DOI: 10.3745/KIPSTB.2008.15-B.4.355

구조화 문서 검색을 위한 다단계 역색인 기법

김 종 익†

요 약

XML로 대표되는 구조화된 문서의 검색을 위해서는 구조 조인 기법이 많이 사용되며 구조 조인 기법을 사용하기 위해서는 구조 조인에 참여하는 엘리먼트들을 추출하는 과정이 선행되어야 한다. 이 과정을 위해서 일반적으로 동일한 태그 값을 가지는 엘리먼트들을 리스트 형태로 추출해 주는 역색인을 사용한다. 하지만 이러한 기존의 기법은 경로 질의 내의 부모-자식 관계나 조상-후손 관계를 비교적 비용이 비싼 구조 조인으로 모두 처리해야 하기 때문에 경로의 길이가 길어질수록 질의 처리 비용이 크게 증가하는 단점을 가지고 있다.

본 논문에서는 기존의 역색인과는 달리 엘리먼트 추출과정에서 부모-자식 관계에 있는 엘리먼트들을 처리할 수 있는 단계별 역색인을 제안한다. 본 논문에서 제안하는 단계별 역색인은 경로 질의 내의 부모-자식 관계를 가지는 엘리먼트 쌍(pair)들의 리스트를 추출해 준다. 또한 단계별 역색인으로부터 추출된 엘리먼트 쌍들의 리스트를 처리하기 위해 기존의 구조 조인과는 다른 변형된 구조 조인 기법을 제안하며 실험을 통해 제안된 기법이 기존의 기법보다 2배에서 4배 가량의 성능향상이 있는 것을 확인하였다.

키워드 : XML, 구조 조인, 다단계 역 색인

A Multi-level Inverted Index Technique for Structural Document Search

Jongik Kim†

ABSTRACT

In general, we can use an inverted index for retrieving element lists from structured documents. An inverted index can retrieve a list of elements that have the same tag name. In this approach, however, the cost of query processing is linear to the length of a path query because all the structural relationships (parent-child and ancestor-descendant) should be resolved by structural join operations.

In this paper, we propose an inverted index technique and a novel structural join technique for accelerating XML path query evaluation. Our inverted index can retrieve element lists for path segments in a parent-child relationship. Our structural join technique can handle lists of element pairs while the existing techniques handle lists of elements. We show through experiments that these two proposed techniques are integrated to accelerate evaluation of XML path queries.

Key Words : XML, structural join, multi-level inverted index

1. 서 론

XML[1]이 인터넷 문서 교환의 표준으로 채택 되면서 XML을 효율적으로 관리하기 위한 연구가 계속되고 있다. XML 데이터의 주요한 특징은 데이터의 구조가 매우 유연하며 데이터 내에 데이터의 구조에 대한 정보를 포함하고 있다는 것이다. 이러한 특징으로 인해 다양한 종류의 구조화된 문서들이 XML을 통해 표현될 수 있고 현재 많은 응용 분야에서 XML을 이용해 정보를 표현하고 있다.

XML 데이터는 일반적으로 트리(tree)로 표현되며, XML 데이터 내의 각 엘리먼트와 속성(attribute)들은 트리의 각 노드(node)로 대응되며 XML 데이터 내의 엘리먼트와 속성들의 포함관계는 트리 내의 노드들을 이어주는 에지(edge)로 대응된다. 본 논문에서는 트리로 표현된 XML 데이터를 데이터 트리라고 부른다. 기본적으로 XML에 대한 질의는 XML 데이터 내의 엘리먼트들의 포함관계에 대한 나열(sequence)로 표현되며 이는 데이터 트리 내의 경로에 해당하게 된다. 본 논문에서는 XML 엘리먼트들의 직접적인 포함관계를 부모-자식 관계라 하며 /(slash)를 이용해 표현하며 간접적인 포함관계를 조상-후손관계라 하며 //(double slash)를 이용해 표현한다. 예를 들어, restaurant[name=Italian]//manager 라는 질의를 생각해 보자. 이 질의는 이름이 Italian인 식당의 주인을 모두 찾아내는 질의로 restaurant/[name=Italian] 과 restaurant//manager라는 두 가지의 포함관계 나열로 이루어진다. 본 질의는 XML 데이터 내에 있는 다음과 같은 패턴을 가지는 식당 엘리먼트를 모두 찾

※ 이 논문은 2007년도 전북대학교 지원 신임교수 연구비에 의해 연구되었음
† 정 회 원 : 전북대학교 전자정보공학부 전임강사
논문접수 : 2008년 2월 14일
수 정 일 : 2008년 3월 3일
심사완료 : 2008년 3월 7일

아준다.

<restaurant>
<name> Italian </name>
… (임의의 엘리먼트들)
<manager> … </manager>
…(임의의 엘리먼트들)
</restaurant>

XML에 대한 경로 질의는 일련의 포함 관계를 계산함으로써 처리 될 수 있으며 이진의 포함관계는 경로 질의 처리의 기본 연산이 된다. 이러한 이진의 포함관계를 계산하는 과정을 구조 조인이라 부른다. 위에 예로 든 질의는 Italian을 텍스트 값으로 가지는 name 엘리먼트들을 추출하여 restaurant/name의 포함관계를 계산하고, 이를 만족시키는 restaurant 엘리먼트들과 manager의 포함관계를 조사함으로써 결과를 얻어낼 수 있다. 즉, 2회의 구조 조인 연산으로 예제의 질의를 처리할 수 있다.

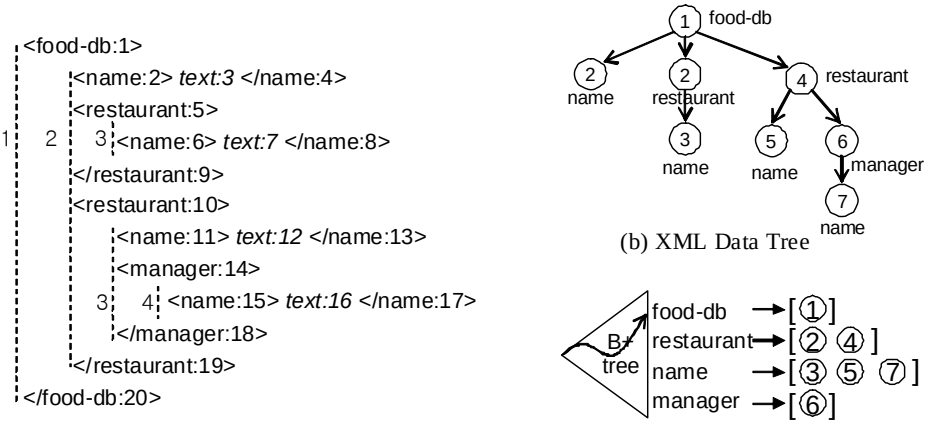
본 논문에서는 XML 경로 질의 처리의 기본이 되는 구조 조인의 성능 향상에 대해서 다룬다. 일반적으로 구조 조인은 다음과 같은 두 가지 단계를 통해 계산된다.

