連載講座

キー検索技法—I

静的ハッシュ法とその応用†

青江　順一††

1. まえがき

計算機による情報処理において“探す”ことは，最も基本的な処理として頻繁に行われている．この探索処理のなかでも，キーを見出しとしてその関連情報であるレコードを探すキー検索処理は，データベースシステムや自然言語処理システムなど非常に多くの分野に応用されている．しかし，各応用分野で生じる検索処理への要求は多彩であり，能率的検索のためには要求に応じてデータの格納と検索構造を変化させる必要がある．この理由で，キー検索法は非常に多くの研究成果が報告されており，実際の応用では各検索法の特徴を十分に把握して，分野にうまく適合した検索法を選択しなければならない[5)~8)]．

本連載講座では，キー検索法を格納と検索構造に基づいて表探索と探索木法に分類し，その特徴と応用分野を4回で説明する．最初の2回は表探索法として代表的なハッシュ（hashing）法を，後半の2回では探索木法を取り上げる．第1回の講座では，古くから知られている方法で，ハッシュ表の大きさが変化しない静的（static）ハッシュ法を，第2回の講座では近年研究が活発化している方法で，ハッシュ表の大きさが変化する動的（dynamic）ハッシュ法を説明する．第3回の講座では探索木法である2分探索木法とB木法を中心に説明し，第4回ではキーを構成する文字（あるいは値）を桁単位に探索を進めるトライ法を説明する．

以下，2.でキー検索法全体を概観し，キー検索法の分類と後に続く連載講座の相互関係を説明する．3.では今回の講座の話題である静的ハッシュ法の基本的原理をハッシュ表での衝突もふまえて説明した後，衝突の解消法として，4.で完全ハッシュ法を，5.で開番地法と連鎖法を説明する．また，6.ではハッシュ関数の計算法について触れ，7.でハッシュ法の特徴と応用分野をまとめる．

2. キー検索法とは

2.1 キー検索法の概観[1)~8)]

格納と探索の対象となる情報の単位をレコード（record）と呼ぶ．レコードはフィールド（field）の1個以上の組からなっており，キー（key，見出し）フィールドにはレコードを一意に識別するキーが存在する．たとえば，キー“retrieve”に対する和訳情報を内容フィールドとしてもつレコードは図-1のようになる．

キー検索の基本操作は指定されたキーの値を手掛かりにそのキーに対応するレコードを探し出すことである．以後簡単のために，単独キー検索だけを対象とし，レコードはキーで代表されるものとして話を進める．

2.2 キー検索法の分類

キー検索法は，図-2のように分類できる[5),11)]．

表探索法は，キーを表に蓄えて検索する方法であり，キーを逐次探索する線形（linear）探索の最悪の時間計算量[1)]（worst-case time complexity）は，n 個のキーに対して $O(n)$ となる．また，アル

キーフィールド	内容フィールド
retrieve	取り戻す，償う，想起する，検索する

図-1　キー“retrieve”に対するレコードの例

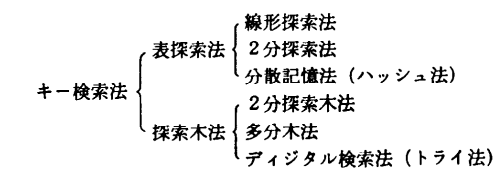

図-2　キー検索法の分類

† Static Hashing and Its Applications by Junichi AOE (Dept. of Information Science and Intelligent Systems, Faculty of Engineering, The University of Tokushima).

†† 徳島大学工学部知能情報工学科

ファベット順に並べておいて探索領域を2分の1に狭めていく2分 (binary) 探索法の計算量は，$O(\log_2 n)$ となる．

ハッシュ法（分散 (scatter) 記憶法）は代表的な表探索法であり，キーをハッシュ表に分散記憶し，ハッシュ関数による番地計算によりキーを探索する．ハッシュ表へのキーの分散が一様ならば，探索はきわめて高速になる．しかし，固定されたハッシュ表の大きさでは，予期されないキーの増加に対応しきれなくなり，検索速度が著しく低下する．このようにハッシュ表の大きさを固定した手法を静的ハッシュ法と呼ぶ．ハッシュ表を局所的に大きくして，この静的ハッシュ法の欠点を解決する動的ハッシュ法の研究が活発になってきている．したがって，連載講座の最初の2回は，この静的と動的ハッシュ法をそれぞれ取り上げる．

