10. ハッシュ法†

井田哲雄††

1. ハッシュ法とは

ハッシュ法とは，探索法の一つである．ある鍵Kが与えられた時，Kに依存して生成する番地列によって探索場所を決定し，Kを含むレコードを探索する手法をハッシュ（探索）法と呼ぶ．ハッシュ（hash）とは鍵から番地を生成する操作を指す．本来は，切り刻む，混合するという意味である．鍵から番地への変換を行う関数をハッシュ関数という．

ハッシュ法の基本的な考え方は，ハッシュ関数列を適当に選択することにより，レコードを記憶領域内になるべく衝突のないよう分散して格納し，探索時に，少ない記憶参照回数で，目的とするレコードを探索しようとするものである．

ハッシュ法の考察は，

① ハッシュ関数列（ハッシュ番地列生成アルゴリズム），

② 生成された番地列による目的レコードへのアクセスアルゴリズムと，アクセスアルゴリズムが使用するデータ構造（ハッシュ表），

③ レコードの登録，削除，再配置等のハッシュ法におけるサブアルゴリズム，

の3点にわたって行う必要がある．これら3点は互に密接な関係がある．

2. ハッシュ法の適用分野

ハッシュ法の適用分野としては，次のものが代表的である．

① コンパイラ，アセンブラにおける記号表操作，

② データベースの検索，

③ 構造データ（例えば，リスト，集合）の高速同定および部分的構造の共有．

①，②におけるハッシュ法の適用手法は自明であろう．③の場合は，レコードが鍵のみよりなり，しかも，鍵が構造データの部分構造を指すポインタであることに着目する．

3. ハッシュ表

ハッシュ関数によって生成される番地によってアクセスされる記憶領域をハッシュ表と呼ぶ．鍵を含むレコードが表の内部に格納される場合，この表を直接ハッシュ表と呼ぶ．レコードの番地が格納される場合，間接ハッシュ表と呼ぶ．直接，間接ハッシュ表いずれの場合も，生成された一つの番地でアクセスされるレコード格納用領域の一分割単位をバケット（bucket）と呼ぶ．バケットの大きさとは，そのバケットに格納可能なレコードの総数をいう．**図-1**にハッシュ表の構成を示す．

4. ハッシュ法の分類

ハッシュ法は，ハッシュ表の構造，ハッシュ番地生成アルゴリズムという2つの観点から分類できる．（**図-2** 参照）

4.1 ハッシュ表の構成

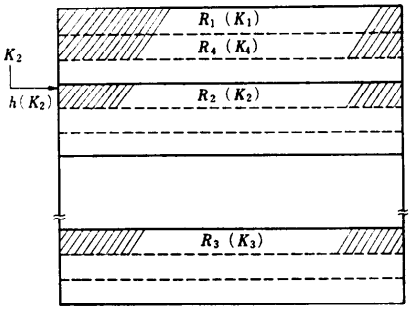
直接あるいは間接ハッシュ表のいずれを用いるかによって，直接ハッシュ法あるいは間接ハッシュ法に分類される．前者はハッシュ表の大きさが比較的小さい場合によく用いられる．後者は，対象となるデータベースの規模が大きい場合あるいは，鍵の構造が一様でない場合に用いられる．

4.2 ハッシュ番地生成アルゴリズム

ハッシュ番地生成アルゴリズムは，鍵の衝突の対処法によって，まず2つに分類できる．ここで鍵の衝突（collision）というのは，ハッシュされた番地のバケットがすでに詰っており，そのバケットに新たなレコードの格納ができない状態をいう．衝突を解消するには，①あふれ領域を設けてそこにレコードを格納し，ハッシュ表よりポインタで接続する，あるいは②衝突状態が起らなくなるまで，次々に新たなハッシュ番地

