経歴･オフセット法による XML 文書の一実装方式

天木 圭太† 西野 裕臣‡ 都司 達夫† 樋口 健†

†福井大学工学研究科 〒910-0011 福井市文京 3-9-1
‡福井大学工学部 〒910-0011 福井市文京 3-9-1
E-mail: {amaki,nishino,tsuji,higuchi}@pear.fuis.u-fukui.ac.jp

あらまし 我々は，経歴オフセット法と呼ぶ多次元データセットのエンコード方式に基づいて，XML 木の構造更新に対して再ラベル付けを行う必要がない XML 木の構造エンコーディングの方式を提案している．この方式では XML 木の構造に従ったエンコードが行われるが，ノード名の連接であるノード名パス式が扱われていない．したがって，XPath などによる検索要求を受け付けることはできない．ここでは，XML 文書を入力として，XML 木の構造をエンコード/デコードするデータ構造とノード名パス式の構造をエンコード/デコードするデータ構造を構築する．２つのデータ構造によるエンコード結果を相互に参照することにより，パス式の検索と軸の指定による構造検索を効率よく行う方式を提案する．

キーワード XML 文書，多次元データ，記憶構造，XML 検索

An Implementation Scheme of XML Documents Based on History-offset Encoding

Keita AMAKI† Hiroomi NISHINO‡ Tatsuo TSUJI† and Ken HIGUCHI†

†Graduate School of Engineering, University of Fukui 3-9-1 Bunkyo, Fukui-shi, Fukui, 910-8507 Japan
‡Faculty of Engineering, University of Fukui 3-9-1 Bunkyo, Fukui-shi, Fukui, 910-8507 Japan
E-mail: E-mail: {amaki,nishino,tsuji,higuchi}@pear.fuis.u-fukui.ac.jp

1. はじめに

我々は，経歴オフセット法と呼ぶ多次元データセットのエンコード方式に基づいて，XML木の構造更新に対して再ラベル付けを行う必要がないXML木の構造エンコーディングの方式を提案している[3]．この方式は，XML 木を多次元拡張可能配列[1][2]に埋め込むことによりXMLノードのエンコードを行うが，同種他方式と比べ，ノードの追加場所と順序に関わらず，ラベルの記憶コストが低く，検索も高速である．

この方式ではXML木の構造に従ったエンコードが行われるが，ノード名の連接であるノード名パス式が扱われていない．したがって，XPathなどによる検索要求を受け付けることはできない．ここでは，XML文書を入力として，XML木の構造をエンコード/デコードするデータ構造とノード名パス式の構造をエンコード/デコードするデータ構造を構築する．２つのデータ構造によるエンコード結果を相互に参照することにより，軸の指定による構造検索とパス式の検索とを効率よく行う方式を提案する．

2. 経歴オフセット法と HOMD

経歴オフセット法は拡張可能配列の概念[1]を元にした多次元データのエンコード方式である．通常，拡張可能配列は主記憶上に領域が確保されデータの格納が行われる．しかし，多次元データセットを多次元配列の位置情報にマッピングする場合には，通常，有効な配列要素が少なく疎配列であり，記憶領域の無駄が多い．経歴オフセット法では配列の実体は持たず，論理的に配列を表現する．拡張された部分配列には，何番目に拡張されたかを表す経歴値を与え，論理配列の構造を表す次元毎に存在する経歴値テーブルと呼ぶ補助テーブルに格納する(図 1)．論理配列の配列要素は経歴値と，その経歴値を持つ部分配列内の位置を示すオフセット値により一意に表すことができる．この二つの値の組＜経歴値，オフセット値＞が配列要素のエンコードであり，高次元の多次元データのタプルであってもこの２つのスカラー値の組で配列要素を表現できる．

部分配列の各次元のサイズによってオフセット値が指し示す部分配列内の位置が変化するため，各部分配列のサイズの情報を係数ベクトルとして記憶する．

ここで，経歴オフセット法による論理拡張可能配列の座標から経歴値とオフセット値に変換する方法と，経歴値とオフセット値から論理拡張可能配列の座標

(添字の組）に変換する方法を，例をあげて説明する．まず，座標を経歴値とオフセット値に変換する方法について．図１の論理拡張可能配列において座標（1, 2）を変換する場合を考える．一次元の添字１の経歴値は１，二次元の添字２の経歴値は４であることが経歴値を持つテーブルから分かる．拡張可能配列の要素が属する部分配列は，その要素の各次元の添字に対応する経歴値の中で最大である経歴値が対応する部分配列を識別する経歴値である．つまり，例では経歴値４の部分配列 *S* に要素（1, 2）が含まれる．さらに最大経歴値以外の経歴値に対応した添字と，*S* のサイズの情報を持つ係数ベクトルから *S* のアドレス関数を計算し *S* における要素のオフセット値を求めることができる．例ではオフセット値は１である．よって，経歴値とオフセット値の組は <4, 1> となる．

