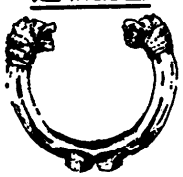
連載講座



キー検索技法—II

動的ハッシュ法とその応用†

青江　順一††

1. まえがき

前回の講座では，ハッシュ表の大きさが変化しない静的 (static) ハッシュ法について説明した．静的ハッシュ法は，キー集合の性質（最大個数，構成文字など）が予測できる分野に対して高速な検索が実現できるが，この性質が予知できない分野では，ハッシュ表を大きく取りすぎると記憶領域が無駄になり，逆に小さく見積もると，衝突（あふれ）が頻発して検索効率が低下する．過小見積りに対する一つの解決策は，新しいハッシュ関数を用意して，ハッシュ表を大きく再構成することであるが，この方法は大変時間がかかるので実用的ではない[1]．

もう一つの解決法が，本講座で概説する動的 (dynamic) ハッシュ法である．この方法は，キーの性質が予測ができない環境においても，ハッシュ関数とファイル構造を局所的に再構成してハッシュ法の高速検索を維持させようとするものである．必然的にこのような環境では，レコード件数が十分に多く，ランダムアクセス可能な2次記憶を利用する場合を仮定する．

動的ハッシュ法の研究は，1980 年代から急速に活発になり，解説記事としては Enbody ら[1]のものが非常によくまとまっている．文献調査と分類は，著者の論文集[2]や Gonnet[3] のハンドブックを参照するとよい．

以下，**2.** で動的ハッシュ法を説明するための準備を行い，**3.** で拡張 (extendible) ハッシュ法と線形 (linear) ハッシュ法の基本概念を例により説明し，あふれ処理に対する空間効率を説明する．**4.** と **5.** では拡張ハッシュ法と線形ハッシュ法に対するファイル管理法をそれぞれ説明する．**6.** では，動的ハッシュ法の応用分野と特徴をまとめる．

2. 動的ハッシュ法の基礎

2.1 静的ハッシュ法と動的ハッシュ法の評価基準

動的ハッシュ法で使用される用語は，基本的に静的ハッシュ法と同じであるが，一つのハッシュ番地に対応するレコードの格納場所バケット (bucket) は，ページ (page) と呼ばれる場合が多い．ページは固定長であり，2次記憶上に格納される．次に，静的ハッシュ法と動的ハッシュ法の評価基準の差を説明する．

静的ハッシュ法では，キーを発見するための平均探索回数が検索時間の評価基準となっていたが，動的ハッシュ法では2次記憶を使用するので，時間評価は目的とするキーを探索するために必要なページのディスクアクセス回数で決定される．すなわち，ディスクアクセスコストはハッシュ法で必要な主記憶上の内部処理コストより十分に大きいからである．また，静的ハッシュ法の空間的評価は，衝突を解消するために必要な記憶領域を基準としていたが，動的ハッシュ法ではレコードが格納されたファイル全体の空間効率が重要な基準となるので，この空間効率を定義しておく．

レコード数を n，ページ数を w，ページに格納できる最大レコード数を b で表すとき，動的ハッシュ法での空間効率は，

$$n/(wb)$$

で表される．よい空間効率を維持するためには，アクセス時間を犠牲にする必要がある．この問題はファイル管理技法として **4.** と **5.** で説明する．

† Key Search Strategies—Dynamic 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

表-1　ハッシュ関数の例

K	$w(k)$	$H(K)$
JAN	217	011001
FEB	205	001101
MAR	224	100000
APR	227	100011
MAY	231	100111
JUN	237	101101
JUL	235	101011
AUG	221	011101
SEP	232	101000
OCT	230	100110
NOV	243	110011
DEC	204	001100

2.2 動的ハッシュ法のハッシュ関数

ハッシュ表の大きさを動的に拡張あるいは縮小させるためには値域が変化するハッシュ関数が必要である．動的ハッシュ法では，キー K を十分に大きい m 長のビット列に写像する関数

$$H(k)=b_{m-1}\cdots b_2b_1b_0$$

(b_i ($0\leqq i\leqq m-1$) は，0 または 1 のビットを表す)

を定義し，$H(K)$ に対して末尾の i 個のビット列を抽出したビット列を関数

$$h_i(K)=b_{i-1}\cdots b_2b_1b_0$$

として定義する．

例として，1回目の講座で使用した英語月名の3文字省略語のキー集合と ASCII による内部コード値の総和 $w(K)$ を利用する．簡単のために，関数 $H(K)$ を $w(K)$ のビット列（後尾6ビット分）とする（表-1）．

