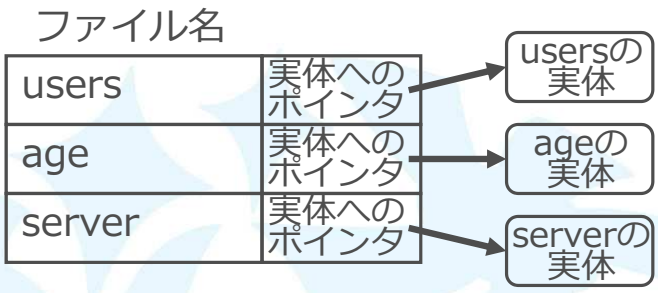
ファイルの名前付け
3
ディレクトリ情報の管理

名前を付ける

- ファイルに名前を付けて
  名前でアクセスしたい
  - 名前がないと？
    ディスク上のブロック番号でアクセスする？
    - 「２台目のディスクのブロック237をread」
      ブロック番号を覚えるのはやっかい
      そもそも１つのデータが１ブロックに収まらない
      ので複数ブロックを覚えておく必要がある？
- ⇒ とにかく名前を付けよう

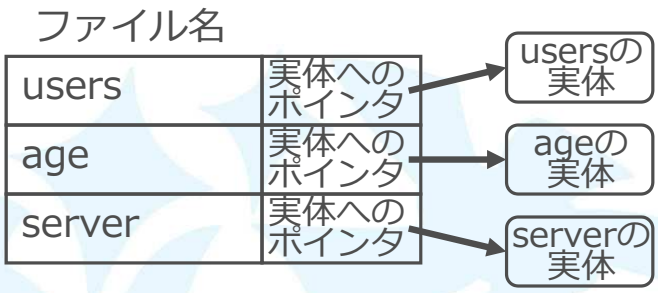
ディレクトリ情報の管理

- 案①：表に並べて持つ



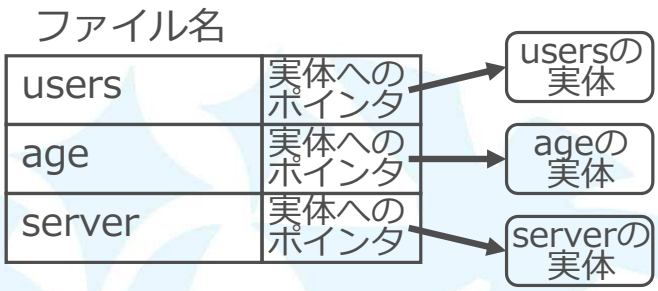
ディレクトリ情報の管理

- 案①：表に並べて持つ
  - ファイル名での検索は先頭から順にスキャン



ディレクトリ情報の管理

- 案①：表に並べて持つ
  - ファイル名での検索は先頭から順にスキャン
    ⇒　後のものは時間かかる

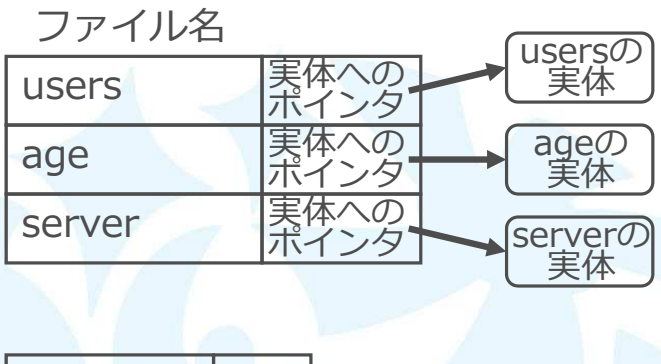


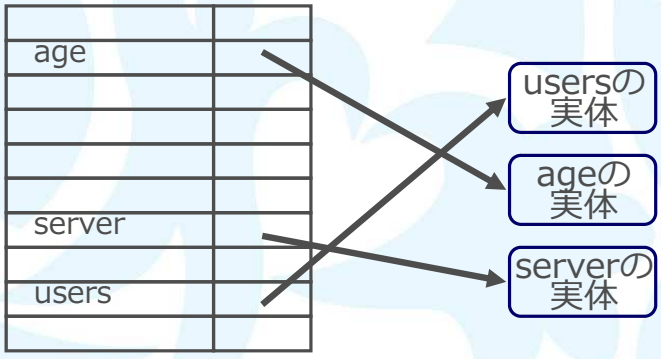
