挿入制限のないコード VLEI を用いた XML ラベリング手法の検討とその評価

小林　一仁[†]　　横田　治夫[††,†]

† 東京工業大学 大学院 情報理工学研究科 計算工学専攻
†† 東京工業大学 学術国際情報センター
E-mail: †kkobayashi@de.cs.titech.ac.jp, ††yokota@cs.titech.ac.jp

あらまし　近年 XML ドキュメントの効率のよい検索が求められている．そのため、XML ドキュメントを関係データベースに格納する手法が注目されている．関係モデル上で包含関係を表現するための XML ラベリング手法が提案されているが，更新への対応は十分ではない．特に挿入に関しては，一定の間隔を持たせることである程度の挿入を低コストで行える手法も提案されているが，これらの間隔を持たせる手法では間隔を使いきった時再構成が必要となる．我々は要素を容易に、無制限に挿入できる VLEI コードを提案した．本稿は XML ラベリング手法である Dewey Order や前順後順法等に VLEI コードを適用し，検索と挿入の性能について評価を行う．

キーワード　XML，半構造データ，性能評価．問い合わせ処理，アクセスパス

Evaluation of XML Laveling Methodss using Endless Insertable Code VLEI

Kazuhito KOBAYASHI[†] and Haruo YOKOTA[††,†]

† Department of Computer Science Graduate School of Information Science and Engineering in Tokyo Institute of Technology
†† Global Scientific Information & Computing Center in Tokyo Institute of Technology
E-mail: †kkobayashi@de.cs.titech.ac.jp, ††yokota@cs.titech.ac.jp

Abstract　Recently, it is strongly demanded to search XML documents efficiently. Storing XML documents into a relational database is one of good approaches. A number of methods have been proposed to handle the XML parent-children relationship within the relational model. However, these methods do not well consider update operations. To reduce the cost of insertions, several methods keep intervals between labeling numbers, but they require whole database reconstruction when the intervals are used up. We have proposed the Variable Length Endless Insertable (VLEI) code to reduce the cost of reconstruction. In this paper, we propose methods to apply the VLEI code to XML labeling methods, preorder-postorder and Dewey order methods, and compare their performance for search and insertion operations.

Key words　XML, Semi-Structured Data, Performance Evaluation, Query Processing, Access Path

1. はじめに

近年様々な場面でデータを表現，交換するため XML が使われるようになった．それに伴い，XML ドキュメントを検索する機会が多くなり，効率のよい XML ドキュメントの検索が求められている．そのため，XML をデータベースに格納することが必要とされている．なかでも関係データベースに格納する手法が，注目されている [1]．関係データベースに格納することで，関係データベースの様々な機能が利用できる．しかし，XML はタグにより表現された包含関係をもっている．XML ドキュメントを検索する際には，この包含関係に従った検索ができなければいけないが，関係モデルのみではこの包含関係を表現することができない．そのため，包含関係を関係モデル上で表現する様々な手法が提案されている．主に，包含関係を木構造と見て，木構造の先祖子孫判定手法を使うラベリング手法が使われている．図 1 に示す前順後順法 [2] や図 2 に示す Dewey Order [3] などがその例である．

しかし，挿入が行われると，図 3 のように，多くのノードの値を割当し直す再構成をしなければいけない．関係データベースで多くの値をつけ直す処理は大変コストが高い．そこで，我々は挿入が行われても，他のノードの値を付け直すことなく挿入するノードの値を作成できる挿入制限のないコード，VLEI(Variable Length Endless Insertable) コードを提案した [4]．

本稿では，この VLEI コードを上で挙げたラベリング手法に

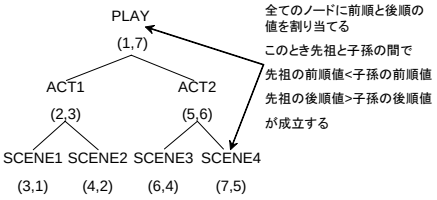

図 1　前順後順法

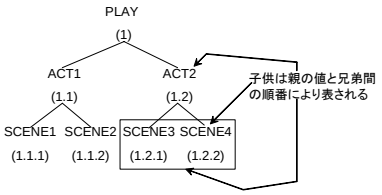

図 2　Dewey Order

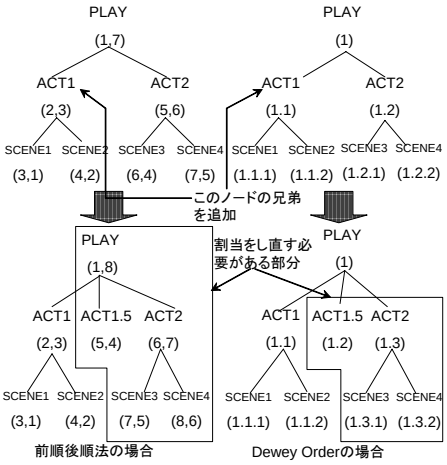

図 3　挿入が行われたときの再構成の例

適用し，XML ドキュメントを関係データベースに格納して性能を評価する．また挿入を考慮した他手法と比較をして，VLEI コードの優位性を示す．

本稿の構成は次の通りである．まず 2 節でこれまでに提案されている関連研究を述べる．次に 3 節で VLEI コードの概略について説明を行う．4 節では包含関係を表現するラベリング手法を挙げ，それらに対してどのように VLEI コードを適用するのか説明を行う．5 節で実験について述べ，6 節で実験に関する考察を行い，7 節で本稿をまとめる．

