情 報 学 基 礎　4−2
（１９８７・２・２０）

データベース符号化の試み

中野良平　　木山稔
（ＮＴＴ　電気通信研究所）

データベースが全て固定長であれば、データベース管理は簡単で高速になる。可変長データが短い固定長符号に１：１に変換できれば、データベース管理の簡単化、高速化だけでなく、データベースの圧縮、機密保護も期待できる。本稿では、符号化の論理レベルのアルゴリズムとして符号化情報が少量で済む木構造を用いた方法を採用し、格納レベルのアルゴリズムとしてデータの増加に柔軟かつ効率的に対処できる動的ハッシュを用いた符号化法を提案する。符号化／復号化処理にＩ／Ｏ処理を入れず、大規模化が進む主記憶上でのみ高速に実行することを狙う。提案した符号化法を電話帳データベース等に適用した実験結果について報告する。実験の結果、上記狙いの実現性が確認できた。

EXPERIMENT OF ENCODING DATABASES

Ryohei NAKANO　　Minoru KIYAMA

NTT Electrical Communications Laboratories
1-2356, Take, Yokosuka-shi, Kanagawa-ken, 238-03 Japan

If databases consisted of only fixed-length data items,how easy and fast database management would be ! Encoding a variable-length data item to a fixed length code not only makes it possible, but brings about both database compaction and database encryption. This paper proposes a new encoding approach, which uses oriented tree in representing variable-length items and dynamic hashing to cope with consecutive data addtion. The present method is intended to run on large main memory without disk I/O operations, which leads us to high-speed encoding/decoding. The approach was tested using yellowpages database and so on, and the experiment showed that the approach is feasible with up to middle-scale databases and gave us several findings about encoding databases.

1. はじめに

『もしも、データベース中のデータが全て固定長だったら、どんなに楽だろう！！処理は簡単で速くなるのに…。』　データベース管理メカニズムの開発者は誰しもそのように考える。しかし、現実には可変長データを無視することはできない。事実、可変長が扱えない商用のデータベース管理システムは皆無といってよい。

84	12	4	412860	8	681.3.06	4	427
13	The MIT Press			17	The Art of Prolog		
…				…		…	
	10	L.Sterling		9	E.Shapiro		

図1　　可変長データの形式

近年、データベースのアクセスインタフェースの高度化が進む一方で、アクセス処理の性能劣化が問題になり、データベースマシンの研究開発が続けられている。可変長データはそこでも問題になる。図1に示すような可変長データを分解して個々の値を取り出す処理が、扱うデータ分だけ繰り返す必要があり、その処理負荷が馬鹿にならず、飛躍的な性能向上を阻む1つの大きな要因となっている。

そこで、可変長データを比較的短い（例えば4バイト位）固定長符号に1対1に変換できないかと思い至る。現に、インタプリタやコンパイラでは、文字列を固定長の符号（多くの場合アドレス）に対応させて処理の高速化を図っている。しかし、データベースでは対象とするデータ量がインタプリタ等の対象であるプログラムに比べて余りにも多いので、どのような符号化が望ましいのか明らかでない。

データベースを上記の背景から固定長の符号に変換する研究はこれまで余り行われていない。データベースマシン研究で提案された符号化法［1］は数少ないその種の研究の1つで、拡張可能ハッシュを用いているが、符号化対象の扱いに関しては、ハッシュする際常套的に用いられる集合型としている。

本稿では、符号化対象の扱い（論理レベルのアルゴリズム）に関しては、符号化のための情報が少なくて済む木構造を用いた方法［2］を採用し、格納レベルのアルゴリズムとしてはデータの増加に柔軟かつ効率的に対処できる動的ハッシュ［3］を用いた符号化法を提案する。同法は、符号化／復号化処理からI／O処理を取り除き、大規模化が進む主記憶上でのみ高速に実行することを狙っている。更に、その符号化法を実際に運用されている電話帳データベースと図書館データベースに適用した結果について報告する。

2. 符号化の方法

2．1　符号化の位置付け

データベースの符号化／復号化は、データベース利用者が意識する必要もないし、また意識させるべきでもないので、データベース管理メカニズムの中に閉じて処理する（図2）。即ち、利用者とのインタフェースでは可変長／固定長データが混在し、符号化／復号化機構が可変長／固定長のデータ変換を司る。より内側のデータ処理機構は固定長符号を含む固定長データのみを処理対象とするため、機構がシンプルとなり、演算処理の高速化が望める。

