大規模木構造データに対する正規パス式の効率的な処理方式の検討

只石 正輝 †　森嶋 厚行 ‡　田島 敬史 ††

筑波大学 図書館情報専門学群 †　筑波大学大学院 図書館情報メディア研究科 ‡　京都大学大学院 情報学研究科 ††

1. はじめに

本稿では，エッジラベル付き大規模木構造データに対する正規パス式を効率よく評価するためのディスク上でのデータ格納方式について提案する．XML や各種オントロジデータなど，近年はエッジラベル付きの大規模木構造データの扱いが重要となっている．また，正規パス式は半構造データ言語[1] 等の文脈で議論されてきたが，近年 XML データに対しても Regular XPath 式という形でまた注目されており[2]，その効率よい処理が求められている．

一般に，正規パス式は，子要素を求める演算と，閉包を求める演算に展開できる．本稿では閉包に関しては特に需要の大きいと考えられる「特定ラベルの繰り返し」のみに焦点を当て，下記の 2 つの演算を扱う．

(1) a $\overset{label}{\rightarrow}$ X: ノード a から label を持つエッジ 1 度辿ることで得られるノード集合

(2) a $\overset{label+}{\rightarrow}$ X: ノード a から label を持つエッジを 1 度以上辿ることで得られるノード集合

演算の結合は・を用いて記述する．例えば，a $\overset{t1}{\rightarrow}$ ・ $\overset{t2}{\rightarrow}$ X はノード a から t1 を持つエッジを 1 度辿り，さらにその解となるノードから t2 を持つエッジを 1 度辿ることで得られるノード集合を表す．

本稿では，特に木を構成するノードのディスク上での配置順序に着目し，少ないディスクアクセス数でこれらの演算を処理する手法を提案する．提案手法は，J. Banerjee らの研究[3] におけるラベル無し木の処理の一般化になっているが，我々の知る限り，木構造データに対する正規パス式の処理に関してこのようなアプローチで取り組んだ研究は存在しない．

2. 格納方式

本節では，まずディスク上でのノード配置順序について提案し，次に格納方式の詳細について説明する．

2.1 ディスク上でのノード配置順序

正規パス式を効率よく処理するための我々のアイデアは，解となるノード集合が出来るだけ連続して並ぶようにノードをディスク上に配置する事である．これにより，ディスクのランダムアクセス数が減少する．

比較のために，まずは最も単純なノードの配置順として，深さ優先順にノードを配置することを考える．この場合，各正規パス式の解は連続して並ぶとは限らない．例えば，図 1 左の木構造のノードを深さ優先順で配置した場合，ノード 101 からラベル a を 1 度以上たどって到達可能なノード集合 (101 $\overset{a+}{\rightarrow}$ X) の解は，途中でラベル b を経由しないと到達しないノードを配置する必要があるため，連続して配置されない．したがって，ランダムアクセスの回数が増加する．

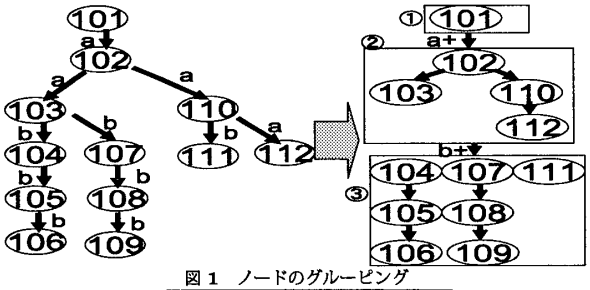

図 1　ノードのグルーピング

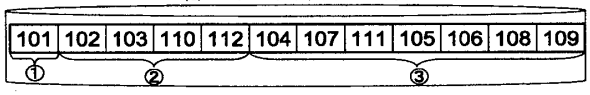

図 2　提案手法のディスクイメージ

次に，我々が提案するノード配置順を説明する．具体的には (1) ノード集合をグループに分割し，(2) 各グループ内のノードに順序をつける．下記ではこの順に説明する．

(1) ノード集合のグルーピング.

