データベースシステム 106－11
（1996. 1. 25）

ハッシュ法を用いた類似キー検索ファイル

楊 昇達†, 田中 榮一††

† 神戸大学大学院工学研究科 †† 神戸大学工学部

〒657 神戸市灘区六甲台町 1-1

あらまし B-木やハッシュ法に類名表記を用いた類似キー検索ファイルが提案されている．これらの方法は類似キーの検索効率はよいが，ファイルの記憶利用率が良くない．これを改善するため部分拡張の技法を用いた拡張可能ハッシュ法及び線形ハッシュ法でファイルを作り，記憶利用率が高く，しかも類似キーの検索効率が上記のものと等しいものを実現した．約 23 万語の英単語で実験し，記憶利用率が高いことを確認した．2 種類のハッシュ法とも同じ記憶利用率が得られ，拡張ステップ数が 16 のとき約 95[%] であった．

Similar Key Search Files Using Hashing Techniques

Sheng-ta YANG † and Eiichi TANAKA††

†Graduate School of Engineering, Kobe University

††Faculty of Engineering, Kobe University

Nada, Kobe 657, Japan

Abstruct Similar key search files based on a B-tree and an extensible hashing were proposed. However, the storage utilizations of those files are around 66 ~ 68 [%] and should be improved. In this paper two files with high storage utilizations for similar key search are described. One is based on an extensible hashing and the other on a linear hashing with partial expansion technique. Computer simulations on around 230 thousand English words show that the storage utilization of the file with 16 expansive steps is about 95[%].

1. まえがき

データベースシステムをキーで検索する場合，入力キーあるいはデータベースシステムの中のキーが誤っていて，一致するキーがないときでも，類似したキーが検索できるデータ構造が望ましい．B-木やハッシュ法に類名表記を用いた類似キー検索ファイルが提案されている．これらのファイルはキーの検索・挿入・削除が効率良く行なえるだけでなく，類似したキーの検索も効率良く行なえる．しかし，ファイルの記憶利用率が低い．そこで本論文では部分拡張の技法を用いた拡張可能ハッシュ法のファイルを提案する．このファイルはハッシュされたバケツが拡張ステップ数を単位とした部分拡張を行なえるようにしたもので，部分拡張をしないハッシュ法に比べ，記憶利用率が高いファイルを構成することができる．約23万語の英単語で実験した場合，拡張ステップ数が16のときファイルの記憶利用率は約95[%]であった．また，このファイルの類似キーの検索率は上記のB-木やハッシュ法を用いたファイルと等しい．部分拡張の技法を用いた線形ハッシュ法でも，拡張可能ハッシュ法を用いた場合と同じ記憶利用率が得られることも示した．

2. ハッシュ法

静的なハッシュ関数を用いたハッシュ法は，ファイルサイズが予め定まっているため，キーの数の変化に対応してない。そこで，拡張可能ハッシュや線形ハッシュのような動的ハッシュ法が提案されている．

2.1 拡張可能ハッシュ法[1]

拡張可能ハッシュ法は，キーのハッシュ値の始めのdビットを辞書順に並べてディレクトリ構造にし，ポインタが指すページを開く．この場合，ディレクトリには2^d個のポインタができる．$d=3$の場合の例を図1に示す．各々の葉のバケツには圧縮されたキーが納められている．図1の場合，ハッシュ値が'00 ...'や'1 ...'で始まるキーの数が少ないため，それぞれ1つのバケツで適合するすべてのキーを納めている．

バケツがオーバーフローを起こした場合，圧縮されたキーを拡張する場合とディレクトリを拡張する場合の2通りある．

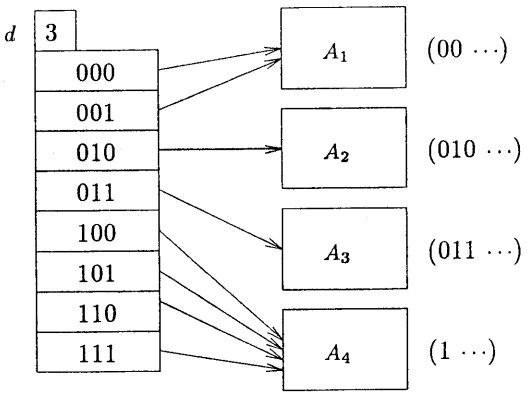


図1: 拡張可能ハッシュ法のファイル構成

- ポインタが2本以上来ているバケツのとき，圧縮されたキーを拡張して，2つのバケツに分割する．
- もしポインタが1本しか来ていないバケツのときは，上記と同じようなディレクトリの分割はできない．この場合，まず最初にディレクトリサイズを2倍にし，有効なハッシュ値のビット数を1増やす．そうすると，バケツには2本のポインタが来ることになるので，上記と同様の分割操作を行なうことができる．