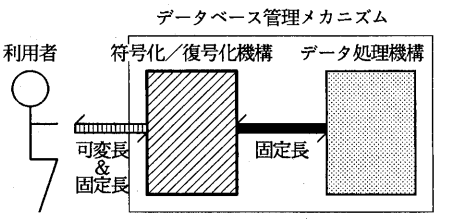


図2　符号化／復号化の位置付け

2．2　符号化の条件

可変長データを変換して生成される符号はどのような条件を満たすべきであろうか。

（1）固定長符号であること

主にデータベース圧縮の観点から、データベース中のデータをコンパクト符号（Huffman code等）を用いて符号化する研究、事例は少なくない。しかし、ここでは値の固定長化を目論んでいるので、可変長データを生成することになるコンパクト符号では都合が悪い。

（2）１：１変換であること

符号化のソース（可変長データ）とターゲット（固定長符号）とは１：１に対応する必要がある。これは、データ処理機構が変換された符号を対象に演算することを考えれば明らかである。例えば、検索条件の中に『住所が“横須賀市武”であること』が含まれていたとする。即ち、関係論理［４］風に表現すると、

… x［住所］＝“横須賀市武” …

今、“横須賀市武”の符号が、例えば１０００とするとき、データ処理機構で処理すべき検索条件は

… x［住所］＝１０００ …

となる。“横須賀市武”以外のものが１０００として符号化されてはならないとともに、“横須賀市武”が１０００以外に符号化されてはならないことが分かる。

（3）順序性保持（order-preservation）は必須ではない

符号化対象は主に可変長（勿論、固定長でも長大なときは符号化すべき）文字列であり、固定長になる数値データを符号化する必要はない。検索式中に大小比較が現れた場合、符号化しない数値データに対してならば問題ないが、文字列の場合には、符号化で順序性が保持されてないと、符号を大小比較しても意味がない。しかし、データベースの符号化では、符号化対象が次々と追加されるので順序性の保持は困難である。

現実には大小比較は、多くの場合、数値データに対してであり、文字列を対象とした場合は少ないと思われる。例えば、知識処理で数多く現れるユニフィケーションは等号演算に対応する。従って、必要性は少ないが、どうしても文字列を大小比較したいという場合には、少し工夫が必要である。文献［１］で示されているように、x［書名］＞Ａのような場合には、＞Ａを満たす文字列集合を求め、それらに対応する符号集合を得て、それらとの一致検索に置き換えるか、符号の中に部分的に順序性を表わす領域を設ける等である。

２．３　符号化の効果

データベースを符号化する効果は、次の４点にある。即ち、（ａ）データ処理機構の単純化、（ｂ）データ処理機構の高速化、（ｃ）データベースの圧縮、（ｄ）データベースの機密保護である。前２項は既に述べた。それ以外に、長大な可変長文字列が短い固定長符号に変換されるので、データベースが圧縮される。また、符号は一種の暗号であり、暗号を解く鍵である符号化情報を別管理して洩れないようにしておけば、データベースの機密保護にも役立つ。後述の方法で生成される符号は、暗号法としては、コードブック方式に当たるが、その一般的解法はない。今の場合、コードブックのエントリ数は極めて多い（３バイトとして2^{24}＝数10^{7}、４バイトとして2^{32}＝数10^{9}）ので、適当な仮定を置いて総当りで調べるとしても不可能に近い（10^{7}の階乗の組み合わせ！）し、解明結果の真偽を判定することも難しい。また、一部が判明しても他との相関がないので、そこから解明されてしまうこともない。従って、暗号としての強度は強いと思われるが、データベースを覗かれた場合、符号の分布からある程度類推される部分が出ることは考えられる。

