韓國 컴퓨터情報學會 論文誌
第12卷 第5號, 2007. 11.

2007-12-5-1-8

동적 XML 문서에서 효과적인 레이블링을 위해 형제순서 값을 갖는 프라임 넘버링 기법

이 강 우*, 이 준 동**

A Prime Numbering Scheme with Sibling-Order Value for Efficient Labeling in Dynamic XML Documents

Lee KangWoo *, Lee JoonDong **

요 약

동적인 XML 문서에서 빈번히 발생하는 갱신에 대한 고려를 하지 않는 레이블링 기법들은 XML 문서 트리의 갱신이 일어날 때 마다 변화된 레이블 정보를 반영하기 위해서 전체 XML 트리를 재탐색하여 전체 노드의 레이블을 다시 계산하는 리레이블링(relabeling)과정이 필요하다. 이러한 리레이블링은 갱신이 빈번히 일어나는 동적인 XML 문서에서는 비용이 상당히 크다는 단점이 있다. 이런 단점을 해결하기 위해 리레이블링 과정이 필요 없는 레이블링 기법으로 프라임 넘버 레이블링 기법(prime number labeling scheme)이 제안되었다. 그러나 프라임 넘버 레이블링 기법은 문서가 갱신될 때 XML 문서 트리의 노드 간 형제순서(sibling order)를 갱신하는 문제는 고려하지 않고 있다. 이러한 형제순서의 갱신과정은 XML 문서 트리의 많은 부분을 재탐색하고 재기록 하여야 하므로 많은 비용이 필요하게 된다. 따라서 본 논문에서는 XML 문서 트리의 재탐색과 재기록이 필요 없이 형제순서를 유지할 수 있는 형제순서 값을 갖는 프라임 넘버 레이블링 기법을 제안한다.

Abstract

Labeling schemes which don't consider about frequent update in dynamic XML documents need relabeling process to reflect the changed label information whenever the tree of XML document is update. There is disadvantage of considerable expenses in the dynamic XML document which can occurs frequent update. To solve this problem, we suggest prime number labeling scheme that doesn't need relabeling process. However the prime number labeling scheme does not consider that it needs to update the sibling order of nodes in the tree of XML document. This update process needs much costs because the most of the tree of XML document has to be researched and rewritten. In this paper, we propose the prime number labeling scheme with sibling order value that can maintain the sibling order without researching or rewriting the tree of XML documents.

▸ Keyword : 동적 XML 문서(Dynamic XML document), 레이블링(Labeling), 프라임넘버링(Prime Numbering)

• 제1저자 : 이강우, 교신저자 : 이준동
• 접수일 : 2007. 9.18, 심사일 : 2007. 10.8, 심사완료일 : 2007. 10.20.
* 한라대학교 컴퓨터공학과, ** 강릉대학교 컴퓨터공학부

Ⅰ. 서 론

XML(eXtensible Markup Language)은 인터넷상에서 이루어지는 자료 표현이나 정보교환에 있어서 실질적인 표준으로 사용되어 왔다. 웹상의 XML 문서 저장소(repository) 수가 급속도로 증가하면서 XML 문서에 대한 효율적인 질의와 저장이 가능한 시스템의 개발이 요구되고 있다. 이러한 요구를 충족시키기 위해 W3C에서는 XPath와 XQuery와 같은 질의어를 XML 문서를 효과적으로 처리하기 위한 언어로 추천하였다[1]. XML 문서에 대한 질의(query)는 데이터의 경로(path)를 표현하는 XPath와 XQuery 등을 사용하며, 실제 질의처리를 위해서는 XML 트리를 탐색하여 원하는 데이터를 찾게 된다[2,3]. 예를 들어, "Resource / Artist / Painter / Michelangelo"라는 질의를 처리하기 위해서는 대상이 되는 XML 문서 트리를 탐색하여 "Resource"와 "Artist"를 조상(ancestor)으로 하고, "Painter"를 부모로 하는 "Michelangelo"에 해당하는 패턴(pattern)을 찾아야 한다. 이러한 패턴을 XML 문서 트리에서 찾기 위해서는 전체 트리를 모두 탐색하여 각각의 노드 사이의 관계가 해당 질의에 맞는지를 찾는 과정이 필요하다. 그러나 XML 문서트리에서 질의에 해당하는 패턴을 찾는 검색 비용은 상당히 크다. 따라서 XML 문서 트리의 탐색 공간(search space)을 줄여 탐색 시간을 줄이는 기법과 부모-자식 관계, 조상-후손 관계, 형제 관계를 효율적으로 결정할 수 있는 기법이 필요하다[4].

