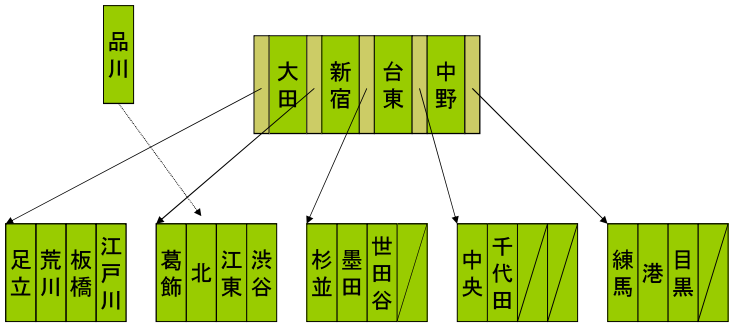
- Bayer によって 1970 年に提案された
  - Bayer's Tree (B 氏木)であって、Balanced Tree ではない
- 条件
  - どのノードの子の個数も k+1 以上である。
    - ただし、根と葉はこの限りでない。
  - どのノードの子の個数も高々 2k+1 である。
  - 根は 2 以上の子を持つ。
    - ただし、根自身が葉である場合はこの限りでない。
  - 根から葉に至る経路の長さは、どの葉でも同じである。

Btreeの特徴

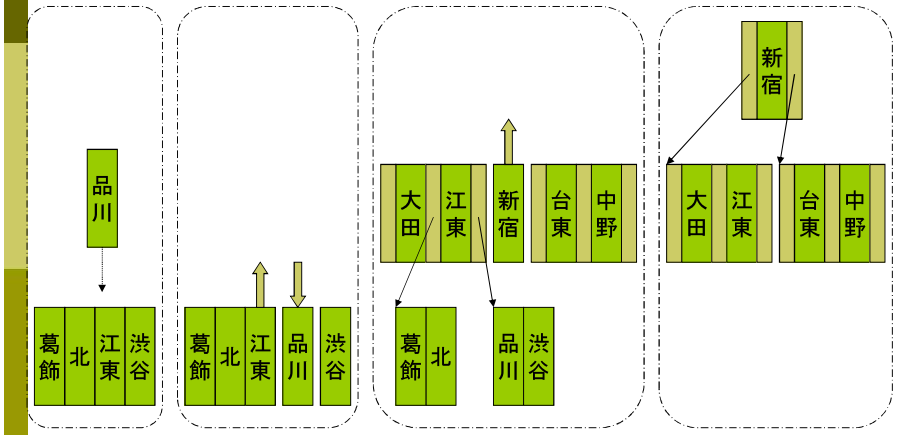
- 木のノードをページに対応
  - どの葉まで行くにも同じ回数のディスクアクセス
- ページの利用効率を規定
  - k レコードないし 2k レコードを含む
    - 利用効率が半分以下にはならない
- 上の条件を満たすために、木の高さが変わる
  - 2k を越えるとノードを分割し、中間の値を親のノードに入れる
  - 親のノードも再帰的に同じ処理をし、根まで行くと高さが増える