２．４　木構造型符号化法

符号化法の論理レベルのアルゴリズムは、有向木（oriented tree）［５］を用いた方法とする。即ち、符号化対象の文字列群を有向木で表現し（図３）、各エントリ（動的ハッシュにおけるノードと区別するため、及び連想メモリを用いて実現するとき１エントリに当たるのでこの用語とする）に符号を１：１に割付ける。一方、相異なる文字列を各々独立して持つ方法（集合型と称す）が常套的に考えられるが、前者の木構造表現では、先頭から一致した文字列が共用できるので、集合型に比し符号化情報が縮小できる利点がある。符号化情報の縮小は符号化処理を全て主記憶上で行う場合には極めて重要なポイントとなる。縮小の程度は実データベースで評価する。以下、木構造符号化法について説明する。

文字列の符号化時には、まずそれが既に登録済みでないか検索する。検索はルートエントリから一致を判定して行き、最後まで一致したならば登録済みとなる。

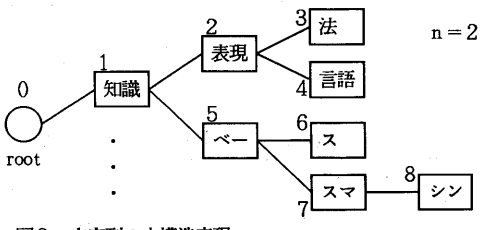
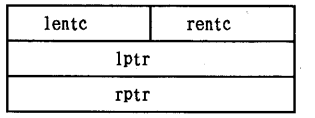


図３　文字列の木構造表現

エントリを１段進める時には、上位エントリの符号と次の部分文字列（ｎ文字）をペアにして一致を調べる。
復号化時には、最後の符号から逆に辿ることにより、元の文字列が後から得られる。
文献［２］では、各エントリには１文字を対応させているが、長大な文字列の場合、辿る回数が増える問題が生じるので、本稿では、１エントリにｎ文字を対応できるようにし、妥当なｎを調べることとした。

２．５　動的ハッシュ法

符号化法の格納レベルのアルゴリズムは、連想メモリを用いる方法とハッシュを用いる方法がある。前者は後者に比し、より高速な検索が可能であり、前述の大小比較の際の文字列群の抽出が容易という利点があるが、反面、小容量、高価という難点がある。ここでは、大容量化を重視し、ハッシュによる方法とする。
データベースの符号化では、符号化対象が次々と追加されることを考慮する必要がある。従って、ハッシュテーブルが動的に拡張できるハッシュ法が望ましい。そのようなハッシュ法として動的ハッシュ［３］やリニアハッシュ［６］が知られている。しかし、リニアハッシュは、アルゴリズムはより簡潔であるが、各バケットを均等に分割するので、分布に偏りがある場合には、空間効率が良くないと想定されるので、オーバフローしたバケットを分割する動的ハッシュ法を採用した。なお、文献［３］では、内部ノードと外部ノードを用意しているが、図４のようなノードを考えれば、後者をなくすことができる。後で分かるようにノードの占める量は少なくなく、これによりノード量を約半分に減ずることができる。図３の文字列群の格納例を図５に示す。
ノードを辿る時、右か左の選択において、ほぼ１／２の確率で０、１を生成する関数が必要になるが、主に性能上の理由から通常組み込まれている乱数生成器を使わず、ランダムビット生成器を用いた。ただし、今の場合には長いビット列のランダム性よりは、最初の１０ビット位までのランダム性の方がデータをバラつかせる観点やバケットの分割時にデータを均等に分ける観点から重要である。ランダムビット生成器はその点を考慮し、文献［７］（p.28）に若干の修正を加えた。即ち、下位ビットにより有効な情報が現れるので右シフトとし、更によりランダム性を出すためオーバフロービットに拘わらず排他的論理和をとることとした。



lentc,rentc : バケットのエントリ数.
-1のとき、ポイント先はノード
lptr,eptr : ノードまたはバケットへのポインタ
図４　動的ハッシュにおけるノード