次に経歴値とオフセット値の組から座標に変換する方法について，図１の経歴値 <4, 1> を添字に変換する場合を考える．まず経歴値を記憶しているテーブルを参照することによって，経歴値４から次元２の添字２が求まる．そして，経歴値４の部分配列の係数ベクトルから部分配列のサイズが分かるので，オフセット値から添字１が求まり経歴値とオフセット値を論理拡張可能配列の座標（1, 2）に変換することができる．

n 次元拡張可能配列では配列に新たな次元を動的に追加して *n*+1 次元とすることができる(次元拡張)．次元拡張時には，既存次元の経歴値テーブルと係数ベクトルテーブルはそのまま利用でき，次元拡張前の配列座標の再エンコードは必要ない．追加する新たな次元方向に対して，経歴値テーブルと係数ベクトルテーブルを設け，拡張前の *n* 次元配列の追加次元での添字は 0 となる．以後の次元サイズの拡張は *n*+1 次元の拡張可能配列として同じ手順で拡張することができる．

上述の経歴オフセット法に基づく多次元データセットの実装データ構造を HOMD（History-Offset implementation scheme for Relational Table）[2]（図２）と呼ぶ. HOMD では, 多次元データの各属性, 属性値, タプルをそれぞれ論理的な拡張可能配列の次元, 添字, 配列要素の座標に対応させる．HOMD は配列の各次元について，上述の経歴値テーブル，係数ベクトルテーブルおよび CVT（key-subscript ConVersion B+Tree）を持つとともに唯一の RDT（Real Data Tree)を持つ．各次元の CVT は B+木であり，当該次元の属性値をキーとして，その属性値に対応する添字をデータとして持つ．また，RDT は＜経歴値，オフセット値＞の組を格納する B+木であり, 経歴値が上位ビットにオフセット値が下位ビットに配置されることにより，そのシーケンスセットには＜経歴値，オフセット値＞の組が経歴値の順にソートされている．