3. 拡張ハッシュ法と線形ハッシュ法

本章では，動的ハッシュ法である拡張ハッシュ法と線形ハッシュ法の基本概念を説明する．ただし，表の縮小は，拡張方法より容易に理解できるので，説明は省略する．

3.1 拡張ハッシュ法[1),4)~15)]

いま，$H(K)$ の末尾の1ビットに対応する $h_1(K)$ なるハッシュ関数を選んで，キー JAN, FEB, MAR を格納したハッシュ表を図-1 の（a）に示す．ただし，ページに格納できるキーの最大数は2とする．

$h_1(K)$ が1ビットであるので，ハッシュ表の大きさは2であり，ハッシュ表の要素はページへのポインタをもつ．すなわち，$h_1(K)=0$ に対応するページ a にはキー MAR が，$h_1(K)=1$ に対応するページ b にはキー JAN と FEB が格納される．

ここで，ハッシュ表の全域深さは，ハッシュ表全体の番地計算に必要なビット数を表し，各ページの局所深さはそのページを他のページと区別するために必要な番地計算のビット数を表す．

さて，図-1 の（a）に対して，さらにキー APR (100011)，h_1(APR)=1 を追加する場合を考えると，ページ b であふれが生じる．そこで，2ビット分のハッシュ関数 $h_2(K)$ を採用して，ハッシュ表の大きさを2倍に成長させる（図-1 の（b））．図-1 の（b）では，番地 01 と 11 は唯一のページに対応するが，ページ a には複数の番地 00 と 10 が対応する．これは，ページ a が将来キーの追加

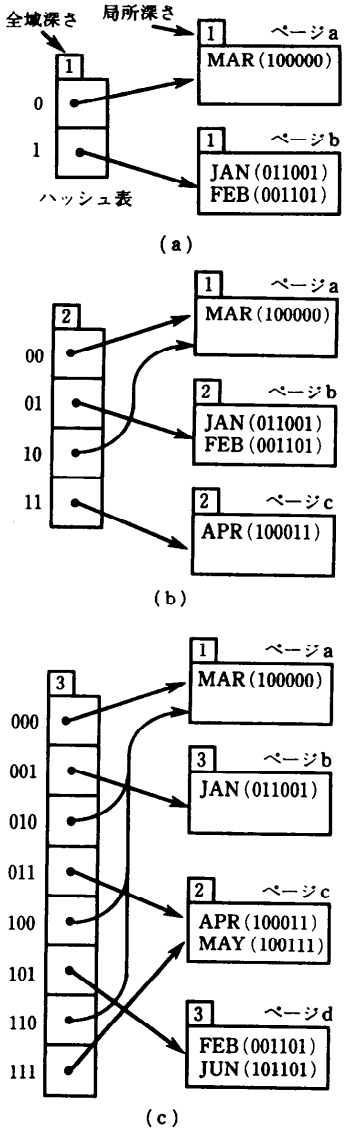


図-1　拡張ハッシュ表の例

で領域があふれたとき，番地 00 と 01 にページを分割できるチャンスがあることを意味する．このことは，ページ a の局所深さ 1 が全域深さ 2 より小さいことからも分かる．さらに，キー MAY (100111) と JUN (101101) の追加を考えるとき，キー MAY はページ c に追加できるが，キー JUN はページ b に追加できない．しかも，ページ b の局所深さと全域深さはともに 2 で等しいので，ページ分割のチャンスがない．したがって，3 ビット分のハッシュ関数 $h_3(K)$ により，再びハッシュ表を成長させて，キー JUN を追加する (**図-1** の (c))．

このように，ハッシュ表は成長を繰り返して大きくなるが，ハッシュ表が大きくなるにつれて各ページの局所的深さはバラバラになり，拡張したハッシュ表の多くの番地が特定の（局所深さの浅い）ページに集中するので，ハッシュ表も冗長になる．したがって，ハッシュ表のデータ構造として，ビット列を 2 進木 (binary tree) 構造で表現する場合が多い[5),24),33)]．この構造はトライ (trie) ハッシュと呼ばれるが，このトライは対象文字をビットに限定したものである．トライの詳細については最後の講座で改めて説明する．

図-1 の(c)に対して，残りの全てのキーを追加したトライによる拡張ハッシュ表を**図-2** に示す．

図-2 の例では，全域深さは 5，最小の局所深さはページ c の 2 であり，本来ならばページ c には $2^{5-2}=2^3=8$ 個の番地が対応するはずであるが，図-2 ではノード 6 だけがページ c に連結されている．キー OCT (100110) を検索してみると，まずルートノード 1 よりスタートし，最後尾のビット 0 の枝（ノード 1 から 2）を辿り，次の 2 番目のビット 1 に対する枝をノード 2 から 6 に辿ると，トライの葉を経由してページ c にアクセスし，キー OCT が発見される．