[엘리먼트 리스트 추출] 구조 조인의 첫 번째 단계에서는 특정 태그 이름을 가지는 모든 엘리먼트들을 하나의 리스트 형태로 추출한다. 구조 조인을 계산하기 위해서는 부모 또는 조상에 해당하는 엘리먼트 리스트와 자식 또는 후손에 해당하는 엘리먼트 리스트가 필요하다. 예를 들어, restaurant//manager라는 구조 조인을 계산하기 위해서는 restaurant라는 태그를 가지는 모든 엘리먼트들을 추출한 리스트와 manager라는 태그를 가지는 모든 엘리먼트들을 추출한 리스트가 필요하게 된다. 본 논문에서는 구조 조인에 사용되는 리스트 중 부모 또는 조상의 역할을 하는 엘리먼트들의 리스트를 AList라 부르며 자식 또는 후손의 역할을 하는 엘리먼트들의 리스트를 DList라 부른다. 엘리먼트 리스트는 역 색인(Inverted Index)이라 부르는 자료구조에 의해 쉽게 추출될 수 있다. 역 색인은 [태그 이름, 엘리먼트 리스트] 쌍을 B+ 트리 등에 저장한 색인 구조를 말하며, 색인의 키값으로 태그이름을 사용한다. (그림 1(c))는 (그림 1(a))와 같은 XML 데이터에 대한 역색인의 예를 보여준다.

[포함관계를 가지는 엘리먼트 쌍 추출] AList에 있는 각 엘리먼트 e1과 DList에 있는 각 엘리먼트 e2를 비교하여 e1이 e2를 포함하는 경우 (즉, 데이터 트리 내에서 e1 노드가 e2 노드의 부모 또는 조상인 경우) (e1, e2)를 결과에 포함한다. XML 데이터 내의 임의의 두 엘리먼트가 포함관계에 있는지 조사하기 위해 XML 데이터 내의 각 엘리먼트를 위치를 통해 표현한다. 각 엘리먼트는 (StartPos, EndPos, Level)의 세가지 값을 가지게 되며 StartPos는 XML 데이터의 시작부터 현재 엘리먼트의 시작 태그 전까지의 단어의 숫자를 값으로 가지며 EndPos는 데이터의 시작부터 현재 엘리먼트의 끝 태크 전까지의 단어의 숫자를 값으로 가진다. 또한 Level은 현재 엘리먼트의 포함된 깊이 값을 나타낸다. 예를 들어 (그림 1(a))의 첫 번째 restaurant엘리먼트는 (2, 6, 2)의 값으로 표현된다. 이와 같이 엘리먼트들을 위치로 표현하는 경우, e1의 StartPos가 e2의 StartPos보다 작고 e1의 EndPos가 e2의 EndPos보다 크면 엘리먼트 e1이 엘리먼트 e2를 포함한다는 것을 쉽게 알 수 있다. 또한 e1과 e2의 Level의 차이가 1인지를 확인하여 e1과 e2가 직접적인 포함관계(부모-자식 관계)를 가지는 지를 알 수 있다.

기존의 연구들은 구조 조인을 위한 위의 두 단계 중 두 번째 단계인 포함관계를 가지는 엘리먼트 쌍 추출에 대한 성능향상에 대해 집중하고 있으나, 엘리먼트 추출 단계에서 큰 사이즈를 가지는 엘리먼트 리스트가 추출될 경우 두 번째 단계의 계산 능력을 향상시켜도 전체 성능 향상에 한계가 있다는 문제점을 가지고 있다.

본 논문에서는 엘리먼트 리스트 추출 단계에서 하나의 태그를 이용해 역색인을 만드는 대신 부모-자식 관계로 이루



(a) XML Document

(b) XML Data Tree

(c) Inverted Index

(그림 1) XML 데이터 및 역 색인 예제

어진 일련의 태그 리스트(부분 경로)를 이용하여 역색인을 구성하되, 역색인을 유연하게 관리하기 위하여 부분 경로의 길이를 적응적(adaptive)으로 조정할 수 있는 방법을 제안한다. 또한 본 논문에서 제안하는 역색인에 의해 얻어지는 엘리먼트 리스트를 효과적으로 처리할 수 있는 구조 조인 기법에 대해 제안한다. 제안된 기법을 사용함으로써 역색인으로부터 얻어지는 엘리먼트 리스트의 크기를 줄이고 구조 조인의 횟수를 줄일 수 있어 전체적인 질의 처리 능력을 향상시킬 수 있다.

본 논문은 다음과 같은 기여도(contribution)를 가진다.

1. 구조 조인을 위한 엘리먼트 리스트 추출 시 기존의 역색인과 달리 부모-자식 관계를 가지는 부분 경로를 통해 엘리먼트 리스트를 추출할 수 있는 방법을 제안함으로써 엘리먼트 리스트의 크기와 구조 조인의 횟수를 줄일 수 있다.
2. 역색인에 사용되는 부분 경로의 길이를 적응적(adaptive)으로 조정함으로써 역색인을 탐색하는 부담(overhead)를 조정할 수 있는 방법을 제시한다.
3. 제안한 역색인으로부터 얻어지는 엘리먼트 리스트를 이용해 질의 결과를 계산할 수 있는 구조 조인 기법을 제시한다.
4. 실험을 통해 기존의 구조 조인 기법과 본 논문의 구조 조인 기법을 비교하고 제안하는 기법의 우수성을 입증한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구에 대해 살펴보고3장에서는 본 논문에서 제안하는 다단계 역색인 기법 및 이를 이용하는 질의 처리 기법에 대해서 설명한다. 4장에서는 실험결과를 설명하고, 5장에서는 결론 및 향후 연구를 기술한다.

2. 관련연구

XML 데이터는 주로 트리 형태의 데이터 모델로 표현되며 보다 일반적으로는 그래프 형태로 표현될 수 있다. XML에 대한 질의 언어는 데이터 트리 내의 경로를 이용하여 표현되며 특히 경로 내에 패턴을 사용함으로써 데이터 트리 내의 임의의 경로를 쉽게 표현할 수 있게 해 준다. XML에 대한 표준 질의 언어인 XQuery는 [2]에서 찾아볼 수 있으며 XQuery 내에서 사용되는 표준 경로 질의 표현인 XPath는 [3]에서 찾을 수 있다.

XML 데이터를 위한 색인 기법은 데이터 트리 내에 반복되는 부트리(sub-tree)들을 하나로 묶음으로써 데이터의 구조 정보를 포함하는 작은 크기의 색인 트리를 만드는 방법이 많이 사용되었다. 이러한 색인 기법들은 단일 경로로 표현된 질의를 찾는 데에 효과적이지만 분기(branch)를 가지는 경로로 표현된 질의를 처리할 수 없다는 단점을 가지고 있다. 이러한 단점을 보완하기 위해 구조 조인 기법[4,5,6]이 소개되었다.