노드와 노드 사이의 관계를 결정하기 위해서 XML 문서 트리의 모든 노드들에 조상-후손 관계를 나타내는 레이블을 추가한다[7]. 즉 효율적인 질의 처리를 위해 XML 문서 트리에 XML 문서 트리의 각 노드에 대한 레이블(label)과 XML 문서 트리에 대한 구조적 색인(index) 정보를 추가로 포함하고 있다. 이러한 레이블과 색인정보는 XML 문서 트리의 각 노드에 부여되어 임의의 두 노드 간의 관계를 빠르게 결정할 수 있도록 한다. 이와 같은 XML 문서트리의 각 노드에 레이블을 부여하고 이 레이블 정보를 이용하여 노드 사이의 관계를 표현하고 결정할 수 있는 레이블링 기법(labeling scheme)이 연구되고 있다[1,4,10,11]. 레이블링 기법은 노드들 간의 조상-후손 관계 및 형제(sibling)들 간의 순서를 결정할 수 있도록 하는 색인을 위한 인코딩(encoding)이라고 할 수 있다.

그러나 동적인 XML 문서에서 빈번히 발생하는 갱신에 대한 고려를 하지 않는 기존의 레이블링 기법들은 XML 문서 트리의 갱신이 일어날 때 마다 변화된 레이블 정보를 반영하기 위해서 전체 XML 트리를 재탐색하여 전체 노드의 레이블을 다시 계산하는 리레이블링(relabeling)과정이 필요하다. 동적인 XML 문서의 변경된 정보를 반영할 때 다른 노드의 레이블에 영향을 주지 않는 레이블링 기법으로 프라임 넘버 레이블링 기법(prime number labeling scheme)이 있다[4]. 이 기법은 XML 문서 트리에서 각 노드의 레이블을 조상-후손 관계를 나타내는 값으로 전위우선순위(preorder traversal) 순으로 할당된 소수(prime number) 값과 부모레이블 값을 곱한 결과로 할당함으로써 문서의 갱신 시 다른 노드의 레이블에 영향을 주지 않는다. 그러나 문서가 갱신될 때 XML 문서 트리에서 형제들의 순서를 고려하는 갱신 비용은 XML 문서 트리의 많은 부분을 재탐색과 재기록에 대한 많은 비용을 필요로 한다. 따라서 본 논문에서는 XML 문서 트리의 재탐색과 재기록이 필요 없이 형제순서를 유지할 수 있는 형제순서 값을 갖는 프라임 넘버 레이블링 기법을 제안한다.

Ⅱ. 관련 연구

XML 문서 데이터베이스에서의 질의처리에는 부모-자식 간의 관계를 결정하는 것뿐만 아니라 조상-후손 관계를 결정하는 정보를 포함하여야 한다. 이는 XML 문서에 대한 DTD(Data Type Definition)가 존재하더라도 문서의 정확한 구조를 사용자가 알지 못하기 때문이다. 탐색을 기반으로 하는 질의 처리에서는 대응되는 후손 노드를 찾을 때까지 조상 노드를 유지하고 있어야 한다. 이는 질의처리 시간이 상당히 길게 할뿐만 아니라 많은 저장 공간을 필요로 한다. 따라서 조상-후손 관계를 결정하는데 있어서 발생하는 문제를 해결하기 위해 XML 문서 트리의 각 노드에 자신의 위치를 나타내는 레이블을 사용하는 방법이 제안되어 왔다. 이 방법들은 조상-후손 관계와 부모-자식 관계를 쉽게 확인할 수 있으며, 이러한 레이블링 방식의 대표적인 종류로는 범위기반 레이블링 기법(containment labeling scheme), 전위기반 레이블링 기법(prefix labeling scheme), 프라임 넘버 기반 레이블링 기법(prime number labeling scheme) 등이 있다[1]. 이와 같은 레이블링 기법은 다음의 특징을 갖는다[4].

1) 결정적(deterministic) : 두 노드 간의 관계는 그들의 레이블을 검사함으로써 간단하고 빨리 결정될 수 있어야 한다.
2) 동적(dynamic) : XML 파일의 수정은 XML 문서 트리에 있는 노드의 리레이블링(rela- beling)을 요구하지 않아야 한다.

3) 간결(compact) : 레이블의 크기는 메인 메모리에 적당한 최소한의 크기여야 한다.
4) 유연(flexible) : 모든 종류의 XQuery와 XPath 기능을 지원해야 한다.

2.1 범위기반 레이블링

〔6〕등은 각 노드에 세 개의 값(start, end, level)을 할당하는 넘버링 기법을 사용한다. 두개의 노드 u와 v 사이에서 u.start 〈 v.start 그리고 u.end 〈 v.end 관계가 성립하면 u는 v의 조상이다. 즉 u의 구간이 v의 구간을 포함(containment) 한다고 할 수 있다. u가 v의 조상(ancestor)이고 v.level - u.level = 1 이면, 노드 u는 노드 v의 부모이다. 예를 들어 〈그림 1〉에서 노드 A와 B의 관계를 살펴보면 A.start(4) 〈 B.start(5) 그리고 A.end(6) 〈 B.end(9) 이므로 노드 A는 노드 B의 조상이고, B.level(3) - A.level(2) = 1 이므로 노드 A가 노드 B의 부모가 된다.

