2003－DBS－131　(6)
2003／7／16

挿入制限のないXML範囲ラベリング用コード

小林 一仁 [*]　小林 大 [*]　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

〒 152-8552 東京都目黒区大岡山 2-12-1

kkobayashi@de.cs.titech.ac.jp, daik@de.cs.titech.ac.jp, yokota@cs.titech.ac.jp

概 要

従来のデータベース技術の適用可能性から、関係データベースに XML を格納する手法が注目されている。さらに XML の問い合わせに対しては、タグの指示を含むことから、タグの包含関係に従った検索が重要となるが、関係データベースに格納された XML に対してタグの包含関係を高速に検出する手法として範囲ラベリングが提案されている。これまで提案されている範囲ラベリングの手法は検索に対する有効性は十分であるが、更新に対しては番号変更に関する問題点を持っていた。この問題を解決するために本論分では挿入などの更新に強い番号付けを提案する。bit 列をうまく利用することで、無限の挿入を可能にする。

キーワード： XML, 範囲ラベリング, 可変長コード , 更新

Code for XML Range Labeling without Limitation of Inserts

Kazuhito KOBAYASHI[*]　Dai KOBAYASHI[*]　Haruo YOKOTA[†,*]

[*] Department of Computer Science Graduate School of Information Science and Engineering
Tokyo Institute of Technology

[†] Global Scientific Information & Computing Center Tokyo Institute of Technology

2-12-1 Oh-Okayama, Meguro-ku Tokyo, 152-8552 JAPAN

kkobayashi@de.cs.titech.ac.jp, daik@de.cs.titech.ac.jp, yokota@cs.titech.ac.jp

Abstract

For the possibility of applying the traditional database techniques, approaches for storing XML documents in a relational database are in the spotlight. It is important for XML queries to retrieve items corresponding to stractual relationships of tags because the queries include specification of the tags. To detect the stractual relationships of tags, the range labeling methods were proposed. In spite of usefulness of the methods in retrieval, they have a problem of numbering within insert operations. To solve the problem, we propose a bit sequence based coding method and algorithms for inserting and numbering of the range labeling for XML with the code. We also discuss the usefulness of the code.

KEYWORD: XML, Range Labeling, Valiable Length Code, Update

1 はじめに

XML はその表現能力から、Internet をはじめさまざまな場面においてデータを表記したり、交換するためのデファクトスタンダードな記述言語になっている。記述された内容や交換された内容を格納しておくことは重要であり、また単に格納するだけでなく検索の機能も必要であることから、XML で記述された文書をデータベースに格納する要望が高まっている。

これまでデータベース処理は関係データベースを中心に発展してきていることから、関係データベースに XML を格納して、検索を効率よく行うアプローチが提案されている [9, 7]。関係データベースにおける技術を利用することで、XML 文書中の要素値などのみを対象とした単純な検索は効率良く行うことができる。しかし、XML 文書はタグによって構造化されていることが特徴であり、問合わせにおいてもタグで示された内包関係に従った検索が必要となる。

このため、XML 文書の包含関係をすばやく検索するための索引手法が提案されている。Li と Moon は要素の順番とサイズを索引とした包含関係判定手法を提案している [6]。あらかじめ範囲を順番とサイズに適切な間隔を持たせることによって、挿入による更新が複数の要素に及ぶ頻度を下げ、効率的な挿入が行われるように工夫している。また、Cohen らは包含関係判定手法として範囲数え上げ手法を提案している [3]。彼らは、Dietz 数え上げ手法 [4] を適用して、ノードとリーフに対して前順と後順の値を割り振り、先祖が子孫に対して、より小さい前順の値とより大きい後順の値を持つ性質を利用することによって、効率的な包含関係判定を実現している。

[6] や [3] のアプローチは、内包関係に従った検索には十分有効であるが、更新における番号付けにおいて問題がある。XML 要素の挿入を行うことで、文書全体に渡る大幅な番号変更が要求され、更新コストが高くなってしまうのである。