ディレクトリ情報の管理

- 案①：表に並べて持つ
  - ファイル名での検索は先頭から順にスキャン
    ⇒　後のものは時間かかる

早い方法として

- 案②：名前＝アドレスにする



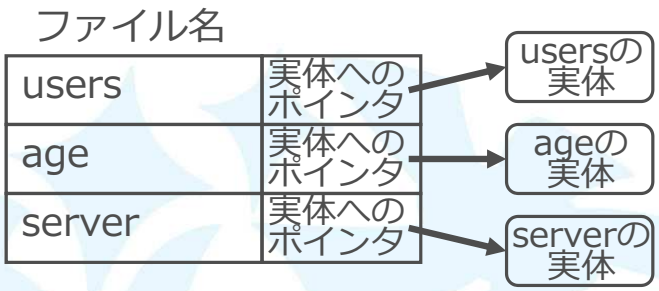


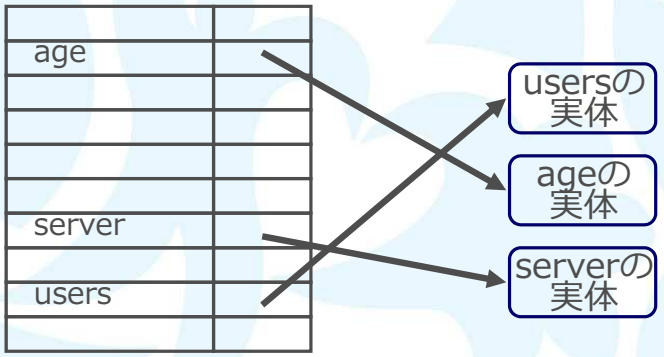
ディレクトリ情報の管理

- 案①：表に並べて持つ
  - ファイル名での検索は先頭から順にスキャン
    ⇒　後のものは時間かかる

早い方法として

- 案②：名前＝アドレスにする
  - 名前の文字コードを表の行番号と対応する



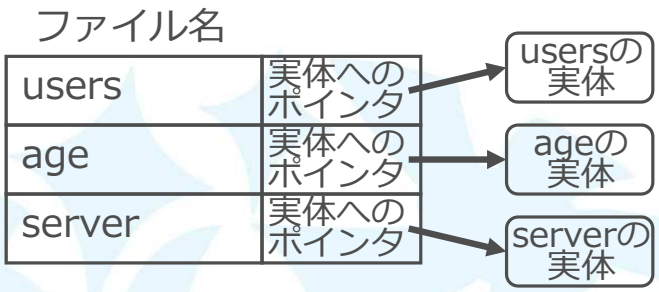


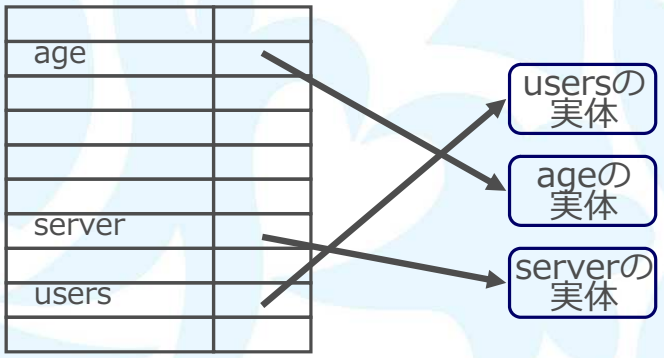
ディレクトリ情報の管理

- 案①：表に並べて持つ
  - ファイル名での検索は先頭から順にスキャン
    ⇒　後のものは時間かかる

早い方法として

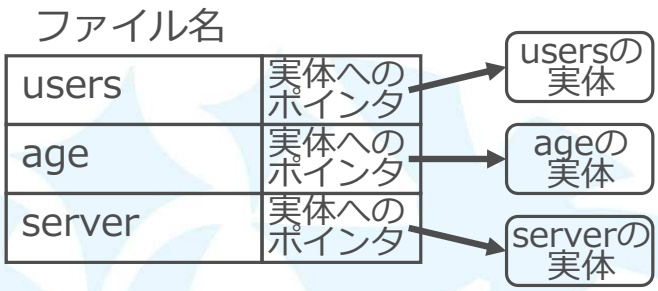
- 案②：名前＝アドレスにする
  - 名前の文字コードを表の行番号と対応する
    ⇒名前から直接表が引ける