범위기반 기법은 신속히 부모-조상 관계를 결정할 수 있으나 XML 트리에 노드가 삽입될 경우 트리에 있는 모든 노드의 리레이블링 과정을 야기한다. 또한 구간의 크기를 결정하는 것도 쉽지 않다. 작은 구간은 잦은 리레이블링이 일어나며, 큰 구간을 설정하는 것은 저장 공간의 낭비를 야기한다.

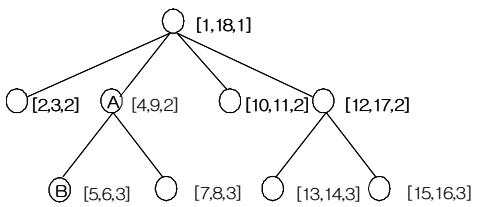

그림 1. 범위기반 레이블링
Fig 1. Containment labeling

2.2 전위기반 레이블링

전위기반 레이블링은 노드의 레이블을 자신의 레이블에 구분자(delimiter)와 부모의 레이블을 전위(prefix)에 붙여 생성한다. 두 노드 u와 v사이에서, label(u)가 label(v)의 전위에 있다면 노드 u는 노드 v의 조상이다. label(v)의 왼쪽부분인 label(u)를 제거했을 때, label(v)에 전위 부분이 남아있지 않다면, 노드 u가 노드 v의 부모이다. 주요 전위기법에는 정수를 사용하는 방법(integer based scheme)과 바이너리 스트링을 사용하는 방법(binary string based scheme) 두 가지가 있다.

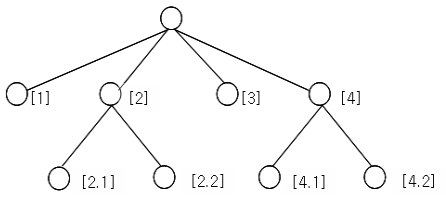

그림 2. 전위기반 레이블링
Fig 2. Prefix labeling

정수를 기본으로 하는 전위기법을 사용한 〔1,4,6〕은 노드의 n번째 자식 노드에 정수 n을 레이블로 부여한다. 그리고 n은 자식노드의 완전한 레이블을 구성하기 위해 구분자와 함께 부모 레이블의 뒤에 붙여서 자식노드의 레이블을 구성한다. 예를 들면 〈그림 2〉와 같이 레이블이 생성된다.

유사한 기법으로 Cohen 등은 노드의 레이블로 Binary String을 부여한다. Binary String의 각 문자는 1 비트로 저장되고, 트리의 root는 빈 스트링(empty string)을 갖는다. root의 첫 번째 자식은 '0'으로 레이블 되고, 두 번째 자식은 '10', 세 번째 자식은 '110', 네 번째 자식은 '1110'으로 각각 레이블이 부여된다. 구간기법과 비교하여 전위기법은 삽입된 노드의 형제들의 후손에 대해서만 리레이블링 과정이 필요하며, 수정과정에서 좀더 동적인 방법이라 할 수 있으나 리레이블링 과정을 회피할 수는 없다.

2.3 프라임 넘버 기반 레이블링

〔4〕 등은 프라임 넘버(prime number)를 XML 트리의 레이블로 부여하는 방법을 제안하였다. 기본적인 프라임 넘버 기법은 XML 트리의 노드를 프라임 넘버를 사용해서 레이블링을 한다. 각 노드의 레이블은 두수의 곱으로 이루어지는데, 첫 번째 값은 이 노드가 가지는 고유한 레이블인 자신의 self-label이 되고, 두 번째 값은 부모의 레이블인 parent_label이 된다. 이 방법은 프라임 넘버를 재활용할 수 없고, 레이블링이 진행될수록 프라임 넘버가 점차 큰 프라임 넘버를 가지게 될 가능성이 높은 단점을 가지고 있다.

III. 효율적인 형제순서 정보유지 기법

3.1 프라임 넘버 레이블링의 문제점

본 장에서는 형제순서 값을 갖는 프라임 넘버 레이블링 기법(prime number labeling scheme)의 조상-후손 관계를 결정하는 방법을 살펴보고, 갱신이 일어날 경우의 형제간의 순서 결정법과 이 기법의 문제점을 살펴본다.