3.2 線形ハッシュ法[1),16)~30)]

拡張ハッシュ法ではあふれページ自体が分割されるが，線形ハッシュ法での分割対象ページは，あふれとは無関係に順番に選ばれる．この方式は，ハッシュ表を各ページへのポインタとして利用する拡張ハッシュ法と異なり，ハッシュ関数値が直接ページ番号に対応する．この意味で，拡張ハッシュ法はディレクトリ方式，線形ハッシュ法はディレクトリレス方式と呼ばれる．例として，**図-3** の（a）に示す 2 ページ構成から説明する．

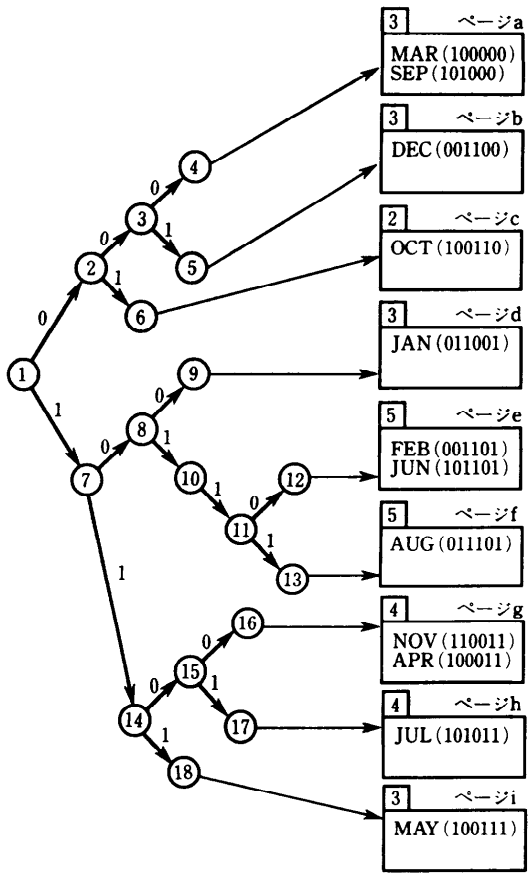


図-2 拡張ハッシュ表の 2 進木表現（トライ）例

ここでもページの大きさは 2 とする．この状況で，キー APR (100011) が追加されると仮定すると，ページ b があふれることになるが，ページ分割の順序は分割ポインタ p で指されているページ a であるので，ページ a がページ A に分割される (**図-3** の(b))．ただし，この時点でもページ b のあふれは解消されないので，キー APR はページ b のあふれ領域に格納される．さらに，キー JUL (101011) が追加されると再びページ b へのあふれを生じるので，分割要求がポインタ p の指すページ b で起こり，**図-3** の（c）が得られる．このように，ページ数が倍加すると，分割ポインタ p は最初のページ a を指し，同様な操作が繰り返される (**図-3** の（d))．

最初のページ数を N，ページ総数の倍加の回数を L，$0 \leq p < N \times 2^L$ なる分割ポインタ p に対して，キー K に対するページ番号 addr は，次の

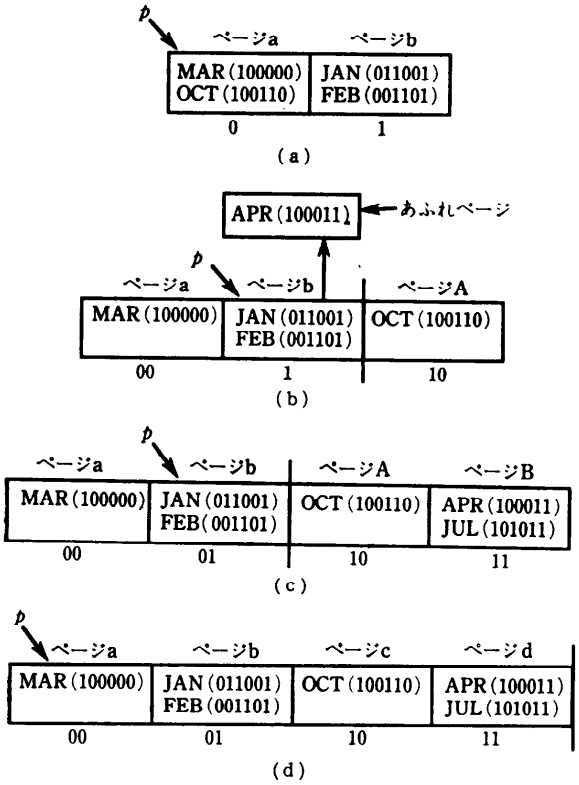


図-3　線形ハッシュ法の例

ように計算される[1].