挿入時に要素の更新が広い範囲に及ぶことを抑制する手法として、順序の値にあらかじめ適当な間隔を与える手法 [10] や順序の値を浮動小数点数で表現する手法 [1] が提案されている。しかし、これらの手法では、予め用意された間隔を使い切った場合には、広い範囲で要素の順序付けの再定義が必要となる。

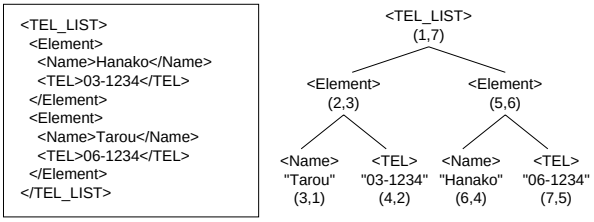

図 1: XML のノードに前順と後順の値を割り当て例

そこで本論文では、挿入が起こってもほかの節点の前順や後順の値が変わらないコードを用いた番号付けのためのコードを提案する。bit 列を用いた可変長コードを用いることで、bit 単位で拡張に可能であり、任意の個所に任意の個数の要素を挿入することができる。また、提案コードに基づく挿入のアルゴリズムと XML の範囲ラベリングにおける番号付けへの適用アルゴリズムを提案し、その有効性を議論する。

以下に本論文の構成を示す。第 2 章では、本論文で対象とする範囲ラベリングの手法の簡単な紹介と挿入における問題点を述べる。第 3 章では、bit 列を用いた挿入制限のない可変長コードの定義と、挿入アルゴリズム、および範囲ラベリングに対する番号付けへの適用アルゴリズムを提案し、その正当性を証明する。第 4 章では提案するコードを用いて番号付けをした手法と自然数を利用して番号付け場合のコストを見積もり比較し、第 5 章で本論文をまとめる。

2 範囲ラベリング手法

本論文では、Cohen らによる範囲ラベリングによる包含関係の検索手法 [3] を前提とする。XML 文書の内包関係が構成する木構造に Dietz 数え上げ手法 [4] を適用して、XML のタグの範囲毎に前順と後順の値を割り振り、内側の範囲の番号が外側の範囲の番号に対して、より小さい前順の値とより大きい後順の値を持つ性質を利用する。

実際に簡単な XML に番号付けを行った例を図 1 に

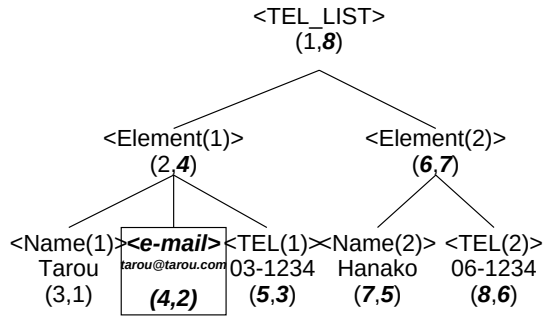

図 2: XML 木に要素を挿入した場合の例

示す。図の左側の文書に対して右側のような木が構成され、その木に対して前順と後順の値を割り当てる。次に、この前順と後順の値と、要素の名前、値を属性とする関係データベースを作成する。この XML において Element のタグの中にある Hanako を探すような、XPath[8] では//Element[Name="Hanako"] と表記される問合わせの場合には、要素毎に格納した関係データベースに対して、以下の SQL を発行することで高速に求めることができる。

```
SELECT *
FROM TEL_LIST AS T1, TEL_LIST AS T2
WHERE T1.要素の名前="Element"
AND   T2.要素の値="Hanako"
AND   T1.前順の値<T2.前順の値
AND   T1.後順の値>T2.後順の値;
```

しかし、このような範囲ラベリングの手法は、挿入において、番号を付け直さなければならないという問題が生じる。例えば、<Name>Tarou</Name>, <TEL>03-1234</TEL>の間に<e-mail>tarou@tarou.com</e-mail>を追加すると図 2 の斜体で示すようにこの要素以外の 6 つのノードまで前順後順の値が変わり、値を書き換える必要がある。

