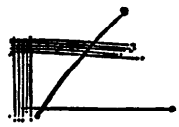
展　望



ハッシングの技法と応用†

西　原　清　一††

1. まえがき

ハッシングによるキー探索 (key search) の技法は，今日，ハッシュ法 (hash method) あるいは分散記憶法 (scatter storage technique) と呼ばれ，データ探索の基本技術の1つとしてよく知られている．この方法は，ファイルへレコードを追加する際，レコードにつけられている識別名，すなわちキーを用いて，実在の表内の格納番地を決定する方法である．探索時には，格納と同様の手続きにより目的のレコードに到達できる．レコード総数に関係なく，数回程度の表アクセス動作により探索が完了する．原理的には平均1～2回の試行錯誤で充分である．探索に要するアクセス動作の平均回数は，表の総エントリ数に対するレコード総数の割合，すなわち表占有率によって決まってくる．このようにレコード総数に関係しないという点が，ほかの探索法と比較して，ハッシュ法の際立った特徴といえる．しかし，この性質を原理通りに成り立たせるにはいくつかの前提条件が満たされている必要があり，またそのために多くの技法が提案されてきた．

本稿は，ハッシュ法の基本技術を概説し，応用例について紹介するものである．この分野は独特の用語が多く，また不統一であるので，まず次節で用語の説明と概説を行う*．ハッシュ法の問題は大きく，衝突処理と分散化手法の2つに分けられるが，それぞれ**3**と**4**で述べる．最後に**5**と**6**では，今後の展望を含めて応用例を述べる．

2. ハッシングの基本技法

ここでは諸用語の定義を行いつつ，基本的な手法を概説する．ハッシュ法における問題点や評価基準についても述べる．

まず格納・探索の対象となる情報の単位を**レコード** (record) という．レコードは，**フィールド** (field) の1個以上の組から成っており，各フィールドに具体的な値（実現値）を与えることにより定まる情報の論理的な単位である．レコードを一意に識別するような値を**キー** (key) という．キーは，1つの（あるいは特別な場合には複数の）フィールドに実現値として設定されるのが普通である．処理の対象となる情報の集まりを**ファイル** (file) といい，レコードの有限個の集合である．**ハッシュ表** (hash table) または単に表 (table) とは，計算機システムの記憶領域に実現されるファイルのことである．表は有限個の**エントリ** (entry) から成り，各エントリには**番地** (address) がつけられている．エントリの総数Mを**表サイズ** (table size) という．各エントリには一般に1つ以上のレコードを格納できるが，とくに複数個の場合は**バケット** (bucket) ということがある．各バケットに格納できるレコードの個数の最大値は通常一定であり，この値を**バケット・サイズ** (bucket size) という．1つのバケットに，バケット・サイズを超えた個数のレコードの格納要求が生じた場合，**あふれ** (overflow) が生じたという．以後，簡単のために，表の番地は0からM-1までの通番とし，エントリにはちょうど1つのレコードを格納できるものとする．また，キーとレコードは1対1に対応するものであるから，特に区別せずに用いる．同様に番地とエントリも特に区別しない．

ハッシュ法は，キーを手がかりにしてレコードの格納番地を決定する方法である．具体例として，つぎのようなキーの集合を格納することを考える．これらのキーは PASCAL の予約語から採った：

SET, CASE, AND, DO, DIV, MOD

上は，格納要求の生起順に並べてあるものとする．表サイズは9，番地は0から8までとする (**図-1**)．最初，表は空である．**ハッシュ関数** (hash function) h_oとは，キーから番地への写像，すなわち**キー番地変換** (key-to-address transformation) を行う関数である．可能なキー，例えば長さ8以下の文字列の全集合Sを

† Hashing Techniques and their Applications by Seiichi NISHIHARA (Institute of Information Sciences and Electronics, University of Tsukuba).

†† 筑波大学 電子・情報工学系

* 用語は'的確'で'短い'ものが良いと考え，英語をカナ書きしたものもある．

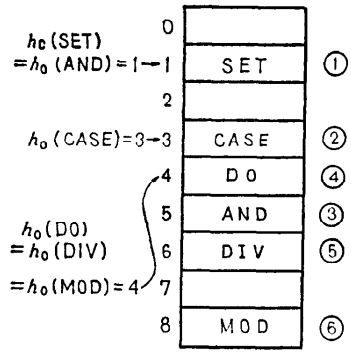


図-1 ハッシュ表（キー格納後）

キー空間 (key space) といい，番地の集合Aを**番地空間** (address space) という．ハッシュ関数は，

$$h_o : S \longrightarrow A$$

と表わされる．あるキーKに関数 h_o を施すことをハッシング (hashing) または，ハッシュ符号化 (hash coding) といい，得られた番地 $h_o(K)$ を**ハッシュ番地** (hash address, home address) という．

いま仮に，h_o を，キーの第1文字に注目し，そのアルファベットの順番を表サイズ9で割った余りをとる関数としよう．例えば，h_o(SET)$=[19]_{\mathrm{mod}9}=1$．さらに，h_o(CASE)=3 であるから，SET と CASE は問題なく格納される (①, ②)．1つのエントリを調べる操作を**探査** (probing) という．h_o(AND)=1 ゆえ第1番地を探査すると，すでに SET が詰まっている．このような現象を**衝突** (collision, conflict) という．互いに同じハッシュ番地をもつキーは，**同族**[123] (synonym, 同義語) という．衝突が生じている限り，空のエントリが見つかるまで，替りの番地を調べ続ける．仮に，キーKについて替りの番地を，$[h_o(K)+2\cdot i]_{\mathrm{mod}M}$ $(i=1,2,\cdots)$ のアルゴリズムで調べてゆくものとしよう．Mは表サイズ (=9)．このような替りの番地列を**番地系列** (sequence of addresses) といい，生成アルゴリズムを**走査方式** (search method) という．キーの探索時にも同じ番地系列をたどってゆく．K=AND の番地系列は，1, 3, 5, …となり，2度の衝突ののち，空番地（第5番地）が見つかる(③)．このとき SET, CASE, AND が1つのかたまりとなっているのが観察される．このように，本来同族でないキー同志が，互いの番地系列の重複部分において干渉しあってかたまりを作ってしまう現象を**第1種クラスタ** (primary clustering) という．これは，格納時のみならず，探索においても処理効率を低下させる大きな原因である．第1種クラスタは，一度発生するとますますその成長が加速される傾向がある．表が全部レコードで詰まった状態とは，表全体が1つのクラスタになってしまった極端な場合であるといえる．レコードの格納を続けよう．DO, DIV, MOD のハッシュ番地はすべて4となり，同族である．既述のように，番地系列は 4, 6, 8, …となり，④, ⑤, ⑥のように格納される．ここで MOD の格納手続きに注目すると，まず同族キーが入った第4，第6番地を訪問したのち，ようやく空番地を求めて走査が始まっている．結局，これら3つのキーはやはり1つのかたまりを成している．このように，番地系列が全く同じであるゆえに生ずる同族キー同志のかたまりを，**第2種クラスタ** (secondary clustering) という．これも，第1種クラスタほどではないが，格納・探索の効率を低下させる一因である．第1種および第2種クラスタは，独立した現象ではなく，相乗的に干渉し合って効率低下に拍車をかけている．1つのキーの探索・格納に要する表探査回数の平均値を，**平均探査回数** (average probe number, 平均探索路長[123]) という．番地系列において，同じ番地が出てくるまでの長さを**探索周期** (search period) という．表サイズに対する表内のレコード総数の割合を**表占有率** (load factor) という．表中に格納されるキーを探索することを**成功探索** (successful search)，そうでない場合を**不成功探索** (unsuccessful search) という．上記よりハッシュ法の問題点をまとめると，'衝突の処理' と '優れたハッシュ関数の選定' の2点となる．