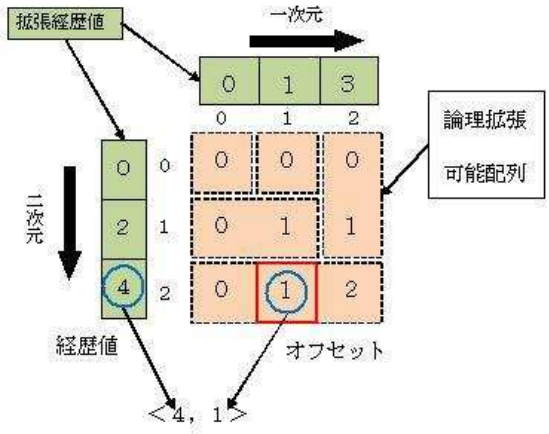

図 1 経歴オフセット法の概念図

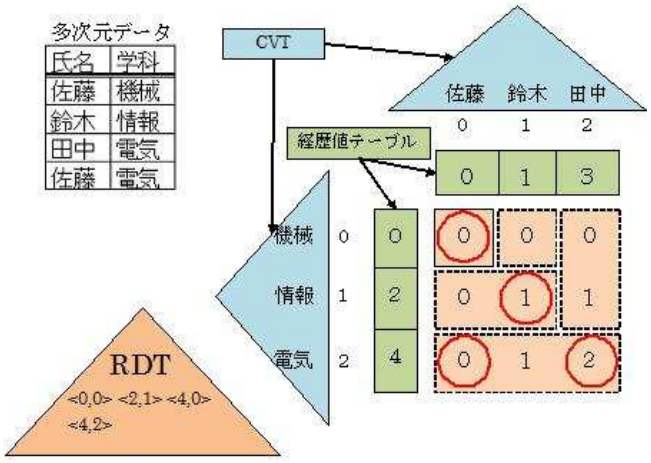

図 2 HOMD

3. XML 文書木ノードへのラベル付け方式[3]

本研究で用いる XML 文書木のノードのラベル付けには前節で述べた経歴オフセット法の考え方を用いる(図１). XML 文書木のノードの存在する木の各高さレベルを論理拡張可能配列の各次元に割り当てる．各高さレベルに存在する兄弟ノードに対して左から順にその木の高さに対応する次元の添字を１，２，３，…と割り当てる．この時，各ノードはノードの添字とその祖先のノードの添字とで論理拡張可能配列上の座標と見ることができる．この座標値は祖先の座標を含んでいるため，座標値を比較することでその座標値を持つノードが比較する座標値を持つノードの祖先，子孫，兄弟であるかの判別ができる．また，共通の祖先も二つの座標値で共通して持つ添字を調べることで簡単に求めることができる．

ノードの座標値は論理拡張可能配列の１要素に対応することとなる．この配列の１要素は前節で述べたように経歴オフセット法により経歴値とオフセット値

として表現できる．この２つの値の組をラベルとして用いて，図 2 の HOMD データ構造の内 RDT に格納する．

XML 文書の属性や PCDATA についても要素ノードと同様に論理拡張可能配列の１要素として扱う．属性はその属性を持つ要素ノードの子ノードとして扱う．格納したタグ，属性，PCDATA をそれぞれ識別するために，RDT に経歴値とオフセット値の組であるラベルを格納する際に，ラベルの上位２ビットを識別用のビットとする．そのラベル値を持つのが XML 文書のタグに当たるのならば 00，属性に当たるのなら 01，PCDATA に当たるのなら 10 としてラベル値によって識別ができるようにする．属性の属性値，PCDATA のテキストの内容はそれぞれ格納用に別ファイルを用意し, RDT のその属性や PCDATA のラベル値対応するデータ部に別ファイルのオフセットを格納する．こうして, ラベル値を知ることで, XML 文書でのタグ, 属性, PCDATA の判断や実データの参照が可能となる．

ノードの挿入によってある木の高さでのノードの最大の分岐数が変化した場合にその木の高さに対応する次元方向に配列の拡張が起こる．このように，分岐数の増大に対しては論理的な配列の拡張を行うことによって動的な更新に対応する．また，木の高さが増加して, 次元数が大きくなった場合には HOMD の次元拡張機能により, 再ラベル付けの必要なしに対応できる．このとき，ラベル値は２つのスカラー値で表現でき，内部的に固定長で扱うことができる．これらの利点は各次元サイズが固定の通常の多次元配列では獲得できない．

この経歴オフセットを用いたラベル付けによる XML 木の格納構造を nHOMD，その RDT を nRDT，経歴値とオフセット値の組を nID と呼ぶ．

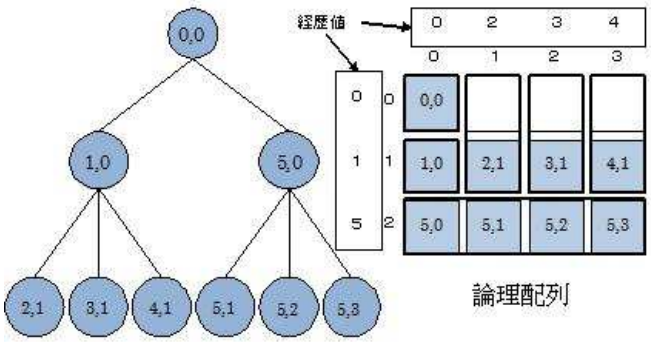

図 3 経歴オフセット法によるラベル付け

4. 文書順の保持

XML の重要な情報として，文書順が存在する．前節で述べたラベル付けの方法は文書順ではなく出現順にノードを論理配列に格納するため，文書順にノードを格納した場合はノードに対応する配列添字がそのまま文書順を表現するが，新たなノードが追加されると，添字が文書順を表さなくなる．そのため，ラベルだけで文書順を保った XML 文書を復元することができない．

そこで，文書順の情報を OS（Ordered Sequence）テーブル（図４）と呼ばれる補助データ構造によって保持する．OS テーブルは，１次元の配列で表される．OS テーブルの添字は論理拡張可能配列の添字を表し, OS テーブルの要素の値には, 文書順でその要素の次の兄弟ノードの添字を格納する．この OS テーブルを順に辿ることで，文書順を再現することができる．このとき, OS テーブルが表す兄弟ノードの長子のノードも OS テーブルと共に記憶しておく, こうすることで先頭の兄弟ノードから OS テーブルを辿る操作を行うことができる. これら OS テーブルの情報をその OS テーブルの対象となる兄弟の親ノードの nID から求められるように，nRDT のデータとして格納する．