경로질의 처리를 위해 [5,6]에서는 XML 데이터의 각 엘리먼트들을 위치를 이용해 표현하는 방법과 위치로 표현된 엘리먼트 리스트들을 조인하는 구조 조인 방법을 소개하고 있다. [7]에서는 그래프 구조로 표현되는 데이터에 대한 위치 표현 방법에 대해서 소개하고 이를 이용한 그래프 데이터의 구조 조인 기법에 대해서 설명한다. [8]에서는 조인을 통해 나오는 결과물의 정렬 순서를 고려하는 구조 조인 최적화 기법에 대해 설명하며, [4]에서는 [8]에 소개된 기법을 기반으로 하여 여러 개의 구조 조인 연산을 한번에 처리할 수 있는 방법을 보여주고 있다.

구조 조인의 최적화를 위해 [9,10,11,12]에서는 구조 조인에 참여하는 엘리먼트 리스트 자체에 색인을 만듦으로써 불필요한 엘리먼트들의 스캔을 줄여 구조 조인을 최적화 하는 방법들을 제시하고 있다. [9]에서는 B+ 트리를 이용해 엘리먼트 리스트에 대한 색인을 만들었고, [10,11]에서는 XR-트리 기법을 제시함으로써 B+ 트리 보다 향상된 성능의 엘리먼트 리스트 색인을 만드는 방법을 설명하고 있다. [12]에서는 엘리먼트들의 분포 정보를 이용한 엘리먼트 트리 기법과 이를 이용하는 구조 조인 기법을 제시하였다. 그 외에 구조 조인 기법의 성능을 향상 시키기 위해 구조 조인 결과의 크기를 예측하는 연구[13]및 조인 순서를 선택함으로써 최적화하기 위한 연구[14]등이 소개되었다.

하지만, 제안된 대부분의 구조 조인 기법은 엘리먼트 추출을 역색인에 의존하며 보다 효과적인 엘리먼트 추출에 관한 기법에 대해 언급하고 있지 않다.

3. 다단계 역색인 및 구조 조인 기법

본 절에서는 본 논문에서 제안하는 다단계 역색인 기법에 대해서 설명한다. 3.1에서는 2단계 역색인 기법에 대해 소개함으로써 다단계 역색인 기법의 기본 개념에 대해서 설명하고 3.2 에서는 3.1에서 설명한 2단계 역색인 기법을 일반화한 다단계 역색인 기법에 대해서 설명한다. 3.3에서는 다단계 역색인을 이용한 질의 처리 방법에 대해서 알아본다.

3.1 2단계 역색인 기법

구조 조인 연산을 수행하기 위해서는 반드시 엘리먼트 리스트를 추출하는 과정이 선행되어야 한다. 예를 들어, book//name 과 같은 경로 질의의 일부분을 처리하기 위해서는 태그 이름이 book인 엘리먼트들을 모두 포함하는 엘리먼트 리스트와 태그 이름이 name인 엘리먼트들을 모두 포함하는 엘리먼트 리스트를 추출해야 한다. 대부분의 기존의 연구들은 역색인을 이용해 구조 조인에 필요한 엘리먼트 리스트를 추출한 후 조인 연산의 최적화에 대한 내용만을 다루고 있다. 하지만 구조 조인의 성능은 엘리먼트 리스트의 크기에 종속적이며, 일반적으로 역색인을 통해 추출되는 엘리먼트 리스트에는 구조 조인의 결과로 포함되지 않는 엘리먼트가 다수 존재할 수 있다는 문제가 있다. 앞의 book//name의

예에서 name에 해당하는 엘리먼트 리스트에는 조인 결과에 도움을 주지 못하는 다수의 엘리먼트들이 존재할 수 있다. 왜냐하면 데이터 내의 많은 엘리먼트들이 자식 또는 후손 엘리먼트로 name이라는 엘리먼트를 가질 수 있기 때문이다. 예를 들어, 데이터 내의 사람과 관련된 엘리먼트도 name이라는 후손 엘리먼트를 가질 수 있으며 빌딩, 식당, 도시 등 대부분의 엘리먼트들이 name이라는 후손 엘리먼트를 가질 가능성이 매우 크다는 것을 쉽게 알 수 있다.

이러한 역색인을 이용한 엘리먼트 추출 기법의 문제를 해결하기 위해서 본 절에서는 2단계 역색인 기법을 제안한다. 또한, 앞서 언급한 바와 같이3.2절에서는 본 절에서 제안하는 2단계 역색인을 일반화하여 다단계 역색인 기법을 제안한다. 2단계 역색인은 다음의 두 가지 단계를 통해 만들어진다. 첫번째 단계로, XML 데이터 내의 서로 다른 각 태그 이름마다 하나의 색인 노드를 생성하고 색인 노드의 이름을 해당 태그 이름으로 한다. 두 번째 단계로, 각 색인 노드 L1에 대해, 데이터 트리 내에 L1/L2인 경로가 존재하는 경우에 색인 노드 L2를 생성하고 색인 노드 L2를 색인 노드 L1의 자식 노드로 만든다.

(그림 2(b))는 (그림 2(a))의 데이터 트리에 해당하는 2단계 역색인을 보여준다. (그림 2(c))의 2단계 역색인의 물리 구조에 대해서는 3.2에서 설명한다. 각 색인 노드는 길이가 1 또는 2인 부모-자식 관계의 부분 경로와 대응된다. 각 색인 노드는 엘리먼트 쌍의 리스트를 포함하고 있다. 예를 들어, 부분 경로 A/B에 대응되는 역색인 노드는 [(2,7), (2,8), (4,10), (4,11), (6,13), (6,14)]의 여섯 개의 엘리먼트 쌍을 가지는 리스트를 포함한다. 이는 데이터 트리 내에 A/B의 부분 경로가 여섯 개가 존재하며 리스트 내의 각 엘리먼트 쌍은 데이터 트리 내에서 부분경로 A/B가 시작하는 노드와 부분 경로 A/B가 끝나는 노드를 나타낸다. (그림 2(b))에서 회색으로 표시된 부분은 기존의 구조 조인에서 사용하는 역색인과 동일하다는 것을 쉽게 알 수 있다. 회색으로 표현된 부분은 경로의 길이가 1인 부분 경로를 포함하는 색인 노드들로서 기존의 역색인은 1단계 역색인이라 부를 수 있다.

3.2 다단계 역색인 기법

다단계 역색인 기법은 기존의 역색인 기법인 1단계 역색인과 3.2에서 설명한 2단계 역색인을 n단계로 확장할 뿐만 아니라 자주 사용되는 경로와 자주 사용되지 않는 경로를 구분하여 색인을 적응적으로 변경시킨 구조 조인을 위한 일반적인 다단계 색인이라고 볼 수 있다. 즉, 다단계 역색인은 기존의 역색인에 구조 정보를 추가하였으며 색인 구조의 크기를 유연하게 조정할 수 있는 새로운 색인 기법이라 볼 수 있다. 본 논문에서 제안하는 다단계 역색인 구조를 보다 이론적으로 설명하기 위해 다음과 같은 몇 가지 정의를 한다.