探索木法のうち2分探索木法では，キーの値が左部分木のキー＜根のキー＜右部分木のキーなる関係に従って格納される．探索は木の根から始め，各節のキーの値と比較しながら，左右の部分木へと辿り，途中でキーが一致すれば探索を終了する．したがって，探索木が左右うまくバランスしている場合は，2分探索と同じ検索コストとなる．多分木は，ノードからの分岐が $m(>2)$ 個以上存在する場合であり，キーの値の順序関係は2分探索木と同様に保存される．多分木の探索時間計算量は $O(\log_m n)$ となる．探索木法は上記の順序関係により，キーの順検索 (order search) が自然に実現できるが，ハッシュ法ではキー分布の仮定や更新頻度を考慮する必要があり，一般的ではない．

ディジタル検索は，トライ (trie) 法に代表されるように，キー自身の文字（または，値の桁）単位に探索木を作るので，探索木法の範疇に入るが，探索法はキーの値を一度に比較する上記の手法とは異なる．トライ法はその特徴により，自然言語辞書検索などに応用されている．したがって，第3回の講座では2分探索木法とB木法を，第4回目ではトライ法を取り上げる．

3. ハッシュ法とは

1970 年代を中心としたハッシュ法は，西原[10]，弓場[11]，Knuth[8] にまとめられているが，1980 年以後に多くみられる完全(perfect)ハッシュ法[15)~28)]の研究動向の紹介は少ない．したがって，本講座では完全ハッシュ法を主体的に取り上げ，上記の解説で参照可能な 1970 年代のものは，その説明を簡略化する．

3.1 ハッシュ表とハッシュ関数

ハッシュ法で用いられるハッシュ表の各要素はバケット (bucket) と呼ばれ，番地が付けられている．バケットに格納できるキーの総数をバケットの大きさと呼び，その大きさを越えた個数のキーの格納要求が生じた状況を衝突(collision, conflict)という．バケットの番地は，キー K に対するハッシュ関数 h により $h(K)$ として計算する．すなわち，ハッシュ表の大きさを M とするとき，ハッシュ関数はキー集合を0から $M-1$ の番地の集合に写像するキー番地変換 (key-to-address transformation) 関数である[31]．

英語の月名の3文字省略語をキーとし，大きさ $M=7$ のハッシュ表の例を考える．簡単のために，K の各文字の ASCII による内部コード値の総和を $w(K)$ とし，ハッシュ関数を次のように定義する（**表-1**）．

$h(K)=w(K) \bmod M$; mod は剰余演算子

いま，JAN, FEB, MAR, …順にバケットの大きさが1であるハッシュ表に格納するとき，JAN が格納されている番地0に MAR の格納要求が起こるので，衝突が起こったことになる．同じハッシュ関数値をもつこのようなキー JAN, MAR, MAY を，同族 (synonym) という．**図-3** には，残りのキーも含めた衝突の例を示す．

3.2 衝突の解決法

ハッシュ法の研究目的は，衝突の問題を解決することに帰着するが，次を考慮する必要がある．

（a） キーの検索と更新に対するハッシュ表の

表-1 ハッシュ関数の例

K	$w(K)$	$h(K)$
JAN	217	0
FEB	205	2
MAR	224	0
APR	227	3
MAY	231	0
JUN	237	6
JUL	235	4
AUG	221	4
SEP	232	1
OCT	230	6
NOV	243	5
DEC	204	1

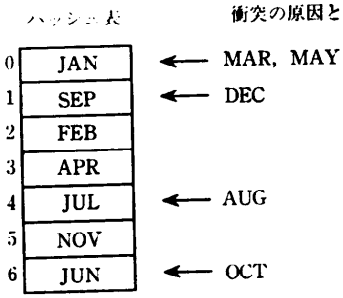

図-3　衝突の例

平均探索回数 (average probe number), すなわち時間効率.

(b)　ハッシュ表や余分な領域に関する空間効率.

(c)　適用するキー集合に対する制約などの応用性.

衝突の解消法は次のように分類できる.

(1)　完全ハッシュ法

衝突の起こらないハッシュ関数を工夫する手法であって, 最悪の探索回数が常に1回となり, 高速な検索が可能であるが, 小さくてしかも追加と削除が起こらない (静的) キー集合に適用範囲が制約される.

(2)　開番地 (open addressing) 法[10]

衝突した番地に代わる新たな番地を再計算する方法であり, 追加と削除のある (動的) キー集合に対して適用できるが, 表がいっぱいになると探索効率が悪くなる.