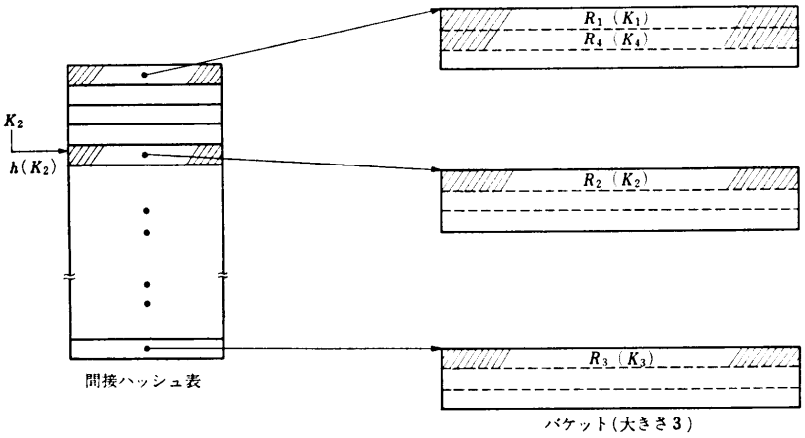
† Hash methods by Tetsuo IDA (Institute of Physical and Chemical Research).

†† 理化学研究所



(a)　直接ハッシュ表

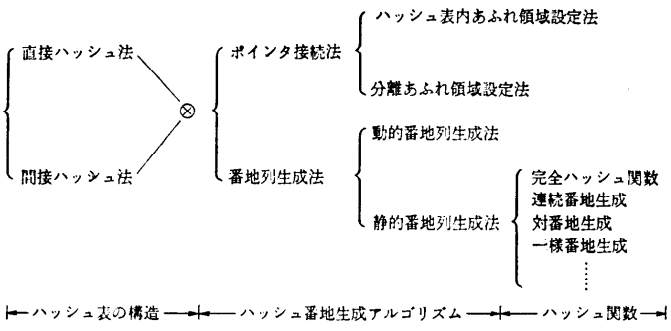
この図はバケットの大きさ3の直接ハッシュ表である．$R_i(K_i)$ で鍵 K_i を含んだレコード R_i を示す．直接ハッシュ表では，あらかじめ割当てられた連続記憶領域を通常は用いる．



(b)　間接ハッシュ表

この図では間接ハッシュ表として，一次元連続領域を示すが，必ずしも，連続領域である必要はない，木構造のものも提案されている．

図-1　ハッシュ表の構成



注）⊗は，左右の分類が直積の関係にあることを示す

図-2　ハッシュ法の分類

を生成する，の2通りの方法がある．①の方法をポインタ接続法(chaining)，③の方法を番地列生成法(open addressing)と呼ぶ．ポインタ接続法では，あふれ領域をハッシュ表の中に設定するか，あるいは別個にあふれ領域を設定するかによって，さらに2つに分類できる．番地列生成法では，衝突が生じた時のハッシュ表の占有率によって，ハッシュ関数を動的に変化させるものと，そうでない（固定した）方法がある．さらにハッシュ関数をどのようなものにとるかによって，ハッシュ法は細分される．

5. ハッシュ法・サブアルゴリズム

具体的なハッシュ法として，最も単純な，バケットの大きさ1の直接ハッシュ法のうち番地列生成法を例にとって説明する．ハッシュ表を $H[0:M-1]$，鍵を K，ハッシュ関数列を $h_0, h_1, \ldots, h_{M-1}$ とする．ハッシュ表の各バケットは3つの状態をもつ．空白状態，削除状態，および鍵が格納された占有状態である．この3つの状態を示すために，空白状態，削除状態を仮想的鍵とみなし，各々 e, d と表わす．そして，取扱われるすべての鍵 K は $K \neq e$ かつ $K \neq d$ であるとする．

5.1 ハッシュ表の初期化*

```
for i=0 to M−1
    H[i]←e
```

5.2 鍵の挿入

アルゴリズム I

```
for i=0 to M−1
    if H[h_i(K)]=d or H[h_i(K)]=e
    then H[h_i(K)]←K
        return
error ("hash table full")
```

5.3 鍵の登録

アルゴリズム I*

```
del←false
for i=0 to M−1
    if H[h_i(K)]=d
        if del then loop
        else j←i
            del←true
    else if H[h_i(K)]=e
        then if del then H[h_j(K)]←K
            else H[h_i(K)]←K
            return
        else if H[h_i(K)]=K
            then error ("key already exists")
error ("hash table overflow")
```