新たなノードが挿入された場合には，その兄弟ノードの OS テーブルの内容を書き換えることで，文書順を保つことができる．この時, OS テーブル上の多くとも二つの値を書き換えることで文書順を保つための更新を行うことができる．

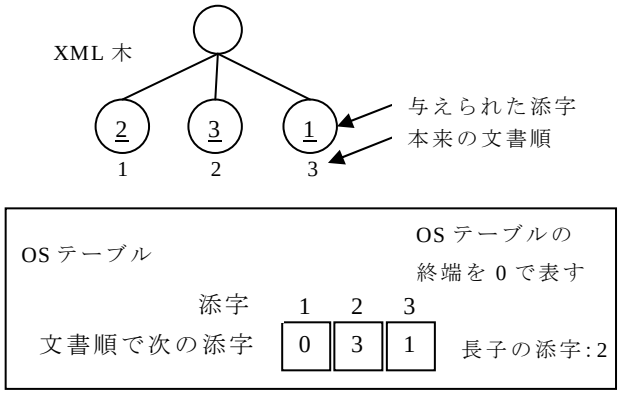

図 4 文書順保持のデータ構造

5. パス式の格納方法

前節では XML 文書木の木構造としての格納方式を説明したが，ノード名（タグ名）やノード名パス式の格納方式を扱っていない．ここでは，XML 文書木のノード集合の構造を格納管理する nHOMD とは別にタグ名パス式を格納管理するための構造を 3 節で述べた HOMD を用いる. HOMD を用いることで重複したパス式は同じキー値（＜経歴値，オフセット＞）として扱うことができる．

XML 文書木の木の各高さレベルを HOMD の論理配列の各次元に割り当てる．それぞれの次元の添字は，

その次元と対応する木の高さに属するノードの名前に対応させる．同じ名前のノードが同じ次元に複数存在する場合にも，一つのノード名を一つの添字に割り当てる．属性についても同様に属性名をそれぞれの添字に対応させる．PCDATA については PCDATA であることを表す名前を与え添字に割り当てる．

ノードの名前から論理配列の添字への変換は CVT を用いて行う．こうすることで，XML 文書木の一つのノード名パス式が HOMD の論理配列内の一要素に対応する．論理配列の一要素は前述したように，経歴値とオフセット値として表すことができる．

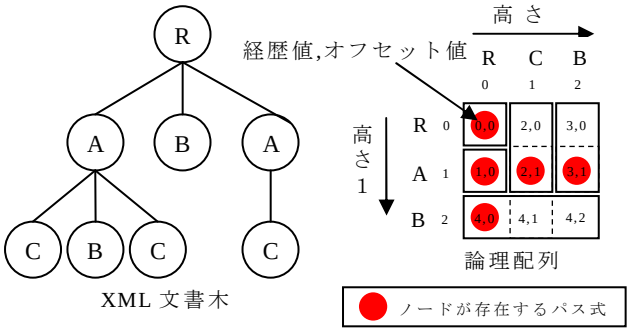

図 5 パス式を格納する HOMD

さらに，複数存在するノード名を一つの添字で表現するので，同じパス式を持つノードは一つの経歴値とオフセット値の組によって表される．図 6 におけるように XML 文書木を冗長性を排したパス式情報の木として表現することになり，パス式情報をコンパクトに表現することができる．