2.2 線形ハッシュ法[2]

線形ハッシュ法は連続したアドレスを用いることができるファイル構造である．拡張可能ハッシュ法と同じように，オーバーフローが生じたときにバケツの分割を行なう．しかし，線形ハッシュ法では，オーバーフローしたバケツを分割させるのではなく，一定の順序に従って分割を行なう．ここで，ファイルのレベルをdとする．

図2に線形ハッシュの構造を示す．まずハッシュ関数H_dが$s=2^d$個のアドレスを作るとする．そこにキーが挿入されていき，オーバーフローが起きたときに，ファイルの最後に新しいバケツを追加する．分割するバケツはポインタpが指すバケツである．いま，k番目のバケツをB_kと書く．図2(a)の場合，初期状態として，ポインタpはB_0を指している．B_0がハッシュ関数H_{d+1}でB_0とB_sとに分割された後，

ポインタ p は B_1 へと移動する．このように，H_{d+1} で B_p が B_p と B_{s+p} とに分割されて，p は 1 ずつ移動していく．最後にバケツの数が 2^{d+1} になったとき，つまり，ポインタ p が B_s にあるとき，レベル d を 1 増やしてポインタ p を再び B_0 に戻し，次のレベルの分割が始まる．

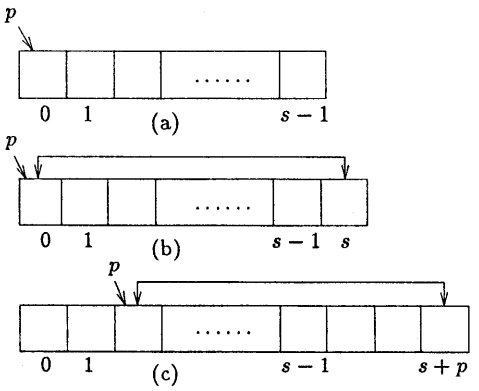


図 2: 線形ハッシュ法のファイル構成

2.3 ファイル構成

ハッシュ法により分類されたキー集合は 2 次記憶上に格納される．バケツ B を 2 次記憶の読み書きの単位とし，バケツはいくつかのブロックをつなぎ合わせたものとする．

いま，バケツ B のサイズを $S(B)$，ブロックのサイズを b とすると，$S(B) = b,\ 2b,\ 3b, \cdots,\ rb$ とバケツのサイズを拡張することができる．つまり，バケツがオーバーフローしたときに，すぐにバケツを分割するのではなく，$S(B) = rb$ のバケツがオーバーフローしたとき，はじめてバケツを分割する．

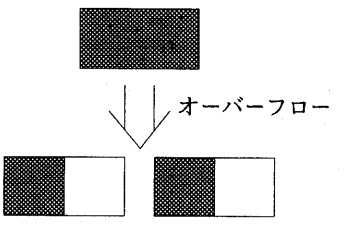


図 3: バケツの変化 $(r = 1)$

$r = 1$ の場合，バケツの変化は図 3 のようになる．バケツサイズは常に一定の大きさであるため，バケツがオーバーフローした場合，キーは分割され 2 つのバケツに分属し，分割された直後のバケツの記憶利用率は 1/2 になる．