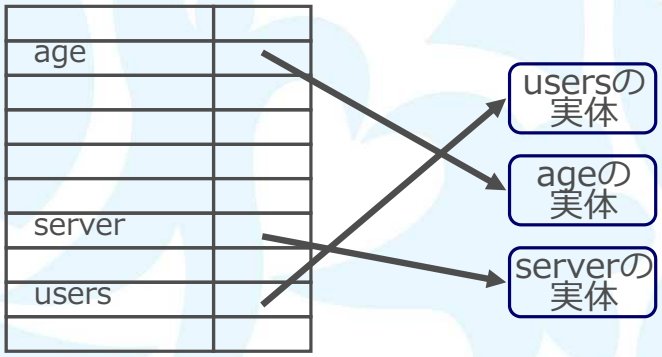
ディレクトリ情報の管理

- 案①：表に並べて持つ
  - ファイル名での検索は先頭から順にスキャン
    ⇒　後のものは時間かかる



早い方法として

- 案②：名前＝アドレスにする
  - 名前の文字コードを表の行番号と対応する
    ⇒名前から直接表が引ける
  - すかすかになってしまう

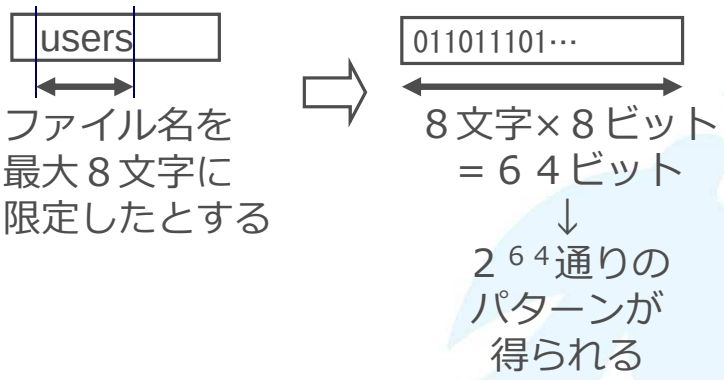


名前をアドレス(行番号)にするには

ファイル名を
最大８文字に
限定したとする

８文字×８ビット
＝６４ビット

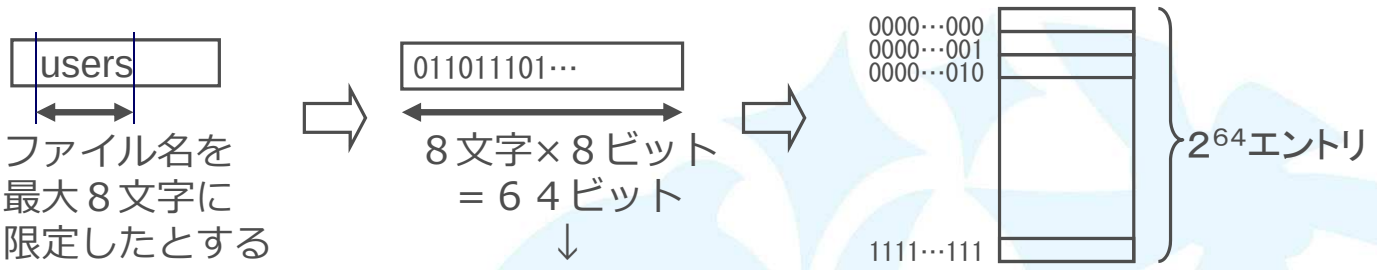
名前をアドレス(行番号)にするには



ファイル名を
最大８文字に
限定したとする

８文字×８ビット
＝６４ビット
↓
２６４通りの
パターンが
得られる