この HOMD を用いてパス情報を格納した構造を pHOMD，その RDT を pRDT，経歴値とオフセット値の組を pID と呼ぶ．

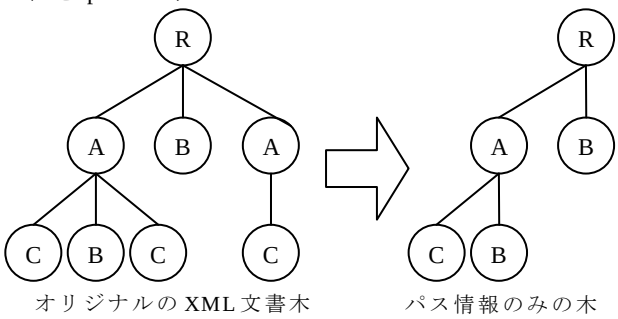

図 7 HOMD 上で表現されるパス情報のみの木

6. nID とパス式の相互変換方法

nHOMD における nID を対応するノード名パス式に変換するには，nID と pID を対応させる必要がある．そこで，nID を格納する nRDT に pID を格納する．nRDT において nID をキーとして，その nID で指定されるノードのパス式に対応する pID をデータとして格納する．pID は pHOMD の係数ベクトルを用いて各次元の添字にデコードされる．pHOMD の論理配列の添字はノード名に対応しているので，ルートノードから対象のノードまでのパス式上のノード名集合が求まる．こうして，nID からパス式への変換を行うことができる．

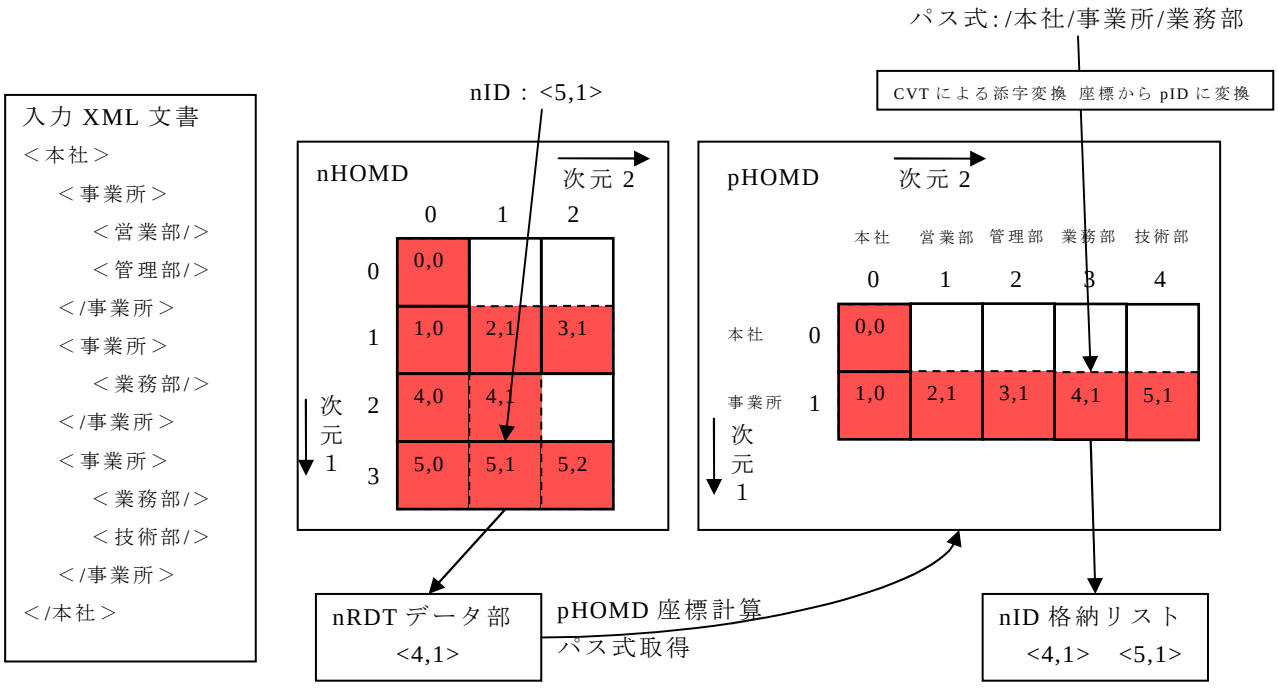

図 6 nID と pID の相互参照

このようにしてパス式を求めることができるので，検索等で求めた nID を元の XML 文書のタグ名パス式にデコードすることができる．

逆に，パス式を nID に変換するには pID から nID を求める必要がある．6 節では nID から pID を求める方法を述べたが，新たに pHOMD において pID と nID を関連付ける必要がある．そこで，pRDT において，キーである pID に対してデータとして対応する nID を格納する．pID は一つのノードパス式を表すが，pID 一つに対して，複数のノードが存在し得るので，一般には nID は複数存在する．そこで，それぞれの pID に対して二次記憶中の一定サイズのディスクブロックをリスト要素としたリスト構造に nID を順次格納する，pRDT にはこのリスト構造の先頭ブロックのアドレスを格納する．こうして，二次記憶のシーク回数を抑制して，対応する nID 集合を取り出すようにする．