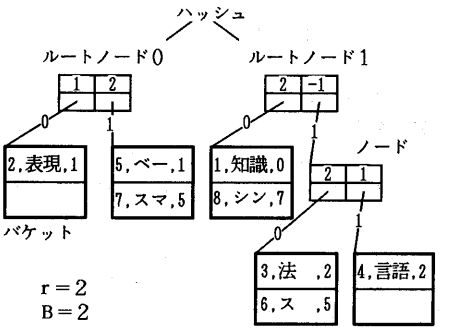


図５　動的ハッシュによる格納

3. 実験結果

３．１　実験環境

前述の符号化法アルゴリズムをプログラム化し、８メガバイトの実メモリを実装した計算機環境で符号化実験を行った。データベースとしては、東京都２３区職業別電話帳データベースとＮＴＴ横須賀研究センタ図書館データベースを用いた。前者は１種類のリレーション（レコードタイプ）で、３８７バイトのタプル（レコード）約１００万件から成り、総量約３８７メガバイトのデータベースである。図６にタプル例を示す。後者は数種のリレーションから成り、総量１０メガバイト近いデータベースである。

名義（掲載名）	名義（主掲載）	電話番号	住所１	住所２	住所肩書	職業付記
電気通信科学館		03-241-8080	千代田、大手	2-2		
電電共済トラベルサービス		03-501-5520	千代田、内幸	1-1		

図６　電話帳データベースのタプル例（符号化対象属性のみ）

3．2　木構造型　ｖｓ．集合型

符号化法の論理レベルのアルゴリズムとして、木構造型と集合型を採り上げ、図書館データベースに対する所要エントリ数を比較した（図７）。考察（４．３節）で示すように、１エントリに格納する文字数ｎを固定したとき、エントリ数は所要メモリ量と線形関係にある。

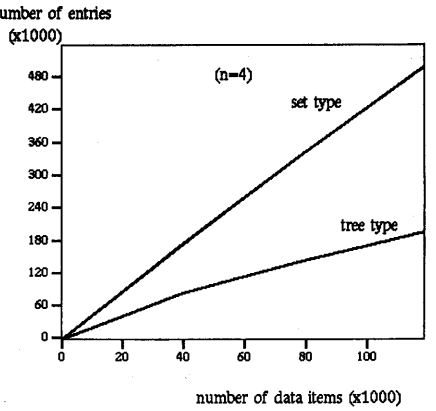


図7　木構造型　ｖｓ．集合型のエントリ数比較

3．3　符号化単位

有向木の１エントリに格納する文字数ｎを増せば、エントリを辿る回数は明らかに減り、更に束ねる効果が出て、総エントリ数も減少する。図書館データベースの実測結果を図８に示す。

ｎを増せばエントリ数は減るが、一方では１エントリの大きさが増えるため、所要メモリ量はトレードオフの関係を反映する。図書館データベースの実測結果を図９に示す。

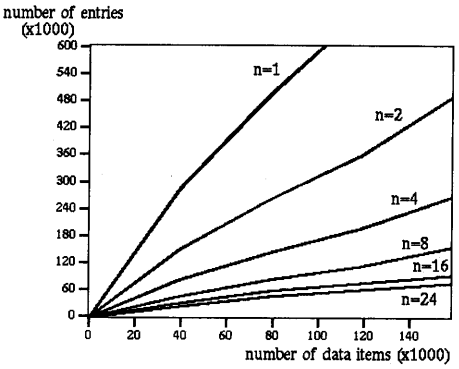


図8　符号化単位によるエントリ数比較

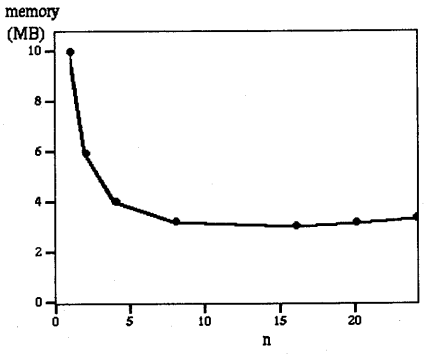
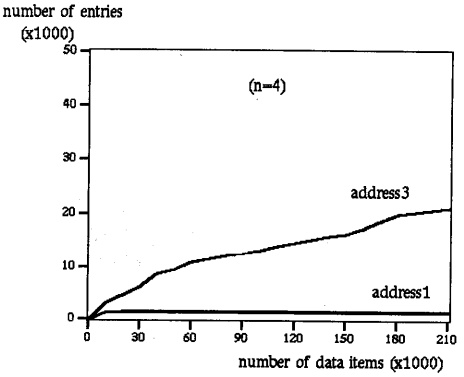