2. 関 連 研 究

XML ドキュメントを前順後順などの値を用いて包含関係を表現し，関係データベースに格納する場合，再構成のコストを下げるために前順後順などの値の間を開けることで，ある程度の挿入を容易に行う方法として [5] や [6] などがある．[5] では最初に予めノード間にスペースを作っておき，挿入によりスペースが狭くなってきたら動的にスペースを空ける手法を提案し，[7] で性能評価をしている．また [6] では挿入するノードを浮動小数点で表現する手法 (QRS) を提案している．また，オーバーフローを起こしたときにバルクロードを高速に行う手法も

アルゴリズム makeInsertValue(v_l, v_r)
入力: 挿入する要素の左側のコード v_l, 右側のコード v_r (但し $v_l < v_r$)
出力: 挿入された要素のコード v_i

if $length(v_l) \leqq length(v_r)$
$v_i = v_r \cdot 0$
else
$v_i = v_l \cdot 1$
endif
return v_i

図 4　アルゴリズム 1. 挿入する要素の決定

提案している．本稿では，提案手法と上記スペースを空ける手法及び浮動小数点を用いる QRS 法との比較を行う．

包含関係を表現するための XML のラベリング方法として，前順後順法及び Dewey Order 以外では, [8] は XML の木構造をバイナリ tree にマッピングし，その値を用いて効率よく containment ジョインを行う手法を提案している．しかし，これは挿入に関してまったく考慮されていない．また前順後順法で効率よく大小関係を判定するための手法として [9] はスタブリストを用いる手法を提案している．

3. VLEI コード

3.1 VLEI コードの概略

本節では我々の提案している VLEI(Variable Length Endless Insertable) コードの概略を述べる．詳しくは [4] を参照されたい．VLEI コードは 1 から始まる 0,1 の可変長の bit 列で，次の大小関係を満たす．

定義 1. VLEI コードの大小関係

v を 1 から始まる 0,1 の bit 列としてとして，以下の大小関係を定義する．

$v \cdot 0 \cdot \{0|1\}^* < v < v \cdot 1 \cdot \{0|1\}^*$

1 を基準点としてこの値よりも小さいか大きいかを 0 と 1 をつけて表現する．例として，11 は 111 より小さく，110 よりは大きいという大小関係を持っている．

この VLEI コードを用いれば図 4 のアルゴリズムにより，あるコードとあるコードの間を示すコードを容易に作成することができる．たとえば，11 と 111 の間に要素を挿入するためこの間を表すコードを求める場合は図 4 のアルゴリズムにより 1110 が得られる．これは VLEI コードの定義から $11 < 1110 < 111$ が成立する．このように挿入する要素の値を他の要素の値を変更することなく容易に求めることができる．

3.2 VLEI コードの偏り制御

一箇所への集中した挿入により，VLEI コード長に偏りが発生する．偏りが発生すると，

- 大小関係に関わる bit 数が増加
- 記憶容量が増加

などの弊害が発生する．そこで偏りを平坦化を行う．しかし平

```
アルゴリズム NNum2VLEI(a_i, N)
入力: 自然数 a_i,
      要素の数 N
出力: VLEI コード v_i
int m = ⌈log_2 N⌉
int P = 2^m
variable bit v_i = 1
int x = a_i − P
while( x ≠ 0)
    P = 1/2 P
    if(x > 0)
        v_i = v_i · 1
        x = x − P
    elseif(x < 0)
        v_i = v_i · 0
        x = x + P
    endif
endwhile
return v_i
```

図 5　アルゴリズム 2. 自然数から VLEI コードへのマッピング

坦化のために全体を再構成しなおすことはコストが高い．そこで木を部分的に平坦化する手法として AVL 木の 1 重回転と 2 重回転 [10] があるが，VLEI コードの 2 分木でこれらの操作を行う場合，1 重回転，2 重回転を行う部分の全ての値の書き直しが必要になる．また 1 重回転，2 重回転で減らすことができる偏りの量は 1 である．また長さ 2 以上の偏りが発生するたびに均衡化をしなければいけないので挿入，削除のコストが大きくなってしまう．そこで簡単な偏りを平坦化する手法を提案した [4]．しかしこれらの手法では平坦化できない偏りが生じる可能性がある．そこで自然数を使った前順後順法に対して，初期状態として長さをできるだけ短かい VLEI コードを割り当てる図 5 のアルゴリズムを用いて読み込みと書き込みの 2× 要素数回の I/O で，最大 $2^{m+1}-1$ の要素に対して最大長 m の長さの VLEI コードを割当をしなおす手法を提案する．まず割り当てをしなおす領域を次のように決める．

（1） VLEI コードを 2 分木と見て，各節点に葉節点までの最長の長さを記す

（2） 自分の子供の節点における上の 1 における長さの差がある一定の閾値を越えるところで一番根に近いところを探す

領域が決まったら図 6 のアルゴリズムで値を振りなおす．

この再構成法を行う際に重要になるのが閾値をどのように設定するかである．AVL 木のように閾値を 2 に設定すると頻繁に全ての値の再構成を行ってしまう可能性がある．また逆に閾値を高く設定すると偏りをうまく調整できない可能性もある．適切な閾値の設定は今後の課題とする．