データの探索手法の評価基準として，1) 処理速度，2) 記憶効率，3) 汎用性（動作環境に適した手法を選択できること）の3つが考えられる．すなわち，この3つの評価基準を念頭におきつつ，先の2つの問題を解決することが目標であり，これから諸条件が導かれる．ここで，ハッシュ法に要求される種々の条件をまとめてみる：

1）平均探査回数が少ないこと．なるべく第1種，第2種クラスタが発生しないような衝突処理を行うこと．

2）総記憶容量が小さくて済むこと．リンク・フィールドなどに余分な記憶量を要する場合があるが，平均探査回数とのかねあいで評価すべき条件である．

3）表サイズとして選べる値に(例えば素数などの)制限が少ないこと．また探索周期が充分大きいこと．探索周期が最大値，すなわち表サイズに等しいとき，**全表的** (full table search) という．

4）ハッシュ関数の計算が迅速に行えること．また，表内になるべく一様に分布させるような変換であること．

5）つぎのような諸動作環境に応じうる柔軟性を備えていること：

・ 操作の種類は，成功探索，不成功探索，格納，削除のうちどこまでを含むか，とくに削除の有無は，技法や効率に大きく影響する．

・ ハッシュ表の動的な再編成を行うか．

・ メモリ階層の有無．磁気ディスクなどのファイル装置やバケット方式を採用するかどうか．

・ キーによって探索頻度に偏りがあるか．

ハッシュ法を現実の仕事に適用するに当っては，上記の諸条件の軽重を見極めて種々の技法を取捨選択することが重要である．

3. 衝突処理

3.1 開番地法

衝突の処理に用いられる番地系列は，$h_1(K)$, $h_2(K)$ …と書くことができる．**開番地法** (open addressing) における走査方式すなわち $h_i(K)(i\geq 1)$ の決め方として，つぎの2つの方法が代表的である：

i）**線形走査法** (linear search)

$$h_i(K)=[h_o(K)+d\cdot i]_{\mathrm{mod}\,M} \qquad (O1)$$

ii）**2次走査法** (quadratic search)

$$h_i(K)=[h_o(K)+a\cdot i+b\cdot i^2]_{\mathrm{mod}\,M} \qquad (O2)$$

ここに，h_o はハッシュ関数，d, a, b は定数，Mは表サイズである．整数 d を**ハッシュ増分** (hash increment) という．a, b は整数でなくても，$a\cdot i+b\cdot i^2$ が整数であればよい．線形走査法 (O1) は，第1種，第2種クラスタともに発生するので処理は遅い．探索周期は，表サイズMとハッシュ増分dを互い素にとっておけば最大値Mになる．2次走査法 (O2) は元来，第1種クラスタを除くために提案されたものである (効率は後述)．一見2次走査法の方が複雑に見えるが，実際のアルゴリズムでは乗算を避けることができるので，線形走査法と大差ない．探索周期の短いことが2次走査法の問題点であったが[76]，整数論的な性質の解明とともに実用的な改良が種々提案されている[1),6),32),35),57),83)]．

つぎに，開番地法の改良について述べる．ハッシュ増分dについてつぎの性質がある：

$$\begin{cases} \cdot d\equiv \text{定数} \longrightarrow \text{第1種，第2種クラスタともに発生,} \\ \cdot d\equiv d(h_o(K)) \longrightarrow \text{第2種クラスタのみ発生,} \\ \cdot d\equiv d(K) \longrightarrow \text{クラスタは発生しない.} \end{cases}$$

この点に注目して種々の改良が提案されたが[14),15),70)]，表サイズが素数でなければならないなどの制限があった．実用的には，表サイズMを2のべき乗にとり，番地系列を，

$$h_i(K)=[h_o(K)+(2\cdot g(K)+1)\cdot i]_{\mathrm{mod}\,M} \qquad (O3)$$

とすれば，クラスタのない簡潔なアルゴリズムが得られる．ただし，$0\leq g(K)<M/2$．ハッシュ増分が奇数であるため，常に表サイズに対して素となる．このように，ハッシュ増分がキーに依存するような技法を**2重分散** (double hashing[65)]) という．また2次走査法についても，探索周期を含む種々の改良がなされ，上の方法と同じ処理効率が得られている[32),57),83)]．現在，クラスタの発生がなく，表サイズが素数という制限が無く，しかも探索周期が全表的な方法は上記のほかにもいくつか提案されている[63)]．これらをまとめて**2重分散法** O3 と総称することにする．

上記3つの方法による成功探索と不成功探索の各平均探査回数は，**表-1** のようにまとめられる．

3.2 連 鎖 法

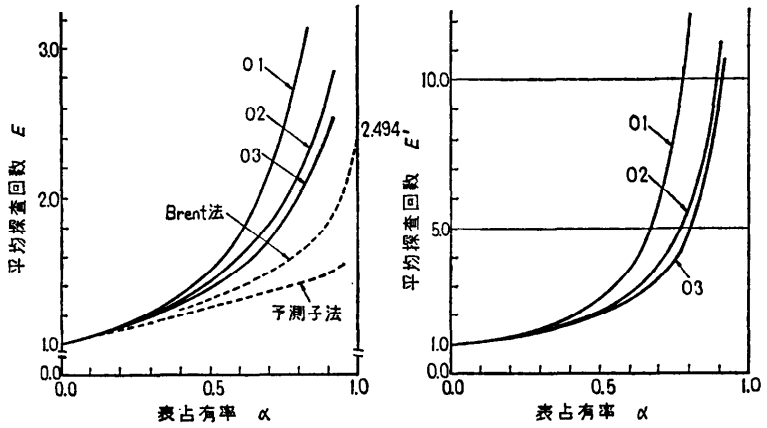
格納時に衝突を起こしたキー同志をポインタでつなぐ方式を**連鎖法** (chaining) という．各エントリにポインタ用のフィールドが必要となるので，記憶効率に限っていえば効率は低下する．しかし，開番地法に見られるような試行錯誤的探査を行わない分だけ，探索時間は短くなる．

新たにキーを格納しようとしてそのハッシュ番地を調べるとすでにほかのキーが入っており，それが同族でないことがある．このときの対処の仕方に2通りの方法がある．1つは，すでに入っていたキーが属する連鎖の末尾要素として新キーを登録する方法である．これを**併合連鎖法** (coalesced chaining, C1) という．1つの連鎖中に同族でないキーが混在しうるため，連鎖長が肥大化する傾向がある．この現象は第1種クラスタと類似の現象である．ほかの1つは，すでに入っ

