1S−11　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

佐藤　雅博　青江　順一
徳島大学

1．はじめに

今日，データベースのようにたくさんのキー（情報）からなるファイルを扱う処理の必要性が高くなっている．そのファイルの保守のためには，情報の検索・挿入・削除といった処理を行わなければならない．これらの処理を行うアルゴリズムの一つとして，静的(static)ハッシュ法，動的(dynamic)ハッシュ法が用いられている．

静的ハッシュ法は，ハッシュ表やハッシュ関数をあらかじめ定義するので，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．しかし，この性質が予知できない分野では，ハッシュ表の大きさによって，記憶領域の無駄の発生や，衝突（あふれ）の頻発による検索効率の低下が起こる．

一方，動的ハッシュ法は，キーの性質が予測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する事によりハッシュ法の高速検索を維持させようとする手法である．

本稿では，この動的ハッシュ法の一つであるコンパクト2進木(Compact Binary Trees,CB-Trees)[2]と呼ばれる手法の改良法（以下，本手法と略記する）を述べる．

以下，2．でCB-Treeと，そのもととなった2進デジタル検索木(Binary Digital Search Trees,BDS-Trees)について概要を示し，3．で本手法についての解説を行う．4，5．では本手法の評価および，今後の課題について記す．

2．　従来の手法

2．1　BDS−Tree

BDS-Treeとはその名の示すとおり，キーを2進数表現にしたトライ(trie)構造である．トライとは，木構造の一種であり，枝の選択をキーの全体ではなくその一部に基づいて行うものである．また，枝の選択はキーによって一意に決定されるので，木を構成するノードには比較のためのキーを持たないという特徴がある．

図1にBDS-Treeの例を示す．

2．2　CB−Tree

BDS-Treeで，図1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必要となる．Jongeら[2]はノードから必ず2本のアークがでることを保証するために，ダミーページをつけ，コンパクトなデータ構造を提案した．これが，CB-Treeである．

CB-Treeの求め方は，トライに対して木の先行順(preorder)走査により，○印の内部ノード通過時にビット0を，ページ通過時にビット1とそのページに登録したキーがあるレコードへのポインタを出力していくことを繰り返せば良い．このCB-Treeを図1のBDS-Treeについて作成すれば，次の結果が得られる．なお，aaaaはキーAの，bbbbはキーBのポインタに対応している．また，DDDDはダミーノードである．

0001aaaa01DDDD001bbbb1cccc1DDDD1dddd01eeee1ffff

（但し，上のようなビット表現ではわかりにくいので，図は全てBDS-Treeで示している．）

CB-Treeでの検索方法は，先行順走査に従って行われ，挿入と削除は，ビットシフト演算で行われる．

BDS-Tree，CB-Treeの詳細について興味のある方は，参考文献[2],[3]を参照されたい．

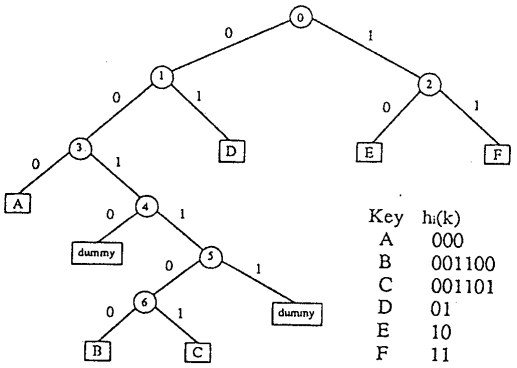

図1　BDS-Treeの例

3．　本手法（改良CB-Tree）について

2．で述べたCB-Treeは，たいへんコンパクトなデータ構造であるが，その反面，検索・挿入・削除の処理時間が大きいという欠点がある．これはCB-Treeの操作が全て先行順の性質に従ったビット操作が基本となっているためである．例えば，アルファベット26文字からなるキー10,000個を大きさ10のページに登録する場合，配列を用いたトライ構造では，最低でも26,390もの大きさのポインタ用配列を用意しなければならないが，CB-Treeではレコードへのポインタ（16ビットと仮定する）を含めて最低約18,500ビットで表現できる．これは，1ページあたり約19ビットで，トライ構造の場合の10%程度と非常に小さい値であるといえる．しかし，最悪の検索コストは，2^{10}に比例する．

A Compression of Directories for Extending Hashing
Masahiro SATO and Jun-ichi AOE
Tokushima University