3.3　VLEI コードのパッキング手法

VLEI コードは可変長の bit 列により表現されるが，可変長の bit 列は管理が大変である．PostgreSQL などでは可変長の bit 列を扱うことができるが，プログラミング言語 Java では予め用意されている格納領域に格納できる可変長の bit 列は 32bit までで

```
アルゴリズム:PartialReconstruct(V[i],v)
入力: 昇順で並べた再構成を行う部分の VLEI コードのリスト V[i],
      再構成を行う部分の根の VLEI コードの値 v
出力: 再構成が行われた VLEI コードの配列 V'[i]
length = V[i] の配列の長さ;
for(int j = 0; j <length(v); j + +)
    v' = NNum2VLEI((i + 1),length);
    v' = v' の先頭の 1 を取ったコード;
    V'[i] = v · v'
endfor
```

図 6　アルゴリズム 3. 部分再構成法

ある．そこで int などの固定長の記憶領域を複数利用して VLEI コードを配列として分割格納する手法を提案する．

まず可変長の VLEI コードを int などの固定長の記憶領域に格納するために VLEI コードの後ろ又は前に任意の bit を入れて長さを合わせなければいけない．そこで次のように固定長にパックしたパックド VLEI コードを定義する．

定義 2. パックド VLEI コード

VLEI コードを v，その長さを l，固定長の記憶領域の大きさを b とし，パックド VLEI コード v' を次のように定義する．

$v' = (v$ の先頭から 1 をとったもの $) \cdot 0 \cdot (1)^{b-l-1}$

VLEI コードの先頭の 1 は，基準点を示すものであり，この値をつけたままパックすると，全ての値の先頭が 1 となる．またパックしたものはこの次の値から一番下の bit までで元と同じ大小関係を示すことができる．このことはこの後に示す定理により示す．よって先頭の 1 が冗長となるためこの値を削除した．パックしていない VLEI コードの場合、先頭の 1 を削除すると、サイズが固定されていないことから VLEI コードの開始位置が特定できないため、削除することができない．

例として固定長の長さを 8 として以下の 4 つの VLEI コードをパックド VLEI コードに変換をすると次のようになる．

100100 => 00100011
10010 => 00100111
1001 => 00101111
10101 => 01010111

次にパックド VLEI コードは元の VLEI コードと同じ大小関係を持つことを示す．

定理 1. VLEI コードを v，その長さを n，格納する固定長の bit 列の長さを l，パックド VLEI コードを v_n とする．v の先頭の 1 を取った bit 列を v' とする．$v_n = v' \cdot 0 \cdot (1)^{l-n-1}$ ただし $l > n$ とする．この v_n の各値を単純比較した結果は元の VLEI コードの大小関係を比較したものと同じものになる．

証明 任意の 2 つの VLEI コード v_1，v_2 があったとする．これに対してパックド VLEI コードを v_1'，v_2' とする．

まず VLEI コードの大小関係判定方法は次の通りである．

A) 一番左の bit から順にそれぞれの値を比較していき，どちらか一方が 1 で他方が 0 になった時 1 の方が大きい．

B) どちらかが最後の bit まで到達した場合，もう一方の次の bit を見て、0 であれば最後の bit まで到達したほうが大きい．1 であれば最後まで到達したほうが小さい．

ここで元の VLEI コードが VLEI コードの比較判定方法 1,2 のそれぞれで大小関係を判定されるとき，パックド VLEI コードがどのように判定されるかみていく．

（1） 元の VLEI コード v_1，v_2 が左から $k$bit 目で比較方法の A で大小関係が判定される場合

このとき v_1' と v_2' において左から 1〜$k-2$bit 目まで同じ値で，左から $k-1$bit 目でどちらかが 0 で，もう一方が 1 になっている．またこの時，右からの位置も等しくなる．よって単純比較を行うと右から $v-k$ 番目が 1 になっている方が大きくなり，元の VLEI コードを比較した時と同じ大小関係になる．

（2） 元の VLEI コード v_1，v_2 が B で判定される場合

v_1 の長さを l_1，v_2 の長さを $l_2(l_1 < l_2)$ とする．ここで v_1' と v_2' は左から 1〜l_1-1 まで値が同じになる．そして v_1' の左から l_1 番目の bit は 0 になる．

- **v_2' の左から l_1 番目の bit が 1 である場合**

このとき，右からの位置も等しくなるので単純比較で $v_1' < v_2'$ となり元の VLEI コードを比較したときと同じ大小関係になる．

- **v_2' の左から l_1 番目の bit が 0 である場合**

v_1' は左から l_1+1〜n まで全ての bit が 1 である．また v_2' は $l_1 < l_2$ で，v の後ろに 011･･･を付けたものであるから l_1+1〜n までの間に必ず 1 個は 0 がある．よって単純比較を行うと $v_1' > v_2'$ となり元の VLEI コードを比較したときと同じ大小関係になる．

