4F-4

階層型ハッシュ法の辞書データベースへの応用*

蓮井洋志†

室蘭工業大学情報工学科‡

1 はじめに

ハッシュ法はデータベースを実現する際に検索が高速なことで有名な方法である。この方法は、キー文字列をハッシュ数に変換し、ハッシュ数と対応づけられたハッシュテーブルのインデックスに格納する。

ハッシュ法ではハッシュ数を計算するハッシュ関数やハッシュテーブルの大きさなどを登録するデータによって考えなければならない。しかし、登録するデータの数が決定できない場合がある。例えば、形態素解析システムの辞書には未登録語がある。そのため、文書によっては解析できなくなる。逐次、未登録語を登録する必要がある。登録する語の上限の数はわからない。そのために、多くのデータが衝突をおこす結果となる。衝突防止のために、ハッシュ法に工夫が必要になる。ダブル法、チェイン法 [1] が有名である。

ダブル法ではハッシュテーブルに登録する単語の数の上限を知らないといけない。チェイン法では、衝突した単語はリストにする。衝突した単語が多くなる場合は検索が低速となる。そこで、我々は登録するデータの総数を知らない場合でも実現できる階層型ハッシュ法を提案する。ハッシュ数が一致する異なった単語がある場合、そのノードから小さいハッシュテーブルを作り、その子テーブルに前とは異なったハッシュ関数でハッシュ数を求め、それを使ってその子テーブルに登録する。従来のハッシュ法と異なって、登録する単語数がハッシュテーブルの大きさより遥かに大きい場合でも高速な検索が期待できる。

本研究では、形態素解析システム WordClass の辞書データベースに階層型ハッシュ構造を応用した。このデータ構造をファイルシステムに応用することで高速で大容量のデータベースを作ることが可能である。

本稿では、2 節では階層型ハッシュ構造について説明し、3 節では辞書データベースへの応用について説明し、4 節では階層型ハッシュ法について他のデータ構造と比較しながら考察する。

*Application for Dictionary Database with Hierachical Hash Method

†Hiroshi Hasui

‡Department of Computer Science and Systems Engineering in Muroran Institute of Technology

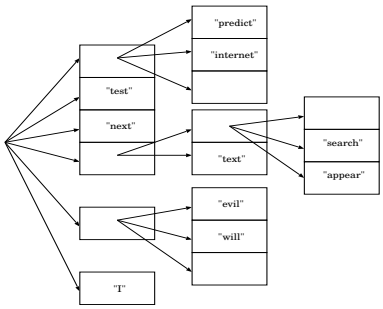

図 1: 階層型ハッシュ構造

2 階層型ハッシュ法

2.1 階層型ハッシュ構造

図 1 は階層型ハッシュ構造である。親ハッシュテーブルから、子ハッシュテーブルがカスケード状に並ぶ。そのために同じハッシュ数の単語が衝突する可能性がない。

2.2 ハッシュ関数

ハッシュ関数は以下のとおりである。C 言語で書いた。layer が階層数で string が文字列 indexsize である。%は剰余算である。INT_MAX は整数型変数の最大値、number は予備ハッシュ数である。

```
int hash(int layer, char *string){
  int                           i, co = 1;
  int                           number = 1;

  for(i = layer % strlen(string);
                      i < strlen(string); ++i){
    number = number * co + string[i] * 255;
    co++;
    if(number > INT_MAX / 256)
      number = number % indexsize;
  }

  for(i = 0; i < layer % strlen(string); ++i){
    number = number * co + string[i] * 255;
    co++;
    if(number > INT_MAX / 256)
      number = number % indexsize;
  }
  //  number 予備ハッシュ数
  return number % indexsize;

}
```