addr←$h_L(K)$;
if addr<p **then** addr←$h_{L+1}(K)$;

また，ページの分割が起こったとき，変数 p と L は次のように変更される.

p←p+1;
if $p=N\times 2^L$ **then**
　begin
　　L←L+1;
　　p←0;
　end;

逆に，ページの収縮が起こったとき，変数 p と L は次のように変更される.

p←p−1;
if p<0 **then**
　begin
　　L←L−1;
　　p←$N\times 2^L-1$;
　end;

3.3　あふれ処理と空間効率[1),31)~42)]

線形ハッシュ法ではあふれ領域（ページ）によるあふれ処理が必ず必要なのに対して，拡張ハッシュ法ではあふれ処理の導入は任意である．しかし，いずれの方法にしろ，各ページをできるだけ満杯に近い状況(高い空間効率)で管理することは重要であるので，あふれ処理と空間効率について少し触れておく.

特にあふれ処理を行わないで，いっぱいになった主ページを単純に2ページに分割する方法では，ページに格納されているキーのハッシュ番地が一様ならば，分割された二つのページにはレコードがほぼ半分詰まっている状態となるので（図-4 の（a)），これらのページの空間効率は 50% となる．この繰り返しにより各ページは 50%～100% の空間効率になる．Fagin[4], Larson[18], Mendelson[10] らの理論的解析により，この値は 69% に収束することが分かっている.

次に，主ページと同じ大きさのあふれページを使用する場合を考える（図-4 の（b)）．この方式では主ページがあふれると次にあふれページにあふれ分を格納し，あふれページもいっぱいになると主ページを分割する．この分割では，$2b$ 分のレコード領域が $3b$ 分の領域に格納されるので，分割後の空間効率は $2b/3b$ より 67% になり，あふれ処理を行わない場合より向上する．しかし，アクセス時間はあふれページの参照のために遅くなる[1]．より空間効率を向上させるためには，あふれページの大きさを小さく（たとえば $b/2$) にする（図-4 の（c)）．この場合のレコード領域は $3b/2$ であるから，これを $2b$ の領域に分割すると空間効率は 75% になる．また，$b/2$ サイズのあふれページを二つにするとさらに空間効率は向上する（図-4 の（d)）．しかし，この手法では，分割後の主ページもあふれる確率が高くなるので，空間効率が単純に向上する保証はない．必然的に高い空間効率を得るためには，あふれ領域を連鎖することになり (Mullin[25], Larson[35])，当然ながらアクセス時間の低下を招く結果となる.

4. 拡張ハッシュ法のファイル管理技法

本章で紹介する技法は，次章で述べる線形ハッシュ法にも適用できる場合が多々ある.