정의 1. 데이터 트리 내의 임의의 두 노드 a와 b사이에 경로 $p=t_1/t_2/\cdots/t_n$가 존재한다면 노드 쌍 (a,b)는 경로 p를 만족한다고 한다.

예를 들어, (그림 2(a))에서 노드 쌍 (4, 16)은 경로 *A/B/C*를 만족하며 (10, 18)은 경로 *B/C/D*를 만족한다.

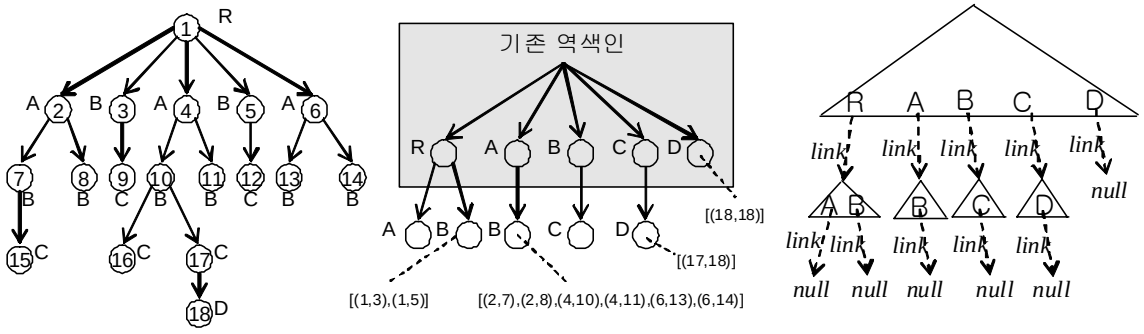
정의 2. 경로 $p=t_1/t_2/\cdots/t_n$에 대해, E_p는 데이터 트리 내에서 경로 p를 만족하는 모든 노드 쌍을 포함하고 있는 집합을 나타낸다.

예를 들어, (그림 2(a))에서 경로 *A/B/C* 의 $E_{A/B/C}$는 { (2, 15), (4, 16), (4, 17) } 이며, 경로 *C/D*의 $E_{C/D}$는 { (17, 18) }이 된다.

정의 3. 경로 $p=t_1/t_2/\cdots/t_n$ 에 대해, CE_p는 (E_p, *link*)를 의미하며 *link*는 색인 노드에 대한 포인터로 초기값으로 *null* 값을 가진다.

정의4. 데이터 내의 서로 다른 태그 t에 대해, t를 키값으로 하여 CE_t ($E_t \neq \emptyset$)를 B+트리 등의 구조를 이용하여 색인한 것을 1단계 역색인이라 부르며 I^1으로 표시한다.

정의 4에서 정의한 바와 같이 1단계 역색인은 데이터 내의 모든 태그들에 대해 생성된다. 이때, 데이터 내의 각 태그들은 길이가 1인 경로라 볼 수 있다. k단계 역색인 I^k는 이전 단계인 k-1단계 역색인 I^{k-1}을 이용해 다음과 같은 2단계의 과정을 통해 만들어진다.



(a) 데이터 트리 (b) 2단계 역색인 (c) 2단계 역색인 물리구조

(그림 2) 2단계 역색인 예제

1. 경로의 길이가 k-1인 각각의 경로 p에 대해 서로 다른 이름을 가지는 각 태그 t에 대해, $E_{p/t} \neq \emptyset$ 인 $E_{p/t}$에 대해 $CE_{p/t}$를 만들고 t를 키값으로 하여 $CE_{p/t}$들을 B+트리 등의 색인 구조를 통해 색인한다. 이렇게 만들어진 색인을 $I_{p/t}$라 한다.
2. I^{k-1}의 각 말단 노드 (leaf node) CE_p의 link가 1에서 만든 색인 $I_{p/t}$를 가리키도록 한다.

(그림 2(c))는 이와 같이 만들어진 2단계 역색인의 물리 구조를 보여준다. 그림에서 삼각형은 B+ 트리를 나타내며 B+ 트리의 각 노드들은 t를 키값으로 하여 $CE_{p/t}$ 및 자신의 하위 B+ 트리에 대한 포인터를 저장한다.

정리1. k 단계 역색인 I^k는 길이가 1부터 k인 모든 경로 (e.g. t_1,t_1/ t_2,…, t_1/ t_2 /…/ t_k)를 색인하며 각 경로 p에 대해 CE_p인 색인 노드가 존재한다.

증명) I^1은 정의 4에 의해 길이가 1인 모든 경로를 색인하며 각 경로 p에 대해 CE_p인 색인 노드를 가진다. I^{k-1}이 길이가 1부터 k-1인 모든 경로를 색인하며 각 경로 p에 대해 CE_p를 가진다고 가정하자. k단계 역색인 구성단계 1, 2와 같이, I^k는 I^{k-1}의 각 단말노드에 경로의 길이가 k인 모든 경로를 색인하는 역색인을 붙여서 만들어지므로 본 정리를 만족한다.

적응형 다단계 역색인은 다단계 역색인은 색인 내에 거의 사용되지 않는 경로들을 삭제하고 자주 사용되는 경로를 추가함으로써 만들어 진다. 본 논문에서는 경로의 사용 빈도를 알아보기 위해 [15]에서 소개된 것과 유사한 정의5와 같은 support의 개념을 사용한다.

정의 5. 데이터 내의 임의의 경로 p에 대해, support(p)는 전체 질의 내에 p를 부분 경로로 가지는 질의의 비율을 나타낸다. support(p) 는 0과 1사이의 값을 가지게 되며 minSup은 사용자가 지정한 최소한의 support값을 의미한다.

정리 2. 두 개의 경로 P와 Q에 대해 prefix(Q) = P 이면 support(P) >= support(Q) 이다.