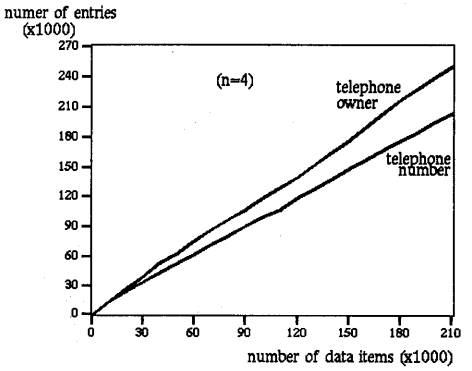
図9　符号化単位による所要メモリ量の変動

3．4　データベースの符号化特性

符号化する文字列を木構造で表現する背景には、先頭からの部分文字列の共用が多くなりデータの増加に対してそれ程符号化情報容量が増大しないのではないかとの期待がある。実際に符号化してみると、期待通りに飽和してくるものと、それ程でもないものがある。電話帳データベースの例を図１０に示す。



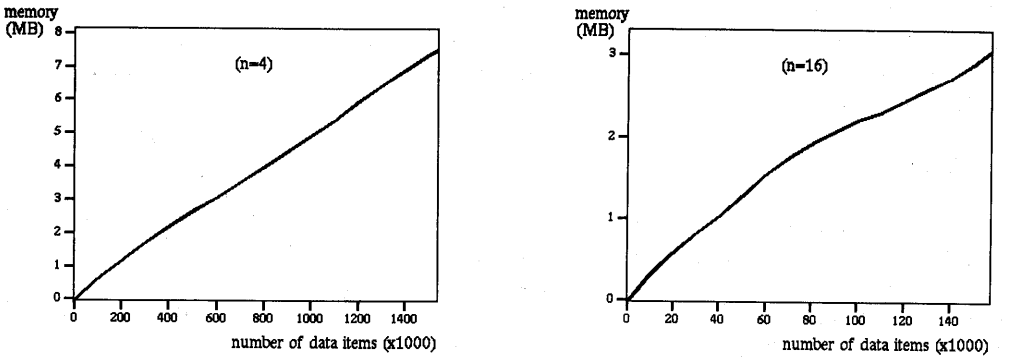
（ａ）飽和するタイプ



（ｂ）容易に飽和しないタイプ

図１０　属性毎の符号化情報容量の伸び

符号化できる全属性を対象に、電話帳データベースの一部（約１／５）、及び図書館データベース全体を符号化した際の符号化情報総容量の伸びを消費するメモリ量を尺度にして図１１に示す。

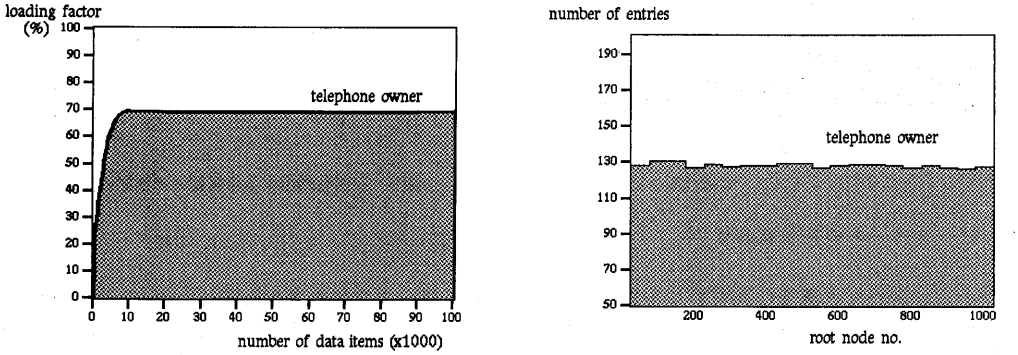


（ａ）電話帳データベース　　（ｂ）図書館データベース

図１１　符号化情報総容量の伸び

３．５　動的ハッシュの特性

ハッシュでいつも問題になるのは、バケットへのデータの格納率（loading factor）である。電話帳データベースの属性（名義）で測定した格納率の推移を図１２（ａ）に示す。また、もう１つのポイントは、ハッシュによって均等にデータが分散されているかという点にあるが、その様子を図１２（ｂ）に示す。