3 VLEI コード

上記の挿入時に要素の更新が広い範囲に及ぶ問題に対して、番号付けの値に予め適当な間隔を与える手法 [10] や番号付けの値を浮動小数点数で表現する手法 [1] が提案されている。しかし、予めあけられた間隔以上には挿入できず、浮動小数点数も精度限界が存在する。そもそも、これらの固定長の整数や浮動小数点を使う方法では、プログラミング言語や OS の仕様により数値の上限および下限が存在する。

可変長の bit 列を利用するこどでこの制限を外すことができる。しかし、単に可変長の 2 進数の値を利用しただけでは、任意の連続する値の間に新たな要素を挿入した場合に、付加情報を付けない限り、値間の大小関係を表現することができない。この付加情報のコストは大きい。

そこで、任意の 2 値間に挿入しても、挿入した値を含む 3 値間の適正な大小関係を付加情報を加えず表すことができる *VLEI(Variable Length Endless Insertable)* コードを提案する。VLEI は、1 を基準として、後ろに 0 をつけると 1 よりも小さくなり、逆に後ろに 1 をつけると大きくなるとする。同様にしてある bit 列に対して後ろに 0 をつけるた bit 列は元の bit 列よりも小さい、1 をつけたものは大きい。また 0、1 をつけた後の bit 列はつける前の bit 列とそれ以外の bit 列の大小関係を継承するとする。つまり 110 は 1 よりも大きくて、101 は 1 よりも小さくなる。この規則により bit 列に適切な大小関係を与える VLEI コードを作成することができる。

定義 1. VLEI コード
以下の関係が成立する $v = 1 \cdot \{0|1\}^*$ を VLEI コードと呼ぶ。

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

この大小関係は、図 3 のように 1 を根とする 2 分木と見なすができる。v をある VLEI コードとすると、v は $v \cdot 0$、$v \cdot 1$ の親、逆に $v \cdot 1$ は v の大きいほうの子供、$v \cdot 0$ は小さいほうの子供と呼ぶ。VLEI コードは節点又は葉となり、左に行くほど値は小さくなり、右に行くほど大きくなる。VLEI コードは、要素間の順序関係が保存されることからアルファベティッ

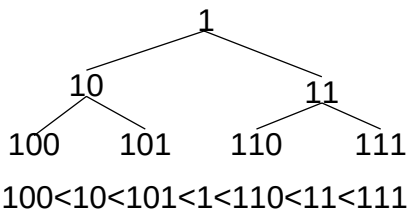

図 3: VLEI の大小関係を 2 分木表現

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
　　$(v_l < v_r)$
出力: 挿入された要素のコード v_i
if $l(v_l) \leq l(v_r)$
　　$v_i = v \cdot 0$
else
　　$v_i = v \cdot 1$
endif
return v_i

図 4: アルゴリズム 1. 挿入する要素を決める

クコード [5] の一種である。

次に、VLEI コードにおいて、任意の場所に要素を自由に挿入できることを証明する。

3.1 挿入

図 4 に挿入する要素の VLEI コードを求めるアルゴリズム 1 を示す。次に、このアルゴリズムによって定められた挿入する要素の VLEI コードがまだ割り当てられていない値であることを示す。

定義 2. VLEI の長さ
$l(v)$ を v の bit 長とする。

定理 1. 任意の隣接する VLEI コード v_l と $v_r(v_l < v_r)$ に対して、

1. $l(v_l) \leq l(v_r)$ のとき、$v_l \cdot 0$ はまだ割り当てられていない値である。
2. $l(v_l) \geq l(v_r)$ のとき、$v_l \cdot 1$ はまだ割り当てられていない値である。