よってパックド VLEI コードを単純比較したものは元の VLEI コードを比較したものと同じ大小関係になる．□

先ほどの例で上げた元の VLEI コードには

100100 < 10010 < 1001 < 10101

という大小関係が成立している．またパックド VLEI コードに変換したものに対しても

00100011 < 00100111 < 00101111 < 01010111

という大小関係が成立していて，元の VLEI コードと同じ大小関係になる．

次に，パックド VLEI コードを配列に分割格納する方法を提案する．格納する配列の固定長の bit 列の長さを n とする．パックド VLEI コードを $n-1$bit で切り分け，各配列の先頭から $n-1$ までの値をこの値とする．またこの配列にパックド VLEI コードが収まらずに次の配列もこの格納に必要な場合はその配列の最後の bit を 0 にする．また最後のブロックには 111･･･で最後の bit まで埋める．このようにすることで次のブロックを使用しない場合には最後の bit が 1 になる．よって次のブロックを使用するときのみ最後の bit が 0 になる．VLEI コードからパックド VLEI コードの配列に分ける処理は図 7 のアルゴリズムで行える．

このアルゴリズムを用いて先ほどの 4 つの VLEI コードに対して固定 bit 列の長さを 4 として分割格納すると次のようになる．

アルゴリズム:VLEI2IntArray(v,b)
入力: VLEI コード v, 格納する配列の固定 bit 列の長さ b
出力: VLEI コードの配列 $B(v)[i]$

$length$ = v の長さ;
for(int $i = 0$; $i < \lceil length/(b-1) \rceil$); $i++$)
$B(v)[i]$ = (v の $(i*(b-1)+1$ から長さ $b-1$ スライスしたもの)･0;
endfor
int $i = \lceil length/(b-1) \rceil$;
int $j = (length-1) mod (b-1)$;
$B(v)[i]$ = (v の $i*(b-1)+1$ から長さ j でスライスしたもの)･0 ･$(1)^{b-j}$;

図 7　アルゴリズム 4.VLEI コードから int の配列への変換

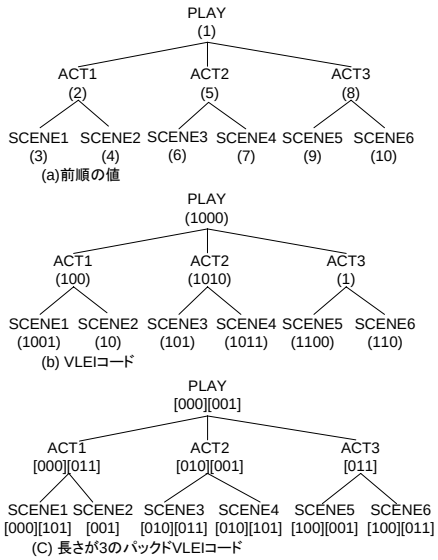

図 8　前順後順法への VLEI の適用

100100 => 0010 , 0001

10010 => 0010 , 0011

1001 => 0010 , 0111

10101 => 0100 , 1011

4. 包含関係表現方法

包含関係を表現するためのラベル付け手法について説明を行う．VLEI コードは可変長の bit 列のためこのままラベル付けに利用するとパフォーマンスに悪影響を与える可能性がある．そこで各々の手法でどのように VLEI コードを利用するか，又はパックド VLEI コードの配列を使うかについて説明も行う．

4.1 前順後順法

前順後順法はまず全てのノードに木構造における前順と，後順の値を割り当てる．ここでノード A がノード B の木構造における先祖であれば以下の大小関係が成立する．

A の前順の値 <B の前順の値

A の後順の値 >B の後順の値

この関係を利用して，XML の包含関係を調べる．

実際に図 8 の (a) のような XML があり，前順の値を割り当てたとする．これに図 5 のアルゴリズム NNUM2VLEI [4] により VLEI コードを割り当てたものが図 8 の (b) になる．固定 bit

```
アルゴリズム:isBiggerArray(a[],b[])
  入力: パックド VLEI コードの配列 a[], b[]
  出力: boolean
  for(int i = 0; i < min(length(a),length(b);i + +)
    if(a[i] > b[i]) return true
    if(a[i] < b[i]) return false
  endfor
```

図 9　アルゴリズム 5. パックド VLEI コードの大小比較

```
アルゴリズム:makeInsertVLEICValueArray(a[],b[])
  入力: 挿入する VLEI コードの前順又は後順の前の配列 a[],
        挿入する VLEI コードの前順又は後順の後ろの配列 b[]
  出力: 挿入する VLEI コードの前順又は後順の配列 v[]
  if(length(a) >length(b))
    for(int i = 0; i < length(a) − 1; i + +)
      v[i] = a[i];
    endfor;
    if(a[i] mod(4)==1)
      v[i] = a[i] − 1;
      v[i + 1] = 011···;
    else
      int j = a[i] に対する VLEI コード;
      j = j ∗ 2;
      v[i] = j に対するパックド VLEI コード;
    endif
  else
    for(int i = 0; i < length(b) − 1; i + +)
      v[i] = b[i];
    endfor
    if(b[i] mod(4)==1)
      v[i] = b[i] + 1;
      v[i + 1] = 011···;
    else
      intj = b[i] に対する VLEI コード;
      j = j ∗ 2;
      v[i] = j に対するパックド VLEI コード;
    endif
  endif
```