拡張ハッシュでは，あふれページの分割がハッ

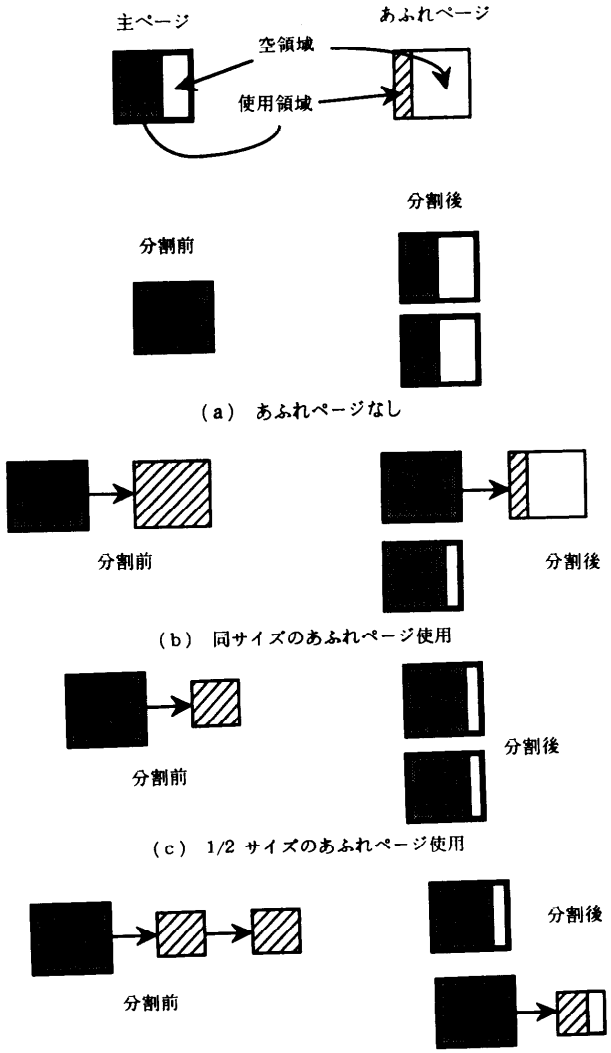


図-4 あふれページの使用と分割状況

シュ表の冗長な倍加を招き，ハッシュ表の無駄使いが起こる．これを防ぐ方法として，あふれページによる分割を遅延する方法が Larson[18] により提案されている[29),30)]．また，Scholl[29)] は静的ハッシュ法でも利用されているように，あふれページを共有することでページ数を減少させている．このことは，より少ないあふれページで同じ空間効率を実現することを意味するので，平均的ディスクアクセス回数を減少させ，空間効率をある程度緩和できる．Veklerov[30)] は，二つのページを一つの組として互いのあふれを共同管理する手法を提案している．

また，Ramakrishna ら[40)]は静的ハッシュ法で紹介した合成完全ハッシュ法をこのマルチノード管理に利用している（**図-5**）．キー K は，一次ハッシュ関数 $H(K)$ により主記憶上のヘッダ表の番地に分配され，次に完全ハッシュ関数 $h(K,R)$ により，p ページから m 個のページグループ t に分配されるので，ディスクアクセスが1回となる．R は，キーのグループを記憶するための，完全ハッシュ関数のパラメタである．

キーの分布が不均一な場合に対して，Lomet[9)] はページへの分配法を提案している．この方法は固定インデックス指数ハッシングと呼ばれ，主ページに連続する複数のページを許すので，マルチページノード方式とも呼ばれている．この方法では，ページノードの追加により，ハッシュ表の倍加を解決しているが，マルチノードの検索・更新作業が余分に必要となる．したがって，マルチノード自体に動的ハッシュ法を利用する手法も考えられる．また，Larson[18)] は各ハッシュ番地に **3.1** で触れたトライを対応させて，ハッシュ表の収縮の代わりにトライを伸縮させる方法を提案している．

拡張ハッシュ法で，図-2 のような2進木表現のトライ構造をインプリメントする場合，ノード数が増加すると記憶量が多くなる．Jonge ら[33)]は全てのノードから2本のアークが出ていれば，コンパクトな表現が可能であることを示し，これを実現するために，1本のアークしかもたないノードに対しては，ダミーページを挿入している．**図-6** に対して，この方法では木の先行順 (preorder) 走査により，○印の内部ノード通過時点でビット0

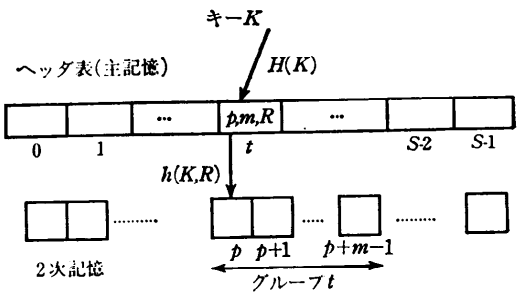


図-5 合成完全ハッシュによる動的ハッシュの実現

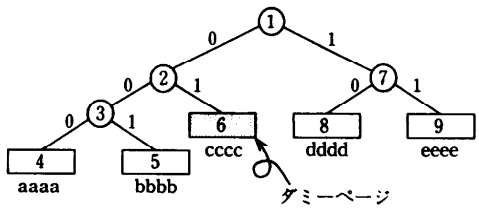


図-6　トライの2進木構造の例

表-2　予想されるハッシュ表の大きさ

n	b					
	0	10	20	50	100	200
10^3	1.50K	0.30K	0.10K	0.00K	0.00K	0.00K
10^4	25.60K	4.80K	1.70K	0.50K	0.20K	0.00K
10^5	424.10K	68.20K	16.80K	4.10K	2.00K	1.00K
10^6	6.90M	1.02M	0.26M	62.50K	16.80K	8.10K
10^7	111.11M	11.64M	2.25M	0.52M	0.26M	0.13M

K=10^3, M=10^6

を，□印のページ通過時点でビット1を出力する．この場合のノードとページ番号の走査順は1，2，…，9となるので，

000111011

が得られ，各ページに対応するビット1の後に番地を格納すると次のコンパクトな表現が得られる．

0001aaaa1bbbb1cccc01dddd1eeee

検索方法は，先行順走査の性質に従って行われ，挿入と削除は，ビットシフト演算で行われる．ただし，最悪の場合，検索では全ビットを走査し，挿入と削除では全ビットの移動を行う必要があるので，大きな木構造では，このビット処理時間に対する改善が必要となる点に注意しなければならない．この欠点に対して，佐藤ら[11]は，木構造を分割して適用する手法を提案している．