証明 背理法で証明する。

1. 隣接する v_l と $v_r(v_l < v_r)$ に対して、その外に $v_r \cdot 0$ が既に割り当てられていたと仮定する。ここで、定義 1 より $v_r \cdot 0 < v_r$。また、$v_r \cdot 0$ は v_r に 1bit 加わえたものなので、$l(v_r) < l(v_r \cdot 0)$。さらに、条件の $l(v_l) \leq l(v_r)$ より、$l(v_l) \leq l(v_r) < l(v_r \cdot 0)$ となり、$v_r \cdot 0 \neq v_l$ となる。一方、$v_l < v_r$ と $l(v_l) \leq l(v_r)$ から $v_l = v_c\{\cdot 0 \cdot s_1 | \varepsilon\}$、$v_r = v_c \cdot 1 \cdot s_2$ (ここで $v_c = 1 \cdot \{0|1\}^*$、$s_1 = \{0|1\}^*$、$s_2 = \{0|1\}^*$ とする) と表すことができる。よって $v_c\{\cdot 0 \cdot s_1 | \epsilon\} < v_c \cdot 1 \cdot s_2 \cdot 0 = v_r \cdot 0$ となる。これより $v_l < v_r \cdot 0 < v_r$ となり、v_l と v_r が隣接という仮定に反する。よって $v_r \cdot 0$ はまだ割り当てられていない。
2. 同様にして証明可能。□

3.2 自然数から VLEI へのマッピング

図 2 にあるような自然数を使った範囲ラベリングに対して、初期状態として長さをできるだけ短かい VLEI コードを割り当てたい。そこで最大 $2^{m+1} - 1$ の要素に対して最大長 $m + 1$ の VLEI コードを割り当てる方法を提案する。

ここでは、[3] により予め全ての要素の数と前順、後順の自然数での番号がわかっていると仮定する。図 5 に示すアルゴリズムにより要素 N の自然数の列 $[a_1, a_2, ..., a_i, ..., a_N]$ に対して、VLEI コードを割当てることができる。

次にこのアルゴリズムで作成したものが重複がないことと $a_i < a_j$ を満たしていれば $v_i < v_j$ を満たすことを示す。

定理 2. 重複のない自然数の列 $[a_1, a_2, \ldots, a_N]$ に対して、アルゴリズム 2 で作成した $[v_1, v_2, \ldots, v_N]$ は重複がない。

証明 上のアルゴリズムの中では

$$a_i = 2^m h_1 + 2^{m-1} h_2 + \ldots + 2h_m + h_{m+1} \qquad (1)$$

という計算を行っていると見なすことができる。ここで h_l は VLEI コードの左から l 番目の bit が 1 であ

```
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
v_i = x
return v_i
```

図 5: アルゴリズム 2. 自然数から VLEI へマッピング

れば 1、0 であれば-1、存在しなければ 0 を表している。ここで

$$2^m > \sum_{j=1}^{m-1} 2^j = 2^m - 1 \qquad (2)$$

から、$|h_{i+1}|2^i > |h_i|2^{i-1} + |h_{i-1}|2^{i-2} + \ldots + |h_1|$ なので、h_i 以降のいかなる組み合わせでも $|h_{i+1}|2^i$ を超える数値は作れない。よって $H_i = [h_1, h_2 \ldots, h_{m+1}]$ とすると、異なる自然数 a_i と a_j に対して、$H_i \neq H_j$ である。また、アルゴリズム 2 より、$H_i \neq H_j$ ならば $v_i \neq v_j$ である。よって重複のない自然数の列 $[a_1, a_2, \ldots, a_N]$ に対して、アルゴリズム 2 で作成した $[v_1, v_2, \ldots, v_N]$ は重複がない。□

定理 3. $a_i < a_j$ を満たしていれば $v_i < v_j$ を満たす。

証明 ここで g_l は h_l 同様に VLEI コードの左から l 番目の bit が 1 であれば 1、0 であれば-1、存在しなければ 0 を表している。

$$a_i \quad = \quad 2^m h_1 + 2^{m-1} h_2 + \ldots + 2h_m + h_{m+1}$$

表 1: 前順と後順に VLEI コードを割り当てた表

前順の値	後順の値	要素の名前	要素の値
100	111	TEL⊥IST	
10	101	Element	
101	100	Name	Tarou
1	10	TEL	03-1234
110	11	Element	
11	1	Name	Hanako
111	110	TEL	06-1234

$$a_j \quad = \quad 2^m g_1 + 2^{m-1} g_2 + \ldots + 2h_m + g_{m+1}$$

ここで $a_i - a_j > 0$ かつ式 (2) より $h_i - g_i$ を $i = 1$ から $m + 1$ まで順に見ていくと値が負になる前に必ず正の値が得られる。よって $a_i < a_j$ を満たしていれば $v_i < v_j$ を満たす。□