(3)　連鎖 (chaining) 法[10]

衝突したキーをポインタで鎖のように連結して格納する方法であり, ポインタの余分な領域を必要とするので, 空間効率は悪くなる. 開番地法と同様に動的キー集合への適用が可能で, 平均探索回数は開番地法よりは少なくなりやすく, 表がいっぱいになっても, あふれ領域を別にとることができる.

4. 完全ハッシュ法による衝突の解消

4.1 完全ハッシュ法

キー集合が既知 (静的) であるとしても完全ハッシュ関数を見つけるのは容易でない[8),14)].

Sprugnoli[28] は, 次の二つの式

$$h(K)=\lfloor(w(K)+s)/N\rfloor$$

$$h(K)=\lfloor((d+qw(K))\bmod M/N\rfloor$$

を満足する関数 $h(K)$ を決定するアルゴリズムを考案した. s, N, d, q は全て整数であり, $\lfloor x\rfloor$ は, $k\leqq x$ なる最大の整数 k を抽出する. しかし, この手法は適用可能なキー集合が小さく実用的なものではない. Berman ら[16]はキーを写像できる最大の整数値 P, キーの個数 k, バケット数 b をパラメータとして, 完全ハッシュ関数の存在を理論的に考察している. また, 適用可能なキー集合を大きくするために, 次の妥協案が提案されている.

その一つは, Brain ら[17]のように, 少しの衝突は起こるがほとんど完全に近いハッシュ関数を決定する方法であり, 約1% 程度の衝突頻度に抑えた准完全 (near-perfect) ハッシュ関数の計算法を提案している. もう一つは, まず1次ハッシュ関数によりキー集合をバケットと呼ばれる部分集合に分割し, 次に2次完全ハッシュ関数でバケットの中のキーを検索する完全合成(composite perfect) ハッシュ法 (分割 (segment) 法) であり, 実用的な実現法である.

Fredman ら[22]の完全合成ハッシュ法は,

$$h(K)=(t*w(K)\bmod p)\bmod m \qquad (1)$$

なる関数を利用する. p はキー K に対する $w(K)$ の最大値より大きい素数; t は定数; m は1次ハッシュ関数ではハッシュ表の大きさ, 2次ハッシュではバケットの中のキーの数 r の2乗に等しい数である. 静的キー集合ならば p と m は既知であるので, 式(1)を満足する t の値を試行錯誤的に決定できる. 表-1 のキー SEP に対して, $t=1$, $p=251$, $m=7$ として1次ハッシュ関数を計算すると,

$$h(\mathrm{SEP})=(1*232\bmod 251)\bmod 7=1$$

となり, 表-1 と同じハッシュ関数値を得る. この例では, 表-2 のヘッダ表の番地に対応する7つのバケットに分割され, そのバケットの始まる番地が検出される. 各バケットの最初の要素はバケット中のキーの総数, 2番目の要素はそのバケットの t の値である. 1次ハッシュよりキー SEP のバケットは1であり, そのバケットは番地 18 より始まることが分かる. そして, バケット1では $m=2^2=4$, $t=6$ であるので,

$$h(\mathrm{SEP})=(6*232\bmod 251)\bmod 4=1$$

が決定され, Bucket 1 のベース番地 20 にこの値1を加えると SEP が格納されている番地21が決定できる.

このほか, Du ら[21]は複数のハッシュ関数を組

表-2 2次ハッシュ表

Address		
0	7	
1	18	
2	24	
3	27	Header table
4	30	
5	36	
6	39	
7	3	
8	1	
9		
10	JAN	
11		
12		Bucket 0
13		
14		
15	MAY	
16		
17	MAR	
18	2	
19	6	
20	DEC	Bucket 1
21	SEP	
22		
23		
24	1	
25	1	Bucket 2
26	FEB	
27	1	
28	1	Bucket 3
29	APR	
30	2	
31	1	
32		Bucket 4
33	AUG	
34		
35	JUL	
36	1	
37	1	Bucket 5
38	NOV	
39	2	
40	1	
41		Bucket 6
42	JUN	
43		
44	OCT	

み合わせた完全合成ハッシュ法を，Cormac ら[20]は動的キー集合に対する完全合成ハッシュ法を提案し，25,000 語の英単語に対する具体的評価を行っている．また，Ramakrishna ら[26]は，m 個のページに分割されたバケットに b 個以内のキーが格納できる場合を完全合成ハッシュの条件として，動的ファイル検索へ応用している．