アルゴリズム I* は挿入すべき鍵の存在の有無を確認し，鍵の登録を行っている．これは，機能的には **5.5** に述べる探索アルゴリズムと **5.2** の鍵の挿入アルゴリズムを組み合せたものと考えられる．

5.4 鍵の削除

アルゴリズム D

```
for i=0 to M−1
    if H[h_i(K)]=K
    then H[h_i(K)]←d
        return
    else if H[h_i(K)]=e
        then error ("key does not exist")
```

5.5 鍵の探索

アルゴリズム S

```
for i=0 to M−1
    if H[h_i(K)]=K
    then return ("successful search")
    else if H[h_i(K)]=e
        then return ("unsuccessful search")
```

5.6 鍵の再配置

応用によっては，次のような理由で，鍵をハッシュ表内に再配置することが必要とされることがある．

（i） 鍵の削除，登録の繰り返しにより，削除状態のバケット数が増大し，アルゴリズムの探索効率が低下するため，削除状態を除去したい．

（ii） ポインタ値が鍵の一部になっている場合，ポインタの指すデータがガーベジコレクション等により移動し，鍵の値が変化してしまった時には，この鍵を一度削除し，再挿入が必要となる．

対応策としては，ハッシュ表を新たに作り，そこにすべての鍵を新たに挿入するのが最も簡単でかつ，再配置後の探索効率がよい．表内再配置を行うアルゴリズムはいくつか考えられるが，次のアルゴリズムが，補助ビットも必要とせず簡単である．

アルゴリズム R

```
for i=0 to M−1
    if H[i]=d then H[i]←e
for i=0 to M−1
    if H[i]≠e then K'←H[i]
                   H[i]←d
                   insert K' using
                     algorithm I
```

5.7 アルゴリズムの拡張および単純化

以上の例を一歩すすめたバケットの大きさ2以上の場合への拡張は読者に委ねる．この時，バケットの削除状態とは，バケットに格納された鍵の少なくとも一つは d であり，しかも e を含まない状態であること，空白状態とは，バケット内の鍵の少なくとも一つは e である状態を指すことに注意する．

バケット数2以上のハッシュ表は，記憶領域がディ

---

* 簡単のため，以下では鍵のみよりなるレコードを考える．

スクのような回転媒体である時のみならず，主記憶に記憶階層をもった大型計算機でのハッシュ法でも有益である．

鍵の削除が起らない場合，例えばある種のコンパイラにおける記号表処理では，バケットの削除状態を考える必要がない．このとき **5.1～5.6** のアルゴリズムは著しく単純化される．

6. ハッシュ関数

鍵の集合を$\boldsymbol{K}$とする．ハッシュ関数 $h_i(i=0,1,\cdots)$ は $\boldsymbol{K}$ から，整数の集合 $\boldsymbol{N}=\{0,1,\ldots,M-1\}$ の上への写像である．h_0 が $\boldsymbol{K}$ から $\boldsymbol{N}$ への一対一写像である時，h_0 は完全ハッシュ関数という．この時 $h_1,\cdots$ は不要である．しかし，このような h_0 は実用上作ることが困難であるか，あったとしても M を非常に大きくとる必要があることが多い．

ハッシュ関数の性質としては，

① h_i $i=0,1,\cdots$ が一様であること（特に h_0），

② h_i と h_j $i\neq j$ が独立であること，

③ $h_i(K)$ の計算が容易であること，

が望まれる．

①は直感的には，すべての鍵が，ハッシュ表に同一密度で分散配置されることである．②は，ある時点（例えば $h_i(K)$ の計算で）鍵の衝突が起った時，以降のハッシュ番地列 $h_{i+1}(K),\cdots$ の値が $h_i(K)$ に依存しないことが要求されることを意味する．③は，ハッシュ番地の計算時間は表探索に費す時間との関連でもって考えなくてはならないことを含蓄する．表がいかなる記憶媒体にとられるかによって，計算に費やしうる時間の制約は異なる．①～③の条件を完全に満たすハッシュ関数列を選ぶのは，実際には，困難である．応用によって適当なハッシュ関数列を選ぶことが多い．以下にいくつかの例を示す．