3.1.1 부모-자식 관계 결정

기본적인 프라임 넘버 레이블링 기법은 XML 문서 트리의 노드에 프라임 넘버를 사용하여 레이블을 부여 한다. 프라임 넘버의 특성으로 1과 자기 자신으로만 나누었을 때 나머지가 0인 성질을 이용하여 XML 트리의 각 노드에 프라임 넘버 레이블을 부여한다. 먼저 XML 문서 트리의 루트 노드에 1을 레이블로 부여하고, 이하 서브 트리를 구성하는 각 노드에 프라임 넘버를 트리의 전위순회 순서대로 부여한다. 각 노드의 레이블은 두 수의 곱으로 이루어지는데, 첫 번째 값은 자신의 셀프 레이블(self-label)이 되고, 셀프 레이블은 프라임 넘버 리스트({2, 3, 5, 7, 11, 13, ...}) 값을 트리의 전위순회 순서로 루트 노드를 제외한 각 노드에 차례로 부여하며, 이는 노드가 가지는 고유한 레이블이 된다. 그리고 두 번째 값은 부모의 레이블이 된다. 기존의 레이블링 기법과 같이 쉽게 두 노드 사이의 조상-후손 관계를 확인할 수 있다. 이 방식의 경우 한 노드가 다른 노드의 조상임을 확인하기 위해서는 이 노드의 레이블로 다른 노드의 레이블을 나누었을 때 나머지가 0인지 아닌지를 확인하면 된다. 즉, 나머지가 0이 되면 조상이라는 것을 확인할 수 있다. 예를 들어 〈그림 3〉에서 ⓖ 노드의 경우 레이블로 [91]이 부여되는데, 이는 부모 노드(ⓒ)의 레이블 [7]과 자신(ⓖ)의 셀프 레이블 (13)의 곱으로 결정된다. 또한 조상-후손 관계 확인은 자손(ⓖ) 레이블을 부모(ⓒ) 레이블로 나누면 즉, [91]을 [7]로 나누어 나머지가 0이므로 ⓒ와 ⓖ 노드는 조상-후손 관계임을 알 수 있다.

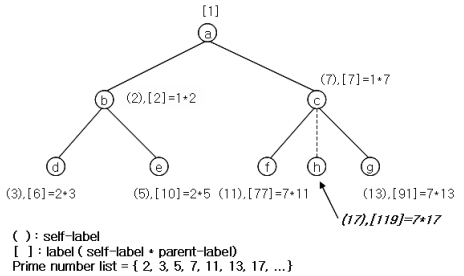

그림 3. 프라임 넘버 레이블링
Fig 3. Prime number labeling

이와 같은 방식으로 레이블이 각 노드에 부여되면 새로운 노드의 삽입과 삭제 시에도 다른 노드의 레이블 값에 영향을 주지 않고 새로운 레이블을 부여할 수 있다. 예를 들어 〈그림 3〉에서 ⓕ 노드와 ⓖ 노드 사이에 새로운 노드 ⓗ가 삽입되는 경우를 살펴보면 다음과 같다. 노드 ⓗ의 셀프 레이블은 17(지금까지 부여된 프라임 넘버 다음 번째 프라임 넘버)이 되고, 레이블은 셀프 레이블 [17]과 부모 레이블 [7]의 곱인 [119]가 된다.

3.1.2 순서화된 트리에서의 레이블링

XML 문서에서 엘리먼트는 고유의 성질에 의해 순서를 갖는다. 세 개의 장으로 이루어진 책에 대한 XML 문서가 있다고 가정해보자. 그러면 이 XML 문서는 세 개의 장(chapter) 태그를 갖게 된다. 순서화된 XML 문서 트리의 예를 보면 〈그림 4〉와 같다.

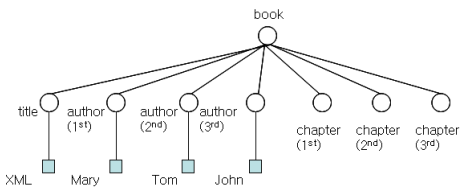

그림 4. 순서화된 문서 트리
Fig 4. An ordered XML tree

XML 문서에서 순서 유지 문제는 순서에 민감한 질의(order-sensitive query) 처리에서는 매우 중요한 문제이다. 예를 들어 질의 "book[author[2] = 'John']"는 제2저자가 'John'인 책을 검색한다. XML 문서에서 순서를 유지해야 하는 필요성은 많은 XML 레이블링 기법에서 문제를 일으킨다. 예를 들어 〈그림 4〉에 나타난 XML 문서 트리에 제2저자를 새로운 저자로 삽입할 경우, 'Tom'과 'John'을 세 번째, 네 번째 형제(sibling) 관계인 세 번째와 네 번째 저자로 밀어내야 한다. 이와 같이 XML 문서 트리에서 새로운 노드의 삽입 시 새로운 노드에 대한 레이블 부여 방법에 따라 새로운 노드의 레이블은 항상 기존의 노드보다 큰 셀프 레이블를 부여 받게 된다. 따라서 새로운 노드가 형제들 사이에 삽입된다면 형제 노드들 중에 새로 삽입된 노드가 가장 큰 레이블을 갖게 되고, 이는 노드에 부여된 레이블 값만으로는 형제 순서를 결정할 수 없게 된다.