4.2 最小完全ハッシュ法

最小完全 (minimal perfect) ハッシュ法[14]は，m 個のキー集合を連続した m 個の整数値に1対1に写像する完全ハッシュ関数を発見することであり，最悪の探索回数が1で，しかもハッシュ表の大きさがキーの数と一致する理想的な探索表の実現を目標とする．

この手法としては，Cichelli[19]の方法がよく知られており，関連研究も多い．この方法では，次式を満足する文字 c の値 VAL(c) を決定する．

$$h(K)=\mathrm{VAL}(\mathrm{First\_Letter}(K))+\mathrm{VAL}(\mathrm{Last\_Letter}(K))+\mathrm{Length}(K)$$

表-3 に英語の月名（フルスペル）に対する VAL と最小完全ハッシュ値を示す．たとえば，JUNE に対して次の番地が計算できる．

$$h(\mathrm{JUNE})=\mathrm{VAL}('\mathrm{J}')+\mathrm{VAL}('\mathrm{E}')+\mathrm{Lenghth}(K)=0+5+4=9$$

Cichelli の方法の特徴は計算が簡単で，しかも関数がマシンの文字コード値に依存しないことであるが，VAL の決定手続きの時間計算量がキーの数に対して，指数関数的に増加し，適用できるキー集合も少なく，英単語では 45 個程度が限界と言われている．もちろん，45 個以下の任意のキー集合に対して関数が求まる保証もない[15),24)]．この方法に対して，Haggard ら[23]，Sager[27]は，ハッシュ関数で考慮する文字情報を広げる方法を提案し，また Cercone ら[18]は文字位置の指定をユーザと対話して選択できる支援システムを開発し，関数決定の効率化を計っている．Cichelli に基づく最小完全ハッシュ関数は，現状の手法で 500 個程度までのキー集合に対して適用可能と言われている[14),27)]．また，Lewis ら[14]は，完全最

表-3 Cichelli による最小完全ハッシュ関数

文字 c	VAL(c)	K	$h(K)$
A	0	JANUARY	7
D	−7	FEBRUARY	2
E	5	MARCH	10
F	−6	APRIL	11
H	5	MAY	3
J	0	JUNE	9
L	6	JULY	4
M	0	AUGUST	12
N	−3	SEPTEMBER	8
O	−1	OCTOBER	6
R	0	NOVEMBER	5
S	−1	DECEMBER	1
T	6		
Y	0		

小ハッシュ関数の決定に対して，衝突の起こったキーだけを格納する特別なバケットを定義し，2000個の英単語に適用している．

5. 開番地法と連鎖法による衝突の解消[2)~13)]

5.1 開番地法

図-3 のハッシュ表で，JAN（番地0），FEB（番地2）の格納後，MAR（番地0）の格納による衝突状況を考えてみる．衝突後第 i 番目に計算される再ハッシュ関数を

$$h_i(K)=(h(w(K))+2i) \bmod 7$$

と定義すると，番地系列 2, 4, 6 を生成するので，番地4に MAR が格納でき，衝突は解消されるが，この番地系列には同族でないキー FEB を含む重複が生じる(**図-4**)．この重複は1次クラスタ（primary clustering）と呼ばれ，平均探索回数の増加の原因となる．さらに，APR（番地3），MAY（番地0）の格納を続けると，MAY は番地6に格納される．このように番地0をアクセスする同族のキーは同じ番地系列 2, 4, 6 上で重複し，これは2次クラスタ（secondary clustering）と呼ばれ，ハッシュ法の効率を低下させる．

ハッシュ表の大きさに対するキーの総数の割合を占有率（load factor）α と呼び，α が大きくなるとクラスタの発生率は高くなる．したがって，開番地法ではクラスタ問題の研究が中心となっている．