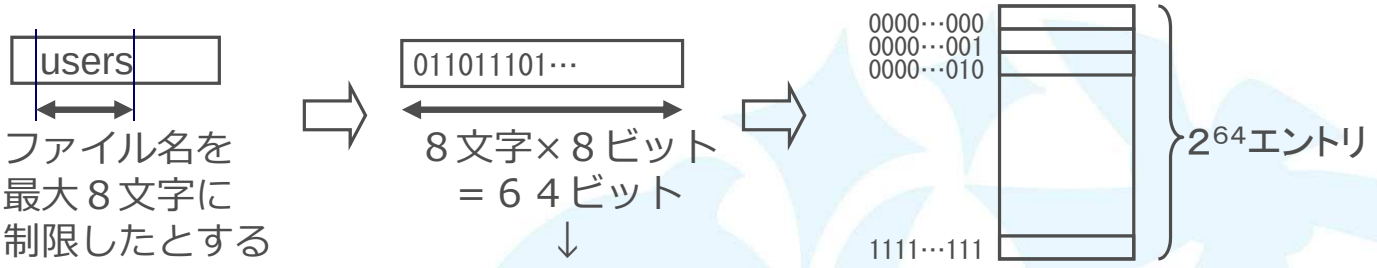
名前をアドレス(行番号)にするには



ファイル名を
最大８文字に
限定したとする

８文字×８ビット
＝６４ビット
↓
２６４通りの
パターンが
得られる

名前をアドレス(行番号)にするには

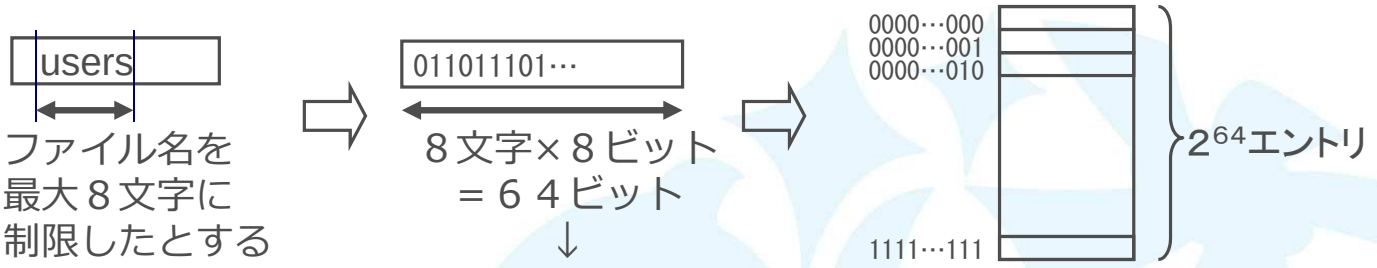


ファイル名を
最大８文字に
制限したとする

８文字×８ビット
＝６４ビット
↓
２６４通りの
パターンが
得られる

2^{64}はおよそ16×10^{18}だが
実際に使うのは数千個とか

名前をアドレス(行番号)にするには



ファイル名を
最大８文字に
制限したとする

８文字×８ビット
＝６４ビット
↓
２６４通りの
パターンが
得られる

2^{64}はおよそ16×10^{18}だが
実際に使うのは数千個とか

⇩