증명) 경로 P는 경로 Q의 prefix이므로 경로 Q를 부분 경로로 가지는 질의가 질의 집합에 나타난다면 경로 P 또한 해당 질의의 부분 질의가 되므로 성립

```
deleteNodes ( ){
    for(k = n; k > 1; k--)
foreach path p in I^n whose length is k
if(support(p) < minSup) remove the last node of p
from I^n;
}
```

알고리즘 1. 적응형 다단계 역색인을 위한 노드 삭제 알고리즘

n단계 역색인에 대해 적응형 다단계 역색인을 만들기 위해 support값이 minSup을 넘지 못하는 색인 내의 경로들에 대해 색인 노드를 알고리즘 1과 같은 방법으로 경로의 길이가 1이 될 때까지 단계적으로 삭제한다.

```
insertNodes () {
    PathQueue: 길이가 n보다 큰 경로들을 support 값
    으로 내림차순 정렬한 큐
    while (PathQueue is not empty AND number of
    inserted nodes < number of deleted nodes) {
      p = PathQueue.pop();
      q = find prefix path of p from I^n;
      append nodes in (p - q) to the end of q in I^n;
    }
}

} // end of IndexedJoin
```

알고리즘 2. 적응형 다단계 역색인을 위한 노드 삽입 알고리즘

알고리즘 1에 의해 support 값이 minSup보다 작은 경로들의 노드가 삭제된 n단계 역색인은 알고리즘 2에 의해 노드들을 추가하게 된다. 본 논문에서는 추가되는 노드의 개수를 삭제된 노드의 개수와 동일하게 하여 색인의 크기가 동일하게 유지되도록 하였다. 정리 2에 의해 알고리즘 1에 의해 노드가 삭제된 경로들은 새로운 노드를 추가할 대상 경로가 되지 못한다. 따라서 알고리즘 2는 길이가 n보다 큰 경로들에 대해서 발생빈도가 큰 경로 순으로 다단계 역색인에 추가한다.

3.3 다단계 역색인을 이용한 경로 질의 처리 기법

본 절에서는 3.2에서 제안된 다단계 역색인과 구조조인을 이용하는 질의 처리 방법 방법에 대해 알아본다. //A/B//C/D인 경로 질의를 생각해 보자. 기존의 구조 조인을 이용하는 경로 질의 처리 방법은 각 태그에 대해 4회의 역색인 탐색이 필요하며, ((A⋈B)⋈C)⋈D)의 4회의 조인 연산을 필요로 한다. 이러한 질의는 2단계 역색인을 이용하면 다음과 같은 방법으로 처리할 수 있다.

1. 경로 질의 내의 조상-후손 축(//)을 분리자로 하여 경로 질의를 여러 개의 부분 경로 질의로 분리한다. 즉, //A/B//C/D는 A/B와 C/D의 두개의 부분 경로로 분리된다.
2. 2단계 역색인을 이용해 A/B에 대한 엘리먼트 리스트인 $E_{A/B}$를 추출하고, C/D에 대한 엘리먼트 리스트인 $E_{C/D}$를 추출한다.
3. $E_{A/B}$와 $E_{C/D}$에 대해 구조 조인연산을 수행하여 질의의 결과를 얻어낸다.

위와 같이 2단계 역색인을 사용하는 경우 2회의 색인 탐색과 2회의 조인 연산만으로 질의의 결과를 얻을 수 있다. 이때, 기존 역색인보다 다단계 역색인의 크기가 더 크므로

역색인 탐색 비용이 약간 더 커질 수 있지만, 조인에 참여하는 엘리먼트의 수가 현저하게 줄어들게 되며 역색인 탐색 횟수 및 조인 횟수가 절반으로 줄어들어 더 좋은 성능을 보일 수 있다.

경로 분할 및 다단계 역색인 탐색

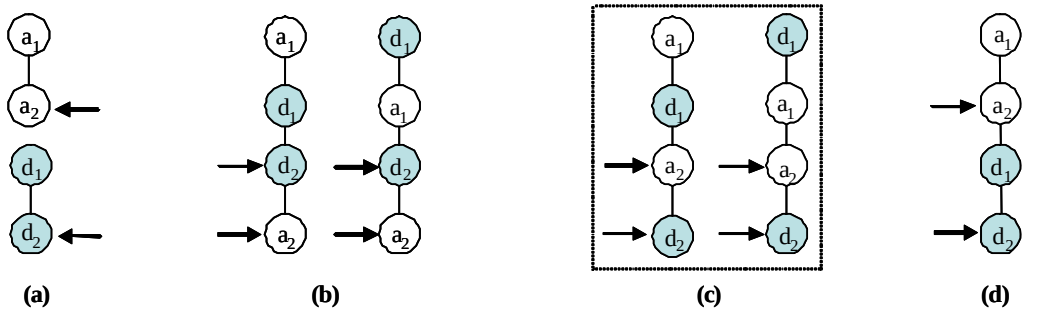
일반적으로 다단계 역색인을 사용하는 경우에도 위와 같이 조상-후손 축(//)을 분리자로 하여 경로 질의를 분리하고 분리된 부분 경로에 대한 엘리먼트 리스트를 다단계 역색인을 통해 추출한다. 이때 분리된 부분 경로의 길이가 다단계 역색인의 길이보다 긴 경우 분리된 부분 경로를 다시 분리하는 과정을 필요로 하게 된다. 예를 들어, //A/B/C//D인 질의를 생각해 보자. 이 질의는 A/B/C 와 D의 두 개의 부분 경로로 분리된다. 이때 다단계 역색인이 A/B/C경로를 색인하고 있지 않은 경우 부분 경로 A/B/C는 추가로 분리되어야 한다. 경로 A/B/C 는 {A/B, C}, {A, B/C}, {A, B, C} 등으로 분리될 수 있으며, 분리될 수 있는 경우의 수는 경로의 길이를 L이라고 할 때 2^{L-1} -1개 존재한다. 본 논문에서는 부모-자식 축으로 연결된 부분경로를 앞에서부터 순서대로 다단계 역색인에 색인되어 있는 길이만큼 잘라내는 방법을 사용한다. 예를 들어 2단계 역색인의 경우 A/B/C와 같은 부분 경로는 A/B, C의 두 개의 부분경로로 분리한다. 적응형 다단계 역색인의 경우 색인된 경로마다 길이가 다를 수 있으므로 검색해야 하는 부분 경로로 전체를 다단계 역색인의 입력으로 넣은 후에 다단계 역색인에서 검색되는 길이만큼씩 잘라 사용한다. 예를 들어, 길이가 n인 부분 경로 T_1/ T_2 /…/ T_n 의 경우 검색방법은 다음과 같다. 다단계 역색인의 루트 노드에서 T_1을 검색하고 검색 결과로 나온 색인 노드 N_1에서 T_2를 검색하며, 같은 과정을 반복하여 색인 노드 N_i에서 T_{i+1}을 검색한다. 색인노드 N_i에서 검색이 실패하는 경우 T_1/…/ T_i 로 경로를 분리하고 T_{i+1}/…/ T_n까지 동일한 방식을 다시 적용한다. 부모-자식 관계로 묶여 있는 부분 경로 p가 앞의 설명에 따라 $p1$과 $p2$로 분리될 경우 다단계 역색인을 통해 E_{p1}과 E_{p2}를 추출하고 E_{p1}과 E_{p2}에 대해 부모-자식 관계에 대한 구조 조인을 수행함으로써 E_p를 얻어낼 수 있다. 이때, 부분 경로 p = T_1/ T_2 /…/ T_n를 $p1$ = T_1/…/ T_i와 $p2$ = T_i/…/ T_n으로 분리하게 되면 E_{p1}에 속한 각 엘리먼트 쌍의 두번째 엘리먼트와 E_{p2}에 속한 각 엘리먼트 쌍의 첫번째 엘리먼트가 같은 지를 비교함으로써 보다 손쉽게 E_p를 계산할 수 있다.