なお，レコード数nとページに格納するレコード数bに対して，Flajolet[5]は拡張ハッシュ表の大きさ（要素数）に関する精密な見積りを表-2のように与えているので，応用段階で参考になる．

5. 線形ハッシュ法のファイル管理技法

線形ハッシュ法では，分割遅延は暗黙的に行われているが，あふれページとは無関係に分割候補が選択されるので，空間効率は約60%程度となる[1]．

線形ハッシュの問題点は，分割ポインタpの左右のページで空間効率の不均一が生じることである．すなわち，分割待ちになっている右側のページは，先に分割処理されているページに比べて多くのあふれが生じる．Litwin[23]による制御分割(controlled splitting)では，ファイル全体の空間効率があるしきい値を越えたとき，ページ分割することで，空間効率の向上を計っているが，各ページあふれはしきい値まで限りなく行われるので，高いしきい値をとるとアクセス時間は低下する．

Ramamohanarao ら[28]は，主ページのあふれ処理を2次線形ハッシュで解決する方法を提案している．2次ハッシュのあふれは，3次レベルと再帰的に受け継がれるので，再帰線形ハッシング(recursive linear hashing)と呼ばれている．この再帰回数は，ハッシュ番地の分布が一様ならばほとんど3回以内で解決できることも示されている．

Larson[19]はファイル拡張を小刻みに行うことにより，あふれ連鎖を短くする部分展開[31),38)](partial expansion)を提案している．部分展開に関する展開途中のアドレス計算法としては，Ramamohanarao ら[27]の文献も合わせて参照されたい．

キー分布の不均一をより効果的に解決する方法として，スパイラル記憶(spiral storage)法がMullin[26]とLarson[21]により紹介されている．この手法はファイル成長過程で左端のページを削除し，右端により多くのページを追加し，不均一な空間効率を解決する手法である．この方法では，一様に分布するキーを指数関数的に分布するように変換して，変化する連続ページを管理する．**図-7**と**図-8**によりこの仕組みを説明する．

まず，キーKに対する$0 \leqq h(K) < 1$なる一様ハッシュ関数を定義するとき，図-7の(a)に対する$h(K)$はx軸上のc_1からc_1+1の区間内の値xに写像される．これは，値xの小数部が$h(X)$に一致するように変換されることを意味し，次の計算式で決定される．

$$x = \lceil c_1 - h(K) \rceil + h(K)$$

さらに，次式でy軸上のページ番地1，2に写像される．

$$y = \lfloor b^x \rfloor$$

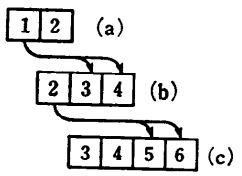


図-7　スパイラル記憶のページ拡張

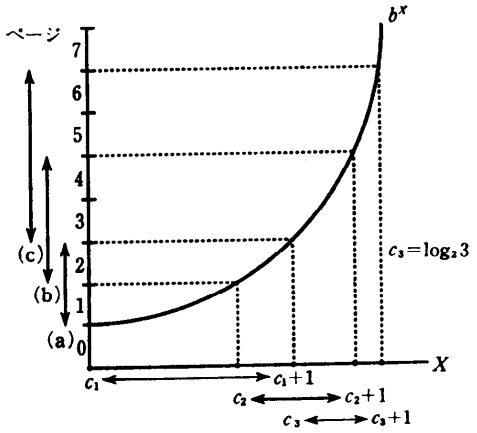


図-8 スパイラル記憶の区間とページ番地の対応例

次に，図-7 の(b)に対しては，区間は c_2 から c_2+1 へ右へシフトするので，y 軸上のより広いページ 2, 3, 4 に写像される．同様に，図-7 の(c)は，区間 (c_3, c_3+1) に対応する．新しい区間は拡張前の先頭ページに 1 を加えた値となるので，$c_1=0$ よりスタートし，$y=b^x$ 逆関数より，$c_2=1$, $c_3=\log_2 3$ へと変化する．

ファイルを縮小する場合は，x 軸上で区間を左にシフトしていけばよい．この方法では，削除されたページが再利用されないので，無駄なように思えるが，これは論理ページ上での問題であり，連続した実ページとして管理する手順も紹介されている[1),21)]．

このほか，Larson[20)] はあふれ処理を静的ハッシュ法の開番地法で行う方法，Mullin[25)] はあふれレコードを別の主ページに格納するあふれ連鎖法を提案している．