이를 해결하기 위해 〈그림 5〉와 같이 형제들 간의 순서를 결정할 수 있는 형제 순서 정보를 나타내는 SC 테이블(Simultaneous Congruence)을 유지한다. SC 테이블은 self-table에서 보통 5개의 그룹으로 이루어진 프라임 넘버 중 가장 큰 값을 나타내는 Max prime 값과 중국인나머지 정리(Chinese remainder theorem)를 적용한 SC 값을 부여한다. SC 테이블을 이용하여 형제간의 순서 결정을 하는 방법은 해당 노드에 부여된 'self-label' 값으로 자신이 속한 그룹의 SC 값을 나누어 그 나머지인 'order number'

값으로 형제간의 순서를 결정한다.

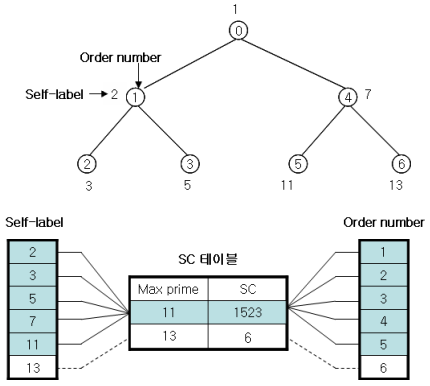

그림 5. SC 테이블을 갖는 프라임 넘버 레이블링 기법
Fig 5. Prime number labeling scheme with SC table

이러한 형제간의 순서를 결정하기 위한 추가적인 정보 유지비용은 동적인 XML 문서인 경우 XML 트리를 재탐색하여 새로운 값을 할당해야 하는 리레이블링(relabeling)을 발생하게 하고, 이 비용이 상당히 크다는 점이다. 또한 새로운 노드의 삽입 시 SC 값을 다시 계산해야 하는 비용과 해당 그룹에 속하는 노드들과 SC 테이블간의 연결정보를 재계산(recalculating)해야 하는 비용이 필요하다.

3.2 형제순서 값을 갖는 프라임 넘버 레이블링 기법

본 논문에서는 XML 문서 트리에서 새로운 노드의 삽입에 따른 리레이블링(relabeling) 또는 SC 테이블의 유지비용을 줄이기 위해 SC 테이블을 사용하지 않고 형제순서(sibling order) 값을 노드의 레이블과 함께 부여하는 형제순서 값을 갖는 프라임 넘버링 기법을 제안한다.

3.2.1 사전식 순서

사전식 순서(lexicographical order)는 단어를 순서화하는 방법으로 이러한 예는 전화번호부나 영어사전에서 흔히 볼 수 있다[10]. 바이너리 스트링에 대한 사전식 순서는 다음과 같이 정의되고, 사전식으로 순서화된 두 바이너리 스트링 사이에 새로운 값이 삽입될 경우는 〔예제〕에 나타나 있다. 이와 같은 사전식 순서를 갖는 바이너리 스트링을 형제 노드들 간의 순서를 유지하기 위해 형제순서 값으로 사용한다.

〔정의〕 사전식 순서 : $\ll$

주어진 두 바이너리 스트링 S_L과 S_R에서 만약 두 스트링 정확하게 같다면 S_L과 S_R의 사전식 순서는 동일하다. S_L과 S_R이 다음의 두 가지 경우에 사전식 순서로 S_L이 S_R보다 작다($S_L \ll S_R$)라고 한다.

- S_L과 S_R의 사전식 순서의 비교는 한 비트씩 왼쪽에서 오른쪽으로 진행한다. 만약 S_L의 현재 비트가 0이고 S_R의 현재 비트가 1이면 $S_L \ll S_R$ 이고 비교 중지한다.
- S_L은 S_R의 전위(prefix)이다.

〔예제〕 주어진 두 스트링 "0011"과 "01"의 사전식 순서는 "0011" $\ll$ "01"이다. 왜냐하면 "0011"의 두 번째 비트는 "0"인 반면 "01"의 두 번째 비트는 "1"이기 때문이다. 또 다른 예로 "01"이 "0101"의 전위이기 때문에 "01" $\ll$ "0101"이다.

3.2.2 형제순서 값 결정

형제순서(Sibling Order; SO) 값의 부여 방법은 새로운 노드가 XML문서 트리에 삽입될 때 삽입되는 위치의 바로 이전 노드의 SO 값을 이용하여 새로운 노드의 SO 값을 결정한다. SO 값은 새로운 노드가 삽입되는 위치에 따라 세 가지 경우가 있다.

〔Case 1〕 첫 번째 노드 앞에 위치하는 경우
〔Case 2〕 두 노드 사이에 위치하는 경우
〔Case 3〕 마지막 노드 뒤에 위치하는 경우

따라서 삽입되는 노드의 SO 값 결정은 다음과 같이 세 가지 경우로 나뉘어 결정한다.