1. Root ノードから，ラベル t のみを辿って得られるノード集合，すなわち Root $\overset{t+}{\rightarrow}$ X の解を一つのグループとする．
2. t とは異なるラベルが出現したとき，1. で得られたグループを一つの大きな Root ノードとみなし，再び 1. を適用する．
3. 全てのノードがいずれかのグループに属したとき，各グループに対して深さ優先順に番号を振り，その順にグループに含まれるノードをディスクに配置する．(各グループ内でのノードの配置順序は次に説明する)

上記提案手法に従って，図 1 左の木構造をグループ化したものが図 1 右となる．また，各ノードを配置したディスクイメージを図 2 に示す．

(2) 各グループ内でのノード順序.

各グループ内のノード順序には工夫が必要である．なぜなら，101 $\overset{a+}{\rightarrow}$ ・ $\overset{b}{\rightarrow}$ X のように，グループ内の一部のノードを解とする正規パス式も存在するからである．これらの問合せについても，解に含まれるノードができるだけ連続するような配置順序が望ましい．

提案手法では，以下のルールに従ってグループ内のノードを配置する．

1. 親が存在しないノード及び親が別のグループに属するノードをグループの先頭から順に並べる．これらの順序は，それらの親ノードの親グループ内での順序に従う．
2. 1. で並べた各ノードの子孫を兄弟ごとにグループ化する．
3. 1. で並べたノードの直後に，2. で得られた各兄弟グループを深さ優先順に配置する．

図 2 は，以上のルールに従ってノードが配列されている．図中で，101 $\overset{a+}{\rightarrow}$ ・ $\overset{b}{\rightarrow}$ X の解に含まれているノードは連続して並んでいる．また，103 $\overset{b}{\rightarrow}$ X の計算の際は，まず {104, 107}を取得し，次に 105 以降のノードに連続してア

On an Efficient Processing Scheme of Regular Path Expressions for Large Tree Structures

Masateru TADAISHI† Atsuyuki MORISHIMA‡ and Keishi TAJIMA††

Sch. of Library and Information Science, Univ. of Tsukuba.† Grad. Sch. of Library, Information and Media Studies, Univ. of Tsukuba.‡ Grad. Sch. of Informatics, Kyoto Univ.††

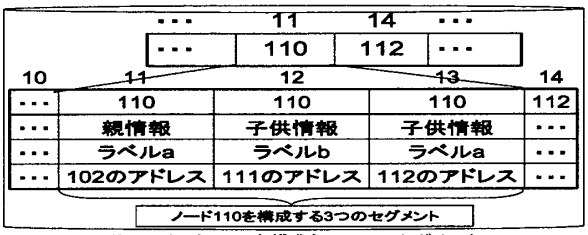

図 3 ノード 110 を構成する 3 つのセグメント

クセスする.

2.2 格納方式の詳細

まず，各ノードの格納方式の詳細について説明する．各ノード毎に，次のように，ノードの ID, 親，子供の情報を記録する．(1) ノード ID は固定長の領域を取る．(2) 親の情報としては，そのノードの親のアドレス，及び親からのエッジラベルを記録する．(3) 子供の情報としては，エッジラベルごとに子供のアドレスを記録する．ただし，提案手法では，特定のラベルを持つ子供は連続して並ぶため，各ラベルの最も先頭の子供のアドレスのみを記録すればよい．例えば，図 1 左の木構造におけるノード 103 の場合，ラベル b を持つ子供は複数存在するが，そのうち，最も先頭の子供，104 のアドレスを 103 の子供情報として記録する．以降，この最も先頭の子供のアドレスを firstChild と呼ぶ．

一般に，あるノードは複数のラベルの子供を持つ (例，図 1 のノード 110) ため，なにも工夫をしないとノードの格納サイズは可変長となり，ノードのアドレス管理が煩雑となる．この問題に対処するため，本提案手法では，ノードに記録する親と子供情報を，複数の固定長のデータに分割する．この固定長のデータをセグメントと呼ぶ．

図 1 のノード 110 の場合，ラベル a の親情報，ラベル a 及びラベル b の子供情報の計 3 つの情報を図 3 のように 3 つのセグメントで記述する．ここで，各セグメントは，ノードの ID, そのセグメントが格納する情報が親の情報か子供の情報かを示すフラグ，ラベルのタイプを表す固定長の情報，親または子供のアドレスで構成される．