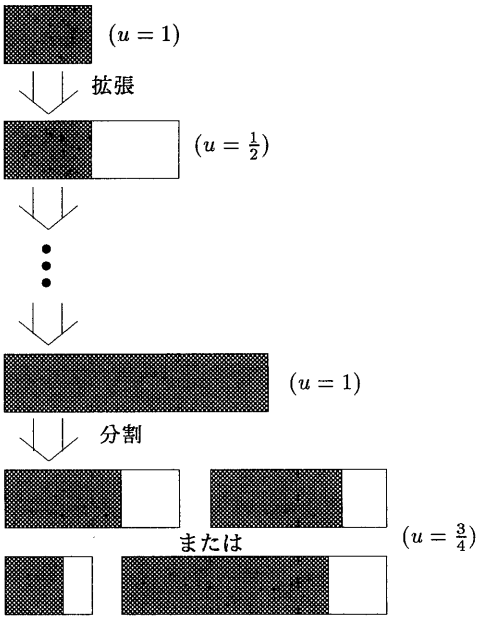


図 4: バケツの変化 $(r = 3)$

一方，$r > 1$ の場合，分割時の空領域が相対的に小さいので，分割直後のバケツの記憶利用率は $r = 1$ の場合に比べて高くなる．例として，$r = 3$ のときのバケツの変化を図 4 に示す．キーのバランスにより，分割した 2 つのバケツは $S(B) = 2b$ のバケツ 2 つに分割されるか，$S(B) = b$ と $S(B) = 3b$ のバケツに分割される．いずれの場合も，分割された直後のバケツの記憶利用率の平均は 3/4 となる．

3. 類名表記を用いたファイルの構成法

3.1 類名表記

Σ を文字の集合とし，次の条件を満たす複数の部分集合 $\{S_i\}$ に分類する．ϕ を空集合とする．

(i) $\Sigma = \bigcup_i S_i$

(ii) $i \neq j$ のとき，$S_i \bigcap S_j = \phi$

例えば，Σをアルファベットの集合とし，次のように2つの部分集合に分類する．

$$\begin{aligned} A &= \{a,b,c,d,e,f,g,h,i,j,k,l,m\} \\ B &= \{n,o,p,q,r,s,t,u,v,w,x,y,z\} \end{aligned}$$

このとき，部分集合 A，B を類名と呼ぶ．また，キーを類名を用いて書いたものを類名表記と呼ぶ．例えば，キー gold の類名表記 E(gold) は ABAA である．

3.2 データ構造

本節では類名表記を用いたファイルのデータ構造を述べる．類名表記で分類されたキー集合は2次記憶上にバケツごとに記憶される．

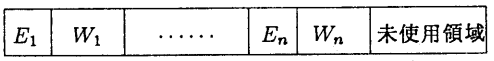


図 5: バケツの構造

バケツの構造は，図 5 のようにキーの類名表記 $E_1,\cdots,E_n$ とその類名表記を持つキー集合 $W_1,\cdots,W_n$ とを交互にならべて構成する．このファイルは以下の条件を満たす．

(1) キーは長さ毎に違うファイルに記憶されている．

(2) 1つのバケツ当りのキーの個数 kは，キーの長さを lとすると，

$$0 \le k \le \lfloor S(B)/l \rfloor - 1$$

となる．

(3) 1つのバケツ当りに類名表記またはキーが記憶されている領域を使用領域 u とすると，バケツには類名表記とそのキー集合が n 組記憶されているので，

$$u = \sum_{i=1}^{n} (|W_i| + 1) \cdot l$$

となる．ここで，$|W_i|$ は W_iの要素数である．

図 6 と図 7 にこれらを満たすファイルの構成例を示す．ただし，線形ハッシュ法で用いる類名表記は，最後尾の類名がキーの先頭の類名を表している．すなわち，通常の類名表記の逆順である．