実際に図 1 の XML に対してこのアルゴリズムを利用して前順と後順の VLEI を求めたものを表 1 に示す。

3.3 VLEI の比較

範囲ラベリングで包含関係を調べるために各 VLEI コード間の大小関係の比較を行う必要がある。そこで次の方法で 2 つの VLEI コードの大小を決定できる。

1. 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。
2. 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。

この方法では VLEI コードの値によって大小関係がわかるまで何回 bit を比較すればいいのか変わってくる。そこで平均の比較回数を求めてみる。ある長さの VLEI が 2 つある。一様に bit が割り当てられている状態では 2bit 目が存在する要素のうち半分の要

表 2: [2] における平均の bit 比較回数

ファイル名	要素数	平均値	ファイル名	要素数	平均値
com_err.xml	3153	3.171	m_for_m.xml	4876	3.512
dream.xml	3361	3.099	hen_viii.xml	4903	3.501
tempest.xml	3757	3.019	hen_vi_3.xml	4907	3.500
john.xml	3926	3.000	m_wives.xml	4958	3.482
titus.xml	3932	3.000	hen_v.xml	4971	3.477
macbeth.xml	3975	2.997	all_well.xml	5031	3.458
pericles.xml	4000	2.995	hen_vi_2.xml	5046	3.454
rich_ii.xml	4116	3.973	win_tale.xml	5050	3.453
two_gent.xml	4141	3.950	lll.xml	5056	3.451
merchant.xml	4145	3.946	r_and_j.xml	5080	3.444
hen_vi_1.xml	4334	3.797	hen_iv_2.xml	5325	3.364
timon.xml	4399	3.793	lear.xml	5984	3.233
j_caesar.xml	4455	3.717	troilus.xml	6077	3.223
as_you.xml	4522	3.679	othello.xml	6194	3.203
t_night.xml	4568	3.654	rich_iii.xml	6224	3.195
taming.xml	4675	3.599	coriolan.xml	6285	3.181
much_ado.xml	4727	3.574	a_and_c.xml	6347	3.167
hen_iv_1.xml	4825	3.532	hamlet	6636	3.116

素が左から 2 番目の bit 値が 1 で残り半分のものが 0 である。よって 2bit 目で大小関係がわかる確率は $\frac{1}{2}$ となる。よって k 番目で大小関係がわかる確率は $\frac{1}{2^{k-1}}$ となる。次に長さ n と長さ n の VLEI を比較したとき何 bit 目で大小関係がわかるかの期待値を出してみる。比較回数の期待値 $p(n)$ は次の式のようになる。

$$p(n) = \sum_{k=2}^{n} k\frac{1}{2^{k-1}} = 4 - \frac{n-1}{2^n}$$

となり、$n = \infty$ としても $p(n) = 4$ となる。よって 4bit 目、言い換えると 3 回の bit の大小関係を比較することで VLEI の大小関係がわかる。

実際の XML ファイルに対して比較回数を調査した。対象は [2] の各 XML ファイルに対して、各要素に VLEI を割り当て、後順のすべての VLEI を総当りで比較をし、何回比較して結果が出たかその平均値を出した。結果を表 2 に示す。要素数とは XML のファイルにあるすべての要素の数であり、平均値とはそのファイルにおいて何回 bit の大小関係を比較したかの平均値である。