다단계 역색인을 이용하는 구조 조인 기법

부모-자식 축으로 이루어진 부분경로 p에 대해, 다단계 역색인의 검색을 통해 얻어지는 E_p는 정의 2와 같이 엘리먼트 쌍의 리스트이므로 기존의 구조 조인 기법을 직접 적용할 수 없다. 다단계 역색인을 통해 얻어진 두 개의 엘리먼트 리스트 AList와 DList를 생각해 보자. AList 내의 엘리먼트 쌍 a = (a_1, a_2)와 DList 내의 엘리먼트 쌍 d = (d_1, d_2)에 대해서 a_2가 d_1의 조상 엘리먼트인 경우 (a_1, d_2)가 조인 결과에 추가되게 된다. 기존의 구조 조인과 같이 정렬 기반의 조인을 수행하려면 AList는 각 엘리먼트 쌍의 두 번째 엘리먼트인 a_2로 정렬되어야 하며 DList는 각 엘리먼트 쌍의 첫 번째 엘리먼트인 d_1으로 정렬되어야 한다. 하지만, 조인의 결과는 (a_1, d_2)이므로 이러한 경우 조인의 결과는 정렬이 되지 않아 이후 조인을 수행하기 위해서는 정렬을 다시 해야 하는 문제가 발생한다. 본 논문에서는 이러한 문제를 해결하기 위해 다음의 속성을 사용한다.

속성 1. AList 내의 엘리먼트 쌍 a = (a_1, a_2)와 DList 내의 엘리먼트 쌍 d = (d_1, d_2)에 대해서, 만약 a_2가 d_1의 조상 노드이면 a_2는 d_2의 조상 엘리먼트 이다. 즉, a_2가 d_2의 조상 엘리먼트가 아니면 a_2는 d_1의 조상 엘리먼트도 아니다.

일반적으로, 구조 조인을 위해 엘리먼트 리스트를 정렬하는 이유는 조인의 결과에 속하지 않는 엘리먼트를 이후 조인 연산에서도 배제하기 위한 목적을 가진다. 속성 1에 따라서 만약 a_2가 d_2의 조상 엘리먼트가 아니면 (a_1, d_2)는 조인의 결과가 될 수 없다는 것을 쉽게 알 수 있다. 즉, a_2와 d_2를 비교함으로써 조인의 결과에 참여하지 않는 엘리먼트 쌍을 찾아낼 수 있다. (그림 3)은 조인하고자 하는 두 개의 엘리먼트 쌍의 가능한 위치관계를 보여준다. 그림에서 엘리먼트 사이에 선으로 연결되어 있는 경우 조상-후손 관계가 있다는 것을 표시하며 위에 위치한 엘리먼트가 조상 엘리먼트이다. (그림 3(d))가 조인의 결과에 속해야 하는 경우를 표시한다. 하지만, 구조 조인 시에 a_2와 d_2를 조사해 조인에 참여하지 않는 엘리먼트를 가려내는 경우 (그림 3(c))의 경



(그림 3) 엘리먼트 쌍의 위치

```
Algorithm StructuralJoin(AList, DList){
a = AList→1st Element;
d = DList→1st Element;
while(the input list are not empty or the stack is not empty){
if(d→2nd.StartPos > stack→top→2nd.EndPos)
stack→pop();
        else if(a→2nd.StartPos < d→2nd.StartPos){
            stack→push(a);
            a = a→nextElement;
        }
        else{
            for(a1 = stack→bottom; a1 ≠NULL; a1 = a1→up)
                if(a1→2nd is an ancestor of d→1st)  append (a1→1st, d→2nd) to OutpuList
            d = d→nextElement;
}
} // end of while
}  // end of IndexedJoin
```

알고리즘 3. 다단계 역색인을 이용하는 구조 조인 알고리즘

우도 결과에 포함될 수 있다. 따라서 조인 연산 시 조인 결과로 계산되는 엘리먼트 쌍에 대해 a_2와 d_1을 이용해 한번 더 걸러주는 과정이 필요하다.

비록 (그림 3(c))와 같은 경우를 결과 출력 리스트에 포함시키기 전에 한번 더 걸러내는 과정이 필요하지만 다단계 역색인을 통해 얻어지는 엘리먼트 리스트의 크기가 작고 조인 결과로 참여하지 못하는 엘리먼트 쌍들은 대부분 (그림 3(a))와 같은 경우임을 고려하면 본 논문의 구조 조인 알고리즘이 여전히 경쟁력이 있다는 것을 알 수 있다. 알고리즘 3은 본 논문에서 제안하는 구조 조인 알고리즘을 보여준다. 알고리즘 3은 속성 1을 사용하여 엘리먼트 쌍에 대한 두 개의 리스트를 처리하며 [8]과 같이 스택을 이용하는 알고리즘이다. 현재 처리되고 있는 DList 내의 엘리먼트 쌍 d = (d_1, d_2)에 대해, 본 알고리즘에서는 s_2 가 d_2의 조상 엘리먼트가 아닌 스택 내의 모든 (s_1, s_2) 쌍을 제거한다. 그리고, 현재 처리되고 있는 DList 내의 엘리먼트 쌍 d = (d_1, d_2)에 대해 a_2 가 d_2의 조상 엘리먼트인 AList 내의 모든 a = (a_1, a_2)를 스택에 추가한다. 이때, 스택이 비어 있지 않다고 하면 항상 스택의 최상위(top) 엘리먼트 쌍 (s_1, s_2)에 대해 a_2가 s_2의 후손 엘리먼트임을 쉽게 알 수 있다. 마지막으로 본 알고리즘은 스택 내의 모든 엘리먼트 쌍 (s_1, s_2)와 DList에서 현재 처리되고 있는 엘리먼트 쌍 d = (d_1, d_2) 에 대해 만약 s_2가 d_1의 조상 엘리먼트인 경우 (s_1, d_2)를 결과로 출력한다. 본 알고리즘에서 각 엘리먼트 쌍은 (1^{st}, 2^{nd})로 표시되어 있다.