表-1 開番地法の平均探査回数

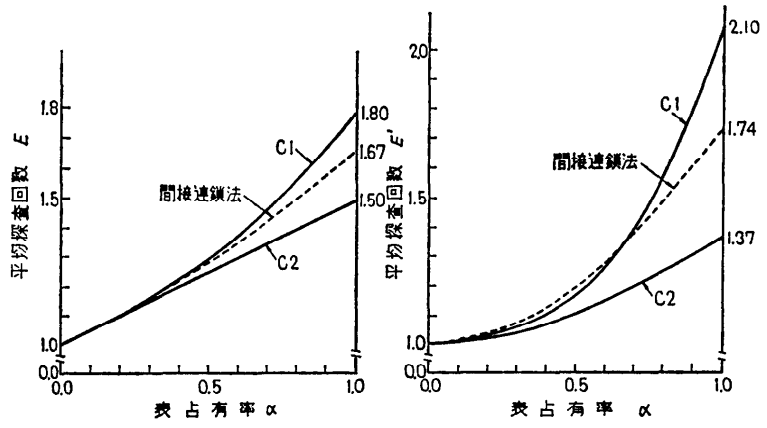
	成功探索の平均探査回数 E	不成功探索の平均探査回数 E'
線形走査法 O1	$\frac{1-\alpha/2}{1-\alpha}$	$\frac{1}{2}+\frac{1}{2(1-\alpha)^2}$
2次走査法 O2	$1-ln(1-\alpha)-\frac{\alpha}{2}$	$\frac{1}{1-\alpha}-\alpha-ln(1-\alpha)$
2重分散法 O3	$-\frac{1}{\alpha}ln(1-\alpha)$	$\frac{1}{1-\alpha}$

α：表占有率



(a) 成功探索　　(b) 不成功探索

図-2 開番地法の平均探査回数



(a) 成功探索　　(b) 不成功探索

図-3 連鎖法の平均探査回数

表-2 連鎖法の平均探査回数

	成功探索の平均探査回数 E	不成功探索の平均探査回数 E'
併合連鎖法 C1	$1+\frac{\alpha}{4}+\frac{1}{8\alpha}(e^{2\alpha}-1-2\alpha)$	$1+\frac{1}{4}(e^{2\alpha}-1-2\alpha)$
分離連鎖法 C2	$1+\frac{\alpha}{2}$	$\alpha+e^{-\alpha}$
多重連鎖法 C3	$2-\frac{1-e^{-\alpha}}{\alpha}$	1

C3 については多重度→∞のときの極限値を示す．α：表占有率

ていたキーをほかの空エントリへ移動させ，新しいキーをその本来のハッシュ番地へ格納する方法である．これを**分離連鎖法** (separate chaining, C2) という．キーの移動操作を避けるため，連鎖の2つ目以降のキーは表外の別領域（あふれ領域）に格納する方法がある．あふれ領域内のレコード数は，表占有率1のとき，表サイズの約 37% ($=1/e$) 程度となる．いずれにしても，分離連鎖法では，連鎖の構成キーはすべて同族である．しかし逆に，同族キーはただ1つの連鎖にまとめられるため，第2種クラスタと類似の効率低下が起こる．連鎖法でこの現象を避けるには，ポインタ・フィールドを複数設ける方法（多重連鎖法，C3）がある．しかし記憶効率は著しく低下するため，特に迅速性を重視する場合にのみ使える．上記各手法による平均探査回数を**表-2** に掲げる．

開番地法，連鎖法の平均探索回数（表-1，表-2）を直観的に比較するために，後述の改良手法とともに**図-**

2, 図-3 に示す.

3.3 動作環境に関連する改良

前節ではハッシュ法の衝突処理の方法として，開番地法，連鎖法の2つについて基本技法を述べた．ここでは種々の操作に注目した諸改良について述べる．

1）成功探索の効率化

開番地法において，衝突回数の極小化を図りつつキーを格納する方法が Brent[21] によって提案された．キー格納時に衝突が発生すると，当該クラスタに属する各キーについて格納をやり直したと仮定してその増加コストを調べ，最も小さいキーを再配置するという着想である．格納は若干遅くなるが，クラスタ内での局所的な最適化であるため，アルゴリズムが簡単でしかも探索時の平均探査回数は大幅に改良される．これを図-2 に示す．表占有率が1の場合でも，約 2.49 という有限値となる点が注目される．

つぎに，線形または2次走査法に適用できる技術として，各エントリに数ビットの予測子フィールドを設ける方法がある[120]．既述のように，第1種クラスタ内には同族でないキーが混在しているのが普通である．予測子は，探索時に目的キーと同族のキーのみを選択的にたどってゆけるように，部分的なポインタの役目をはたすものである．実際には番地系列の順番に関するスキップ幅を設定するため，4～5ビット程度で，連鎖法と同程度の効率が得られる．図-2に5ビットの場合を示す．なお予測子が0のときは，自然な拡張により，同族キーが終ったものと解釈すると，不成功探索を迅速化できる．

また，クラスタの最初のキーの位置を示す情報をハッシュ番地に持たせる擬似連鎖法(pseudo chaining[51])や最適格納アルゴリズムの提案[42),91),124)]がある．後者は，ファイルが静的すなわちキーの集合が固定され，キーに重みがある場合に，平均探査回数を最小にするアルゴリズムを提案したものである．

2）不成功探索の効率化．表内にキーが存在するかどうかを迅速に知りたい場合がある．連鎖法，開番地法いずれにも適した不成功探索の技術として，同族キーを値の大小順に格納する方法がある[4)]．たとえば分離連鎖法では，各連鎖はソートされているので，不成功の平均探査回数は，表占有率1のときでも，約 1.24 回で済む．この他にも，各エントリにフラグやカウンタを設け，衝突の有無や回数を記憶しておく方法がある[4),114),115)]．これらはキーの削除操作の効率化にも有効である．

3）削除操作の処理．分離連鎖法における削除は，リスト処理に類似の方法で解決できる．その他のハッシュ法では多少の工夫が必要である．削除処理の方法には大きく，フラグによる方法，並べかえによる方法の2つの考え方がある．前者は，フラグやカウンタによってその番地が空かどうかを示す方法である．後者は，空番地が発生すると何らかのキーの並べかえを行い，正常な格納状態に戻す方法である．フラグ法は，削除操作は迅速に行えるが，探索時の平均探査回数が増大することがあるので，適用分野としては削除要求の頻度が相対的に高い場合に適している．並べかえ法は，これとちょうど逆の性質を示す．削除アルゴリズムの開発は，動的ファイルを扱えるようにし，ハッシュ法の適用分野を拡大するための重要な課題である[65),113)]．

4）表の再編成．キーの探索だけでなく，格納・削除も混在する動的な状況では，ハッシュ表の再編成が必要とされる．再編成は，ハッシュ関数や走査方式の変更だけでなく，レコード数に応じて表サイズを変える場合も含む．Bays[11)] は汎用的な再ハッシング (rehashing) アルゴリズムを提案している．これは1ビットのフラグを用い，新旧ハッシュ表に重複部分があっても余分な作業領域を使わないで済ます巧妙な方法である．なお表の再編成の問題は，ファイル編成や IR への応用との関連で研究される例が多い[38)]．