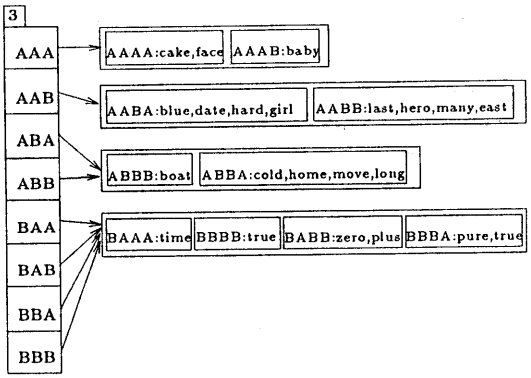


図 6: 拡張可能ハッシュ法を用いたファイル構成例

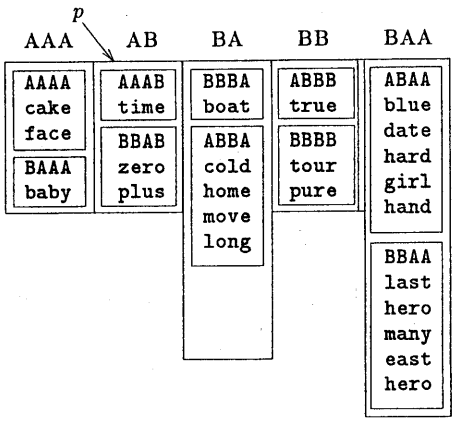


図 7: 線形ハッシュ法を用いたファイル構成例

4. ファイルの操作

ファイルの操作にはキーの挿入・検索・削除がある．

4.1 キーの挿入

拡張可能ハッシュ法と線形ハッシュ法でのキーの挿入方法を述べる．挿入するキーを wとする．

(i) 拡張可能ハッシュ法における挿入方法

(1) wの類名表記 $E(w)$ を作る．

(2) $E(w)$ の先頭から d 桁の類名とディレクトリとを比較し，該当するバケツを探す．

(3) そのブロックが一杯かどうかを確かめ，そうでないときは (7) へ進む．

(4) 一杯になったバケツで，そのバケツが取り得る最大のサイズであれば，そのバケツを指しているポインタの数 p が $p > 1$ のときは (5) へ，$p = 1$ のときは (6) へ進む．また，バケツが取り得る最大のサイズでなければ，バケツのサイズを 1 ブロック拡張し，(7) へ進む．

(5) ファイルの最後に新たなバケツを 1 つ追加し，一杯になったバケツの内容をそのバケツと新しいバケツに分割再配置し，(7) へ進む．

(6) ファイルの最後に新たなバケツを 1 つ追加し，ディレクトリを拡張する．そのとき，一杯になったバケツの内容をそのバケツと新しいバケツに分割再配置し，(7) へ進む．

(7) 類名表記が $E(w)$ であるキーが存在するなら，そのキー集合の最後に wを，存在しなければ，バケツの最後のキーの後ろに $E(w)$ と wを記憶して終了する．

(ii) 線形ハッシュ法における挿入方法

(1) wの類名表記 $E(w)$ を作る．

(2) $E(w)$ の先頭から $d+1$ 桁の類名から，該当するバケツを探す．もしなければ，d 桁の類名から，該当するバケツを探す．

(3) そのブロックが一杯かどうかを確かめ，そうでないときは (8) へ進む．

(4) そのバケツが取り得る最大のサイズであれば (5) へ進む．また，バケツが取り得る最大のサイズでなければ，バケツのサイズを 1 ブロック拡張し，(8) へ進む．

(5) バケツにオーバーフローバケツをつなぎ，(8) の挿入操作を行ない，(6) へ進む．

(6) ファイルの最後に新たなバケツを 1 つ追加し，ポインタ p で示されたバケツの内容をそのバケツと新しいバケツに分割再配置する．

(7) ポインタ p が $2^m - 1$ 番目のバケツを示していれば，p は 0 番目のバケツを，そうでなければ次のバケツを指すようにして終了する．

(8) 類名表記が $E(w)$ であるキーが存在するなら，そのキー集合の最後に wを，存在しなければ，バケツの最後のキーの後ろに $E(w)$ と wを記憶して終了する．

4.2 キーの検索

キー wの検索には，(1) キー wの検索と，(2)wに類似したキーの検索がある．

4.2.1 キーの検索

入力したキーが存在するかどうかの検索方法を次に示す．

(i) 拡張可能ハッシュ法

(1) ディレクトリにある類名表記の長さを d，検索結果を $flag$とする．

(2) wの類名表記 $E(w)$ の先頭から d 桁の類名とディレクトリとを比較し，該当するバケツを探す．

(3) そのバケツ内にある類名表記と比較し，同じ類名表記があれば (4) を行なう．

(4) 同じ類名表記で表されているすべてのキーについて wと比較して，等しいキーがあれば $flag \leftarrow true$ とする．

(5) $flag$が $true$ のとき，キー wは存在し，そうでないときは存在しない．

(ii) 線形ハッシュ法

(1) と (2) を以下のように変更するだけで，その他は拡張可能ハッシュ法と同じである．

(1) ファイルのレベルを d，検索結果を $flag$とする．

(2) 類名表記 $E(w)$ の先頭から $d+1$ 桁の類名から該当するバケツを探す．もし該当するバケツがなければ，$E(w)$ の先頭から d 桁の類名から該当するバケツを探す．