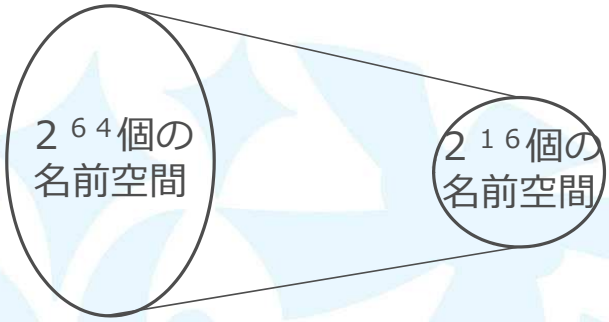
「すかすかになる」

で、ハッシュを使う

- ハッシュとは

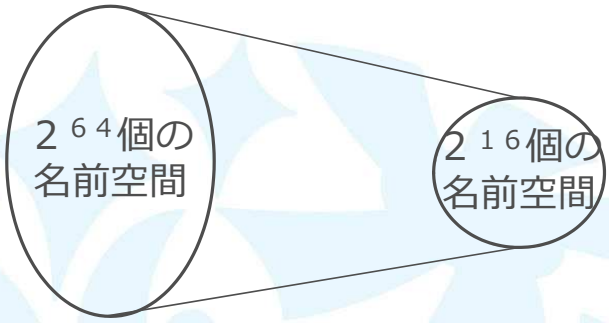
で、ハッシュを使う

- ハッシュとは
  - 広い空間（例：2^{64}）を小さい空間（例：2^{16}）にマップ



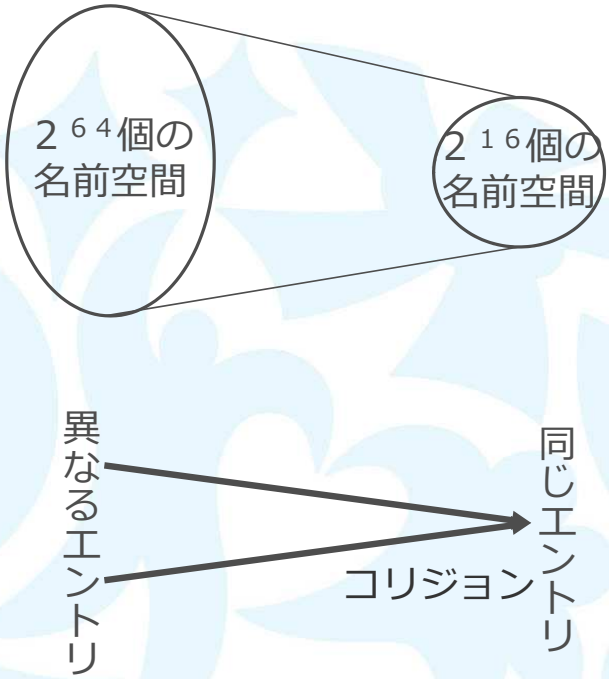
で、ハッシュを使う

- ハッシュとは
  - 広い空間（例：2^{64}）を小さい空間（例：2^{16}）にマップ
  - 適当な関数でマップ



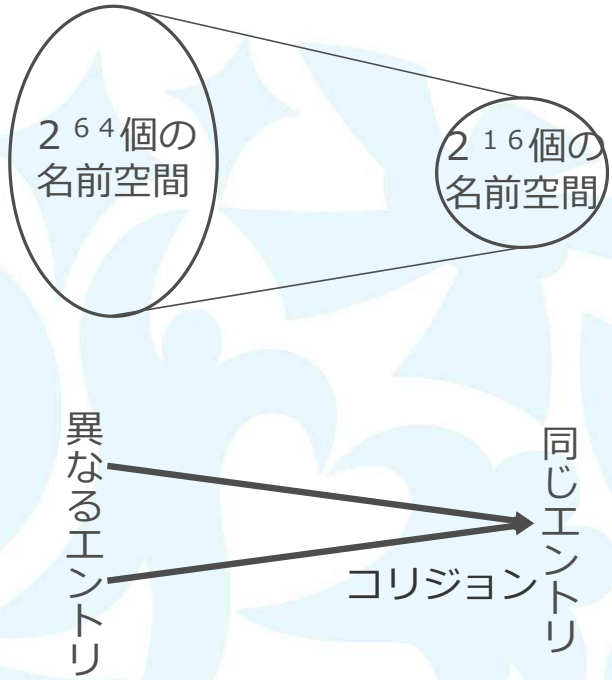
で、ハッシュを使う

- ハッシュとは
  - 広い空間（例：2^{64}）を小さい空間（例：2^{16}）にマップ
  - 適当な関数でマップ
- 結果が重なることもある
  - コリジョン（衝突）