5）格納時に並べかえが不要な分離連鎖法について．あふれ領域を持たない分離連鎖法では，格納時にキーの移しかえが必要なことがある．これを避けるには，2つのポインタ・フィールドを用いるとよい．1つは通常の分離連鎖法と同じく同族の連鎖に用いる．他方は，その番地をハッシュ番地とするキーの連鎖の開始番地を指すのに用いる．この方法（間接連鎖法）によると，平均探索回数は普通の分離連鎖法より少し増加するが，格納や削除が頻繁に起こるような状況には適している．この様子を図-3 に示す．なおこの方法は，擬似連鎖法に似た発想に立つものである．

6）記憶装置の階層について．記憶装置が，高速小容量のものから低速大容量のものへと階層をなしている場合は，特別の配慮が必要である．すなわち，緊急度や頻度の高いレコードやインデクスは高速の記憶装置に配置する．たとえば，探索に使われるキー・フィールドやその一部およびポインタのみを高速記憶に持ち，残りのレコード本体は低レベルの記憶に蓄える方法，また低速記憶装置へのアクセスを並列化する方法

などがあり[62]，種々の解析が行われている．磁気ディスクなどのファイル装置上にハッシュ表を作るときは，バケット方式が有効である．すなわち，1回のアクセス動作で複数個のレコードを取り出し，キー探索の適中率を上げる方法である．バケット方式の各種変形や効率評価については，Peterson[85]，Knuth[65] などの報告がある．

以上のほかにも，高次クラスタ[46),48)]や最適キー配置[14)]の問題，他の技法たとえば木構造と併用する方法，キー操作頻度の偏りの問題など種々の側面について考察されている．また，探索周期の問題は整数論との関連において興味が持たれている[1),9),22),35),95),96)]．

3.4 記憶効率の評価

平均探査回数の比較を単純に表占有率のみについて行うと，連鎖法が最も有効という結論になる．しかし，連鎖法はポインタの分だけレコードが大きくなり，結果的に記憶容量が一定ならば格納できる総レコード数は開番地法より少なくなる．つまり，同数のレコードを格納しても，表占有率では違いがある．したがって，記憶効率は平均探査回数との関連で評価されるべきである．記憶効率については Bays[12] をはじめ2～3の報告がある[115]．2重分散法 (O3) と分離連鎖法 (C2) との比較を，成功探索と不成功探索の各場合について解析した結果を整理するとつぎのようになる．ただし，レコード本体部の大きさに対するポインタの割合を r とする．r の値によってどちらが有効かが決まる．結果のみまとめるとつぎのようになる．

r の範囲	0.0	0.1	0.7	2.8	4.2
成功探索	←C2→		←不定→	←O3	
不成功探索	←C2		→←不定→	←O3	

ここで‘不定’とは，表占有率によってどちらが有効かが決まることを意味する．上の結果から，r が小，すなわちポインタの占める割合が小さいほど，また表占有率が大きいほど連鎖法が有効となり，とくに不成功探索においてその傾向が顕著であることがわかる．なお予測子法では，予測子を 4～5 ビット以上とれば，成功探索は常に連鎖法より有効となる[120]．

ところで分離連鎖法では，ハッシュ番地自身，キーの情報の一部を表現している．したがって情報量の点から，エントリに格納するのはキー全体でなくてもよい．もし簡約されたキーを用いるならば，原理的には連鎖法での記憶効率の低下は生じない．また，簡約されたキーとハッシュ番地から，元のキーを復元することもできる．

4. ハッシュ関数（キー番地変換）

4.1 キー番地変換の意義

ハッシュ関数によるキー番地変換は，一般に多対1の写像となる．すなわち，同族キーによる衝突の発生である．これがクラスタ現象を惹きおこし，効率低下の原因となっている．したがって，衝突の処理技術と衝突の少ないキー番地変換法の開発は，互いに補完的な問題といえる．前節で述べた各ハッシュ法の平均探査回数の解析値は，すべてのキーは表の各番地へ等確率で変換されるという前提のもとで計算した理論値である．また計算機でハッシュ法のシミュレーションを行うときも，区間 [0, M-1] (Mは表サイズ) の擬似一様乱数を用いる場合が多い．したがって理論通りの効率を得るには，ハッシュ関数が上記の性質を満たすものでなければならない．しかし，現実の分布状況は，格納するキー集合に影響されるので，ハッシュ関数を普遍的に評価するのは困難である．キー集合があらかじめ定まっている静的な場合に比べ，削除・格納により変化してゆく動的ファイルの場合は，ハッシュ関数の性質が明らかであっても直ちに適用できるとは限らない．

キー番地変換の研究の方向としては，実在のキー集合についての実験的な評価[72),73)]，キーの構成を考慮した変換法の開発[71),118)]，さらに後述のように分散化そのものを理論的に考察したものなどがある[28),63),97)]．良いハッシュ関数を得るには，関数の計算自体が簡単でしかも衝突が少ないという相反する条件を満たす必要がある．

4.2 分散度の評価

ハッシュ関数を評価するには，それがどの程度キーをアドレス空間内に均等に散らばらせるかがひとつの基準となる．任意のキーをすべての番地へ等確率で変換すると仮定したならば，1つの番地への集中度の観測度数，すなわち同族キーの個数 n は，Poisson 分布 $P(n,\alpha)=e^{-\alpha}\cdot\alpha^n/n!$ に従うと推定される．ただし α は表占有率．1つのファイルを表に登録する場合，各番地ごとにそれをハッシュ番地とする同族キーの集合が定まる．つまりハッシュ番地に関するファイルの同値類別が得られる．これらを濃度ごとにまとめ，同値類の個数を数え，適当に自由度と有意水準を決めることにより，Poisson 分布を示しているかどうかを，カイ2乗適合度検定で調べることができる．この検定に耐えたキー番地変換を‘可用変換’と呼ぶことにする．可

用変換は，前節の平均探査回数の解析値にほぼ近い効率を保証する．つぎに，衝突が全く生じないような変換は，'最適 (perfect) 変換' といい，確実に1回の探査で格納・探索が完了する．

ハッシュ関数 h の評価につぎのような分散係数を用いる方法がある：

$$B_h(k, M)=\frac{1}{k}\sum_{i=0}^{M-1}\frac{n_i\cdot(n_i-1)}{2}$$

ただし，k はファイルの大きさ，M は表サイズ，n_i は番地 i へ変換されるキーの個数である．$B_h(k, M)$ は，分離連鎖法における成功探索の平均探査回数から1を引いた値である．キー探索で探査回数が1より大となるのは，結局その分 (excess cost) だけ，関数 h が偏った変換を行うからである．上式は，1キー当りの偏りの平均値を表わしている．最適変換における $B_h(k, M)$ の値は0，可用変換では約 $\alpha/2$ (ただし $\alpha=k/M$) となる．ハッシュ関数を実用するには，$[0, \alpha/2]$ の範囲にあるのが望ましい．しかしこの評価基準は，動的ファイルについては直ちに適用し難い．一方，静的ファイルについては常に最適変換が存在することは理論的に明らかであるが，関数の計算時間が大きくならないように注意せねばならない．