（i） $M=2^n$ の時

$K=k_0k_1\cdots k_l$ といった文字列のとき

ビット演算により $k_0,k_1,\cdots,k_l$ を混ぜ合わせ．n ビットに縮退させる．この結果を h_0 とする．

Δh として，$k_0,k_1\cdots k_l$ を h_0 の作成とは異なった仕方で混ぜ合せ，しかも結果が奇数となるような写像を選ぶ．そして，

$$h_{i+1}(K)\leftarrow(h_i(K)+\Delta h(K))\wedge 2^n$$

となる h_i $i=2,3,\cdots$ を選ぶ．

（ii） M を素数にとる．

$h_0(K)=K \bmod M$

$\Delta h(K)=K \bmod M'$，ここで M' として M より小さい，M に近い素数をとる

$h_i(K)$ は次のように計算する．

$h_i\leftarrow h_{i-1}(K)+\Delta h(K)$ $i=1,2,\cdots$

if $h_i>M$ then $h_i\leftarrow h_i-M$

（iii） バケットが二次記憶にある時のハッシュ関数

この時は，通常バケットとハッシュ表（普通は間接ハッシュ表）のアクセス時間比は数百倍程度あるため，バケットへのアクセス回数を最少限に抑えるような h_i を選ぶことが最も重要である．

近年，間接ハッシュ表の構造，ハッシュ関数列の選択に，動的ハッシュ法の考えに基づいて，検討を加えたいくつかのアルゴリズムが発表されている．

7. ハッシュ法の性能

ハッシュ法の性能というのは，鍵がハッシュ表の中に存在することが判明するまでに要するハッシュ番地列の平均生成回数（したがってハッシュ表の探索回数に等しい），と不存在を確認するまでに要するハッシュ番地の平均生成回数によって示される．前者を平均成功探索回数，S，後者を平均不成功探索回数，U，と呼ぶ．S, U は，ハッシュ表の占有率 α によって決まる．直接ハッシングを例にとると，$\alpha=N/(Mb)$ である．ここで，N は鍵の総数，b はバケットの大きさである．各種アルゴリズムの性能比較を行う時に注意しなくてはならない点は，単に表内の鍵の総数と，表の鍵格納容量との比を占有率とするのではなく，表の管理のために必要な補助領域をも考慮にいれて，占有率を算出することである．この場合，α 表の使用率といったほうが適切であろう．$S(\alpha), U(\alpha)$ のもつ共通した性質として，α の 1 の近傍における，急激な上昇がある．つまり，表が詰ってくると，ハッシュ法は実質的に逐次探索法に転化してしまう．したがって表の占有率をある値以下に抑えて，表を使用する配慮が必要である．

参 考 文 献

参考文献は，教科書，解説，論文に分類し，著者の名前をキーとして，アルファベット順に並べた．これらの文献は包括的なものではない．ハッシュ法に関して代表的と思われるものを取り上げた．教科書，解説については，ハッシュ法との関連について，注釈を加えた．論文については，内容を示すキーワードを付した．これらのキーワードは，内容を完全には，示していないので，あくまでも一つの目安として使われるこ

とを希望する.

ハッシュ法に関する教科書

[GRI] Gries, D.: "Compiler Construction for Digital Computers" John Wiley and sons, Inc., New York (1972).
コンパイラ作成におけるハッシュ法の適用手法.
[KNU] Knuth, D.E.: The art of computer programming, Vol. 3, Sorting and searching, Addison-Wesley Publishing Co. (1975).
6.4 節全部がハッシュ法の説明に費やされている.
ハッシュ法の入門書として最適. 1973年以前のハッシュ法に関する文献を知る上でも有益.
[KOH] Kohonen, T.: Associative memory, Springer-Verlag, Berlin (1977).
ハッシュ法を用いた連想記憶の実現手法.

ハッシュ法に関する解説

[ML] Maurer, W.D. and Lewis, T.G.: "Hash table methods", Computing Surveys, Vol. 7, No. 1, pp. 5-18 (Mar. 1975).
ハッシュ法に関する基本的な事項を詳細にかつ要領よくまとめている.
[NH] 西原清一: ハッシングの技法と応用, 情報処理, Vol. 21, No. 9, pp. 980-991 (1980).
日本文で書かれた, ハッシュ法に関する解説記事.
ハッシュ法に関して, 基本から応用に至るまで, 簡潔にまとめられている. 130 にのぼる参考文献も掲載されており, ハッシュ表の現状を知る上で有益な解説である.