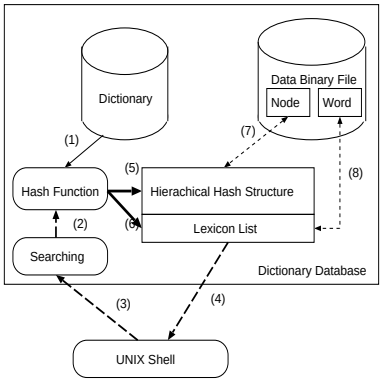

図 2: 辞書データベースシステムの構成

2.3 ハッシュテーブルの大きさ

ハッシュテーブルの大きさは、親テーブルが 10003、子テーブルのサイズは素数である。ハッシュ関数ではテーブルサイズの剰余算を行なうために、テーブルサイズが素数であれば子ハッシュテーブルで衝突がおきにくい。

3 辞書データベースシステム

図 2 に辞書データベースシステムの構成図を書く。システムが起動すると、(1) 辞書ファイルからハッシュ関数が選別して、(5) メモリ上の階層型ハッシュ構造に登録する。(6) 辞書の単語の情報がリストで管理される。(3) シェルが検索要求をシステムに出すと (2) 検索関数が構造型ハッシュ構造をたどって、(4) 検索結果をシェルにかえす。

辞書のデータを一度にメモリに移し、(7) そのイメージを別ファイルに登録する。(8) そのイメージを移すほうが速いために、2 度目以降の検索ではイメージファイルから辞書情報をメモリに直接移す。高速化のためである。

4 考察

ハッシュ法はデータベースを作るとき、ハッシュ関数やハッシュテーブルの大きさなどをデータによって微調整する必要がある。ハッシュ数の衝突を避けるためである。また、ハッシュテーブルの大きさはデータ数によって変えることができれば、メモリを節減できる。UNIX の基本ソフトウェアに gdbm がある。チェイン法を使ってデータベースを作るライブラリである。このライブラリは、辞書データベースシステムだけではなく、さまざまなデータのデータベースとして活用される。しかし、用途が辞書以外の場合でもハッシュ関数やハッシュテーブルは変わらない。こういったデータベースの場合、ハッシュ数の衝突やメモリの問題を解決できない。階層型ハッシュ法では、ハッシュテーブルの大きさが小さすぎても、階層を持つがゆえにチェイン法より高速に探索できる。

辞書データベースのデータ構造としては、TRIE 構造、パトリシア木などが有効である。これは、1 回の探索で文の探索対象の文字以降の単語をすべて探索できる。例えば、「データベースを」という文字列を探索対象とすると、「データ」と「データベース」の 2 語が探索結果となる。このとき、TRIE では 1 回でこの 2 語を探索できるが、ハッシュ法では「デ」「デー」「データ」「データベ」「データベー」「データベース」「データベースを」の 7 回の探索が必要となる。TRIE は 1 回当りの探索時間が多少長くてもハッシュ法よりも高速な探索ができる。しかし、「データ」という単語が辞書にあるかどうかを探索する場合にはハッシュ法が効果的である。

階層型ハッシュ法と比較して、TRIE は非常にメモリを消費する。約 12 万語の辞書に対して階層型ハッシュ法では 11.3MB メモリを必要とするが、TRIE の場合は 68.2MB であった。TRIE は非常に高速なデータ構造であるがメモリの効率は良くない。

辞書ファイルから 68.2MB のデータをロードするのにかかる時間が 11.3MB 読みだすのと比較して大きすぎるために TRIE は探索がいかに高速でもプログラムの実行時間がかえってかかってしまう。

パトリシア木は、文字列「データベースを」の探索に 7 回の探索を必要とするが、データベースに登録する単語の数だけしか、ノードを必要としない。辞書ファイルが小さくて済む。探索には単語の数 n とすると、計算量は $log_2 n$ で、最後にノード中のキーワードと比較して一致しているかどうかを検査する必要がある。階層型ハッシュ法は、ハッシュテーブルのサイズによってちがうが、究極的に計算量は 1 で、最後にキーワードと比較して検索結果があるかどうかを検査する。

階層型ハッシュ構造は埋まっていないインデックスが非常に多くできた。子ハッシュテーブル各々のサイズ、ハッシュ関数によっては未使用のインデックスを減らすことが出来る。将来的に、未使用のインデックスを減らすための研究を行ないたい。

参考文献

[1] Maurer, W. D., and T. G. Lewis. Hash table methods, *Computing Surveys* 7:1, pp. 5-20.