4.3 各種の変換法

キー番地変換の方法は種々提案されており Knott[63] はそのサーベイである．ここではそれらを簡単に述べ，特徴をまとめてみる．

1) 除算法 (division method)

キーの内部コードを2進数値とみなし，表サイズ M で割った剰余を番地とする方法．M が2のべき乗の場合は，まず適当な素数で割った商を再び表サイズで割るなどの工夫をする．いずれにしても，番地決定に，キーの各桁の値が等しく影響するのが望ましい．一般に，M とキーの各桁の重みとが互いに素であることが望ましく，とくに M が素数の場合は除算法が最も良い効率を与える場合が多い[73]．

2) 基底変換法 (radix conversion method)

キーが p 進表現とする．まず全桁をいくつかの組みに分割し，各組の値をあらためて1つの桁と考える．こうして得られた桁列を q 進表現とみなす方法である．p と q は互いに素．Lin[69] の方法は，本方法の特別な場合である．

3) 平方採中法 (mid-square method)

キーを自乗し，中間桁をハッシュ番地とする．広く擬似乱数発生法を応用する方法はこれと同じ範ちゅうに入り，乱数法と呼ばれる．また類似の方法として乗算法[63),65)]があり，Knuth[65] には Fibonacci 法の興味ある記述がある．

4) 折り返し法 (folding method)

キーの構成桁列に折り目をつけ，それによって仕切られた各部分桁列同志を加えたり，排他的論理和をとったりする方法．複数レジスタ間のシフト，加算などで比較的簡単に計算できる．しかし構成桁間に相関がある場合，その影響を受け易いのが欠点である．

5) 代数的符号化法 (algebraic coding method)

キーが i ビットから成っているとする．各ビットを係数として $(i-1)$ 次の多項式を作り，これを適当な $(j+1)$ 次の多項式で割る．得られた剰余の係数を並べて j ビットの番地が得られる．除数の多項式の性質については，誤り訂正符号を作る手法が応用される．

6) 桁解析法 (digit analysis method)

キーの各桁の値の出現頻度や相関に注目し，その偏りの特徴をとらえて関数を決定する方法である．前記5つの方法がキーの各桁の構成にはほとんど考慮をはらわないのに対して，桁解析法は，キーの構成の諸特徴に応じて決まる種々の手法の総称である点に特徴がある．キーの性質を解析するという立場は，2次キーをも組み込んだ '組合せ (combinatorial) ハッシング' への展開へと続くものである[33),118)]．

最後に，キー番地変換法が走査方式（2節）と関連して定められる場合について触れておく．まず除算法において，表サイズを素数とし，商をハッシュ増分とする方法がある[14),15)]，乱数法において，ハッシュ番地のみでなく，続いて発生される乱数列をそのまま番地系列として用いる方法はすぐ思いつくことである．また，番地のすべての順列の集合を想定し，キー空間からこの順列集合への写像という観点から考察した報告もある[107)]．

5. 応用と展望

ハッシュ法は，従来，言語プロセッサの予約語や変数表のような比較的小さいファイルについて応用されてきたが，最近，処理の高速性や内容検索に向くなどの特徴に注目した新たな展開が起こりつつある．本節では，各種の応用や動向について述べる．

1) 直接編成ファイルにおけるアクセス法として従来より研究され，情報検索の技術として使われている[41),82),84),128),132)]．また，TOTAL (Cincom), IDMS (Cullinane), IMS (IBM) などの例にみるように，デ

ータベース管理システムにおいて，逆ファイル機能の実現に用いられている．CODASYL 提案の DDL では，レコードの配置モードとして CALC を指定するとハッシングによる格納が行われる．データベース・マシンへの応用も試みられ，逆ファイルによる検索機能をハードウェア化した例が報告されている[130,135]．

2）言語処理系における変数表の高速探索への応用がハッシング技法の起源であるとされるが，LEAP に始まる初期の関係形式のデータ構造処理や SAIL[131]，SETL[133]，HLISP[98]などの非数値処理言語においては実行時機能として組みこまれている．またハードウェアやファームウェアによる高速化も試みられている[116]．とくに，物理的に分散配置された表に対して並列ハッシングを行い処理速度の向上を図る方向が，井田，後藤の論文[58]を初めとする一連の研究によって開発されている．

3）完全 (perfect) ハッシュ関数，すなわち表への探索をちょうど1回で済ますようなキー番地変換の構成法の研究が，主に静的ファイルあるいは探索と格納を含む準静的なファイルについて行われている[5),29),104]．表サイズが小さくて済み，変換所要時間の短いハッシュ関数がよい．例えば Cichelli[29] では，PASCAL の予約語や標準名（既定義名）のための実用的な変換法が提案されている．

4）キー番地変換の考え方には，レコードの内容と物理的な存在場所とが互いに関連し合っているという意味がある．これは内容検索あるいは連想検索へハッシュ法を適用できることを示唆している．連想検索 (associative retrieval) は，レコードを構成する複数個のフィールドのうちいくつかについてその値を指定したり条件をつけて，適合するレコードをとり出す操作である．連想と類似の用語としては部分マッチ (partial match)，複数キー (multiple key) 検索，属性向き (attribute-oriented) 検索などがあるが，いずれも複数キーを対象にした内容検索が基本である．また基本キー，2次キーの区別をとくにしないで，属性・値対 (attribute-value pair) の観点に立つ点が特徴的で，ここにウィーン方式 (Vienna method)，LEAP[127]，SAIL[30]，関係形式データモデルなどに一連の共通した考え方がうかがわれる．

連想処理へハッシュ法を応用する例は，理論的立場からのものや従来のキー番地変換を拡張する方向など多くの研究がある．Rivest[89] は各フィールドが2進数値であるような固定長レコードについてハミング距離を適合基準とし，誤り訂正符号の性質を用いたハッシング法である．Burkhard & Keller[26] も類似の方法の提案である．組合せ論におけるブロック・デザインのバランス化を応用した部分マッチ検索法が，Rivest[88),90] において提案されている．またこれらの論文では，各種の検索要求を'共通集合型問合せ'という概念で統一し，部分マッチ検索はその一種として位置づけている．上記を発展させた方法やトライ (trie)[65] を併用した手法が，Burkhard[16),23),24] らにより提案されている．また，多重キー・ハッシングの研究[93),113] や，多変量の成分分析における主成分への射影をハッシュ符号とみなす統計的な連想法[68] など多くの報告がある[2),25),26]．レコード本体を大容量記憶装置に格納し，高速記憶の容量を適正に押える方法として，誤りを許すハッシングがある[17]．また，関係形式データベースにおける各種演算を効率的に行うため，演算の対象となる可能性のあるレコードのみを前もって選択的に絞っておく方法もある[128]．これらの蓋然（がいぜん）性あるいは寛容さ（トレランス）を導入した技法は，ハッシングの特徴を生かしたものといえる．一方，広く連想の手法や，隣接分野への関連を述べた成書[129] もある．最後に，キーの値域 (range) 指定による検索を行うためには，キーの大小を保持する (sequence maintaining) ような変換法の必要性が指摘されている[134]．しかし，ハッシングは本来このような応用には不向きであり，今後の検討が望まれる．