再ハッシュ関数 h_i の計算法として，最も簡単なのが線形（linear）分散法であり，次式を利用する．

$$h_i(K)=[h(w(K))+di] \bmod M;\ d \text{ は整数}$$

しかし，この方法は1，2次クラスタが生じ，この1次クラスタの解消法が2次（quadratic）分散法であり，

$$h_i(K)=[h(w(K))+bi^2+ai] \bmod M;$$
$$a,b \text{ は整数}$$

なる式を用いるが，2次クラスタの発生は防げない．1，2次クラスタを発生させない方法は

$$h_i(K)=[h(w(K))+ih_1(w(K))] \bmod M$$

なる式を利用する2重分散（double hashing）法である．h は **6.** で説明される除算法によるものが代表的であり，h と独立な h_1 としては，次式が提案されている[6)]．

$$h_1(K)=1+(w(K) \bmod (M-2))$$

これらの詳細は，西原[10)]，弓場[11)]を参照されたい．

5.2 連鎖法

連鎖法は，分離（separate）連鎖法と併合（coalesced）連鎖法に分類できる．分離連鎖法では，衝突に対して同族のキーをハッシュ表とは別の線形リストに格納するので，同族でないキーが混合されることはない．たとえば，図-3 のハッシュ法に対しては，**図-5** のように衝突は解消される．

図-6 に JAN から JUN まで格納した併合連鎖法に基づくハッシュ表を示す．この方法では，衝突に対してハッシュ表内の適当な空領域（図-6では最も近い空領域を利用）に格納するので，同族でないキーが混合されることがあり，探索回数は多くなる．図-6 では，h(MAY)=0 と h (JUL)=4 なる同族でないキーが連鎖されている．

分離連鎖法では，削除に対して連鎖ポインタを切るだけでよいが，併合連鎖法では削除される

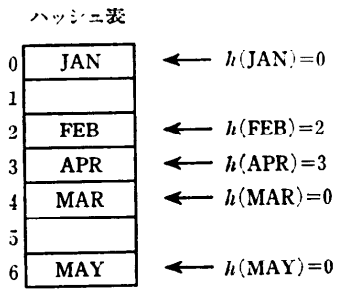

図-4 開番地法の例

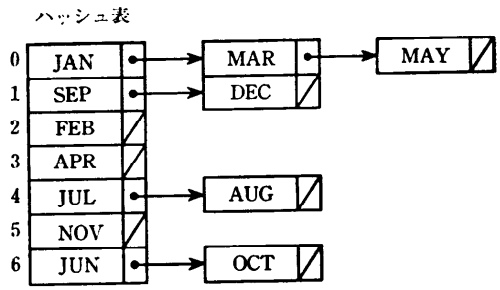

図-5 分離連鎖法の例

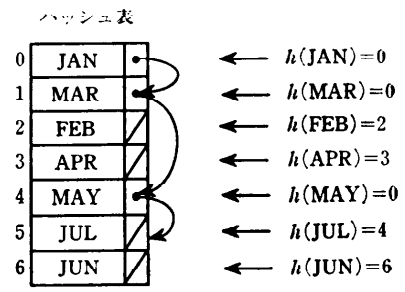

図-6 併合連鎖法の例

表-4 占有率αと平均探索回数

	成功平均探索回数	不成功平均探索回数
開番地法		
線形分散法	$\frac{1-\alpha/2}{1-\alpha}$	$\frac{1}{2}+\frac{1}{2(1-\alpha)^2}$
2次分散法	$1-\ln(1-\alpha)-\frac{\alpha}{2}$	$\frac{1}{1-\alpha}-\alpha-\ln(1-\alpha)$
2重分散法	$-\frac{1}{\alpha}\ln(1-\alpha)$	$\frac{1}{1-\alpha}$
連鎖法		
分離連鎖法	$1+\frac{\alpha}{2}$	$\alpha+e^{-\alpha}$
併合連鎖法	$1+\frac{\alpha}{4}+\frac{1}{8\alpha}(e^{2\alpha}-1-2\alpha)$	$1+\frac{1}{4}(e^{2\alpha}-1-2\alpha)$

$\ln(x)=\log_e x$

キーの場所に削除印を付け，その空領域に同族のキー格納する工夫が必要である．併合連鎖法ではハッシュ表にあふれ領域を用意して，同族でないキーの連鎖を防ぐことが可能であり，衝突したキーを連鎖の最初に連結する場合と最後に挿入する場合をうまく混合してキーの平均探索回数を少なくする手法が，Chen ら[29),30)]により提案されている．

占有率αに対する成功平均探索回数と不成功平均探索回数の関係は，表-4 にまとめられる[8),10),11)]．

表-4 より，開番地法ではαが 0.8 を越えて1に近づくと急激に探索回数が増加するので，表の大きさはキーの数に対して十分余裕をもって設計すべきである．連鎖法はαが1を越えても探索回数は大きく増加しないが，空間効率は開番地法より低下する．