以上の他に，本手法は各ノードのディスク上のアドレスを保持するテーブル (アドレス表と呼ぶ) を必要とする．これは，正規パス式が $101 \xrightarrow{a} X$ のようにノード ID で与えられることを想定しているためである．

3. 処 理 方 法

本章では，正規パス式を具体的な処理方法について説明する．スペースの関係上，$a \xrightarrow{t} X$, $a \xrightarrow{t+} X$ それぞれ単体の場合の処理について説明する．

[$a \xrightarrow{t} X$ の処理]

1. ノード a のアドレスを取得する．
2. ノード a のアドレスへ移動し，ラベル t を持つ firstChild のアドレスを取得する．
3. 取得した firstChild のアドレスへ移動し，親がノード a のアドレスである限り，シーケンシャルに読み続ける．

[$a \xrightarrow{t+} X$ の処理]

1. ノード a のアドレスを取得する．
2. ノード a のアドレスへ移動し，ラベル t を持つ firstChild のアドレスを取得する．
3. 取得した firstChild のアドレスへ移動し，親がノード a のアドレスである限り，シーケンシャルに読み続ける．この際にアクセスした範囲を R3 とする．
4. 範囲 R3 に存在するノードが持つ子供のうち，ラベル t を持つ子供で且つ最もアドレスが小さい子供，minFirstChild のアドレスを取得する．
5. minFirstChild に移動し，ノードを読む．この際，アクセスした範囲を R5 とし，アクセス対象となるノードの親のアドレスが範囲 R3 か R5 のいずれかに属する限りシーケンシャルに読み続ける．また，読み込んだノードのアドレスを随時 R5 に追加する．

4. 実 験

提案手法で正規パス式を評価し，ランダムアクセス数と実行時間を調べた．実験を行った計算機環境は，OS RedHat Linux, CPU Xeon 2.8GHz, メモリ 1GB である．実装は Java 1.6.0 で行った．

実験で用いた木構造は，深さ 22 の完全二分木である．各ノードの 2 本のエッジには，1 つずつ t1, t2 というラベルが付いている．このグラフのノードの数は $2^{22}-1$，ディスクに格納した際のファイルサイズは約 167MB であった．

実験で与えたパス式は以下の 4 つである．

(式 1) Root $\xrightarrow{t1}$ ・ $\xrightarrow{t2}$ X　**(式 2)** Root $\xrightarrow{t1}$ ・ $\xrightarrow{t2+}$ X

(式 3) Root $\xrightarrow{t1+}$ ・ $\xrightarrow{t2}$ X　**(式 4)** Root $\xrightarrow{t1+}$ ・ $\xrightarrow{t2+}$ X

表 1 は，単純な深さ優先順でのノード配置と，提案手法でのノード配置のそれぞれで各式の解を得るために必要であったランダムアクセスの回数である．提案手法では，必要なランダムアクセスの回数が大幅に少ないことが分かる．

	式 1	式 2	式 3	式 4
深さ優先順	3	23	23	213
提案手法	2	2	2	2

表 1 ランダムアクセスの回数

表 2 は提案手法での実行時間である．なお，各実行時間は同じ正規パス式の処理を 5 回行った結果の平均であり，アドレス表への問合せ時間を含めていない．

	式 1	式 2	式 3	式 4
実行時間	9.2ms	11.0ms	11.4ms	22.2ms
解のノード数	1	20	20	210

表 2 実行時間

5. ま と め

本稿では，大規模な木構造データに対して正規パス式を効率的に処理するためのディスク上のノード配置順序について提案した．今後の課題として，木構造以外のラベル付き有向グラフへの拡張などが挙げられる．

参 考 文 献

1) S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. L. Wiener: The Lorel Query Language for Semistructured Data. Int. J. on Digital Libraries 1(1): 68-88 (1997)
2) W. Fan, F. Geerts, X. Jia, A. Kementsietsidis: Rewriting Regular XPath Queries on XML Views. ICDE 2007: 666-675
3) J. Banerjee, W. Kim, S.-J. Kim, J.F. Garza: Clustering a DAG for CAD Databases. IEEE Transactions on Software Engineering 14(11): 1684-1699(1989)