5）集合演算はデータベース・システムや情報検索における基本的な操作である．SETL[133]，LEAP[127] など集合演算機能を有する言語もある．また汎用言語の PASCAL でも集合型データを扱える．集合処理の実現は，比較的小さな集合に限るならばビット列を用いる方法が有効であるが，そうでない場合は通常の逆ファイルの技法が使われる．この点で内容検索や前項の連想処理に共通の機能が必要となる．ハッシングを用いた集合演算処理法についてもいくつか報告され[62),98),121]，また実在の言語でも採用されている[127),133]．ハッシュ技法を用いた集合演算の特徴として，重複要素の検出が容易，記憶容量と処理速度のつり合いが比較的とれている，ソーティングが不要などの点があげられる．また，エントリに数ビットのフィールドを付加して，作業領域を減らす方法も考えられる[121]．

6）ハッシュ技法は，他のデータ構造と組合せたり他のデータ探索技法と併用して効果を上げることが容

易である．その理由は，ハッシュ技法と他の手法例えばリンクによる構造とは互いに独立性の高い技術といえるからである．例えば，ハッシュ表の各エントリの内容が木構造である場合や，あるいはメモリ・バンクのような物理的単位であってもよい．逆に，木の各節がそれぞれ1つのハッシュ表になっているような構成も可能である．ハッシングと他の手法との併用や比較に関する研究も数多い[31),38),75),101),125)]．トライ構造[65)]と関連させて部分マッチ検索への応用を考察した例として Rivest[90)], Burkhard[24)]がある．

6. むすび

ハッシングによるデータの探索技法と応用について概説と展望を行った．ハッシングという言葉もその基本技法もよく知られたものであるが，邦文の本格的な解説は意外に少なく，弓場[122)]の例を知るのみである．しかし，Morris の解説[81)]によって分散記憶技法の問題が明示されて以来，数多くの改良が提案され続け，基本的な知識のみではその有効性や適用範囲を理解したとは言い難い状況にある．たしかに基本技法についてはほぼ完成したといえる面もあり，Severance & Duhne[102)]にみるように，実用の際の設計指針について述べた報告もある．しかし応用の面では今後の研究に期待されるところが大きい．ハッシュ関数を組合せ論的により徹底して解明する方向，連想処理を含む広い範囲の内容検索の実現技術の開発，Bトリー[126)]などすでに実用化されている構造化手法と併用し，ファイル編成やデータベース・システムの効率化を図る方向，推論など知識情報処理のための記憶構造としての可能性，ハードウェア化[116)]やファームウェア化の方向などに新しい展開が予想される．なお，データ実現値と記憶場所との間に連関があるという性質は，いいかえれば論理的側面と物理的側面とが密接に関係づけられているということで，これは人の脳における機能の局在性を思い浮かばせる．また，'探索の迅速性' や '内容検索向き' というハッシングの特徴もこの性質に起因しているといえよう．今後の研究も，これらの特徴を生かす方向で進められてゆくものと思われる．

参考文献

1) Ackerman, A. F.: Quadratic search for hash tables of size p^n, CACM, Vol. 17, No. 3 (1974).
2) Aho, A. V. and Ullman, J. D.: Optimal partial-match retrieval when fields are independently specified, ACM Trans. Database Syst., Vol. 4, No. 2 (1979).
3) Ajtai, M. et al.: There is no fast single hashing algorithm, Inf. Process. Lett., Vol. 7, No. 6 (1978).
4) Amble, O. and Knuth, D. E.: Ordered hash tables, The Comput. J., Vol. 17, No. 2 (1974).
5) Anderson, M. R. and Anderson, M. G.: Comments on perfect hashing functions: a single probe retrieving method for static sets, CACM, Vol. 22, No. 2 (1979).
6) Atkinson, L. V. and Cornah, A. J.: Full period quadratic hashing, Int. J. Comput. Math., 4 (1974).
7) Bandyopadhyay, S. K.: Comment on weighted increment linear search for scatter tables, CACM, Vol. 20, No. 4 (1977).
8) Banerjee, J. and Rajaraman, V.: A dual link data structure for random file organization, Inf. Process. Lett., Vol. 4, No. 3 (1975).
9) Batagelj, V.: The quadratic hash method when the table size is not a prime number, CACM, Vol. 18, No. 4 (1975).
10) Bayer, R.: Storage characteristics and methods for searching and addressing, IFIP, '74 (1974).
11) Bays, C.: The reallocation of hash-coded tables, CACM, Vol. 16, No. 1 (1973).
12) Bays, C.: A note on when to chain overflow items within a direct-access table, CACM, Vol. 16, No. 1 (1973).
13) Bays, C.: Some techniques for structuring chained hash tables, The Comput. J., Vol. 16, No. 2 (1973).
14) Bell, J. R.: The quadratic quotient method: a hash code eliminating secondary clustering, CACM, Vol. 13, No. 2 (1970).
15) Bell, J. R. and Kaman, C. H.: The linear quotient hash code, CACM, Vol. 13, No. 11 (1970).
16) Bentley, J. L. and Burkhard, W. A.: Heuristics for partial-match retrieval data base design, Inf. Process. Lett., Vol. 4, No. 5 (1976).
17) Bloom, B. H.: Space/time trade-offs in hash coding with allowable errors, CACM, Vol. 13, No. 7 (1970).
18) Beyer, J. D.: Comments on 'An improved hash code for scatter storage', CACM, Vol. 11, No. 5 (1968).
19) Bobrow, D. G.: A note on hash linking, CACM, Vol. 18, No. 7 (1975).
20) Bookstein, A.: Double hashing, J. American Soc. for Inform. Sci. (Nov.-Dec. 1972).
21) Brent, R. P.: Reducing the retrieval time of scatter storage techniques, CACM, Vol. 16, No. 2 (1973).