4.2.2. 類似したキーの検索

キー wに類似したキーとは，レーベンシュタイン距離の意味で wと距離が小さいキーであるとする．

入力キーとファイル中のキー間との距離にしきい値d_tを定め，しきい値以下の距離を持つキーのみを類似したキーとする．

(i) 拡張可能ハッシュ法

(1) ディレクトリにある類名表記の長さをd，検索候補をy，レーベンシュタイン距離の最小値をd_{min}，検索結果を$flag$とする．

(2) $D(E(w),s) \leq d_t$となる類名表記sをすべて発生させる．

(3) sの先頭からd桁の類名とディレクトリとを比較し，該当するバケツを探す．

(4) そのバケツ内にある類名表記と比較し，同じ類名表記があれば(5)を行なう．

(5) 同じ類名表記で表されている各々のキーvについてwと比較して，以下の操作を行なう．

- $D(w,v) < d_{min}$のとき，$y \leftarrow v$，$d_{min} \leftarrow D(w,v)$，$flag \leftarrow true$.
- $D(w,v) = d_{min}$かつ$flag = false$のとき，$y \leftarrow v$，$flag \leftarrow true$.
- $D(w,v) = d_{min}$かつ$flag = false$のとき，$y \leftarrow v$，$flag \leftarrow true$.
- $D(w,v) = d_{min}$かつ$flag = true$のとき，$y \leftarrow$未定義.
- $D(w,v) > d_{min}$のとき，何もしない．

(6) 同じバケツ内に発生させたその他の類名表記が含まれていれば，それについても(4)~(5)の操作を行なう．

(7) 発生させたその他の類名表記について，(3)~(6)の操作を行なう．

(8) $flag$が$true$で，かつ，yが未定義でなければ，検索結果yを出力し，yが未定義であれば，最も類似したキーが特定できず，複数個あることになる．また，$flag$が$false$であれば，検索は失敗となる．

以下では，$d_t = 1$と仮定したときの最も類似したキーの検索について述べる．例えば，検索キーをhiroとし，図6のファイルから最も類似したキーを検索するときの手順を示す．E(hiro) = AABBとなり，$D(E(\text{hiro}),s) \leq 1$を満たす$s$を表1に示す．

表1: AABBを1文字誤らせた類名表記

置換誤り	脱落誤り	挿入誤り
BABB	AAABB	ABB
ABBB	BAABB	AAB
AAAB	ABABB	
AABA	AABBB	
AABB	AABAB	
	AABBA	

まず，BABBを調べる．いま$d = 3$であるので，ディレクトリでBABが指しているポインタを調べ，そのバケツを開く．そこで，バケツ内でBABBの下にあるキーとheroとのレーベンシュタイン距離を計算すると，$D \leq d_t$を満たすものが存在しない．よって，この類名による検索は終了する．次に，ABBBを調べる．このように，順に類名表記を見ていくと，AABBのときheroがレーベンシュタイン距離1で見つかり，検索候補となる．脱落誤りのときは類名表記の長さが5となり，キーの長さが5のファイルを検索する．また，挿入誤りのときは類名の長さが3となり，キーの長さが3のファイルを検索する．そして，表1に示した類名についてすべて検索が終わり，しかも検索候補が1つだけであるから，heroを出力し終了する．

(ii) 線形ハッシュ法

図7のファイルについても拡張可能ハッシュ法の場合とまったく同じように検索できる．

4.2.3 キーの削除

ファイル中に既にあるキーwを削除するには，キーの挿入の場合と逆の操作をすれば良い．つまり，wを検索し，見つかれば削除する．削除したとき$E(w)$の下のキーが無くなれば$E(w)$を消去する．また，キーを削除したときバケツが融合し，バケツの最大サイズ以内であればバケツを融合する．

5. 実験結果

提案したファイルを評価するため，2次記憶のアクセス回数，記憶利用率などを調べた．また，類似キーの検索ではレーベンシュタイン距離を計算するため，2次記憶へのディスクアクセス時間に比べ，主記憶上での処理時間は無視できないため，レーベン