図 10　アルゴリズム 6. パックド VLEI コードの挿入値の決定

列の長さを 3 にしたアルゴリズム VLEI2IntArray により値を割り当てたものが，図 8 の (c) となる．また後順についても同様にラベリングを行う．

ここで任意の 2 つの VLEI コード A,B(A>B) に対して図 7 のアルゴリズムを用いて作成したパックド VLEI コード A’,B’ は，各 array の要素 $a[], b[]$ において，$a[i] \geqq b[i]$ が成り立つか，または $a[j] > b[j]$ が成り立つ $j(j > i)$ が存在する．よって，前順後順法の先祖子孫関係判定のための大小関係比較は次の図 9 のアルゴリズム パックド VLEI コードの大小比較で行える．

挿入については，前順と後順において，挿入する要素の前後のパックド VLEI コードを探し，前順後順の両方の値に対して，図 10 のアルゴリズムで挿入する値を決定する．

4.2　Dewey Order

Dewey Order は図 2 のように兄弟間の順番を示す数字とその子供であることを示す識別子によって構成される．Dewey Order を用いてラベリングをしたものに対してノードを挿入すると図 3 のように，兄弟間の順番を示す数字に間がないと兄弟間とその子孫間の値を付け直さなければいけない．兄弟間の順番を示す数字に VLEI コードを利用することで，挿入のコストを低くすることが可能である．以下のように定義した識別子付き VLEI コードで Dewey Order を表現する．

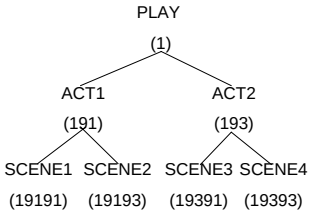

図 11　Dewey Order への VLEI の適用

定義 3. 識別子付き 8 進 VLEI コード

兄弟間の順番を VLEI コードで表現した物を 8 進数の数値で表し，ドットの代わりに 9 を用いて識別子を表現する．つまり識別子付き 8 進 VLEI コードは次のように定義される．

ノードの値=”親のノードの値”+9+”兄弟間の順番の 8 進 VLEI コード”

兄弟間の VLEI コードを 8 進数で表現するため識別子は 8 か 9 を使うことができる．今回は 9 を識別子として使用する．なお，兄弟間の VLEI コードは，兄弟間の順番と兄弟の一を引数として，NNUM2VLEI で変換することで得られる．図 2 に対してこの定義で番号付けを行うと図 11 のようになる．

この Dewey Order で子孫を探すときには先祖の VLEI コードに 9 を付けたもので始まる VLEI コードを探せばよい．逆に親を検索する際には子供の VLEI コードから一番下の桁の 9 から一番下の桁の数を取った VLEI コードを検索すればよい．またその上の親を探すときには下から 2 番目の桁の 9 から一番下の桁の数まで取った VLEI コードを検索すればよい．例えば，図 11 の中で，ACT1 の子孫を捜すときは VLEI コードが 1919 で始まるものを探せばよい．また SCENE3 の親を探すときは VLEI コードが 193 のものを探せばよい．さらに SCENE3 の 2 つ上の親を探すときは VLEI コードが 1 のものを探せばよい．

挿入する値は図 12 のアルゴリズムで決める．親の子供の中で一番長さが小さいものから順にまだ使われているかどうかを調べ，使われていない値を探す．使われていたらその値よりも小さく，長さが 1 大きい値について再帰的に調べていく．

5.　実　　験

関係データベースにそれぞれの手法における包含関係を表現するための値を格納し，それぞれの手法で表 1 に挙げた XPath [11] を検索したときの時間と，偏った 1000 回の挿入を行った時の時間を測定した．また今回データベースには包含関係を表現する値と，ノードの名前，値，そして各ノードを特定するための

アルゴリズム:insertNodeInDewey(*a*,*b*)
入力: Dewey Order における VLEI コードの配列 *a* 挿入のタイプ *b*
出力: 挿入するノードの VLEI コードの値 *c*

if(b==子供)
c=”*a*”+91;
else if(b==兄弟)
c=”*a*”+1;
endif
while(true)
if(*c* がまだ使われていない値である)
break;
endif
c=”*c*”+0;
endwhile

図 12　アルゴリズム 7.Dewey Order における挿入する値の決定

表 1　検索を行った XPath

Q1	/PLAY//LINE
	一回の先祖子孫関係の判定
Q2	/PLAY/ACT/SCENE/SPEECH/LINE/STAGEDIR
	複数回の先祖子孫関係の判定
Q3	LINE[STAGEDIR=”Aside”]
	葉ノードから親ノードの検索

ID を格納した．

VLEI コードを用いたラベリング手法を既存の手法と包含関係表現に必要な bit 数ごとに比較を行った．

- 64bit
  - QRS(float)　前順後順法で前順後順の値を float に格納
  - sparse 前順後順法で前順後順の値と値の間を空けて int に格納
  - VLEI(1)　前順後順法で前順後順の値を VLEI コードに変換し int に格納
  - VLEI(Dewey)　Dewy Order の値を識別子付き 8 進 VLEI コードに変換し long に格納
- 128bit
  - QRS(double)　前順後順法で前順後順の値を double に格納
  - VLEI(2)　前順後順法で前順後順の値を VLEI コードに変換し 2 個の int 領域に分割格納
- 192bit
  - VLEI(3)　前順後順法で前順後順の値を VLEI コードに変換し 3 個の int 領域に分割格納