Btree の例（k = 2 の場合）

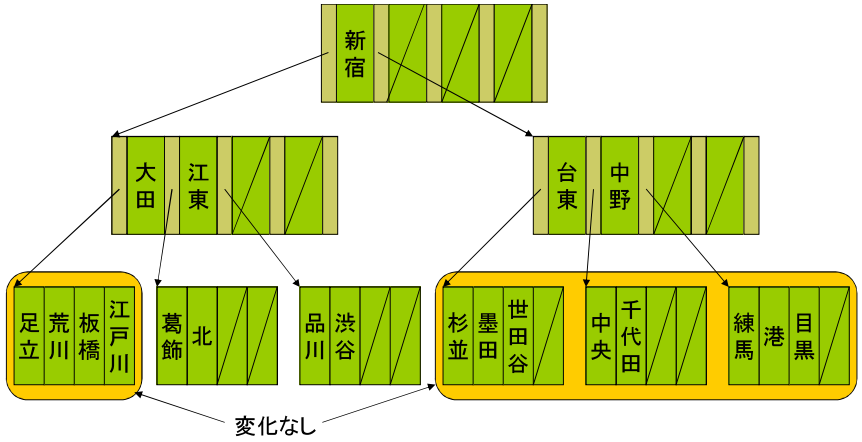
- 簡単化のため、キーのみ示す。



Btree への挿入



挿入後のBtree



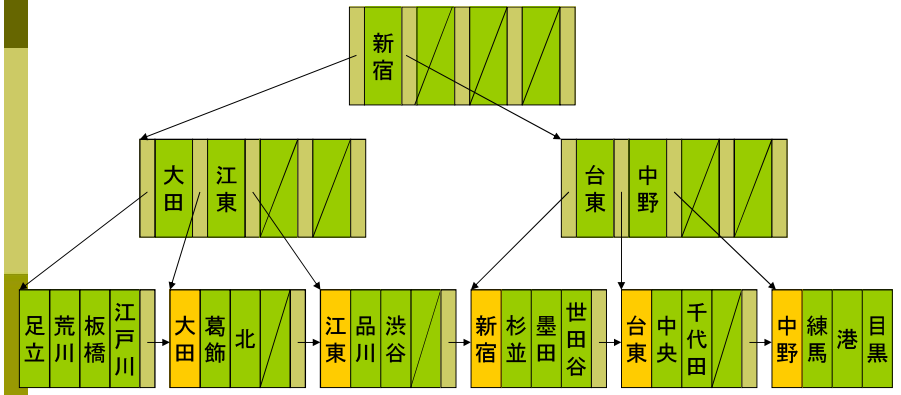
Btree の改良（1）

- B*tree
  - ノードの分割時、自分自身だけでなく、兄弟のノードを見る
  - 兄弟がいっぱいになったら、親と2ノードの分(4K+1+1)を3ノードと2つの親に分る
  - 各ノードは 4k/3 のレコードを含むから、効率は 2/3 以上
- 二次索引
  - レコードの代わりに TID を格納する

Btree の改良（2）

- 実際のデータベースでは隣接するデータを連続してアクセスすることが多い
  - Btreeでは上のノードまで戻る必要がある
- B^+tree
  - データの順アクセスが可能なように改良
    - レコードは葉ノードのみに格納
    - 葉ノードの間をリンクで結合
  - VSAM (Virtual Storage Access Method) : IBM

B^+treeの例



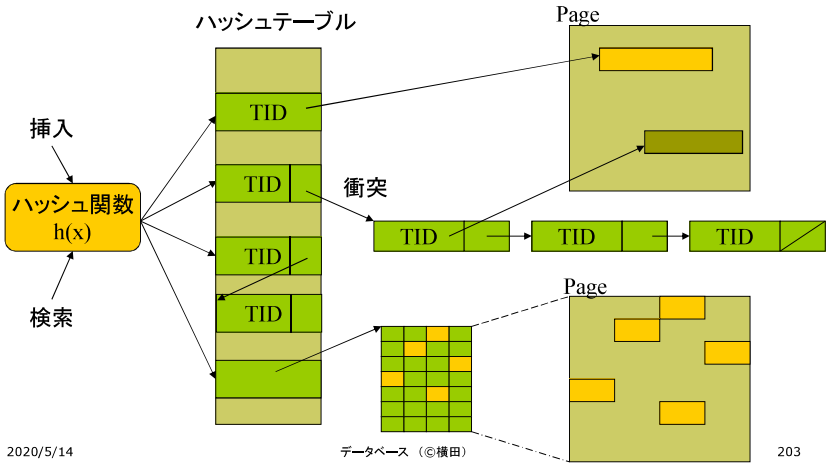
ちょっと考えてみよう(9－2)

- 平均回転待ち時間2ms、平均シーク時間1msで、データ転送性能が10MB/s (512Bのデータに要する転送時間が0.05ms)の性能のHDDを想定します。
  - 高さ3のBtree (あるいはB+tree)の各ノードが512Bのセクタ格納されたとして、ルートから葉までたどる時間はどのくらいでしょうか
    - 実際は、ルート、ルート直下のノードはメモリ上に置かれので、もっと高速にアクセスされます。

ハッシング (Hashing)

- 格納時と検索時に同じハッシュ関数を用いることで、アクセス先を特定
  - ハッシュ表をつくり、TID等の情報を格納
  - ハッシュ関数の例
    - h(X) = X mod n : n はハッシュ表のサイズ
    - X を m 乗したもののビット列の適当な $\log_2$ (n) ビットを使う
    - フォールディング(畳み込み)
  - 衝突(Hash Collision) が発生しない限りは検索コストはO(c)
  - 衝突の対処
    - リストで TID をつなげる／空いてるハッシュエントリーを使う
    - bitmap を使う
    - 拡張ハッシュ (Extendible Hash)

ハッシングのイメージ



拡張ハッシュ(1)

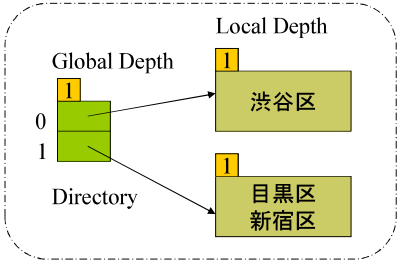
- ハッシュ表のサイズを動的に変化させる(拡張)
  - 2のべき乗で変化(bit 数に対応) (リニアハッシュもある)
  - Depth : ハッシュ値の何 bit 見るかを示す
  - Global Depth と Local Depth の差をdとすると 2^d 分の矢印が集まる

例:

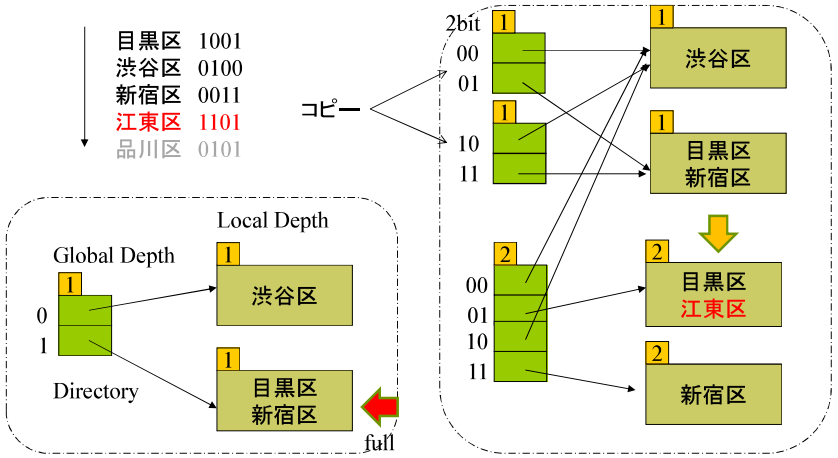
ハッシュ値

目黒区 1001
渋谷区 0100
新宿区 0011
江東区 1101
品川区 0101

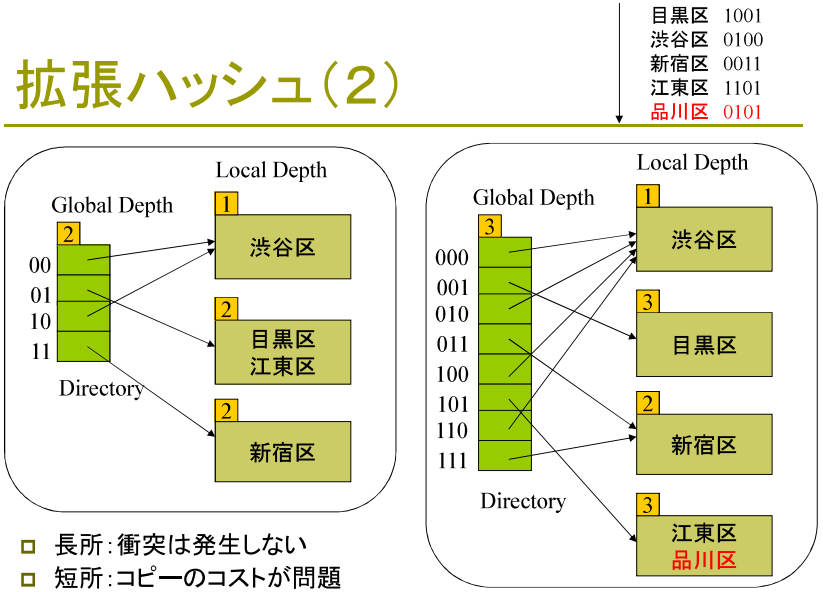
ビット列のLSBもしくはMSBから



拡張ハッシュ（1‘）



拡張ハッシュ（2）



- 長所：衝突は発生しない
- 短所：コピーのコストが問題

ちょっと考えてみよう（9－3）

- 次に来るデータが新宿区と同じブロックに入るとすると、そのデータに対するハッシュ値はどのような値が想定されるでしょう。

スーパーインポーズドコードによる索引

- テキストデータベースのキーワード検索等に使われる

キーワード	シグニチャ
目黒区	00010001
渋谷区	01001000
新宿区	00000101
江東区	11000000
品川区	01100100

検索対象事に
シグニチャのORを取る

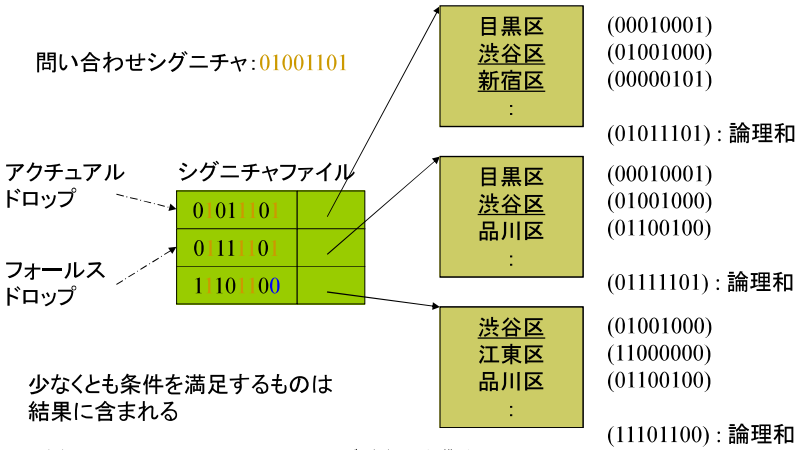
検索要求（AND）

渋谷区	01001000
新宿区	00000101

問い合わせシグニチャ

論理和 01001101

シグニチャファイル



ちょっと考えてみよう（9－4）

- フォールスドロップを減らすにはどうするとよいでしょう。