6. 種々のハッシュ関数[31)~33)]

ハッシュ法における衝突とクラスタの問題は，ハッシュ関数の選択にも関係する．しかし，ハッシュ関数の実現的な評価はキー集合に依存するので，一般的には，任意のキーをハッシュ表の全ての番地に等確率で変換するのがよいハッシュ関数といえる．

Lum ら[31)]の研究には，約 600~30,000 個の種々のキー集合に対して，占有率αとバケット数を変化させた実験的評価が与えられているので，参考になる．

よく知られている関数として，

$$h(K)=w(K) \bmod M$$

で計算する除算 (division) 法があり，一般的にはMが素数とするのがよいとされている．また，キーに定数やキー自身の値を乗じて，その結果のビット列の中央付近からハッシュ表に適合したビット数を抽出する乗算 (multiplicative) 法もよく利用されている．多くの計算機では，除算より乗算が速いので，計算速度は高速になる．また，この方法は乱数法とも呼ばれる．類似の方法として，平方採中 (mid-square) 法がある．

このほか，キーを構成する各桁の頻度に着目して，ハッシュ表の大きさに適合したハッシュ関数を決定する桁解析 (digit analysis) 手法，キーを幾つかの部分に分割して，番地計算をする分割に他の分割の値を畳み込む（シフト演算や論理演算で）折り返し (folding) 法がある．この方法は高速であるが，相関のない分割を選ぶのが難しい場合がある．さらに，キーのp進数表現に対して，それをq進数に変換した値から，必要なビットを取り出す基底変換 (radix conversion) 法，キーの各桁を多項式と考え，これを別の多項式で割る代数符号化 (algebraic coding) 法がある．この除数多項式には，誤り訂正符号の手法が利用できる．

7. 応用分野とまとめ

以上，ハッシュ表自身の大きさが変化しない静的ハッシュ法について概説した．静的ハッシュ法の応用分野では，次の点を考慮する必要がある．

（1） ハッシュ表の大きさとハッシュ関数をあらかじめ決めておく必要があるので，キー集合の性質（最大個数，構成文字など）がある程度予測できる分野．

（2） ハッシュ法は，平均探索回数で評価されるので，最悪の探索時間を強調しない分野．

（3） キーの挿入と削除頻度が高くない分野で，キー集合が予想された個数に増進的に追加される分野．

以上は，かなり厳しい制約ではあるが，これらを十分考慮すれば，ハッシュ法の高速検索の特徴が生かされ，快適な検索環境が利用できる．

具体的には，言語処理系（コンパイラ，アセンブラなど）における記号表の管理は代表的な分野であり，テキスト検索，ファイルアクセス，データベースマシンでの逆 (inverted) ファイル機能への実現もみられる[40),41)]．また，特徴 (signature) ファイルを利用したテキスト検索では，ハッシュ関数は文字列の隣接2文字から特徴ベクトルの

ビット位置に写像する関数として利用されている[4),8)].

キー番地変換の考え方は，連想 (associative) 検索にも適用できる[10),39)]．データベースや情報検索での集合演算への応用についても考えられており[10),38)]，ハッシュ法による集合演算では，ソーティングが不要，重複要素の検出が容易であるなどの特徴を有している．

一般的に，ハッシュ法は順検索には不向きであるが，キー集合に制約を付加することで，Amble ら[35)]，Garg ら[36)]，Chang ら[37)]の順序ハッシングでは順検索を実現し，自然言語辞書検索[34)]として応用されている．

本講座では紹介しなかったが，関連ファイルに対する最適 (best) マッチや部分 (partial) マッチ検索，あるいは多重（属性）キー (multikey, multi-attribute) 検索へも応用される．文献は Aoe[5)] を参照されたい．

完全ハッシュ法は，プログラミング言語の指定語や標準手続き名の探索への応用が代表的であるが，完全合成ハッシュによる動的ファイル検索への応用[26),42)]もみられる．完全ハッシュ法は，最悪の検索時間計算量 $O(1)$ を保証するので，より大きいキー集合に適用可能なハッシュ関数とその効率的決定アルゴリズムの考案に期待がかかっている．

謝辞　本稿に対して種々の有益な指摘をいただいた査読者に感謝いたします．

参考文献

文献は分類してあるが，開番地と連鎖法の 1980 年までの文献は西原[10)]，それ以後は Aoe[5)] にまとめられている．また，集合演算，連想記憶などへの応用についても，西原[10)]と弓場[11)]を参照して補足されたい．