〔Case 1〕 첫 번째 노드 앞에 삽입되는 경우는 삽입되는 노드의 SO 값은 첫 번째 노드의 SO 값의 마지막 비트 값을 "0"으로 바꾸고 여기에 "1"을 마지막에 추가한다.

〔Case 2〕 두 노드 사이에 삽입되는 경우는 왼쪽 노드의 SO 값(SO_L)이 오른쪽 노드의 SO 값(SO_R)보다 크기가 크거나 같을 경우는 삽입되는 노드의 SO 값(SO_N)은 SO_L의 마지막에 "1"을 추가하고, SO_L이 SO_R보다 크기가 작은 경우는 삽입되는 노드의 SO_N은 SO_R의 마지막 비트 "1"을 "01"로 교체한다.

〔Case 3〕 마지막 노드 뒤에 삽입되는 경우의 SO_N은 SO_L의 마지막에 "1"을 추가한다.

새로운 노드의 SO_N을 결정하는 알고리즘은 다음과 같다.

```
Function DetermineSiblingOrder_Pre(SOR)
// Case 1
    SOR : 첫 번째 노드의 SO 값
    SON : 삽입되는 노드의 SO 값
```

```
begin
   SON = replace(lastbit(SOR), "01");
 //replace() : 교체함수
 //lastbit() : 마지막 비트 추출 함수
end
End Function

Function DetermineSiblingOrder_In(SOL, SOR)
// Case 2
   SOL : 왼쪽 노드의 SO 값
   SOR : 오른쪽 노드의 SO 값
   SON : 삽입되는 노드의 SO 값
begin
   if size(SOL) ≥ size(SOR) then
      // size() : SO 값의 비트수를 구하는 함수
      SON = SOL & "1";
      // &는 concatenation 연산자
   else
      SON = replace(lastbit(SOR), "01");
   end if
end
End Function

Function DetermineSiblingOrder_Post(SOL)
// Case 3
   SOL : 마지막 노드의 SO 값
begin
   SON = SOL & "1";
end
End Function
```

위의 알고리즘을 기반으로 하여 세 가지 경우의 삽입에 대하여 예를 〈그림 6〉과 같은 XML 문서 트리에 새로운 노드 ⓐ, ⓑ, ⓒ가 삽입되는 경우의 SO 값 결정과정을 살펴보면 다음과 같다.

〔Case ⓐ〕 SOR이 "01"이므로 마지막 비트 "1"을 "01"로 교체하여 SON은 "001" 된다.

〔Case ⓑ〕 SOL은 "01" 이고 SOR은 "0101" 이고 size(SOL) 〈 size(SOR) 이므로 SON은 "01001" 이 된다.

〔Case ⓒ〕 SOL이 "0101" 이므로 SON은 "01011"이 된다.

새로운 노드의 삽입 후 SO 값의 사전식 순서를 보면 001≪01≪0101 과 01≪01001≪0101≪01011 임을 알 수 있다. 이와 같이 결정된 형제순서 값을 삽입되는 노드의 레이블에 추가함으로써 새로 삽입되는 노드의 레이블에 다음번 노드의 레이블보다 사전식 순서로 작은 값을 갖게 되어 형제간의 순서를 알 수 있다.

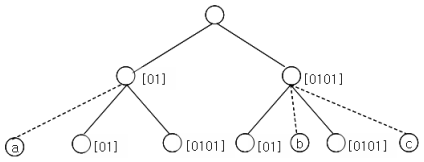

그림 6. 새로운 노드의 삽입 시 SO 값의 결정
Fig 6. Determine of SO value as insert of new node

3.2.3 형제순서 값을 갖는 프라임 넘버링 기법

XML문서 트리에서 부모-자식 간의 관계를 결정하는 방법은 프라임 넘버링 기법과 동일하며, 레이블의 내용은 형제순서를 결정하기 위해 SO 값을 기존 레이블에 추가하여 부여한다. 따라서 레이블의 구성은 〔label, SO〕로 구성된다.

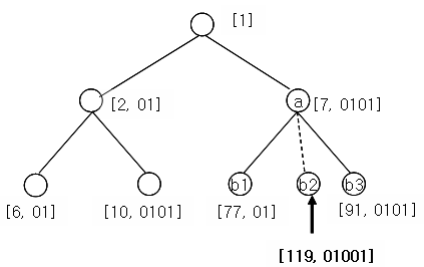

그림 7. 형제순서 값을 갖는 프라임 넘버링 기법
Fig 7. Prime numbering scheme with sibling-order value

〈그림 7〉에서 b2 노드를 삽입할 경우 레이블은 〔119, 01001〕 이다. 여기에서 조상과 후손의 관계 결정은 부모 노드 ⓐ의 레이블인 '7'을 가지고 노드 b2의 레이블 '119'를 나누면 나머지가 '0'이 나오므로 ⓐ와 b2의 관계는 부모-자식 관계임을 알 수 있고, b1, b2, b3에 대한 형제순서는 01≪01001≪0101이 됨을 알 수 있다.

