의미 기반의 XML 키워드 검색을 위한 효율적인 인덱스 구조

(An Efficient Index Structure for Semantic-based XML Keyword Search)

이 형 동 † 김 성 진 † 김 형 주 ††
(Hyungdong Lee) (Sung Jin Kim) (Hyoung-Joo Kim)

요 약 XML 키워드 검색에서의 검색 결과는 일반적으로 질의 키워드를 모두 포함하는 원소 중 가장 구체적 원소들로 정의된다. 키워드 검색의 정확도 향상을 위하여 XML 원소의 레이블과 온톨로지, 개념 모델, 시소러스 등의 의미 정보가 사용되고 있다. 본 논문에서는 의미 정보를 이용하여 검색 결과로 반환 가능한 개념들이 정의되고 사용자가 검색하려는 개념이 해석 가능할 경우 효율적 질의 처리를 위한 계층 인덱스를 제안한다. 계층 인덱스는 각 키워드 포스팅의 XML 원소들을 원소가 속한 개념들의 상하 관계에 따라 구별하여 저장하고, 검색 결과 산출 가능성이 있는 개념에 속한 원소들만을 선별적으로 읽어서 제한된 조합으로 질의 결과 후보가 되는 최소 공통 선조들을 산출할 수 있도록 한다. 본 논문에서는 계층 인덱스의 구성 원리와 구성 방법, 계층 인덱스를 이용한 질의 처리 방법을 기술한다. DBLP의 XML문서와 INEX2003의 XML 문서 집합을 이용한 실험에서 의미 기반 계층 인덱스는 우수한 성능을 나타내었다.

키워드 : XML, 키워드 검색, 계층 인덱스

Abstract Search results of XML keyword search are defined generally as the most specific elements containing all query keywords in the literature. The labels of XML elements and semantic information such as ontology, conceptual model, thesaurus, and so on, are used to improve the preciseness of the search results. This paper presents a hierarchical index for an efficient XML keyword query processing on the condition that returnable search concepts are defined and users' query concepts can be interpreted with the help of the semantic information. The hierarchical index separately stores the XML elements containing a keyword on the basis of the hierarchical relations of the concepts that the XML elements belong to, and makes it possible to obtain least common ancestors, which are candidates for the search results, with selectively reading the elements belonging to the concepts relevant to query concepts and without considering all the combinations of the elements having been read. This paper deals with how to organize the hierarchical index and how to process XML keyword queries with the index. In our experiment with the DBLP XML document and the XML documents in the INEX2003 test set, the hierarchical index worked well.

Key words : XML(eXtensible Markup Language), Keyword Search, Hierarchical Index

1. 서 론

XML 문서에 대한 키워드 검색은 사용자가 문서의 구조적 정보나 복잡한 질의어에 대한 사전 학습 없이도 원하는 정보를 쉽게 검색할 수 있도록 한다. 현재 질의 키워드를 모두 포함하는 원소(element)가 사용자의 질의에 관련된 원소라는 가정이 보편적으로 사용되고 있으며, 관련된 원소 중에서 가장 구체적 원소(또는 다른 관련 원소를 포함하지 않는 원소)들을 검색 결과로 반환하고 있다[1-5]. 이 경우 질의에 무관하거나 의미 없는 원소가 종종 질의 결과로 반환된다. XML 키워드 검

· 본 연구는 BK-21 정보기술사업단과 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성, 지원사업(IITA-2006-C1090-0603-0031)의 연구결과로 수행되었음
† 학생회원 : 서울대학교 컴퓨터공학부
hdlee@idb.snu.ac.kr
sjkim@idb.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
hjk@idb.snu.ac.kr
논문접수 : 2006년 4월 12일
심사완료 : 2006년 6월 22일

색의 정확도 향상을 위하여 XML 원소의 레이블(label)과 온톨로지(ontology), 개념적 모델(conceptual model), 시소러스(thesaurus) 등의 의미 정보가 활용되고 있다[5-10].