今回は実装の関係上 VLEI コードの分割格納数に限界を設け，識別子付き 8 進 VLEI コードで表現した Dewey Order を 64bit の格納領域に格納したため挿入に限界が存在している．

実験環境は表 2 の通りである．

また，今回 DBMS で b-tree によるインデックスをつけなかった．図 13 に表 1 を前順後順法により検索したときにインデックスをつけた時に必要とした時間をつけなかったときに必要とした時間で割った性能差を示す．この図より明らかにインデックスをつけた時の方が速度が遅くなった．これは大小関係を判定する際に MySQL が b-tree を用いないためである．

表 2　実験環境

CPU	Celeron 1.7Ghz
HDD	Seagate ST340016A
Memory	PC-2100 512MB
OS	Windows XP Professional
DBMS	MySQL 4.0.16-nt
Java	J2RE1.4.2

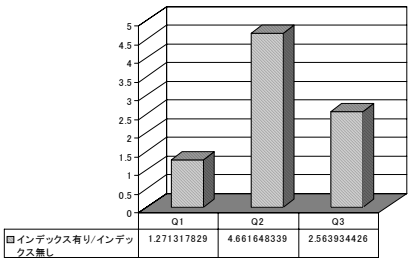

図 13　b-tree をつけた場合とつけない時の速度の比較

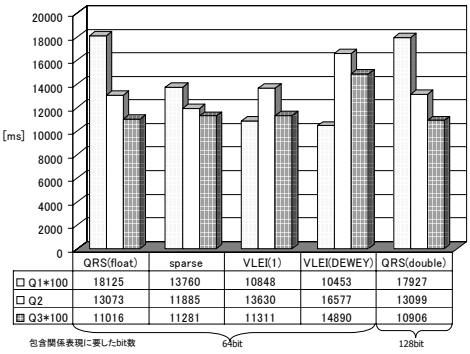

図 14　hamlet.xml を検索するのにかかった時間

表 3　hamlet.xml を格納した時のデータベースのサイズ

VLEI(1)	339[kb]	VLEI(Dewey)	350[kb]
QRS(double)	350[kb]	QRS(float)	350[kb]
Sparce	350[kb]		

5.1　XPath の検索

The Plays of Shakespea [12] の hamlet.xml, r_and_j.xml, lear.xml, othello.xml に対して，表 1 の XPath を検索をした．

図 14 に hamlet.xml を，図 15 に r_and_j.xml を，図 16 に lear.xml を，図 17 に othello.xml を Q2 と，Q1 の 100 回検索と，Q3 の 100 回検索に必要とした時間 (単位はマイクロ秒) を示す．VLEI(Dewey) では VLEI コードの格納に BIGINT(8 バイト整数型) を利用した．またパックド VLEI コードに格納する際，配列 1 個で十分格納できるため配列数 1 個のみを調べた．

図 14，15，16，17 から VLEI(1) は sparce とほとんど時間が変わらなかった．また VLEI(Dewey) は Q1*100 では Normal より速かったが，Q2,Q3*100 では遅かった．