레이블 값은 조상-후손 간의 관계결정에 사용되고, SO 값은 형제간의 순서결정에 만 사용된다. 이들 레이블을 이용한 XML 문서트리에서의 새로운 노드의 삽입과 삭제 알고리즘을 보면 다음과 같다.

```
/*
struct {
      int label; // 레이블
```

```
        char so[20]; // 형제순서 값
}N; // 노드의 레이블
Prev_Sibling(N):노드N의 이전 형제노드를 반환
Next_Sibling(N):노드N의 다음 형제노드를 반환
get_Label( ) : 새로운 레이블 값을 반환
*/
Function Insertion(N)
 begin
  N.label = getLabel();
  if (Prev_Sibling(N) is not exist) then
     next_node = Next_Sibling(N);
     N.SO=DetermineSiblingOrder_Pre(next_node.SO);
  else if (Next_Sibling(N) is not exist) then
     prev_node = Prev_Sibling(N);
     N.SO =DetermineSiblingOrder_Post(prev_node.so);
  else
     prev_node = Prev_Sibling(N);
     next_node = Next_Sibling(N);
     N.SO =DetermineSiblingOrder_In(prev_node.so,
                                    next_node.so);
  end if
  end if
end
End Function
```

삭제 시에는 해당노드를 기존의 방법처럼 순서정보를 계산하거나 제조정해야 하는 과정 없이 삭제만 하면 된다.

Ⅳ. 실험

실험 환경은 cpu가 Intel Pentium 2.0GHz, 메모리 2Gbytes, 운영체제는 Windows XP에서 수행하였다. 구현 언어는 Java(JDK1.6.0)를 사용하였다. XML Parser로는 SAX 2.0.2(SAX2 r3)를 사용하였으며 DBMS는 Access를 사용하였다. 실험 데이터는 Shakespeare 2.0으로 Shakespeare 희곡들의 모음을 XML로 기술한 데이터를 웹상에 공개한 데이터이다[13]. 실험 데이터는 스택 연산을 이용한 SAX parser를 이용하여 XML 문서를 데이터베이스에 레이블이 부여된 형태로 저장하였다.

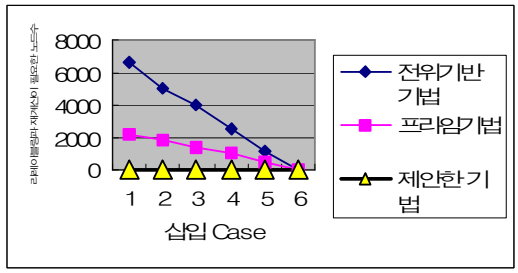

그림 8. 리레이블링과 재계산이 필요한 노드의 수
Fig 8. Number of nodes or values for relabeling or recalculation

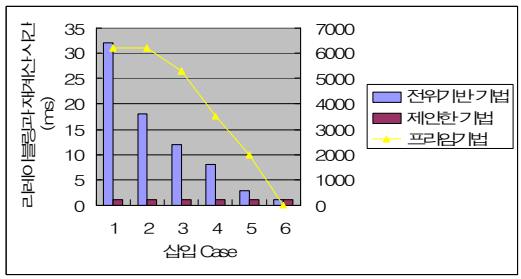

그림 9. 리레이블링과 재계산에 필요한 시간
Fig 9. Processing time for relabeling or recalculation

Shakespeare의 희곡(D8) XML 데이터는 엘리먼트들의 순서가 중요하다. D8에 있는 Hamlet은 5막으로 구성되어 있어 6가지 Case(Act[1] 앞에 새로운 엘리먼트의 집합을 삽입하는 Case, Act[1]과 Act[2] 사이에 삽입하는 Case, ..., Act[4]와 Act[5] 사이에 삽입하는 Case, Act[5] 뒤에 삽입하는 Case)로 실험을 수행하였다. 〈그림 8〉는 기존 기법과 제안한 기법에 적용하였을 때 리레이블링을 수행하는 노드의 수를 나타내었다. 전위기반기법과 프라임 기법에서는 6가지 Case인 모든 Case에 리레이블을 수행해야 하는 노드의 수는 동일하다. 6,636개의 노드를 갖는 Hamlet XML 파일에서 전위기반기법과 프라임 기법은 첫 번째 Case인 경우 6,595개의 노드에 대해 리레이블 과정이 수행되었다. 프라임기법에 대해서는 SC 값의 재계산이 필요한 노드의 수를 나타내었다. SC 값이 3개의 레이블을 그룹화 하여 SC 값을 부여 실험하였다. 여기서 3개의 레이블을 그룹화하여 실험한 것은 4개 이상의 레이블을 그룹화하면 Java에서 64bit를 사용하여 저장할 수 없는 큰 수가 발생하기 때문이다. 3개의 레이블을 그룹화하면 프라임 기법에서 재계산이 필요한 노드의 수는 전위기반기법에서 리레이블이 필요한 노드의 수에 1/3이 된다. 〈그림 9〉는 리레이블과 SC 값의 재계산에 요구되는 시간을 보여