【データ構造も含めて学習するための入門書】

1) Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: Data Structures and Algorithms, Addison-Wesley, Reading Mass., Chs. 4, 5 and 11 (1983) 大野義夫訳，培風館 (1987).
2) 石畑　清：アルゴリズムとデータ構造，2 章（探索）pp. 58-142, 岩波図書 (1989).
3) Lewis, T. G. and Smith, M. Z.: Applying Data Structures, Houghton Miffin Company, Chs. 8 and 10, (1982). 浦　昭二（共訳）培風館 (1987).
4) Standish, T. A.: Data Structure Techniques, Addison-Wesley, Reading Mass., Chs. 3 and 4, pp. 99-181 (1980).

【キー検索法全般の分類と豊富な文献を含む】

5) Aoe, J.: Computer Algorithms—Key Search Strategies—, IEEE Comput. Society Press (1991).
6) Gonnet, G. H.: Handbook of Algorithms and Data Structures, Addison-Wesley, Reading Mass. Ch. 3 (Searching Algorithms) pp. 25-147 (1984). 玄　光男（共訳）啓学出版 (1987).
7) Severange, D. G.: Identifier Search Mechanisms: Survey and Generalized Model, Comput. Surv., Vol. 6, No. 9, pp. 175-194 (1974).

【キー検索法に関する深い考察と評価を含む】

8) Knuth, D. E.: The Art of Computer Programming, Addison-Wesley, Reading Mass., Vol. III, Sorting and Searching, Ch. 6 (Searching) pp. 389-550 (1973).

【ハッシュ法の入門解説】

9) 井田哲雄：ハッシュ法，情報処理，Vol. 24, No. 4, pp. 391-395 (1983).
10) 西原清一：ハッシングの技法と応用，情報処理，Vol. 21, No. 8, pp. 980-991 (1980).
11) 弓場敏嗣：ハッシングによる見出し探索の技法 [I], [II]，信学誌，Vol. 63, No. 1, pp. 15-19, pp. 20-26 (1980).
12) Maurer, W. D. and Lewis, T. G.: Hash Table Methods, Comput. Surv., Vol. 1, No. 1, pp. 5-19 (1975).
13) Price, C. E.: Table Lookup Techniques, Comput. Surv., Vol. 3, No. 2, pp. 49-65 (1971).

【完全ハッシュ関数も含むハッシュ法の入門解説】

14) Lewis, T. G. and Cook, C. K.: Hashing for Dynamic and Static Internal Tables, IEEE Comput., pp. 45-56 (1988).

【種々の完全ハッシュ法】

15) Bell, R. C. and Floyd, B.: A Monte Carlo Study of Cichelli Hash-Function Solvability, Comm. ACM, Vol. 26, No. 11, pp. 924-925 (1983).
16) Berman, F. et al.: Collections of Functions for Perfect Hashing, SIAM J. Comput., Vol. 15, No. 2, pp. 604-618 (1986).
17) Brain, M. D. and Thanp, A. L.: Near-Perfect Hashing of Large Word Sets, Softw. Pract. Exper., Vol. 19, No. 10, pp. 967-978 (1989).
18) Cercone, N., Boates, J. and Krause, M.: An Interactive System for Finding Perfect Hash Functions, IEEE Softw., pp. 39-53 (1985).
19) Cichelli, R. J.: Minimal Perfect Hash Functions Made Simple, Comm. ACM, Vol. 23, No. 1, pp. 17-19 (1980).
20) Cormac, G. V., Horspool, R. N. S. and Kaisers-Werth, M.: Practical Perfect Hashing, Comput. J., Vol. 28, No. 1, pp. 54-58 (1985).
21) Du, M. W., Hsieh, T. M., Jea, K. F. and Shieh, D. W.: The Study of New Perfect Hash Scheme, IEEE Trans. Softw. Eng., Vol. SE-9, No. 9, pp. 305-313 (1983).
22) Fredman, M. L., Komlos, J. and Szemeredi, E.: Storing a Sparse Table with 0(1) Worst-Case