シュタイン距離の計算回数も調べた．実験に用いたキー集合は，UNIX の OS 下にある英単語辞書のうち単語の長さが 1 ～ 16 の約 23 万語を用いた．

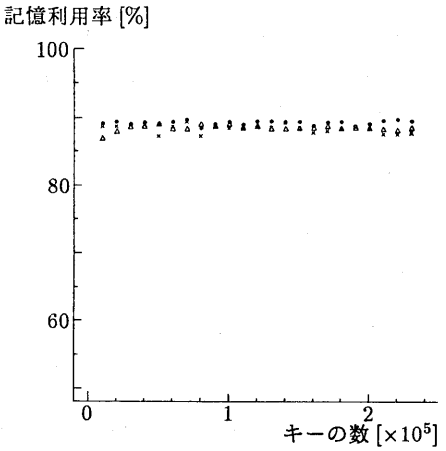
5.1 記憶利用率

キー集合からランダムにキーを選び挿入した．また，2 次記憶での記憶領域の有効利用の評価として，次の記憶利用率が用いられている．

$$\text{記憶利用率} = \frac{\text{全バケツの使用領域サイズ}}{\text{全バケツサイズ}}$$

(i) 拡張ハッシュ法を用いたファイル

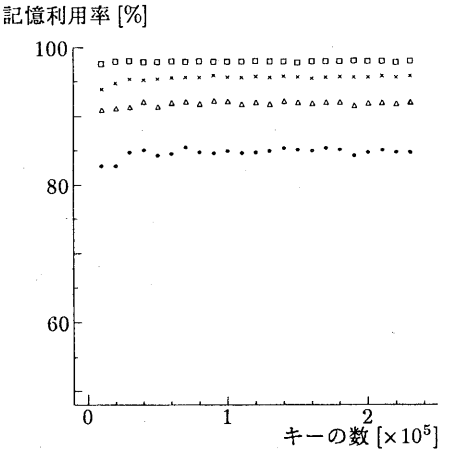
拡張ステップ数を変化させずに，文字の類が異なった場合の記憶利用率を図 8 に示す．この図より，文字の類の分け方が異なっていても記憶利用率はほぼ等しい．つまり，記憶利用率は類の分け方にも関係がないことが分かる．



記号	類名 A の集合	類名 B の集合
•	a,b,c,d,e,f,g,h,i,j,k,l,m	n,o,p,q,r,s,t,u,v,w,x,y,z
△	a,b,c,d,e,f,g,n,o,p,q,r,s	h,i,j,k,l,m,t,u,v,w,x,y,z
×	a,c,e,g,i,k,m,o,q,s,u,w,y	b,d,f,h,j,l,n,p,r,t,v,x,z
□	a,b,c,d,e,n,o,p,q,r	f,g,h,i,j,k,l,m,s,t,u,v,w,x,y,z

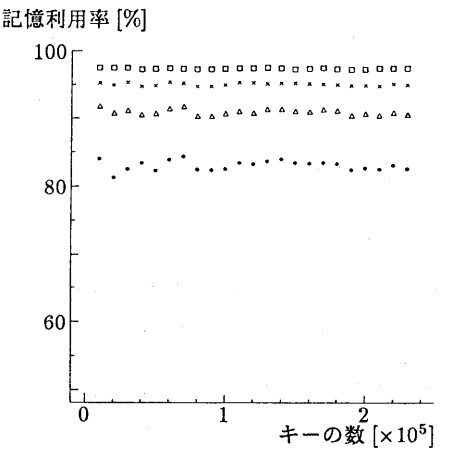
図 8: 文字の類が異なるときの記憶利用率

以下では第 3 節で示した類を用いた．ファイルのブロックサイズが異なるときの記憶利用率を図 9 に示す．拡張ステップ数 rが増加するほど，記憶利用率は上昇している．2 次記憶への読み込み回数，書き込み回数の平均値はそれぞれ 1.000，1.004 となった．書き込み回数が 1 よりも大きいのは，バケツを分割したときに，2 つのバケツを 2 次記憶へ書き込まなければならないからである．



記号	r	ブロックサイズ b	最大バケツサイズ
□	32	128[$bytes$]	4096[$bytes$]
×	16	256[$bytes$]	4096[$bytes$]
△	8	512[$bytes$]	4096[$bytes$]
•	4	1024[$bytes$]	4096[$bytes$]

図 9: 拡張可能ハッシュ法で拡張ステップ数が異なるときの記憶利用率



記号	r	ブロックサイズ b	最大バケツサイズ
□	32	128[$bytes$]	4096[$bytes$]
×	16	256[$bytes$]	4096[$bytes$]
△	8	512[$bytes$]	4096[$bytes$]
•	4	1024[$bytes$]	4096[$bytes$]

図 10: 記憶利用率 (オーバーフローバケツサイズ =256[$bytes$])