またそれぞれの手法に対して，hamlet.xml を格納した時のデータベースのサイズを測定したところ表 3 のようになった．

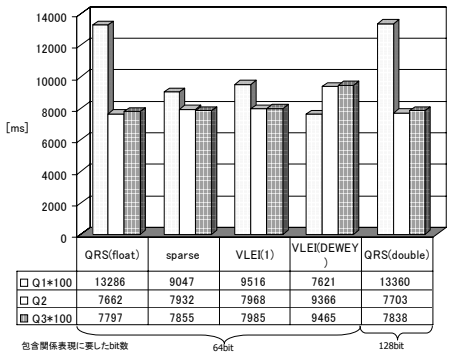

図 15　r_and_j.xml を検索するのにかかった時間

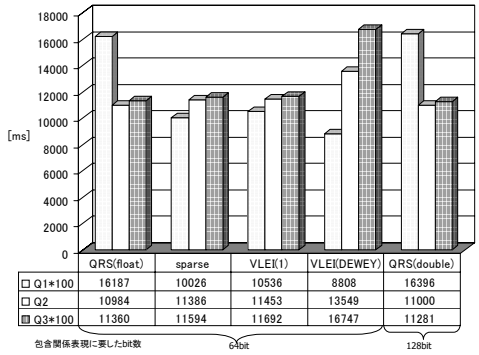

図 16　lear.xml を検索するのにかかった時間

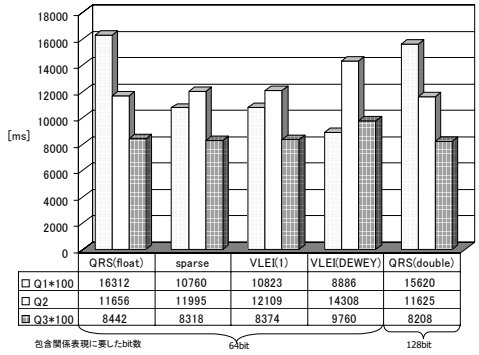

図 17　othello.xml を検索するのにかかった時間

5.2　要素の挿入

VLEI コードの配列を利用した前順後順法と，識別子付き 8 進 VLEI コードを利用した Dewey Order による手法に対して，hamlet.xml を関係データベースに格納し，ある要素の子供に人工的に作成した要素を 1 個追加する挿入を 1000 回行った．要素を選ぶときに theta=0,0.5,1 の Zipf 分布 [13] により作成した乱数を用いた．Zipf 分布は確率分布の一種であり，データベースのアクセス分布のモデル化によく用いられる．引数として theta をとり，この値が大きくなればなるほど偏りが大きくなるような確率分布である．各手法でデータ長を超え，オーバーフローした場合はデータベースを再構成をした．Dewey Order に格納する際には BIGINT を利用した．また比較のため浮動小数点数 float と double を用いた前順後順法 [6] と予め前順と後順に間を空けて格納する前順後順法を比較として測定した．[5] によると

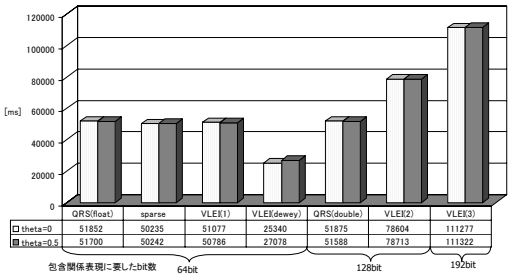

図 18　Zipf 分布 theta=0,0.5 の時，1000 個の偏った挿入に必要とした時間

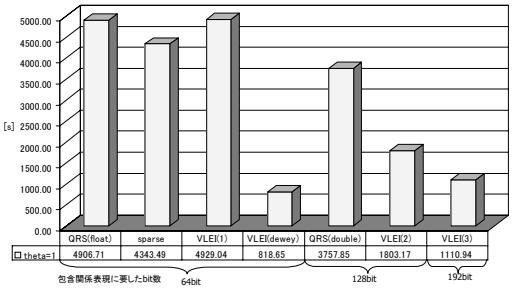

図 19　Zipf 分布 theta=1 の時，1000 個の偏った挿入に必要とした時間

表 4　各手法の再構成の回数

VLEI(1)	6	VLEI(2)	2	VLEI(3)	1
VLEI(Dewey)	3	QRS(double)	5	QRS(float)	6
sparce	6				

間を空ける際に間隔が $[\frac{Width}{|T|+|X|+1}]$($Width$ は使用可能数の最大値，T は更新される XML 木，X は挿入 XML 木) である時が最もよい均衡状態なので，この値で間隔を空けた．また動的に値の割当をし直すことはしなかった．また VLEI コードに偏りが生じても偏り制御をしなかった．

まず図 18 に theta=0,0.5 の時の 1000 個の偏った挿入に必要とした時間を挙げる．このとき，どの手法でもオーバーフローを起こすことはなかった．さらに図 19 に theta=1 の時の 1000 個の偏った挿入に必要とした時間を挙げる．sparce は前順後順の値に間をあけた前順後順法を指す．

この図 18，図 19 から，VLEI(Dewey) が一番速いといえる．また QRS(double) や VLEI(2) では包含関係を表現するのに 1 つの要素ごとに 128bit，VLEI(3) では 192bit 使っているが，VLEI(Dewey) では 64bit しか使っていないことを考えると VLEI(Dewey) が記憶容量の面でも優れている．

この実験の中で theta=0,0.5 のときでは値がオーバーフローをして再構成をすることはなかったが，theta=1 のときではいずれでも再構成を行った．この再構成の平均値は表 4 の通りである．

配列を用いた場合は，配列数を多くすればするほど再構成の数は減った．

6.　考　　察

XPath の検索に関して，Dewey Order は Q1 では前順後順法よりも速かったが，Q2, Q3 では遅い結果となった．これは Dewey

Order でラベル付けを行うと，葉までの深さが深くなるほど葉のラベル値が大きくなる傾向があり，XPath を検索する際に SQL で LIKE を使用したため文字列検索のコストが高くなり遅くなった．しかし Q3 ではプログラムと SQL の改良により前順後順法よりもパフォーマンスが上がる可能性はある．また VLEI(1) は単純比較で大小関係が判定できるため sparce と同じ速度になった．

データベースのサイズに関して，Dewey Order は long を用いて格納しているが，前順後順法では前順と後順の 2 つの int を格納しなければいけないため同じサイズになった．

挿入に関して，識別子付き 8 進 VLEI コードを Dewey Order に適用した場合は他の手法と比べて速い結果になった．これは他の手法では挿入する要素と前順後順において隣り合う 2 つの値をデータベースから探し，その値を基に前順後順の値を決定する．つまり 4 回データベースを検索しなければいけない．これに対して Dewey Order では親の値より挿入できそうな要素の値を決め，この要素の中でまだ使われていない長さが 1 番短いものを探す．最良の場合で 2 回のデータベースの検索で求めることができる．また短いものを探す過程ではまだ使われていないのを確認するだけなのであまりコストがかからない．そのため速かったと思われる．また VLEI コードの配列を用いた場合，再構成が起こらないとき配列の数を多くすればするほど遅くなった．これは挿入する値を決めるために挿入する要素の隣の要素を探す必要があり，この要素を探すときに増えた分の配列に関しても検索したため遅くなってしまった．しかし，再構成が起こるときは配列の数を大きくすればするほど時間は短縮された．これは要した時間のうちほとんどがデータベースの再構成に使われたため，再構成の数が少ない配列の数が大きい物ほど時間は短縮された．