Access Time, J. ACM, Vol. 31, No. 3, pp. 538-544 (1982).
23) Haggard, G. and Kaplus, K.: Finding Minimal Perfect Hash Functions, ACM SIGCSE Bull., Vol. 18, No. 1, pp. 191-193 (1986).
24) Jaeschke, G. and Osterburg, G.: On Cichelli's Minimal Perfect Hash Functions Method, Comm. ACM, Vol. 23, No. 12, pp. 728-729 (1980).
25) Jaeschke, G.: Reciprocal Hashing: A Method for Generating Minimal Perfect Hashing Functions, Comm. ACM, Vol. 24, No. 12, pp. 829-833 (1981).
26) Ramakrishna, M. V. and Larson, P.-A.: File Organization Using Composite Perfect Hashing, ACM Trans. Database Syst., Vol. 14, No. 2, pp. 231-263 (1989).
27) Sager, T.: A Polynomial Time Generator for Minimal Perfect Hash Functions, Comm. ACM, Vol. 28, No. 5, pp. 523-532 (1985).
28) Sprugnoli, R.: Perfect Hashing Functions: A Single Probe Retrieving Method for Static Sets, Comm. ACM, Vol. 20, No. 11, pp. 841-850 (1977).

【開番地法と連鎖法（西原[10]で補足されたい）】

29) Chen, W.C. and Vitter, J.S.: Analysis of Early-Insertion Standard Coalesced Hashing, SIAM J. Comput., Vol. 12, No. 4, pp. 667-676 (1983).
30) Chen, W.C. and Vitter, J.S.: Analysis of New Variants of Coalesced Hashing, ACM Trans. Database Syst., Vol. 9, No. 4, pp. 616-645 (1984).

【ハッシュ関数】

31) Lum, V.Y., Yuen, P.S.T. and Dood, M.: Key-to-Address Transformation Techniques: A Fundamental Performance Study on Large Existing Formatted Files, Comm. ACM, Vol. 14, No. 4, pp. 228-239 (1971). Vol. 15, No. 11, pp. 996-997 (1972); Vol. 16, No. 4, pp. 603-612 (1973).
32) Knott, G.D.: Hashing Functions, Comput. J., Vol. 18, No. 3, pp. 265-278 (1975).
33) Carter, L.J. and Wegman, M.L.: Universal Classes of Hash Functions, J. Comput. Syst. Sci,. Vol. 18, No. 2, pp. 143-154 (1979).

【順検索可能なハッシュ法と応用】

34) 横山，元吉，井佐原：二次記憶上の大規模語彙を用いる自然言語処理システム，情報処理学会論文誌，Vol. 29, No. 6, pp. 570-580 (1988).
35) Amble, O. and Knuth, D.E.: Ordered Hash Tables, Comput. J., Vol. 17, No. 2, pp. 135-142 (1974).
36) Garg. A.K. and Gotlieb, C.C.: Order-Preserving Key Transformation, ACM Trans. Database Syst., Vol. 11, No. 2, pp. 213-234 (1986).
37) Chang, C.C. and Chang, C.H.: An Ordered Minimal Perfect Hashing Scheme with Single Parameter, Inf. Proc. Lett., Vol. 27, No. 2, pp. 79-83 (1988).

【集合演算，連想記憶，逆ファイルなどへの応用】

38) 西原，萩原：ハッシュ技術を用いた集合関数の処理法，情報処理，Vol. 18, No. 1, pp. 11-18 (1977).
39) 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).
40) Lesk, M.E.: Some Applications of Inverted Indexes on The UNIX System, In The UNIX Programmer's Manual, Bell Lab., Murray Hill, N.J. (1978).
41) Salton, G. and McGill, M.J.: Introduction to Modern Information Retrieval, McGraw-Hill, New York (1983).
42) Fox, E.A., Heath, L.S., Chen, Q.F. and Daoud, A.M.: Practical Minimal Perfect Hash Functions for Large Databases, Comm. ACM, Vol. 35, No. 1, pp. 105-121 (1992).

（平成4年6月3日受付）

青江 順一（正会員）

昭和49年徳島大学工学部電子工学科卒業．昭和51年同大学院修士課程修了．同年同大学工学部助手（情報工学科），現在知能情報工学科助教授．工学博士．この間，コンパイラ自動生成系，自然言語処理と理解，知識工学に関する研究に従事し，パーサ，情報検索，ストリングパターンマッチング，データ圧縮に関するアルゴリズムの効率化に興味をもつ．著書「Computer Algorithms-Key Search Strategies」IEEE CS Press など．電子情報通信学会，日本人工知能学会，日本ソフトウェア科学会，日本機械翻訳協会，IEEE, ACM, AAAI, ACL 各会員．