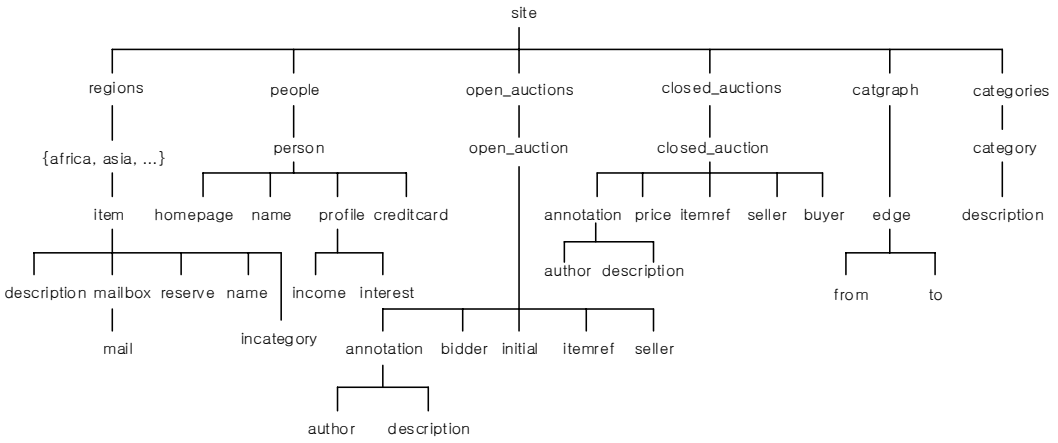
4. 실 험

이 절에서는 본 논문에서 제안한 다단계 역색인과 구조 조인 기법을 이용한 질의처리 성능에 대해 살펴본다. 우선, 본 논문에서 사용된 실험을 위한 플랫폼과 실험 데이터, 그리고 실험에 사용된 경로 질의에 대해 기술하고, 기존의 역색인을 이용한 구조 조인 기법과 성능 비교를 한다.

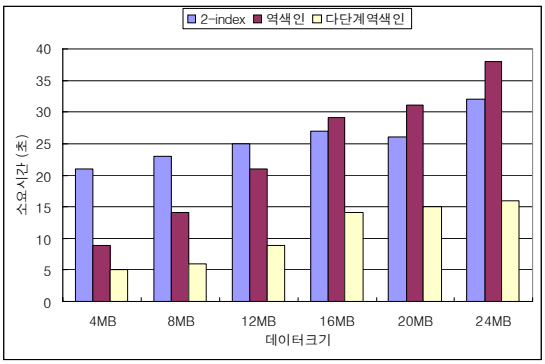
본 논문의 실험을 위해 간단한 객체 저장 시스템을 구현하여 사용하였다. 실험에 사용된 시스템은 4Kbyte 크기의 페이지를 가지고 있으며 저장시스템, 색인 시스템, 버퍼 관리 시스템, 질의 처리 시스템으로 나누어져 있다. 버퍼 관리 시스템은 200개의 메모리 버퍼를 관리하도록 구성하였다. 실험 시스템은 Pentium III 1GHz 512M RAM의 윈도우 XP 시스템 위에 구축하였다. 실험 데이터로는 인터넷 상의 옥션 사이트를 이용해 XML 벤치마크 데이터를 구축한 XMark [16] 데이터 셋을 사용하였다. XMark에서 제공하는 데이터 생성 프로그램을 이용하여 4MB(약 42200 엘리먼트)에서 24MB(약 492900 엘리먼트)까지의 데이터를 생성해 실험에 사용하였다. (그림 4)는 실험에 사용한 XMark 데이터 셋의 DTD 구조를 보여준다.

실험에 사용된 경로 질의는 길이가 5에서 8인 경로를 데이터 트리를 무작위로 탐색하여 생성하였다. 부모-자식 관계로만 연결되어 있는 50개의 경로를 무작위로 추출한 후에 추출된 경로들에서 일련의 태그들을 조상-후손 축으로 교체함으로써 질의 집합을 생성하였다. 표 1은 이와 같은 방법으로 생성된 50개의 질의를 보여준다. 실험에 사용된 각 데이터 셋에 대해 2단계 역색인을 구성한 후에 표 1의 질의 셋을 이용하여 2단계 역색인을 minSup값을 0.02로 하여 적응형 다단계 역색인으로 재구성하여 실험을 수행하였다.

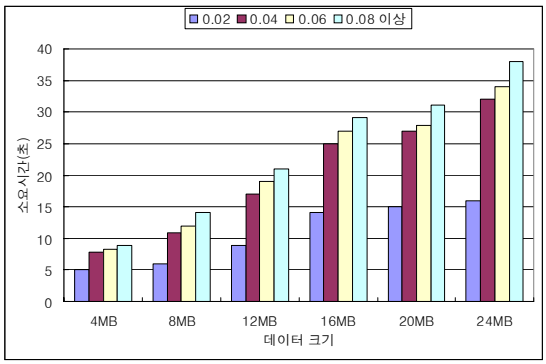
<표 1>에서 볼 수 있듯이 무작위로 추출된 질의 중 90%의 질의에서 부모-자식 관계로 연결되어 있는 부분 경로의 길이가 2를 넘지 않는다. 따라서 적응형 다단계 역색인은 2단계 역색인과 성능차이를 거의 보이지 않는 다는 것을 알 수 있다. 하지만 적응형 다단계 역색인의 경우 2단계 역색인 보다 색인의 크기가 약 50%정도 감소하여 색인 크기에



(그림 4) XMark 데이터 셋의 DTD



(a) 다단계 역색인의 질의처리 성능



(b) minSup 변화에 따른 수행 시간

(그림 5) 실험 결과

〈표 1〉 실험에 사용된 질의

임의 생성 질의	
site/catgraph//to	site//interest/category
site//edge/to	//edge/to
site//id	//person/id
site//reserve	site/people//name
site//closed_auction//person	site//africa//location
site//from	site//person/emailaddress
//catgraph/edge/to	site//category//listitem/parlist/listitem
site/catgraph//to	//category/id
site//current	//closed_auctions/closed_auction//person
site//open_auction//item	site/regions//item/shipping
site//name	site//description/text
site//item/location	site/open_auctions//reserve
site/people//name	site/categories//id
site//itemref/item	//catgraph/edge/to
site//quantity	site//category/id
site/categories//name	site/catgraph//to
//person/emailaddress	site//edge/from
//catgraph/edge/to	//closed_auctions//seller/person
site//to	//closed_auctions//seller/person
site/closed_auctions//item	//closed_auction/date
//people/person/name	site/categories//id
site//closed_auction/type	site//samerica//incategory/category
site/open_auctions//type	site//bidder/increase
site//category/name	site/people//id