22) Burkhard, W. A.: Full table quadratic quotient searching, The Comput. J., Vol. 18, No. 1 (1975).
23) Burkhard, W. A.: Partial match retrieval, BIT, Vol. 16, No. 1 (1976).
24) Burkhard, W. A.: Hashing and trie algorithms for partial match retrieval, ACM Trans. Database Syst., Vol. 1, No. 2 (1976).
25) Burkhard W. A.: Partial-match hash coding: benefits of redundancy, ACM Trans. Database Syst., Vol. 4, No. 2 (1979).
26) Burkhard, W. A. and Keller, R. M.: Some approaches to best-match file searching, CACM, Vol. 16, No. 4 (1973).
27) Buchholz, W.: File organization and addressing, IBM Syst. J., Vol. 2 (1963).
28) Carter, J. L. and Wegman, M. N.: Universal classes of hash functions. J. Comp. and Sys. Sci., Vol. 18, No. 2 (1979).
29) Cichelli, R. J.: Minimal perfect hash functions made simple, CACM, Vol. 23, No. 1 (1980).
30) Claplson, P.: Improving the access time for random access files, CACM, Vol. 20, No. 3 (1977).
31) Coffman, E. G. Jr. and Eve, J.: File structures using hashing functions, CACM, Vol. 13, No. 7 (1970).
32) Day, A. C.: Full table quadratic searching for scatter storage, CACM, Vol. 13, No. 8 (1970).
33) Dubost, P. and Trousse, J.-M.: Software implementation of a new method of combinatrial hashing, STAN-CS-75-511, Stanford Univ. (1975).
34) Dumey, A. I.: Indexing for rapid random access memory systems, Computers and Automation, Vol. 5, No. 12 (1956).
35) Ecker, A.: The period of search for the quadratic and related hash methods, The Comput. J., Vol. 17, No. 4 (1974).
36) Ehrich, H.-D.: Theory of direct-access storage functions, IFIP '74 (1974).
37) Ershov, A. P.: On programming of arithmetic operations, CACM, Vol. 1, No. 8 (1958).
38) Fagin, R. et al.: Extendible hashing—a fast access method for dynamic files, ACM Trans. Database Syst., Vol. 4, No. 3 (1979).
39) Feldman, J. A. and Low, J. R.: Comment on Brent's scatter storage algorithm, CACM, Vol. 16, No. 11 (1973).
40) Friedman, D. P. and Wise, D. S.: Garbage collecting a heap which includes a scatter table, Inf. Process. Lett., Vol. 5, No. 6 (1976).
41) Goble, C. E.: A free-text retrieval system using hash codes, The Comput. J., Vol. 18, No. 1 (1975).
42) Gonnet, G. and Munro, I.: The analysis of an improved hashing technique, Proc. 9th ACM Symp. on Theory of Comp. (1977).
43) Goto, E. et al.: Parallel hashing algorithms, Inf. Process. Lett., Vol. 6, No. 1 (1977).
44) Goto, E. and Kanada, Y.: Hashing lemmas on time complexities with applications to formula manipulation, ACM Symp. on Symbolic and Algebraic Comp. (1976).
45) Groner, L. H. and Goel, A. L. Concurrency in hashed file access, IFIP '74 (1974).
46) Guibas, L, J.: Hashing techniques that exihibit secondary or tertiary clustering, 2nd USA-Japan Comp. Conf. (1975).
47) Guibas, L. J.: The analysis of hashing algorithms, Thesis, STAN-CS-76-556, Stanford Univ. (1976).
48) Guibas, L. J.: The analysis of hashing techniques that exhibit k-ary clustering, JACM, Vol. 25, No. 4 (1978).
49) Guibas, L. J. and Szemeredi, E.: The analysis of double hashing, J. Comp. and Sys. Sci., Vol. 16, No. 2 (1978).
50) Gurski, A.: A note on analysis of keys for use in hashing, BIT, Vol. 13, No. 1 (1973).
51) Halatsis, C. and Philokyprou, G.: Pseudochaining in hash tables, CACM, Vol. 21, No. 7 (1978).
52) Hanan, M. and Palermo, F. P.: An application of coding theory to a file address problem, IBM J. Res. and Dev., Vol. 7, No. 2 (1963).
53) Harrison, M. C.: Implementation of the substring test by hashing, CACM, Vol. 14, No. 12 (1971).
54) Heising, W. P.: Note on random addressing techniques, IBM Syst. J., No. 2 (1963).
55) Herschel, von R. and Jonsson, U. B.: Was ist Hash-Coding?, Elektronische Rechenanlagen, Vol. 17, No. 3 (1975).
56) Hill, E. Jr.: A comparative study of very large data bases, Thesis, Lecture Note in Comp. Sci., 59, Springer (1978).
57) Hopgood, F. R. A. and Davenport,J.: The quadratic hash method when the table size is a power of 2, The Comput. J., Vol. 15, No. 4 (1972).
58) Ida, T. and Goto, E.: Parallel hash algorithms for virtual key index tables, JIP, Vol. 1, No. 3 (1978).
59) Isaac, E. J. and Singleton, R. C.: Sorting by address calculation, JACM, Vol. 3, No. 2 (1956).
60) Johnson, L. R.: An indirect chaining method

for addressing on secondary keys, CACM, Vol. 4, No. 5 (1961).
61) Jones, B.: A variation on sorting by address calculation, CACM, Vol. 13, No. 2 (1970).
62) Kaman, C. H.: Hash coding: techniques and applications, Thesis, Polytecnic Institute of Brooklyn (1974).
63) Knott, G. D.: Hashing functions, The Comput. J., Vol. 18, No. 3 (1975).
64) Kronmal, R. and Tarter, M.: Cumulative polygon address calculation sorting, Proc. ACM 20th Natl. Conf. (1965).
65) Knuth, D. E.: The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison-Wesley (1973).
66) Král, J.: Some properties of the scatter storage technique with linear probing, The Comput. J., Vol. 14, No. 2 (1971).
67) Lamport, L.: Comment on Bell's quadratic quotient method for hash code searching, CACM, Vol. 13, No. 9 (1970).
68) Lee, R. C. T.: Chin, Y. H. and Chang, S. C.: Application of principal component analysis to multikey searching, IEEE Trans. Soft. Eng., Vol. 2, No. 3 (1976).
69) Lin, A. D.:Key addressing of random access memories by radix transformation, SJCC, Vol. 23 (1963).
70) Luccio, F.: Weighted increment linear search for scatter tables, CACM, Vol. 15, No. 12 (19 72).
71) Lum, V. Y.: General performance analysis of key-to-address transformation methods using an abstract file concept, CACM, Vol. 16, No. 10 (1973).
72) Lum, V. Y. and Yuen, P. S. T.: Additional results on key-to-address transform techniques: a fundamental performance study on large existing formatted files, CACM, Vol. 15, No. 11 (1972).
73) Lum, V. Y. et al.: Key-to-address transform techniques: a fundamental performance study on large existing formatted files, CACM, Vol. 14, No. 4 (1971).
74) Lyon, G.: Packed scatter tables, CACM, Vol. 21, No. 10 (1978).
75) Mallach, E. G.: Scatter storage techniques: a unifying viewpoint and a method for reducing retrieval times, The Comput. J., Vol. 20, No. 2 (1977).
76) Maurer, W. D.: An improved hash code for scatter storage, CACM, Vol. 11, No. 1 (1968).
77) Maurer, W. D. and Lewis, T. G.: Hash table methods, Computing Surveys, Vol. 7, No. 1 (1975).
78) McIlroy, M. D.: A variant method of file searching, CACM, Vol. 6, No. 1 (1963).
79) Mendelson, H. and Yechiali, U.: Performance measures for ordered lists in random-acces files, JACM, Vol. 26, No. 4 (1979).
80) Mitra, D.: Solution to the hashing problem for code length 3, Inform. and Contl., Vol. 23, No. 3 (1973).
81) Morris, R.: Scatter storage techniques, CACM, Vol. 11, No. 1 (1968).
82) Mullin, J. K.: An improved index sequential access method using hashed overflow, CACM, Vol. 15, No. 5 (1972).
83) Nishihara, S. and Hagiwara, H.: A full table quadratic search method eliminating secondary clustering, Int. J. Comput. and Inf. Sci., Vol. 3, No. 2 (1974).
84) Olson, C. A.: Random access file organization for indirectly addressed records, Proc. ACM 24th Natl. Conf. (1969).
85) Peterson, W. W.: Addressing for random-access storage, IBM J. Res. and Dev., Vol. 1, No. 2 (1957).
86) Price, C. E.: Table lookup techniques, Computing Surveys, Vol. 3, No. 2 (1971).
87) Radke, C. E.: The use of quadratic residue research, CACM, Vol. 13, No. 2 (1970).
88) Rivest, R. L.: Analysis of associative retrieval algorithms, Thesis, STAN-CS-74-415, Stanford Univ. (1974).
89) Rivest, R. L.: On the optimality of Elias's algorithm for performing best-match searches, IFIP '74 (1974).
90) Rivest, R. L.: Partial-match retrieval algorithms, SIAM J. Computing, Vol. 5, No. 1 (19 76).
91) Rivest, R. L.: Optimal arrangement of keys in a hash table, JACM, Vol. 25, No. 2 (1978).
92) Rosenberg, A. L.: On storing ragged arrays by hashing, Math. Syst. Theory, Vol. 10, No. 3 (1977).
93) Rothnie, J. B. Jr. and Lozano, T.: Attribute based file organization in a paged memory environment, CACM, Vol. 17, No. 2 (1974).
94) Samson, W. B.: Testing overflow algorithms for a table of variable size, The Comput. J., Vol. 19, No. 1 (1976).
95) Samson, W. B. and Davis, R. H.: Search times using hash tables for records with non-unique keys, The Comput. J., Vol. 21, No. 3 (1978).
96) Santoro, N.: Full table search by polynomial functions, Inf. Process. Lett., Vol. 5, No. 3 (1976).