7. ま と め

我々の提案している VLEI コードを XML ラベリング手法である前順後順法と Dewey Order に適用するため，パックド VLEI コードと識別子付き 8 進 VLEI コードを提案し，実際に XML ラベリング手法に適用して，検索時間と挿入時間の計測実験を行った．この実験により VLEI コードを int などの固定長の領域に配列として分割格納した前順後順法は挿入が偏ったとき，他手法と比べ要素の挿入に対して強いことが示された．また識別子付き 8 進 VLEI コードを用いた Dewey Order では偏りに関係なく挿入に関して他手法およびパックド VLEI コードの分割格納を用いた前順後順法比べて優位性を示すことができた．また VLEI コードを用いた Dewey Order の XPath 検索速度は前順後順法によるものとほとんど変わらないことも示すことができた．以上より，識別子付き 8 進 VLEI コードを用いた Dewey Order が VLEI コードを用いた XML ラベリングに適していると言える．

今後の課題として，今回の実験は実際の XML ドキュメントの検索や更新について考慮しなかった．そこで [14] などのような XML ベンチマークなどの研究に基づくベンチマークを行い VLEI コードが実際の更新に対しても強いことを示す．また今回は DeweyOrder を識別子付き 8 進 VLEI コードで表現し 64bit の格納領域に格納したが，XML 木において根から葉までの長さが深くなると 64bit では表現できなくなる．そこで DeweyOrder を 16 進数の識別子付き VLEI コードで表現し，BLOB などのバイナリデータに格納する．また，Dwey Order 以外のインデックス構造またはラベリング手法を検討する．

謝　　辞

VLEI コードの Dewey Order への適用に関して奈良先端科学技術大学の天笠 俊之先生に助言を頂いた．ここに感謝の意を表します．また本研究の一部は, 文部科学省科学研究費補助金特定領域研究 (15017233)，独立行政法人科学技術振興機構 CREST，及び 21 世紀 COE プログラム「大規模知識資源の体系化と活用基盤の構築」の助成により行われた．

文　　献

[1] Igor Tatarinov, Stratis Viglasand Kevin S. Beyer, Jayavel Shanmugasundaram, Eugene J. Shekita, and Chun Zhang. Storing and querying ordered xml using a relational database system. In *Proc. of SIGMOD Conf.*, pages 204–215, 2002.

[2] Paul F. Dietz. Maintaining order in a linked list. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 122–127, 1982.

[3] Online Computer Libraly Center. Introduction to the dewey decimal classification. http://www.oclc.org/oclc/fp/about/about_the_ddc.htm.

[4] 小林一仁 , 小林大 , 横田治夫. 挿入制限のない範囲ラベリング用コード . In **信学技報**, **電子情報通信学会**,*DE2003-13*, 2003.

[5] 江田毅晴 , 天笠俊之 , 吉川正俊 , 植村俊亮. Xml 木のための更新に強い節点ラベル付け手法. In *DBSJ Letters,No.1 in Vol.1,2002*, 2002.

[6] Toshiyuki Amagasa, Masatoshi Yoshikawa, and Shunsuke Uemura. Qrs: A robust numbering scheme for xml documents. In *19th International Conference on Data Engineering (ICDE 2003)*, pages 705–707, 2002.

[7] 江田毅晴 , 天笠俊之 , 吉川正俊 , 植村俊亮. XML のための動的範囲ラベル付け手法:その評価および XRel への適用について . In **情処研報**, **情報処理学会**, *DBS-129-16*, 2002.

[8] Wei Wang, Haihng Jiang, Hongjun Lu, and Jeffrey Xu Yu. Pbitree coding and efficient processing of containment join. In *International Conference on Data Engineering*, pages 391–492, March 2003.

[9] Haifeng Jiang, Hongjun Lu, Wei Wang, and Beng Chin Ooi. Xr-tree: Indexing xml data for efficient structural joins. In *International Conference on Data Engineering*, pages 253–264, March 2003.

[10] G.M.Adel'son-Vel'skiĭ and E.M.Kandis. An algorithm for the organization of information. Doklady Akademiia Nauk SSSR, 146:263-366, 1962. English translation in Soviet Math. Dokl. 3, pp.1259-1263.

[11] World Wide Web Consortium. Xml path language . http://www.w3.org/TR/xpath.

[12] Jon Bosak. The plays of shakespeare in xml. http://www.oasis-open.org/cover/bosakShakespeare200.html.

[13] William J. Reed. The Pareto, Zipf and other power laws. http://linkage.rockefeller.edu/wli/zipf/reed01_el.pdf.

[14] Kanda Runapongsa, Jignesh M.Patel, H.V.Jagadish, Yun Chen, and Shurug Al-Khalifa. Michigan benchmark: Towards xml query performance diagnostics. In *Proceedings of the 29th VLDB Conference*, 2003.