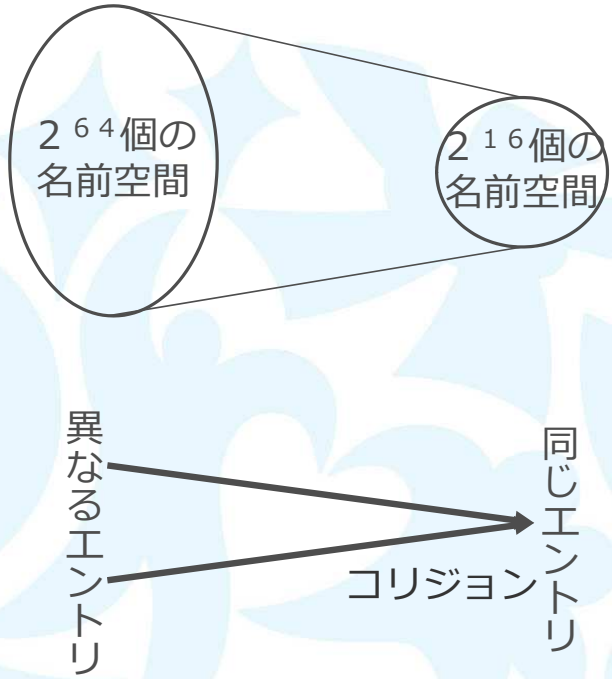
で、ハッシュを使う

- ハッシュとは
  - 広い空間（例：2^{64}）を小さい空間（例：2^{16}）にマップ
  - 適当な関数でマップ
- 結果が重なることもある
  - コリジョン（衝突）
  - 仕方ないので「次候補」



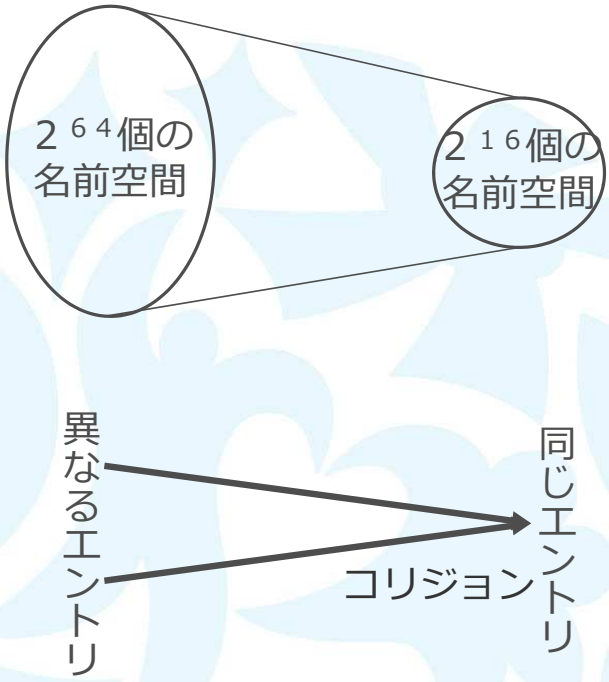
で、ハッシュを使う

- ハッシュとは
  - 広い空間（例：2^{64}）を小さい空間（例：2^{16}）にマップ
  - 適当な関数でマップ
- 結果が重なることもある
  - コリジョン（衝突）
  - 仕方ないので「次候補」
    - リンクによって繋ぐ



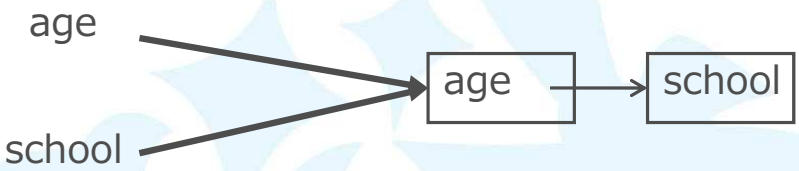
で、ハッシュを使う

- ハッシュとは
  - 広い空間（例：2^{64}）を小さい空間（例：2^{16}）にマップ
  - 適当な関数でマップ
- 結果が重なることもある
  - コリジョン（衝突）
  - 仕方ないので「次候補」
    - リンクによって繋ぐ
    - 次に並べる