例として図 5 の XML 文書木で/R/A/C というパスが指定された場合を考える．まずそれぞれの高さにノード名を分けると，ルートは R，高さ 1 のノード名は A，高さ 2 のノード名は C である．この A と C をそれぞれの次元に割り当てた CVT で添字に変換すると A が添字 1，C が添字 1 に変換される．2 節で述べたように，この添字から，経歴値とオフセット値に変換することができる．結果，経歴値とオフセット値が＜2，1＞と求まるので，これを pRDT から探し，リストから nID を取得する．

[具体例]

図 7 の XML 文書を具体例として説明する．まず，nID からパス式への変換を説明する．図 7 の nHOMD において nID <5,1> は ”/本社/事業所/業務部”のパス式に一意的に対応している．nRDT から nID <5,1>を探す，そのデータ部には pHOMD を構築する時に格納した対応パス式の pID が格納されている．図 7 の場合は pID は<4,1>である．この pID は pHOMD 上で座標に変換できる．その座標値を対応するノード名に変換して対応パス式が得られる．このようにして，nID をパス式に変換することができる．

次にパス式から nID に変換する場合について説明する．まず，指定されたパス式（/本社/事業所/業務部）を木の高さ別のノード名に分解する．そして，分解したノード名を CVT により，添字に変換する．本社，事業所，業務部の添字がそれぞれ 0，1，3 と求まるので座標値は（1,3）となる．この座標値を経歴値とオフセット値に変換すると，経歴値が 4，オフセット値が 1 となるので，経歴値とオフセット値を組み合わせて pID として pRDT から pID <4,1>を探す．pRDT のデータ部からそのパス式を持つノードの nID を格納しているリストを参照し，指定されたパス式に対応する二つの nID <4,1>と<5,1> を得る．このようにしてパス式を nID に変換することができる．

7. nHOMD と pHOMD を用いた構造検索

本節では nHOMD と pHOMD を用いた XML 文書の構造検索について説明する．

前節で述べたように，pHOMD によって/R/A/B のようなパス式から該当するノードの nID 集合を求めることができる．しかし，カレントノードを起点とする軸に沿った構造検索には pHOMD だけでは対応できない．そこで，nHOMD を用いることでより軸検索を行う．ここでは，XML 文書木のカレントノードの nID が与えられたとして，その子ノード，親ノード，兄弟ノードの nID を求める方法を説明する．

子ノードの検索について説明する．まず，カレントノードの nID を nHOMD の論理配列座標値に変換する．カレントノードの nID によって nRDT から OS テーブルを取得する．OS テーブルを長子の添字から順に辿り，カレントノードの座標と OS テーブルから得られた添字を組み合わせることで子ノードの座標値を長子から末子まで順番に得る．子ノードの座標値を再び経歴値とオフセット値にエンコードすることで子ノードの nID が得られる．得られた nID をキーとして，nRDT を探索し，対応する pID を求めることができる．さらに，pHOMD を使用して，6 節で述べた手順により，pID から当該のノード名のパス式にデコードすることができる．

親ノードの検索については，まず，カレントノードの nID を同様に nHOMD の論理配列座標値に変換する．座標値の中で最大次元の添字を 0 にすれば親ノードの座標値となる．求まった親の座標値を再びエンコードして親ノードの nID を求めることができる．得られた nID からノード名のパス式へのデコードは同様に 6 節の手順による．

続いて，兄弟ノードの検索については，まず，カレントノードの nID を nHOMD の論理配列座標値に変換する．親ノードを求める方法でカレントノードの親ノードの nID を求め，その nID をキーとして nRDT から OS テーブルを取得する．その後，子ノードを求める方法と同様にカレントノードの親ノードの子ノードの nID を長子から末子まで求める．そのとき OS テーブルから得られる添字とカレントノードの座標値の最大次元の添字を比較して，一致するまでがカレントノードより文書順で先に存在する兄弟ノード，一致してから末子までがカレントノードより文書順で後に存在する兄弟ノードとなる．このように，nHOMD を用いて，XML 文書木の構造検索を処理することができる．得ら

れた nID からノード名のパス式へのデコードは同様に 6 節の手順による.

