

文字列照合技術に基づく XML データ処理

XML Processing by String Matching Algorithm

喜田 拓也*

宮本 哲†

竹田 正幸†

Takuya Kida

Satoru Miyamoto

Masayuki Takeda

1 はじめに

XML (eXtensible Markup Language) は、最も強力かつ柔軟な汎用データ記述方式の世界標準である。XML の優れた点は、人間にとって可読なテキスト・データ形式でありながら、機械にとっても処理が容易であるように構造化されているところにある。このような特徴を持つ XML 形式によって記述されたデータには、単なる文書データのみならず、画像や音声といったマルチメディアデータや、さらにはネットワークプロトコルのようなデータも存在する。また、こうしたコンテンツ・データのためのデータ、すなわちメタデータもまた XML 形式によって記述されるようになってきた。例えば、図書館学の分野においては、従来より蓄積してきた図書や雑誌の書誌情報 (二次資料データ) を Dublin core[1] と呼ばれる XML 形式に沿って記述しなおそうという動きがある。このように、ありとあらゆる情報が XML 形式によって電子化され、流通されるようになってきた。このような背景をうけ、大量の XML データを効率よく処理する研究がより重要になってきている。

XML データに対して処理を行う場合、一度そのファイルを木構造へ変換し、その木構造のノードを操作する方法が一般的である。そのために用いられる DOM (Document Object Model) は、W3C によって標準化された API 仕様のひとつであり、XML 文書へのアクセスはこの DOM API を通して行われる。XML データから木構造を構築するプログラムは XML パーサと呼ばれ、XML を処理するアプリケーションの多くは XML パーサが提供する DOM API を駆使して XML データを処理している。例えば、ある XML 形式のデータを別の種類のデータに変換する際に用いられる XSLT (eXtensible Stylesheet Language Transformation) プロセッサは、内部で XML パーサを利用し、DOM を通じて多彩な XML データ処理を提供している。

しかしながら、木構造に基づく XML データへのアクセスでは、非常に柔軟な処理を可能とする反面、巨大なデータあるいは量が膨大なデータ群を効率的に処理することができない。なぜならば、ごく単純な処理を行う場合であっても、XML データを必ず木構造へ変換しなければならないため、大量のデータ群に対しては前処理にかかる時間の比率が大きくなるからである。また、木構造を保持するために元データ以上のメモリ領域を必要とするので、巨大なデータに対してはメモリ上での処理が困難となる。

本稿では、木構造に基づくアクセス手法とは異なるアプローチを提案する。本手法は、文字列照合の技術に基づいている。すなわち、XML データを先頭から読み出しながら処理を行うので、前処理の時間が小さく、メモリ消費量の少ないプログラムを作成することができる。そのため、大量の文書群や巨大なデータに対しても効率よく処理を行うことができる。提案する手法では DOM によるアクセスほどには柔軟な処理はできないものの、XML データの階層構造を監視しながら動作するので、例えば、member 要素の下の name 要素の中身を抽出する

といった処理が可能である。

同様の特徴をもつ手法としては、SAX (Simple API for XML) を用いたデータアクセスが存在する。SAX は W3C 標準ではないが、DOM の欠点を補完する事実上の標準 API として広く受け入れられている。SAX も XML データを先頭から読み出しながら処理を行うが、データの階層構造を考慮した処理を行うためには、API 利用者側でその仕組みを用意しておく必要がある。以上のことから、本手法は、DOM を用いる場合ほど柔軟なデータ操作はできないが、SAX を用いるよりは高度な処理が可能で、かつ SAX 並に効率の良いアクセス手段であると位置づけられる。

2 XPath と提案文法

XML データを木構造 (ツリー) で表現するデータモデルでは、要素・属性・名前空間宣言・コメント・文字データ等のすべてがノードで表される。ツリーを深さ優先で走査したノードの並びは、元々の XML データの前方から後方への並びに対応しており、これを文書順 (document order) と呼ぶ。あるノードについて、文書順においてそのノードの前に位置するノードを先行ノード (preceding node) と呼ぶ。また逆に、後続ノード (following node) とは、文書順においてそのノードの後ろに位置するノードをいう。