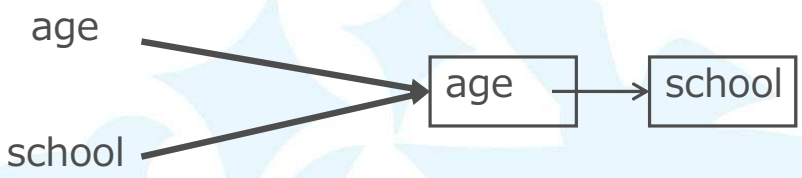
衝突時の「次候補」

1. リンクで繋いでおく方法

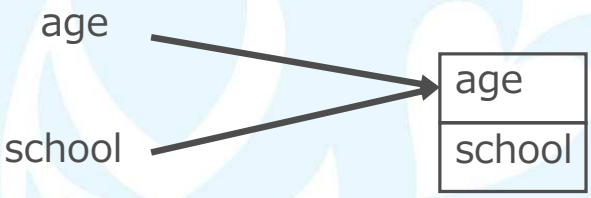


衝突時の「次候補」

1. リンクで繋いでおく方法



2. 次のエントリに書いておく方法



ハッシュ関数の例

- ハッシュ関数の簡単な例を考えてみる

ハッシュ関数の例

- 文字列sに対して
  - 各文字s[i]（のコード値）を足し合わせて
  - 和を97（これは素数）で割った剰余を答とする
    （つまり、97エントリーに分類される）

```
unsigned int hash(char *s) {
  unsigned int h = 0;
  for (i=0; i<length(s); i++) {
    h = h + s[i];
  }
  return h % 97;     %は剰余演算子
}
```

ウィキペディア「ハッシュ関数」より改変

一般論としてハッシュは

一般論としてハッシュは

- 衝突が無ければ
  - とにかくサーチがいらない
    ハッシュの計算だけで（表の行が決まって）
    アクセスできる
  - だから、アクセス時間は表の長さに依存しない

一般論としてハッシュは

- 衝突が無ければ
  - とにかくサーチがいらない
    ハッシュの計算だけで（表の行が決まって）
    アクセスできる
  - だから、アクセス時間は表の長さに依存しない
- 空間を縮めるので、スカスカはなくなる

一般論としてハッシュは

- 衝突が無ければ
  - とにかくサーチがいらない
    ハッシュの計算だけで（表の行が決まって）
    アクセスできる
  - だから、アクセス時間は表の長さに依存しない
- 空間を縮めるので、スカスカはなくなる
- 衝突が起きるときの始末が必要

一般論としてハッシュは

- 衝突が無ければ
  - とにかくサーチがいらない
    ハッシュの計算だけで（表の行が決まって）
    アクセスできる
  - だから、アクセス時間は表の長さに依存しない
- 空間を縮めるので、スカスカはなくなる
- 衝突が起きるときの始末が必要
  - しかも、衝突時は余分な処理時間がかかる
  - なるべく衝突の起きにくいハッシュ関数がよい
    素数による剰余を使うことがある（多い？）

一般論としてハッシュは

- 結構いろいろなところで使われる
  - 表のアクセスの高速化

一般論としてハッシュは

- 結構いろいろなところで使われる
  - 表のアクセスの高速化

- ハッシュ関数をさまざまな工夫

一般論としてハッシュは

- 結構いろいろなところで使われる
  - 表のアクセスの高速化
- ハッシュ関数をさまざまな工夫
  - 計算が簡単で
  - うまく圧縮できて（結果側の集合が欲しい大きさで）
  - 衝突が少ない

一般論としてハッシュは

- 結構いろいろなところで使われる
  - 表のアクセスの高速化
- ハッシュ関数をさまざまな工夫
  - 計算が簡単で
  - うまく圧縮できて（結果側の集合が欲しい大きさで）
  - 衝突が少ない
    - 入り側の発生確率が一様でないこともあり、
      その時は一様なマッピングがよいわけではない

ディレクトリ情報の
管理の考え方が
理解できましたか？

次へ