대한 부담이 현저하게 줄어드는 것을 확인하였다. (그림 5(a))는 실험 데이터에 대한 질의 처리 결과를 보여준다. 본 실험에서는 2-index [17]와 역색인을 이용한 구조 조인 기법 그리고 본 논문의 적응형 다단계 역색인을 이용하였다. 실험 데이터의 크기가 증가할수록 구조 조인에 참여하는 엘리먼트 리스트의 크기도 증가하게 된다. 2-index와 같은 기존의 XML 색인 기법을 사용하는 경우 조상-후손 관계에 대해 색인 그래프 전체를 탐색해야 하는 문제점 때문에 성능이 좋지 않아 지지만 데이터 크기의 증가에 민감하지 않다는 것을 알 수 있다. 다단계 역색인을 통해 엘리먼트 리스트를 추출하는 경우는 기존의 역색인을 이용하는 경우보다 엘리먼트 리스트의 크기 증가율이 작고 조인횟수도 약 2배 가량 감소하게 되어, (그림 5(a))와 같이 성능향상을 가져온 것을 알 수 있다. (그림 5(b))는 minSup의 변화에 따른 질의처리 성능 변화를 보여준다. minSup이 증가함에 따라 적응형 다단계 역색인은 1단계 역색인에 수렴하게 되며 이에 따라 질의 처리 성능이 좋지 않게 된다. 하지만 minSup의 크기가 증가함에 따라 색인의 크기가 감소한다는 것도 알 수 있다. 실험을 통해 알 수 있듯이 XML 데이터에 대한 경로 질의 내에 부모-자식 관계를 가지는 부분질의의 길이는 대부분 2정도 이며 따라서 2단계 역색인을 사용함으로써 질의 처리의 성능을 향상 시킬 수 있다. 하지만, 적응형 다단계 역색인을 사용함으로써 2단계 역색인의 크기를 대폭

감소시킴으로써 공간에 대한 부담을 줄일 수 있다는 것을 확인하였다.

3절에서 언급한 바와 같이 본 논문에서 제안한 구조 조인 기법은 조인의 결과가 되지 못하는 엘리먼트 쌍((그림 3(c)) 경우)을 걸러내지 못해 이를 체크해야 하는 경우가 발생하지만, 본 실험에서는 이러한 경우가 전혀 발생하지 않았다. (그림 3(c))의 경우는 경로 P를 P1과 P2로 분리하는 경우 suffix(P1)과 prefix(P2)가 같아지는 경우로 실 데이터에서는 발생 확률이 매우 적다는 것을 알 수 있다.

5. 결 론

본 논문에서는 XML 경로 질의를 위한 효과적인 다단계 역색인 기법과 이를 이용하는 구조 조인 기법을 제안하였다. 기존의 구조 조인 기법은 태그 중심의 역색인을 사용하기 때문에 부모-자식 관계와 같이 색인을 통해 쉽게 처리될 수 있는 연산에 대해서도 복잡한 조인 연산을 수행해야 하는 단점을 가지고 있다. 이러한 단점을 극복하기 위해 본 논문에서 제안한 방법은 부모-자식 관계를 가지는 부분 질의를 기존의 역색인을 단계별로 확장한 다단계 역색인을 통해 미리 처리하게 함으로써 질의 처리의 성능을 높였다는 것이 특징이다.

본 논문에서는 제안한 다단계 역색인과 구조 조인 기법을 구현하였고, 실제로 사용되는 인터넷 데이터를 기반으로 한 벤치마크 데이터에 대한 실험을 통해 제안한 기법의 성능의 우수성을 보였다. 3.3절에서 언급한 바와 같이 질의에 사용되는 경로를 분할할 수 있는 다양한 방법이 존재한다. 본 논문에서는 가장 단순한 방법을 이용하여 경로를 분할하였으나 향후 최적화된 경로 분할을 통해 질의 처리의 성능을 향상 시키는 방법에 대해 연구할 예정이다.

참 고 문 헌

[1] T. Bray, J. Paoli and C.M. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0," Technical Report, W3C Recommendation, 1998.

[2] D. Chamberlin, D. Florescu, J. Robie, J. Simeon and M. Stefanescu, "XQuery: A Query Language for XML," Technical report, W3C Working Draft, Feb. 2001.

[3] J. Clark and S. DeRose, "XML Path Language (XPath)," Technical report, W3C Recommendation, 1999.

[4] N. Bruno, N. Koudas and D. Srivastava, "Holistic Twig Joins: Optimal XML Pattern Matching," In Proceedings of the ACM SIGMOD International Conference on the Management of Data, pp.310-321, 2002.

[5] Q. Li and B. Moon, "Indexing and Querying XML Data for Regular Path Expressions," In Proceedings of the Conference on Very Large Data Bases, pp.361-370, 2001.

[6] C. Zhang, J. Naughton, D. DeWitt, Q. Luo and G. Lohman, "On Supporting Containment Queries in Relational Database Management Systems," In Proceedings of the ACM SIGMOD International Conference on the Management of Data, pp. 425-430, 2001.

[7] J. Kim and H.-J. Kim, "Efficient Processing of Regular Path Joins using PID," Information and Software Technology, Vol.45, No.5, pp.241-251, 2003.

[8] S. Al-Khalifa, H.V. Jagadish, N. Koudas, J.M. Patel, D. Srivastava and Y. Wu., "Structural Joins: A Primitive for Efficient XML Query Pattern Matching," In Proceedings of IEEE International Conference on Data Engineering, pp. 141-152, 2002.

[9] S.-Y. Chien, Z. Vagena, D. Zhang, V.J. Tsotras and C. Zaniolo, "Efficient Structural Joins on Indexed XML Documents," In Proceedings of the Conference on Very Large Data Bases, pp.263-274, 2002.

[10] H. Jiang, H. Lu, W. Wang and B.C. Ooi, "XR-Tree: Indexing XML Data for Efficient Structural Joins," In Proceedings of IEEE International Conference on Data Engineering, pp.253-264, 2003.

[11] H. Jiang, W. Wang, H. Lu and J.X. Yu, "Holistic Twig Joins on Indexed XML Documents," In Proceedings of the Conference on Very Large Data Bases, pp.273-284, 2003.

[12] J. Kim, "Advanced Structural Joins using Element Distribution," Information Sciences, Vol.176, No.22, pp.1063-1068, 2006.

[13] Y. Wu, J.M. Patel and H.V. Jagadish, "Estimating Answer Sizes for XML Queries," In Proceedings of the International Conference on Extending Database Technology, pp.590-608, 2002.

[14] Y. Wu, J.M. Patel and H.V. Jagadish, "Structural Join Order Selection for XML Query Optimization," In proceedings of IEEE International Conference on Data Engineering, pp. 443-454, 2003.

[15] C.-W. Chung, J.-K. Min and K. Shim, "APEX: an Adaptive Path Index for XML Data," In proceedings of ACM SIGMOD International Conference on the Management of Data, pp. 321-132, 2002.

[16] Xmark, "The XML Benchmark Project." http://monetdb.cwi.nl/xml/

[17] T. Milo and D. Suciu, "Index Structures for Path Expressions," In proceedings of the International Conference on Database Theory, pp.277-295, 1999.

김 종 익

e-mail : jongik@chonbuk.ac.kr
1998년 한국과학기술원 전산학과 (학사)
2000년 한국과학기술원 전산학과 (석사)
2004년 서울대학교 컴퓨터공학과 (박사)
2004년~2007년 한국전자통신연구원 선임연구원
2007년~현 재 전북대학교 전자정보공학부 전임강사
관심분야 : 데이터베이스, XML, 정보검색, 텔레매틱스 등