XML データにアクセスするためには、まず、ツリー中のノードの所在を特定する必要がある。ツリー中におけるノードを選択するのに用いられる表記法としては、XPath (XML Path Language) と呼ばれる W3C 標準がある。これはツリー中のノードの集合を指定する言語を定義している。例えば、

```
/book/chapter[@name = 'introduction']
```

という XPath の式は、book 要素の子のうち、name 属性が 'introduction' である chapter 要素を選択することを意味している。このような式は、主に XSLT プロセッサで処理される XSLT 文書のなかで用いられ、XSLT プロセッサは DOM を駆使してこの式を解決する。

XPath 式で指定されたノードの集合を特定することは、本質的にはツリーの中にあるサブ・ツリーを見つけ出すという問題である。文字列照合機械を用いた探索ではこのようなツリー・マッチングは行えないので、XPath を制限した言語を定義しなければならない。

文字列照合機械は XML データを前から順に処理するので、先行ノードを指定するような式は用いることができない。また XPath の式では、ノードの集合を特定する式 (ロケーションパスと呼ばれる式) の他に、簡単な算術式や文字列操作の式も表現できるが、問題が煩雑になるので省略する。さらに、ノードの指定に述部 (predicates) と呼ばれるフィルタ (上の例でいうと、[@name = 'introduction'] の部分) を付随して指定できるが、この表現を用いるとツリー・マッチングを行う必要が出てくるので、これも省く。以上より、提案手法で処理できる式としては、ロケーションパスのうちで述部を伴わず、後続ノードを特定するものに限る。これでは XPath に比べ、非常に制限された式しか使えないように見えるが、実用上は十分

*九州大学附属図書館研究開発室

†九州大学大学院システム情報科学府 情報理学専攻

に役立つ式を記述できる。XPath の定義を基に、上述の制限を加えた言語を BNF で記述すると次のようになる。

```

LocationPath      ::= '/' RelativeLocationPath
RelativeLocationPath ::= Step
                        | RelativeLocationPath '/' Step
Step               ::= AxisSpecifier NodeTest
AxisSpecifier      ::= AxisName '::'
AxisName           ::= 'attribute'
                        | 'child'
                        | 'descendant'
                        | 'descendant-or-self'
                        | 'following'
                        | 'following-sibling'
                        | 'self'
                        | 'namespace'
NodeTest           ::= QName
                        | NodeType '(' ')'
NodeType           ::= 'node'
                        | 'text'
                        | 'comment'
                        | 'processing-instruction'

```

ここで QName は、Namespaces in XML* で定義されており、ノード名を表す。NodeTest において、node() というのは、任意のノードを意味する。同様に、text(), comment(), processing-instruction() は、それぞれ、任意のテキストノード、任意のコメントノード、任意の処理命令ノードを意味する。この定義の範囲では、例えば、

```
/descendant::cars/child::car/attribute::node()
```

といった式が書ける。これは、データ中にある cars の下の car という要素に含まれる任意の属性を選択する。XPath の省略記法に沿って記述すれば、//cars/car/@* と書ける。

3 文字列照合機械によるノード探索

提案する手法は、スタック付きの文字列照合機械を用いて、XML データをそのままの形で処理を行う。前節で定義した式を入力として、選択されたノードに対応する XML データ上の範囲（位置の組）を出力する。アルゴリズムの詳細は省略するが、概説すると次のようになる。

まず、入力された式から各ノード名を抽出する。それらを文字列パターンとして同時に照合する。ノードの開始地点が見つければ、そのノードのインデックスと出現位置の組をスタックに積む。逆にノードの終了位置が見つければ、それらをスタックから取り出す。例えば、ノードが要素の場合、開始タグが表れたらスタックに積み、終了タグが表れたら取り出す。こうすることで XML の構造を把握した処理を行うことができる。文字列パターンが見つかるごとにスタックの状態を参照し、現在の位置が入力式を満たすノードかどうかを判定する。満たしていれば、開始地点と終了地点の組を出力する。