この結果から、要素数に関係なく、平均で 3〜4 回 bit を比較することで大小関係を判定できることがわかる。結果をもう少し詳しく解析するとほぼ一様に bit が割り当てられている pericles.xml などが 3 より小さい値となっている。しかし、要素の挿入位置に著しい偏りがあり、各 bit 列中の特定位置の bit 値分布が偏ると、この比較回数は大きくなってしまう。rich_ii.xml をみるとほぼ 4 に近い値になっている。この場合には VLEI コードの格納に 13bit 使っている、つまりもし均等に割り当てると 8191 個要素を格納することが可能だが、大体半分の 4116 個しか要素を格納していない。そのため初期番号の割り当てのところで示した方法では 2 番目の bit が 0 になるものが 1 になるものよりもかなり多くなり偏りが生じたため悪くなってしまっている。

一様に bit が割り当てられていてもある部分に集中して要素が挿入すると VLEI コードに偏りが起き、大小関係がわかるまで比較する bit 数が大きくなってしまう。そこで偏りが生じたらなるべく均衡化する必要がある。

3.4 VLEI の偏り制御

前節で述べたように VLEI に偏りがあると比較を行うときに、大小関係がわかるまでに読み込まなければいけない平均の bit 長が長くなる。そこでできるだけ一様に VLEI を割り当てなければいけないが、一から VLEI を割り当て直していては I/O や計算のコストが高くなってしまうので、ある一定の VLEI の偏りを短くすることでなるべく一様になるようにしたい。そこで次に示す方法で長さの偏りを制御する。

方法 1 VLEI コードを 2 章で述べた 2 分木と見立てて考える。

1. ある一定の長さよりも長い VLEI コードがあり、その親の親から根までたどっていく最中にこの長い VLEI コードが含まれていない方の子供が葉であり、両方とも小さい側についているまたは大きい側についている節点が 2 つ連続で続く場合つまり図 6 の丸で囲ったように挿入可能な VLEI コードを持つ親を探す。

2. 葉とより根に近いほうの親の 3 つを根に近い親の長い VLEI コードを含んでいないこの方に移動する。

3. 根に遠い親とその子供たちをひとつ上に移動させる。

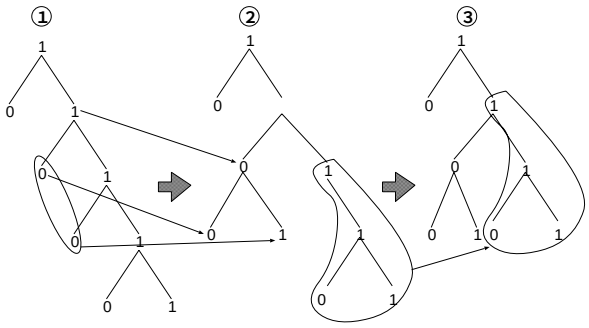

図 6: 方式 1 における偏り制御方法

この作業により bit を 1 つ短くすることができる。

方法 2　v をある VLEI コードとする。ここで $v\cdot 1$、$v\cdot 11$、$v\cdot 111$ が割り当てられていて、$v\cdot 10$ を含む VLEI コードが割り当てられていない時。

1. $v\cdot 1$ を $v\cdot 10$ にする。
2. $v\cdot 11$ を $v\cdot 1$ にする。
3. $v\cdot 111$ を $v\cdot 11$ にする。

同様に $v\cdot 0$、$v\cdot 00$、$v\cdot 000$ が割り当てられているときに同様の方法が使える。

3.5　削除・移動

要素の削除が行われたらその VLEI コードを消すだけで、他の作業は行わない。移動を行うときは挿入を行ってから削除を行うことで実現できる。

4　コストの比較

Cohen らの範囲ラベリング [3] で包含関係を調べるときに、VLEI を用いて番号付けをした手法と自然数を利用して番号付け場合のコストを見積もり比較する。

4.1　検索コストの見積もり

包含関係を調べる際に数の大小を比較しなければいけない。自然数を使う場合では 1 回の整数の大小比較で済む。VLEI コードでは bit が一様に振られている場合でも約 3 回の bit 比較を必要とする。よって自然数と比べておよそ 3 倍比較に時間がかかってしまう。