[具体例]

図 8 の XML 文書を具体例として前述のノードの求め方を説明する. カレントノードを nID <2,0>, 座標値(2,0)のノードとする.

まず, 子ノードを求めるには, カレントノードの nID <2,0> を nRDT から探し, その OS テーブルを取得する. そして, OS テーブルから文書順に添字を得る. 図 8 の例では, OS テーブルから添字 1 と 2 が順に得られる. それぞれをカレントノードの座標値と組み合わせると, 座標値(2,1), (2,2)が得られる. この得られた座標値を nID に変換すると<3,2>, <4,2>が得られる. この二つの nID がカレントノードの子ノードとなる.

親ノードについては, 座標値の 0 でない最大次元の添字を 0 とおくことで求められる. カレントノードの座標値は(2,0)であるので, 0 でない最大次元の添字は 2 である. この 2 を 0 とすると, 座標値(0,0)が求まる. 座標値(0,0)を nID に変換して<0,0>が得られ, 親ノードの nID を求めることができる.

最後に兄弟ノードを求めるには, まずカレントノードの親ノードを求める. 親ノードの nID は<0,0>である. nRDT から求めた親ノード<0,0>を探索し, OS テーブルを取得する. OS テーブルから文書順に添字を読み込む. 例の場合は添字 1, 2, 3 が順に得られる. カレントノードの親ノードの座標値(0,0)と OS テーブルから読み出した添字を順に組み合わせ, nID に変換する. すると, 最初に座標値(1,0), nID<1,0>が求まる. カレントノードが現れるまではカレントノードよりも文書順で前に現れる兄弟ノードである. 次に座標値(2,0), nID<2,0>が求まり, これはカレントノードであるので, 以降の兄弟ノードがカレントノードよりも文書順で後に現れるノードである. 最後に座標値(3,0), nID<5,0>が求まる. このようにして兄弟ノードを求めることができる.

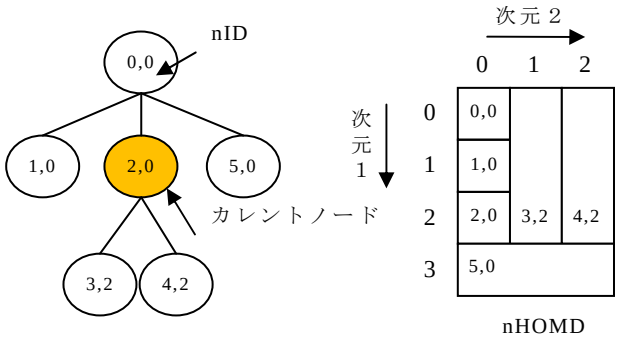

図 8 ノードの求め方

8. 実験

本研究における提案方式にしたがって, 構築したプロトタイプシステムを用いて, 実装した XML 文書について検索時間を求める実験を行った. ~~検索には~~ XPath として表現できるいくつかの検索パス式について, ノードを検索した.

実験に用いた計算機の環境は以下の通りである.

CPU	UltraSPARC-III 750 [MHz]
Memory	512 [MByte]
OS	Solaris 8

実験には XMark[4]によって生成した XML ファイルを用いる. 総ノード数は 388368 であり, その内訳は要素ノードが 212014, 属性ノードが 53512, テキストノードが 122842 である.

実装した各データ構造の記憶サイズを表 1 に示す.

表 1 記憶サイズ

データ名	記憶サイズ[MB]
nHOMD	33.2271
nHOMD の内 nRDT	32.9733
pHOMD	0.02483
pHOMD の内 pRDT	0.01836
pRDT から参照する nID リスト	5.5625
OS テーブル	6.0691
テキストデータ	5.8612
属性データ	0.6368
総記憶サイズ	51.3815

表 1 から記憶サイズの大部分を nRDT が占めていることが分かる. B+Tree である nRDT はその記憶コストがラベル値であるキーのサイズに大きく依存する.

XML 文書のパス情報を記憶している pHOMD のサイズは 5 節で述べたパス情報のみの木を用いるため nHOMD と比較して非常に小さな記憶サイズであることが分かる. 実験に用いた XML ファイルの pHOMD, 即ちパス情報のみの木での総ノード数は 465 である. また, nHOMD や p HOMD の大部分はそれぞれ, nRDT や pRDT が占めている.