この方法では、複数の文字列パターンを照合しながら処理を行うので、Aho-Corasick (AC) 照合機械 [2] のような複数文字列パターン照合機械が必要である。また、スタックに積まれた階層構造を判定するために、ギャップや文字クラスを許した文字列パターン照合機械が必要である（そのような照合機械については [4] を参照）。ノードの終了位置が見つかった際には、後者の照合機械の状態も元に戻さなければならないので、ノードのインデックスと出現位置に加えて、そのときの照合機械の状態も共に積んでおく必要がある。このようにして文字列照合機械を組み合わせることで、XML データの階層構造を考慮したアクセスが可能となる。

*<http://www.w3.org/TR/REC-xml-names/>

この手法の最も有益な応用の一つに、階層構造を考慮したキーワード検索アプリケーションが考えられるが、もともと複数パターン照合機械を用いているので、ノードの探索と同時にキーワードの探索も行うことができる。ただし、ユーザが指定するキーワードがノード名の部分文字列であった場合、誤検出が生じる可能性を考慮しなければならない。例えば style という単語は、</xsl:stylesheet> というタグの一部にマッチするので具合が悪い。もちろんこの問題は、タグ文字列を囲っている「<」と「>」を識別し、その内か外かを表すブール値を一つ記憶しておくという単純な技法によって回避できる。しかしながら、このような余計な処理は照合機械の速度を低下させてしまう。我々は既に、AC 照合機械を拡張することで、この問題を回避するアルゴリズムを開発した [5]。

4 おわりに

本稿では、文字列照合の技術で処理可能な XPath 式のサブセットを定義し、巨大な XML データあるいは大量の XML データ群に対して効率的なアクセス手法を提案した。本手法では、複数文字列パターン照合機械を用いることから、効率よく多数の質問処理を一括して処理するように拡張できる可能性がある。まだ本手法の完全な実装には至っていないが、今後、DOM や SAX を用いた場合との比較実験を予定している。

今回、XPath 式において述部は省略したが、ロケーションパスにおける最右のノードに付随する述部に関しては、フィルタを指定してもツリー・マッチングを行わずにすむ場合があるので、これを組み込むことは今後の課題である。実際、XPath 式のロケーションパスの途中で述部を用いると、とたんにパフォーマンスが低下することが知られており、そうした述部を用いないことが推奨されている。

構造化文書に対する検索ツールとしては、Sgrep [3] がある。これは Nested region algebra というデータモデルに基づいて XML データの階層構造をとらえ、キーワードを検索するツールである。また別の方法として、木構造における根からすべてのパスを関係データベースに蓄えた後に操作する手法も提案されている [6]。これらのツールとの特徴の違いを検証することも今後の課題である。

参考文献

- [1] Dublin core metadata initiative. <http://dublincore.org/>.
- [2] A. V. Aho and M. Corasick. Efficient string matching: An aid to bibliographic search. *Comm. ACM*, 18(6):333-340, 1975.
- [3] J. Jaakkola and P. Kilpeläinen. A tool to search structured text. University of Helsinki. (In preparation).
- [4] G. Navarro and M. Raffinot. *Flexible Pattern Matching in Strings - Practical On-Line Search Algorithms for Texts and Biological Sequences*. Cambridge University Press, 2002.
- [5] M. Takeda, S. Miyamoto, T. Kida, A. Shinohara, S. Fukamachi, T. Shinohara, and S. Arikawa. Processing text files as is: Pattern matching over compressed texts, multi-byte character texts, and semi-structured texts. In *Proc. 9th International Symp. on String Processing and Information Retrieval*, Lecture Notes in Computer Science. Springer-Verlag, 2002. (to appear).
- [6] M. Yoshikawa and T. Amagasa. XRel: a path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology*, 1(1):110-141, August 2001.