表 2: 正検索率

キーの長さ	脱落誤り	挿入誤り	置換誤り
1	-	0.0[%]	-
2	-	3.7[%]	0.0[%]
3	0.0[%]	15.5[%]	1.6[%]
4	0.2[%]	46.9[%]	8.3[%]
5	1.5[%]	72.6[%]	30.7[%]
6	13.2[%]	86.8[%]	57.0[%]
7	36.8[%]	92.3[%]	78.1[%]
8	57.5[%]	95.3[%]	86.5[%]
9	71.2[%]	97.9[%]	91.5[%]
10	78.2[%]	98.5[%]	95.3[%]
11	84.9[%]	98.9[%]	95.6[%]
12	87.5[%]	99.3[%]	97.7[%]
13	90.3[%]	99.9[%]	97.8[%]
14	92.2[%]	99.1[%]	98.7[%]
15	93.2[%]	-	99.2[%]
16	94.3[%]	-	-

表 3: レーベンシュタイン距離の計算回数

キーの長さ	脱落誤り	挿入誤り	置換誤り
1	-	762.7	-
2	-	2254.8	751.2
3	761.4	3830.7	2267.3
4	2289.1	4799.1	3848.8
5	3868.9	4625.9	4807.3
6	4953.4	3914.4	4813.2
7	4816.9	2768.1	3957.3
8	3897.3	1698.4	2755.8
9	2769.6	916.1	1722.8
10	1716.8	457.8	954.0
11	956.3	210.7	475.2
12	479.5	93.0	221.7
13	223.8	40.6	103.1
14	104.7	16.9	46.8
15	46.1	-	19.4
16	20.4	-	-

(ii) 線形ハッシュ法を用いたファイル

第3節で示した類を用いて，ファイルのブロックサイズが異なるときの記憶利用率を図10に示す．2次記憶への読み込み回数，書き込み回数の平均値はそれぞれ1.036，1.005となった．読み込み回数が1よりも大きいのは，オーバーフローバケツがあるファイルではオーバーフローバケツも読み込まなければならないからである．書き込み回数が1よりも大きいのは，バケツを分割したときに，2つのバケツを書き込まなければならないからである．拡張ステップ数rが増加するほど，記憶利用率は上昇している．また，オーバーフローバケツのサイズが異なっていても記憶利用率はほとんど変わらない．これは，ファイル全体に対するオーバーフローバケツの割合が非常に小さいため，ファイル全体の記憶利用率にはほとんど影響していないからである．

5.2 類似キーの検索

類似したキーの検索法の評価には，検索率と検索に要する計算量がある．実験では，各長さのキー集合に対し，置換・挿入・脱落の3種類の誤りのいずれか1種類をランダムに発生させたキーを入力キーとして用い，最も類似したキーの検索実験を行なった．検索結果の評価は，誤ったキーを入力したときの正検索率を調べた．検索結果は，拡張ハッシュ法，線形ハッシュ法とも全く同じで，表2のようになった．

最後に，類似キーの検索効率の評価として，表3に1つの入力キーに対する類似キー検索のためのレーベンシュタイン距離の計算回数の平均を示す．登録されているキーの数が多いキーの長さのファイルを検索すると，レーベンシュタイン距離の計算回数が増加する．

6. あとがき

記憶利用率が高く，類似したキーを検索できるファイルの構成法を提案した．記憶利用率等の理論的検討は今後の課題である．

参考文献

[1] R. Fagin et al., "Extendible hashing-a fastest access method for dynamic files," *ACM Trans.Database Systems*, vol.4, no.3, pp.315-344, 1979.

[2] W. Litwin, "Linear hashing: A new file and table addressing," *Proc. 6th Conf. on Very Large Database*, pp.212-223, 1980.

[3] R. J. Enbody, H. C. Du, "Dynamic hashing schemes," *ACM Computing Surveys*, vol.20, no.2, pp.85-113, 1988.

[4] 楊 昇達, 田中榮一, "類似キー検索のための部分拡張ファイル" 平7関西連大, p.G332, 1995.

[5] S. Kawade, E. Tanaka, "A similar key search file based on extensible hashing," *IEICE Trans. INF. & SYST.*, vol.E78-D, no.9, pp.1218-1220, 1995.

[6] 平出基一, 田中榮一, "B木に基づく類名表記機能ファイルシステムの構成法," 信学論, vol.J77-D-I, no.12, pp.794-802, 1994.

[7] R.A.Baeza, P-Å.Larson, "Performance of B^+-trees with partial expansions," *IEEE Trans. Knowledge and Data Eng.*, vol.1, no.2, pp.248-257, 1989.