준다. 프라임기법(오른쪽 축)이 전위기반기법(왼쪽 축)보다 상당히 많은 시간을 필요로 하는 것을 볼 수 있다. 반면 6가지 모든 case에 대해 본 논문에서 제안한 기법은 리레이블이나 재계산을 필요로 하는 노드는 없다.

V. 결론

순서에 민감한 노드를 XML 문서 트리에 삽입하는 경우 기존의 레이블링 기법에서는 문서의 순서를 유지하기 위해 레이블의 리레이블링 또는 재계산을 필요로 한다. 이와 같은 문제를 해결하기 위해 본 논문에서는 XML 문서 트리에 순서에 민감한 노드(order-sensitive node)를 삽입할 때 노드 레이블의 리레이블이나 재계산이 필요 없는 형제 순서 값을 갖는 프라임 넘버링 기법을 제안하였다. 즉 XML 문서 트리에서 새로운 노드의 삽입에 따른 리레이블링 또는 SC 테이블의 유지비용을 줄이기 위해 SC 테이블을 사용하지 않고 형제순서 값을 노드의 레이블과 함께 부여하는 형제순서 값을 이용하여 프라임 넘버링 기법의 문제점을 개선하였다. 또한 기존의 기법들과 제안한 기법의 실험을 통해 제안한 기법에 대한 성능의 우수성을 입증하였다.

향후 연구로서는 레이블의 크기를 어떻게 줄일 수 있는가와 제안한 기법을 통하여 기존의 질의처리 기법의 성능을 향상시키는 방법 등이 있다.

참고문헌

〔1〕 V. Christophides, D. Plexousakis, M. Scholl and S. Tourtounis, On Labeling Schemes for the Semantic Web, In WWW, 2003.

〔2〕 W3C Working Draft. XML Path Language (XPath)2.0, November 2002.

〔3〕 D. Chamberlin et. al, SQuery 1.0 : An XML Query Language, W3C Working Draft, 2001.

〔4〕 X. Wu, M. L. lee, and W. Hsu. A Prime Number Labeling Scheme for Dynamic Ordered XML Trees. In Proc. of ICDE, pp66-78, 2004.

〔5〕 S. C. Haw, G. S. V. Radha Krishma Rao. Query Optimization Techniques for XML Databases. IJIT Vol.2 No. 2, pp97-104, 2005.

〔6〕 J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, J. Naughton, Relational Databases for Querying XML Documents: Limitations and Opportunities, Proc. of the 25th VLDB Conf., 1999.

〔7〕 P. Buneman, S. Davidson, W. Fan, C. Hara, W. C. Tan, Keys for XML, In WWW, 2001.

〔8〕 W. M. Shui, F. Lam, D. K. Fisher, R. K. Wong, Querying and Maintaining Ordered XML Data using Relational Databases, 16th Australasian Database Conf., 2005.

〔9〕 Q. Li, B. Moon, Indexing and Querying XML Data for Regular Path Expressions, Proc. of the 27th VLDB Conf., Roma, Italy, 2001.

〔10〕 Changqing Li, Tok Wang Ling, Min Hu, Efficient Processing of Updates in Dynamic XML Data, http: \\comp.nus.edu.sg.

〔11〕 V. Christophides, G. Karvounarakis, D. Plexousakis, Optimizing Semantic Web Queries using Labeling Schemes, proceedings of WWW2003.

〔12〕 Arthur M. Keller, Jeffrey D. Ullman, A Version Numbering Scheme with a Useful Lexicographical Order, http: \\cs.stanford.edu.

〔13〕 http://www.oasis-open.org/cover/bosakShakespeare200.htm

저 자 소 개

이 강 우

1997년 2월 : 홍익대학교 전자계산학과 이학박사

1994년 ~ 2002년 : 서남대학교 정보통신공학부 교수

2002년 ~ 현재 : 한라대학교 컴퓨터공학과 교수

관심분야 : 데이터베이스, 유비쿼터스 컴퓨팅

이 준 동

2001년 2월 : 홍익대학교 전자계산학과 이학박사

1997년 ~ 2006년: 원주대학 교수

2007년 ~ 현재: 강릉대학교 교수

관심분야 : 시스템소프트웨어, 유비쿼터스, 실시간처리, 데이터베이스