97) Sarwate, D. V.: A note on universal classes of hash functions, Inf. Process. Lett., Vol. 10, No. 1 (1980).
98) Sassa, M. and Goto, E.: A hashing method for fast set operations, Inf. Process. Lett. Vol. 5, No. 2 (1976).
99) Schay, G. and Raver, N.: A method for key-to-address transformation, IBM J. Res. and Dev., Vol. 7, No. 2 (1963).
100) Schay, G. and Spruth, W.: Analysis of a file addressing method, CACM, Vol. 5, No. 8 (1962).
101) Severance, D. G.: Identifier search mechanisms: a survey and generalized model, Computing Surveys. Vol. 6, No. 3 (1974).
102) Severance, D. and Duhne, R.: A practioner's guide to addressing algorithms, CACM, Vol. 19, No. 6 (1976); Corrigenda, CACM, Vol. 19, No. 9 (1976).
103) Shneiderman, B.: Polynomial search, Software—Practice and Experience, Vol. 3, No. 2 (1973).
104) Sprugnoli, R.: Perfect hashing functions: a single probe retrieving method for static sets, CACM, Vol. 20, No. 11 (1977).
105) Tainiter, M.: Addressing for random-access storage with multiple bucket capacities, JACM, Vol. 10, No. 3 (1963).
106) Toyoda, J. et al.: Analysis of the address assignment problem for clustered keys, JACM, Vol. 13, No. 4 (1966).
107) Ullman, J. D.: A note on the efficiency of hashing functions, JACM, Vol. 19, No. 3 (1972).
108) van der Pool, J. A.: Optimum storage allocation for initial loading of a file, IBM J. Res. and Dev., Vol. 16, No. 6 (1972).
109) van der Pool, J. A.: Optimum storage allocation for a file in steady state, IBM J. Res aud Dev., Vol. 17, No. 1 (1973).
110) van der Pool, J. A.: Optimum Storage allocation for a file with open addressing, IBM J. Res. and Dev., Vol. 17, No. 2 (1973).
111) Webb, D. A.: The development and application of an evaluation model for hash coding systems, Thesis, Syracuse Univ. (1972).
112) Williams, J. G.: Storage utilization in a memory hierarchy when storage assignment is performed by a hashing algorithm, CACM, Vol. 14, No. 3 (1971).
113) 有沢 誠: Residue Hash 法, 情報処理, Vol. 12, No. 3 (1971).
114) 古川康一: コンフリクト・フラブをもったハッシュ記憶法, 情報処理, Vol. 13, No. 8 (1972).
115) 後藤, 郡司: ハッシュ法におけるいくつかの問題—記憶効率, 削除およびハードウェア化について, 第 17 回プログラミング・シンポジウム報告集, 情報処理学会 (1976).
116) 後藤, 井田: ハッシング・プロセッサ, 情報処理, Vol. 18, No. 4 (1977).
117) 伊吹公夫: 見出しを記憶しないファイルの索引法, 情報処理, Vol. 3, No. 4 (1962).
118) 溝口, 首藤: キー・アドレス変換法の一般化と 2, 3 の問題, 電算機研究会資料 EC-73-67～74, 電子通信学会 (1974).
119) 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 (1973).
120) 西原, 萩原: 予測子を用いた Open Hash 法, 情報処理, Vol. 15, No. 7 (1974).
121) 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 (1977).
122) 西原, 北川: キーの構成を考慮したアドレス変換法について, 第 15 回情報処理学会大会 (1974).
123) 弓場敏嗣: ハッシングによる見出し探索の技法 I, および同 II, 信学誌, Vol. 63, No. 1 (1980).
124) 弓場, 星: Knott 法分散記憶表の最適構成, 信学論 D, 61-D, No. 1 (1978).
125) 弓場, 宮川: 最適な Sequence Hash Tree に関する考察, 信学論 D, 56-D, No. 1 (1973).

以下の文献は, ハッシングの記述を本来の目的としたものではない:

126) Comer, D.: The ubiquitous B-tree, Computing Surveys, Vol. 11, No. 2 (1979).
127) Feldman, J. A. and Rovner, P. D.: An algol-based associative language, CACM, Vol. 12, No. 8 (1969).
128) Higgins, L. D. and Smith, F. J.: Disc access algorithms, The Comput. J., Vol. 14, No. 3 (1971).
129) Kohonen, T.: Associative Memory—A system theoretical approach, Springer (1978).
130) McGregor, D. R. et al.: High performance hardware for database systems, in Systems for Large Data Bases (2nd VLDB, eds. P. C. Lockemann and E. J. Neuhold), North-Holland (1976).
131) Reiser, J. F. (ed.): SAIL, AIM-289, A. I. Lab., Stanford Univ. (1976).
132) Schuyler, S. T. et al.: An image matrix for accessing files, Proc. 25th ACM Annual Conf. (1972).
133) Schwartz, J.: On Programming, an interim report on the SETL project, Installment 1: Generalities, Coulant Inst. of Math. Sci., New York Univ. (1973).
134) Wiederhold, G.: Database Design, McGraw Hill (1977).
135) 植村他: 磁気バブル・データベース・マシンの計画, 昭和 52 年電気四学会連合大会 (1977).

(昭和 55 年 5 月 7 日受付)