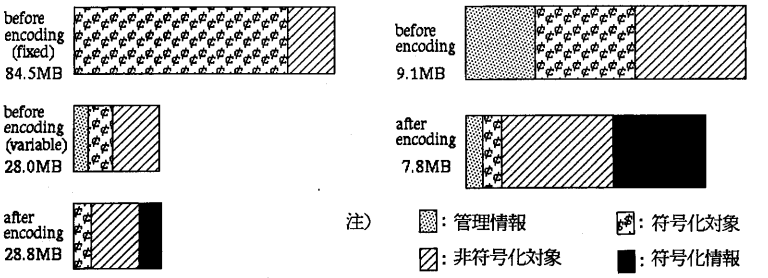


（ａ）格納率の推移　　（ｂ）データの分散状況

図１２　動的ハッシュの特性

３．６　データベースの圧縮効果

データベースの符号化により符号化された部分はコンパクトになることが期待できる。符号化されない部分も含めて全体としてどうなるかを図１３に示す。また、符号化情報の大きさも合わせて示す。



（ａ）電話帳データベース　　（ｂ）図書館データベース

図１３　データベースの圧縮効果

4. 考察

4．1　木構造型 ｖｓ．集合型

木構造表現は、文字列間で先頭から一致した文字列が共用できるので、集合型に比し符号化情報が縮小できることは既に述べた。その効果の程は図７を見ると判然としている。集合型では、主記憶上だけでの高速符号化はとても望めないことが分かる。

有向木の１エントリに格納する文字数ｎを増せば、エントリを辿る回数が減るので、符号化／復号化の処理は速くなる。ｎの増加とともに、総エントリ数も図８のように減るが、１エントリの大きさが増すため、ノードとバケットの総容量（全て主記憶で処理するとした場合の所要メモリ量）は図９に示すようにｎ＝１６位で最少となる。対象とした文字列の平均長は２０バイトであるので、それに近い値が良いように思われる。図８の傾向はデータ特性により異なるが、同じデータベースの場合には各ｎの関係は同じ傾向を示すと想定されるので、規模が増えたからといってｎの値を変える必要はない。いずれにしても、データベースの符号化ではｎ＝１でなく、ｎ＝文字列の平均長が良さそうである。

4．2　データベースの符号化特性と規模増大の影響

符号化する文字列を木構造で表現すれば先頭からの部分文字列の共用量が多くなり、データの増加に対する符号化情報の増分が単調減少するのではないかと予想される。属性毎に調べてみると、図１０（ａ）のように期待通りに減少するものと、図１０（ｂ）のように実験した範囲ではそれ程でないものがある。増分が減らないのは部分文字列の共用が少ないためと思われる。電話番号、名義、書名、等の一意の識別子になるような属性がその例である。電話番号を考えると、これは数字とハイフンの組み合わせに過ぎないので共用の効果がそのうち現れると思われる。事実徐々にではあるが、増分は減って来ている。名義の場合は文字の種類が多いためかなり蓄積しないと、減少傾向は現れないと思われる。特に、この場合データがソートされて入っているので、減少傾向が陽に出ず、始めから低い傾きで直線傾向を示した。このように考えると、どの属性もいずれは減少し始めるが、その時定数がかなり異なるということになる。

データベース全体の符号化の様子は図１１に見る通り、増分が減らない属性のウエイトが大きいため、全体としてほぼ直線傾向にある。電話帳データベースでは、約２１．８万タプルを符号化し、ハッシュテーブルとして約７．５メガバイトのメモリを使用した。直線的に外挿すると、東京都２３区職業別電話帳データベース全体では、約３４．４メガバイトのメモリが必要ということになる。また、図書館データベースでは、約３．２万タプルを符号化対象（符号化しないリレーションは除く）とし、約３～４メガバイトのメモリを使用した。これらから、小規模～中規模のデータベースを、大規模化する主記憶上で高速に符号化／復号化することが可能であると考える。

4．3　動的ハッシュの有用性

動的ハッシュの格納率の期待値はＢトリー並みの６９％であることが解析されている［３］。実験でも、図１２（ａ）に示すように、６９％に収斂することが確認できた。また、ルートノードへのハッシュ関数によるデータ分散の具合は図１２（ｂ）に示すように極めて良い一様性を示している。この一様性により各ルートの下に格納されるデータ数がバランスし、木の段数が程良く押さえられる。