4.2　挿入コストの見積もり

次にある要素を追加した場合のコストを見積もる。VLEI コードを用いた手法では前後の要素を探してその長さを比較して長いほうに 1bit 追加したものをデータベースの中に追加する。よって 2 回の要素の検索と 1 回の大小比較と 1 回の要素の追加ですむ。

次に単純に整数を格納する方で考える。前順の値では挿入する要素の後ろの値すべてが変わってしまう。さらに後順では挿入する要素の後ろだけではなくその要素の親又は先祖の値まで変わる。よって要素の親の検索と、その要素の後順の値の変更、そして挿入する要素の後ろにあるすべての要素の前順の値の変更が必要となる。この VLEI コードを使う手法と比べるとその I/O のコストはかなり高いといえる。

またあらかじめ間を空ける手法と比べても、もし間を使い切ってしまった場合は必ず番号の付け直しをしなければいけないが、この手法ではどんなに挿入しても番号の付け直す必要はない。そのためこの手法は挿入に対してのコストは少ないといえる。

4.3　長さの偏り調整時のコスト見積もり

偏り制御する際の計算および I/O コストを見積もる。

まず方式 1. で図 6 の場合について見積もる。挿入時にある一定の長さ以上の VLEI コードを割り当てたときにマークしておくことで探し出す手間をなくすものとする。まず 1 において挿入可能な子供を捜すのに 5 回木の探索を行っている。次に 2 において 3 つの要素について値を更新している。最後に 3 において 4 つの要素の値を更新している。よって全部

で合計すると 5 回の木の探索と 7 回の値の変更分コストがかかる。

方式 2. を考える。まず $v\cdot 1$、$v\cdot 11$、$v\cdot 111$ のように連続している bit を割り当てた VLEI コードを全探索することで探し出す。次にそれぞれの VLEI コードに対して変更を行っているので 3 回の要素の変更を行っている。よって全探索するコストと 3 回の値の変更分コストがかかる。

5 まとめ

任意の個所に任意個挿入可能な可変長コードである VLEI コードとその挿入のアルゴリズムを提案し、挿入可能性に関して証明を行った。また、Cohen らの範囲ラベリング手法 [3] に VLEI コードを適用するために、自然数から VLEI コードへのマッピングをするアルゴリズムを提案し、その正当性を示した。さらに VLEI コードの比較コストの見積もりと偏りの影響について検討し偏りを平坦化する手法を提案した。自然数を用いた番号付けと比較して、検索コストは高くなるが、挿入コストを抑えることを示した。

今後の課題として、現在の VLEI の問題点として bit 列の先頭が一意に決まっていないため、1bit づつ比較をしていかなければいけない。そこで bit 列を改良し bit の先頭をそろえることで比較を高速化する手法を検討している。また、bit 列の格納方法として、クラスタ化して固定長のデータとして格納することで記憶効率を向上させる方法に関しても、検討を行っている。さらに、現在の偏り平坦化の手法ではある一定の偏りしか調整できない。そこである一部分に対して再構成ができるようにすることで、さまざまな偏りを調整できるようにする必要がある。

参考文献

[1] 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.

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

[3] Edith Cohen, Haim Kaplan, and Tova Milo. Labeling dynamic xml trees. In *PODS*, pages 271–281, 2002.

[4] 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.

[5] Donald E. Knuth. *SORTING AND SEARCHING*, chapter 6. ADDISON WESLEY, 1972.

[6] Quanzhong Li and Bongki Moon. Indexing and querying xml data for regular path expressions. In *VLDB journal*, pages 361–370, 2001.

[7] 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 *SIGMOD*, pages 204–215, 2002.

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

[9] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, and Shunsuke Uemura. Xrel: A path-based approach to storage and retrieval of xml documents using relational databases. In *ACM Transactions on Internet Technology*, pages 110–141, August 2001.

[10] 江田毅晴, 天笠俊之, 吉川正俊, and 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*, pages 35–38, 2002.