線形ハッシュ法の応用を考える場合，Larson[21),22)] による比較実験と評価が参考になるので参照されたい．

6. 応用分野とまとめ

前回の講座で述べた静的ハッシュ法は，確率的な平均値で議論する限りにおいて高速な検索が実現できるが，最悪の検索時間計算量は極端に悪くなる．したがって，キー集合の性質（最大個数，構成文字など）があらかじめ予測できる分野に限定されていた．

今回概説した動的ハッシュ法では，静的ハッシュ法の欠点を解決したものであるので，実時間システムでのファイルシステムにおいて探索時間の上限を保証する分野にも高い安全性で応用できる．しかし，動的ハッシュ法は一般的にキーに対する順検索 (order preserving) には不向きであるので，この性質を要求する場合は次回の講座の探索木法を利用すべきである．ただし，Tamminen[13)]，Hachem[17)] は順検索を可能にする動的ハッシュ法を考察している．動的ハッシュ法の最大の特長は，主記憶上の非常に小さいハッシュ表の領域でページのアクセス回数を 1 回に抑えることができる点であり[16),22),29),34)]，この点は探索木法より優れているといえる．Kelley ら[42)]は，多重キー検索や部分マッチ検索も含めて関係データベースへの検索技法とし動的ハッシュ法を取り上げている．

動的ハッシュ法の実現法は，あふれ処理を中心として非常に多くの研究成果があり，どの方法が優れているかは評価が難しいが，適用の際は空間効率と検索コストのトレードオフを十分に考慮して設計することを心がける必要がある．

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

[動的（拡張，線形）ハッシュ法の入門解説書]
1) Enbody, R. J. and Du, H. C: Dynamic Hashing Schemes, Comput. Surveys, Vol. 20, No. 2, pp. 85-113 (1988). 遠山道夫訳：bit 別冊号，共立出版，pp. 43-68 (1990).
[動的ハッシュ法の文献調査と分類]
2) Aoe, J.: Computer Algorithms—Key Search Strategies—, IEEE Comput. Society Press (1991).
3) 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).
[拡張ハッシュ法関係の文献]
4) Fagin, R., Nievergelt, J., Pippenger, N. and Storing, H. R.: Extendible Hashing—A Fast Access Method for Dynamic Files, ACM Truns. Database Syst., Vol. 4, No. 3, pp. 315-344 (1979).
5) Flajolet, P.: On the Performance Evaluation of Extendible Hashing and Trie Searching, Acta Inf., Vol. 20, No. 4, pp. 345-369 (1983).
6) Kumar, V.: Concurrent Operations on Extensible Hashing and Its Performance, Comm. ACM, Vol. 33, No. 6, pp. 681-694 (1990), Inf. Proc. Lett., Vol. 31, No. 1, pp. 35-41 (1989).
7) Lewis, T. G. and Smith, M. Z.: Applying Data Structures, Houghton Mifflin Company, Ch. 10, (1982). 浦 昭二（共訳）培風館，pp. 253-262 (1987).
8) Litwin, W. A., Roussopoulos, N., Levy, G. and Hong, W.: Trie Hashing with Controlled Load, IEEE Trans., Softw. Eng, SE-17, No. 7, pp. 687-691 (1991).

9) Lomet, D.: Bounded Index Exponential Hashing, ACM Trans. Database Syst., Vol. 8, No. 1, pp. 136-165 (1983).
10) Mendelson, H.: Analysis of Extensible Hashing, IEEE Trans. Softw. Eng., Vol. SE-8, No. 6, pp. 611-619 (1982).
11) 佐藤, 青江: 拡張ハッシングにおけるディレクトリの圧縮アルゴリズム, 情報処理学会第45回全国大会, 4-259 (1992).
12) Tamminen, M.: Extendible Hashing with Overflow, Inf. Proc. Lett., Vol. 15, No. 5, pp. 227-232 (1982).
13) Tamminen, M.: Order Preserving Extendible Hashing and Bucket Tries, BIT, Vol. 21, No. 4, pp. 419-435 (1981).
14) Weems, B. P.: A Study of Page Arrangements for Extensible Hashing, Inf. Proc. Lett., Vol. 27, No. 5, pp. 245-248 (1988).
15) Yao, A. C.: A Note on the Analysis of Extensible Hashing, Inf. Proc. Lett., 11, pp. 84-86 (1980).