ここで、容量を規定する諸量の関係を確認しておく。今、データベースの符号化がある程度進んだ時点において、全体のノード（ルートノードを含む）数をｄ，ルートノード数をｒ、バケット数をｂ，エントリ数をｅ、ノードサイズをＤ、バケットサイズ（１バケットに入るエントリ数）をＢ、エントリサイズをＥ、使用メモリ量をＭとすると、以下の関係が成り立つ。これから、エントリ数は使用メモリ量と線形関係にあることが分かる。

$$b = d + r$$

$$e / (b * B) = 0.69$$

$$M = b * B * E + d * D$$

4．4　データベースの圧縮効果

一般に可変長データは長大なので、符号化により著しく圧縮できると予想される。また、固定長化により、タプル毎の管理情報（タプル長、属性数、属性長等）が不要になる。符号化対象でない属性値は何もしないので変わらない。実測結果（図１３）をみると、電話帳データベースでは、符号化対象データは約３／４に、管理情報は０に、全体として約３／４に圧縮できる。符号化対象データの圧縮効果が小さいのは、データをほとんど含まない属性がある（図６参照）ためで、それらを除いた４属性では約１／2.2 に圧縮できる。また、図書館データベースでは、符号化対象データは約１／５に、管理情報は約１／４に（タプルidが残る）、全体として約１／２に圧縮できる。これに、符号化情報を加算すると、電話帳データベースでは同程度、図書館データベースでは約８５％となる。符号化対象データの圧縮率＝符号長／平均データ長であるから、平均データ長が大きいもの程圧縮効果がある。なお、電話帳データベースでは、元はどの属性も固定長構成であったが、それを可変長構成に変換したものを評価対象とした。元と比べると符号化情報を入れても約１／３に縮小できる。以上から、符号化によりデータベースが圧縮できること、また符号化情報を入れても総容量は増えないことが分かった。

5．おわりに

データベースの符号化法を提案し、それを２種のデータベースに適用した実験結果を述べた。実験から、提案した方法を使えば、小規模～中規模データベースに対しては、大規模化する主記憶上で高速に符号化／復号化することが可能である見通しが得られた。その他、以下のことが明らかになった。

・木構造型は集合型に比し符号化情報が著しく縮小できる。
・有向木の１エントリに格納する文字数ｎは１でなく、文字列の平均長が良さそうである。
・符号化する文字列を木構造で表現すればどの属性もいずれは減少し始めるがその時定数がかなり異なる。
・動的ハッシュの格納率は６９％に収斂する。
・符号化によりデータベースが圧縮でき、符号化情報を入れても総容量は増えない。

本符号化法は小規模～中規模で、可変長文字列を対象としているので、単にデータベース処理だけでなく、記号処理、特に規模増大に伴う深刻な性能問題に直面している知識ベース処理への適用が期待でき、その高速化に役立つと思われる。今後は文献データベース等に対しても実験を行い、より広範に本符号化法の特性を調べる予定である。

最後に、データベースの使用に関し便宜を図って頂いたＮＴＴ電気通信研究所の関係者に感謝致します。

参考文献

[１] Tanaka,Y :A Data-stream Database Machine with Large Capacity, in Advanced Database Machine Architecture, Prentice-Hall,Inc. (1983).
[２] 板野肯三，他:連想記憶に基づくパイプライン型文字列検索アルゴリズム,情報処理学会論文誌,Vol. 26,No.6,pp.1152-1155 (1985).
[３] Larson,P-A :Dynamic Hashing, BIT,18,2,pp.184-201 (1978).
[４] 中野良平，斉藤和巳：ルールに基づいた関係論理／関係代数変換法，情報処理学会データベースシステム研究会資料，86-DB-54 (1986).
[５] Knuth,D.E.:The Art of Computer Programming, Vol.1, Addison-Wesley,Pub. (1968).
[６] Litwin,W.:LINEAR HASHING:A New Tool for File and Table Addressing, Proc.6th VLDB, pp.212 -223 (1980).
[７] Knuth,D.E.:The Art of Computer Programming, Vol.2, Addison-Wesley,Pub. (1969).