ハッシュ法に関する論文

[BR] Brent, R.P.: Reducing the retrieval time of scatter storage techniques, CACM, Vol. 16, No. 2, pp. 105-109 (1975).
番地列生成法の改良アルゴリズム
[CIC] Cichelli, R.J.: Minimal perfect hash functions made simple, CACM, Vol. 23, No. 1, pp. 17-19 (Jan. 1980).
完全ハッシュ関数
[FAG] Fagin, R., Nievergelt, J., Pippenger, N. and Strong, H.R.: Extensible hashing-a fast access method for dynamic files, ACM Trans. Database Syst., Vol. 4, No. 3, pp. 315-344 (Sep. 1979).
間接ハッシュ法, 動的番地列生成法, 二次記憶向きハッシュ法
[FR] Feldman, J.A. and Rovner, P.D.: An algol-based associative language, CACM, Vol. 12, No. 8, pp. 439-449 (Aug. 1969).
連想処理, 主記憶向きハッシュ法
[FUR] 古川康一: コンフリクト・フラグをもったハッシュ記憶法, 情報処理, Vol. 13, No. 8, pp. 533-538 (1972).
番地列生成法の改良アルゴリズム
[GOT] Goto, E.; Monocopy and associative algorithms in an extended LISP, ISTR 74-03, Technical Report of Information Science, Department of Information Science, University of Tokyo (1974).
ハッシュ法の応用, 高速リスト同定アルゴリズム, 属性リストの高速探索法
[GUI] Guibas, L.J.: The analysis of hashing algorithms, Report of Zerox Palo alto Research Center, CSL-76-3 (July 1976).
ハッシュ法の効率の理論的解析
[IG] Ida, T. and Goto, E.; Analysis of Parallel Hashing Algorithm with key deletion, JIP, Vol. 1, No. 1, pp. 25-32 (Apr. 1978).
ハードウェアハッシュ法, 並列ハッシュ法
[KNO] Knot, G.D.: Hashing functions, The Computer Journal, Vol. 18, No. 3, pp. 265-278 (Aug. 1975).
ハッシュ関数の理論的考察
[LIT] Litwin, W.: Virtual hashing: a dynamically changing hashing, Proc. 4th Conf. Very Large Data Bases, pp. 517-523 (Sep. 1978).
間接ハッシュ法, 動的番地列生成法, 二次記憶向きハッシュ法
[LYD] Lum, V.Y., Yuen, P.S.T. and Dodd, M.: Key-to-address transform techniques: a fundamental performance study on large existing formatted filles, CACM, Vol. 14, No. 4, pp. 228-239 (Apr. 1971).
ハッシュ法の効率の経験的比較考察, ハッシュ関数の比較検討
[LAR] Larson, P.: Dynamic hashing, BIT 18, pp. 184-201 (1978).
間接ハッシュ法, 動的番地列生成法, 二次記憶向きハッシュ法
[MOR] Morris, R.: Scatter storage techniques, CACM, Vol. 11, No. 1 pp. 38-44 (Jan. 1968).
ハッシュ法に関する古典的論文, ハッシュ法の基本的考え方, 直接ハッシュ法
[NH] 西原, 萩原: 分割 Residue Hash 表とその連想的検索法, 情報処理, Vol. 14, No. 7, pp. 546-549 (1973).
直接ハッシュ法, ハッシュ法の改良
[OLS] Olson, C.A.: Random access file organization for indirectly addressed records, Proc. ACM National Conference Vol. 24, pp. 539-549 (1969).
鍵の削除問題, バケット, 番地列生成法, 二次記憶向きハッシュ法
[VDP] Van der Poel J.A.: Optimum storage allocation for a file with open addressing, IBM Journal of Research and Development pp. 106-114 (Mar. 1973).
番地列生成法, バケット, 二次記憶向きハッシュ法

(昭和57年12月6日受付)