[線形ハッシュ法関係の文献]

16) Greniewski, M. and Turski, W.: The External Hashing with Limited Internal Storage, J. ACM. Vol. 35, No. 1, pp. 161-184 (1988).
17) Hachem, N. I.: New Order Preserving Access Methods for Very Large Files Derived from Linear Hashing, IEEE Trans. Knowledge and Data Eng., Vol. 4, No. 1, pp. 68-82 (1992).
18) Larson, P. A.: Dynamic Hashing, BIT 18, pp. 184-201 (1978).
19) Larson, P. A.: Performance Analysis of Linear Hashing with Partial Expansions, ACM Trans. Database Syst., Vol. 7, No. 4, pp. 566-587 (1982).
20) Larson, P. A.: Linear Hashing with Overflow-Handling by Linear Probing, ACM Trans. Database Syst., Vol. 10, No. 1, pp. 75-89 (1985).
21) Larson, P. A.: Dynamic Hash Tables, Comm. ACM, Vol. 31, No. 4, pp. 446-457 (1988).
22) Larson, P. A.: Linear Hashing with Separaters —A Dynamic Hashing Scheme Achieving One-Access Retrieval, ACM Trans. Database Syst., Vol. 13, No. 13, pp. 366-388 (1988).
23) Litwin, W.: Linear Hashing: A New Tool for File and Table Addressing, in Proc. of Sixth Conf. Very Large Databases, CS Press, Los Amatios, Calif., pp. 212-223 (1980).
24) Mehlhorn, K.: Dynamic Binary Search Tree, SIAM J. Comput., Vol. 8, No. 2, pp. 175-198 (1979).
25) Mullin, J. K.: Tightly Controlled Linear Hashing without Separate Overflow Storage, BIT 21, pp. 390-400 (1981).
26) Mullin, J. K.: Spiral Storage: Efficient Dynamic Hashing with Constant Performance, Comput. J., Vol. 28, No. 3, pp. 330-334 (1985).
27) Ramamohanarao, K. and Lloyd, J. W.: Dynamic Hashing Schemes, Comput. J. Vol. 25, No. 4 (1982).
28) Ramamohanarao, K. and Sacks-Davis, R.: Recursive Linear Hashing, ACM Trans. Database Syst., Vol. 9, No. 3, pp. 369-391 (1984).
29) Scholl, M.: New File Organization Based on Dynamic Hashing, ACM Trans. Database Syst., Vol. 6, No. 1, pp. 194-211 (1981).
30) Veklerov, E.: Analysis of Dynamic Hashing with Deferred Splitting, ACM Trans. Database Syst., Vol. 10, No. 1, pp. 90-96 (1985).

[ファイル管理関係の文献]

31) Baeza-Yates, R. A. and Larson, P. A.: Performance of B+-trees with Partial Expansions, IEEE Trans. Knowledge and Data Eng., Vol. 1. No. 2, pp. 248-257 (1990).
32) Cooper, R. B. and Solomon, M. K.: The Average Time until Bucket Overflow, ACM Trans. Database Syst., Vol. 9, No. 3, pp. 392-408 (1984).
33) Jonge, W. D., Tanenbaum, A. S. and Riet, R. P.: Two Access Methods using Compact Binary Trees, IEEE Trans. Softw. Eng., Vol. SE-13, No. 7 (1987).
34) Larson, P. A. and Kajla, A.: File Organization-Implementation of a Method Guaranteeing Retrieval in One Access, Comm. ACM, Vol. 27, No. 7, pp. 670-677 (1984).
35) Larson, P. A.: Analysis of Index-Sequential Files with Overflow Chaining, ACM Trans. Database Syst., Vol. 6, No. 5, pp. 671-680 (1981).
36) Leipala, T.: On Optimal Multilevel Indexed Sequential Files, Inf. Proc. Lett., Vol. 15, No. 5. pp. 191-195 (Dec. 1982).
37) Litwin, W. and Lomet, D. B.: A New Method for Fast Data Searches with Keys, IEEE Software, pp. 16-24 (1987).
38) Lomet, D.: Partial Expansions for File Organizations with an Index, ACM Trans. Database Syst., Vol. 12, No. 1, pp. 65-84 (1987).
39) Lomet, D. B.: A Simple Bounded Discover File Organization with Good Performance, ACM Trans. Database Syst., Vol. 13, No. 4, pp. 525-551 (1988).
40) 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).
41) Shneiderman, B.: A Model for Optimizing Indexed File Structures, Int. J. Comput. and Inf. Scie., Vol. 3, No. 1, pp. 93-103 (1974).
42) Kelley, K. L. and Rusinkewlcz: Multiley, Extensible Hashing for Relational Databases, IEEE Software, pp. 77-85 (1988).

(平成4年7月6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, AAA, ACL 各会員.