次に nHOMD を用いた検索条件と検索時間, 検索ヒット数を表 2 に示す. ここで, 検索時間は与えられた検索式を満たすノードの nID 集合を求める時間である. テキストノードや属性ノードの内容とのマッチングの条件, PCDATA や属性の値を参照する必要があるため, location の要素ノードの数や featured の属性ノードの数が増えることで検索時間が増加すると考えられる. また, ワイルドカードである‘*’を含む検索のコストが大きいことが表 1 から見て取れる. これは, そのほかの検索に比べて対象とする XML 文書木の範囲が大きくなっていることが原因であり, 検索範囲が大き

表 2 nHOMD 検索時間測定結果

検索パス式	検索時間[sec]	ヒット数	辿ったノード数
/site/regions/africa/item/description/parlist/listitem	0.09000	65	1055
/site/closed_auctions/closed_auction/price	1.07999	1365	12292
/site/people/person/name	1.94999	3570	21443
/site/regions/*/item/description	3.17998	3045	35735
/site/regions/africa/item	0.01000	77	90
/site/regions/africa/item/location[text()='United States']	0.09999	62	913
/site/regions/africa/item[@featured='yes']	0.05000	7	90
/site/regions/*/item	0.34999	3045	3058
/site/regions/*/item/location[text()='United States']	3.85999	2282	35735
/site/regions/*/item[@featured='yes']	1.86000	303	3058

表 3 pHOMD 検索時間測定結果

検索パス式	検索時間[sec]	ヒット数
/site/regions/africa/item/description/parlist/listitem	0.00999	65
/site/closed_auctions/closed_auction/price	0.02999	1365
/site/people/person/name	0.09999	3570
/site/regions/*/item/description	0.08999	3045
/site/regions/africa/item	0.00999	77
/site/regions/*/item	0.08999	3045

いほど，ノードをいくつも辿る必要があるため nRDT から検索する頻度が増え，結果として二次記憶へのアクセスが増えることで検索時間が増加していると考えられる．そのため，検索範囲となる XML 文書木の範囲を削減する方法が必要である．具体的には検索対象となる木の部分木に注目し，pHOMD を調べて部分木に検索の条件として指定されたノード名が存在しないのであればその部分木の検索を行わないようにする，などの方法が考えられる．

最後に，表 3 に pHOMD を用いて検索パス式で指定されるノードの nID 集合を求める検索を 100 回行ったときの合計時間を示す．pHOMD は文書順の情報を持たないため，文書順を考慮する必要はない．また，この測定では pHOMD のみを用いたため，テキストや属性値との比較は行っていない．表 3 の検索時間は検索を 100 回行ったときの合計時間なので，nHOMD の検索と比べるとほとんど無視できる時間でパス式に一致するノードを求めることができる．よって，表 3 のような pHOMD の検索のみで解決できる検索パス式を判定することにより，検索時間の短縮が可能である．

9. おわりに

多次元データのエンコード方式である経歴オフセット法により，XML 木の構造をエンコード/デコードするデータ構造とノード名パス式の構造をエンコード/デコードするデータ構造を示した．さらに，２つのデータ構造によるエンコード結果を相互に参照することにより，パス式の検索と軸の指定による構造検索を効率よく行う方式を提案した．7 節にて，XPath におけるいくつかの軸の求め方について述べた．7 節で述べたような軸を求める処理を行い，求めた nID に対してさらに軸を求めることを繰り返すことで，XPath の複数のロケーションステップの軸を処理する．さらに，ノードの名前の指定は pHOMD を用いてノードのパス式を求めることで対応する．述語についても，求めた nID に対して与えられた条件を満足するかどうかを調べることによって対応する．XPath による検索システムをより充実させることと，検索範囲が増大した場合の対処方法を考案することは今後の課題である．

参 考 文 献

[1] E.J.Otoo, T.H.Merrett: A Storage Scheme for Extendible Arrays, Computing, Vol.31, pp.1-9, 1983.

[2] Tsuji T., Kuroda M., Tsuji T., Higuchi K.: History offset implementation scheme for large scale multidimensional data sets, Proc. of ACM Symposium on Applied Computing (SAC2008), pp.1021-1028, 2008.

[3] Li B., Kawaguchi K., Tsuji T., Higuchi K.: A Labeling Scheme for Dynamic XML Trees Based on History-offset Encoding, 情報処理学会論文誌: データベース, Vol.3, No.1, pp.1-17, 2010.

[4] XMark : http://www.xml-benchmark.org/