この欠点の解消法として，本稿では，トライ構造の分割による処理時間の改善法を提案する．

まずはじめに，例を用いてこの手法について解説する．図２には，図１と同じキーを登録してあるが，各々のトライ構造の最大全域深さが２となっている．この改良CB-Treeでキー Ｃを検索する場合には，キー Ｃの２進数表現を全域深さ以下となるように分割する．H(k)=001101であるので，00/11/01の３つに分割できる．このうち，１つ目の分割キー00を用いて，Tree1を検索する．このとき，ノード３に，次の木へのポインタがあるので，次の分割キー11を用いて，Tree2を検索する．同様にノード５にポインタがあるので，次の分割キー01を用いてTree3を検索する．これによってレコードへのポインタが見つかるので，そのレコード内に登録されているキーと比較することによって，キー Ｃが登録されているのが確認される．

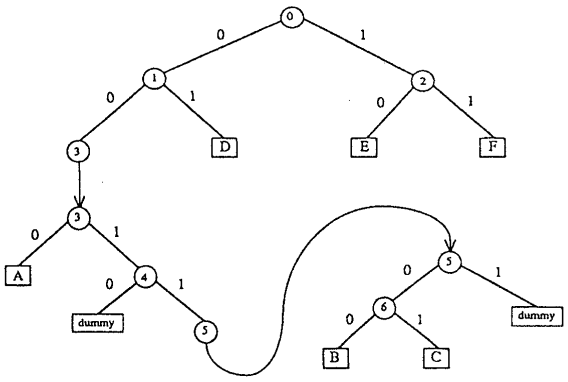

図２　拡張CB-Treeを示したBDS-Tree

一般的には，全てのトライ構造の最大全域深さLを設定し，キーの２進数表現をLビットごとに以下のように分割する．

$H(k)=H_0(k)H_1(k)\ \ldots\ H_n(k)$

$(H_0(k)\sim H_{n-1}(k)$ ： L bit，$H_n(k)$ ： ≦ L bit)

そして，以下のアルゴリズムに基づいて検索を行う．

1) i=0，T=ルートとなるCB-Treeをセットする
2) Tに$H_i(k)$が登録されていなければ検索失敗として終了
3) $H_i(k)$が木へのポインタを持つならば，
   T=次の木，i=i+1として２へ
4) $H_i(k)$がレコードへのポインタを持つのでその中のキーと比較し，同じならば検索成功．

なお，キーの挿入・削除はCB-Treeに準ずるものである．

この改良を行うことによって，分割されたトライ一つ一つの深さに上限をもたせることになる．そのため，各トライの内部にあるノード数が減少し，CB-Treeでインプリメントした際にも，１つのトライあたりビット長が減少するので，その利点であるデータ構造のコンパクトさを失わずに各処理に要する時間を短縮することができる．

４．評価

なお，空間使用量・各操作の時間計算量の概算は以下のようになる．但し，次の変数は以下のような意味を持つ．

n:キーのビット長］
m:分割されたトライの最大全域深さ
k:分割されたトライの最大リンク数
　＝ ⌈n/m⌉（n/m以上である最小の整数）

また，全ての計算は最悪の場合を想定して行い，キーを全て登録したBDS-Treeは完全木であると仮定している．

表１　最悪の場合の計算量

	CB-Tree	改良CB-Tree
記憶量	$O(2^{n+1})$	$O(2^{m+k})$
検索	$O(2^{n+1})$	$O(k\cdot 2^{m+1})$
挿入	$O(2^{n+1})$	$O(k\cdot 2^{m+1})$
削除	$O(2^{n+1})$	$O(k\cdot 2^{m+1})$

この表だけではわかりにくいので(m,n,k)=(32,8,4)としたときの例を示す．キーのビット長が32となるキーを全てCB-Treeに登録するために必要な領域を計算すると，記憶領域の消費量は，CB-Treeと比較して約７０％程度増加する．しかし，CB-Treeが元来非常にコンパクトであるので，これだけの増加でも充分実用的な値である．また，各処理に要する時間については，パラメータの設定や，実際に使用されているキー集合の性質によって若干の変動はあるものの，CB-Treeの最悪の処理時間は大幅に改善される．

５．むすび

本稿では，動的ハッシュ法の一つであるコンパクト２進木の改良法を示した．もともとコンパクト２進木は，データの格納方法にほとんど無駄がないため，ディレクトリ方式などと比較すると使用する記憶領域量では大変有利であるが，速度が犠牲になるという欠点があった．その欠点を改善する手法を示した訳であるが，理論的な評価によれば，使用する記憶領域量が増加するが，処理速度が向上できることがわかった．

今後は，このアルゴリズムを実際にインプリメントして，様々なキー集合に対しての実際のテストを行って行きたい．また，更に改良が可能であるかも調べて行きたい．

参　考　文　献

1) 青江順一：キー検索技法－動的ハッシュとその応用－ 情報処理 Vol.33, No.10, (1992). 【掲載予定】
2) R.J.Enbody, et al.: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).
3) W.D.Jonge, et al.:Two Access Methods using Compact Binary Trees, IEEE Trans. Softw. Eng., Vol.SE-13, No.7, pp.799-810, (1987).