XML 키워드 검색 시스템들은 효율적 질의 처리를 위하여 각 키워드들에 대한 포스팅(posting)을 인덱스에 유지 관리한다. 두 개 이상의 키워드가 질의되면, 검색 시스템은 각 키워드의 포스팅에 색인된 원소들의 조합으로 최소 공통 선조를 산출하고, 산출된 최소 공통 선조가 관리자에 의해 수립된 검색 결과 정의에 부합할 경우 이를 검색 결과 원소로 반환한다. 최악의 경우 질의 키워드의 포스팅에 색인된 원소들간의 모든 조합에 대해 최소 공통 선조가 검색 결과 대상으로 고려될 수 있으며, 한 키워드의 포스팅은 다른 키워드의 포스팅에 색인된 원소 개수만큼 반복적으로 읽힐 수 있다. 그러므로 디스크 접근을 줄이고, 효율적인 질의 처리를 위한 연구가 진행되어 왔다.

본 논문에서는 온톨로지, 개념 모델, 시소러스 등의 의미 정보를 이용하여 검색 결과로 반환 가능한 개념들이 정의되고 사용자가 검색하고자 하는 개념 해석이 가능할 경우 효율적 질의 처리를 위한 계층 인덱스를 제안한다. 계층 인덱스는 각 키워드 포스팅의 XML 원소들을 원소가 속한 개념들의 상하 관계에 따라 구별하여 저장하고, 검색 결과 산출 가능성이 있는 개념에 속한 원소들만을 선별적으로 읽어서 제한된 조합으로 질의 결과 후보가 되는 최소 공통 선조들을 산출할 수 있도록 한다. 따라서, 검색 결과가 되지 않는 불필요한 최소 공통 선조 조합 연산을 줄임으로써 검색 속도를 향상시킬 수 있다.

본 논문에서는 의미 기반 계층 인덱스의 사용에 앞서 다음을 전제한다. 첫째, XML 문서는 트리로 모델링 된다[3,5,6,10]. XML 원소는 트리의 노드가 되고 원소 이름은 노드의 레이블이 된다. 둘째, 노드의 레이블은 노드가 포함하는 데이타의 개념을 표현한다[7-9]. 셋째, 질의 결과로 반환 가능한 원소들의 레이블(또는 개념)이 정해진다[6,10]. 검색 결과 개념은 관리자의 검색 서비스 전략에 따라 임의로 결정되거나, 온톨로지나 개념적 모델 등의 부가 정보를 통하여 검색 결과로 반환 가능한 개념들이 사전에 정의될 수 있다[7-9]. 넷째, 검색 질의에서 사용자가 검색하려는 개념 해석이 가능하다. 검색 개념은 명시적으로 주어지거나 암묵적으로 주어질 수 있다.

의미 기반의 계층 인덱스는 다음의 특징을 가진다. 첫째, 주어진 질의 개념들에 따라 비교 유닛을 구성하여 동일 비교 유닛에 속한 원소들의 조합만으로 질의 결과 원소를 얻는다. 따라서 검색 시스템은 질의 키워드의 포스팅에 색인된 모든 원소들의 불필요한 조합 연산을 피하고 검색 결과 산출에 필요한 최소 공통 선조 산출 회수를 줄일 수 있다. 둘째, 한 키워드의 포스팅에서 원소를 포함하지 않는 비교 유닛이 있을 경우 다른 키워드들의 인덱스에서 동일한 비교 유닛에 해당하는 원소들을 읽지 않는다. 따라서 검색 결과 산출에 필요한 I/O 회수를 감소시킬 수 있다. 셋째, 검색 결과로 반환되지 않는 원소를 불필요하게 인덱스에 색인하지 않는다.

논문 구성은 다음과 같다. 2장에서는 본 논문과 관련된 기존의 연구들에 대해 알아본다. 3장에서는 의미 기반 XML 키워드 검색을 설명하고 연구 범위를 기술한다. 4장에서는 의미 기반 계층 인덱스의 구성과 이를 이용한 검색 방법을 제안한다. 5장에서는 의미 기반 계층 인덱스의 효율성을 실험으로 보인다. 6장에서는 결론을 맺고 향후 계획을 기술한다.

2. 관련 연구

XML 키워드 질의 처리를 위해 최소 공통 선조를 효율적으로 구하기 위한 연구로, [3]은 DIL(Dewey Inverted List) 구조의 인덱스를 제안하고 효율적으로 질의 결과를 검색하는 방법을 기술하였다. 이는 듀이 원소 식별자에 따라 원소들을 정렬하고 각 인덱스를 순차적으로 비교하여 질의 결과를 산출하여 한 포스팅이 두 번 이상 읽히지 않도록 하였다. 하지만 대용량의 문서들의 모든 원소를 처리하는 것은 비용이 크다. [4]는 XML 원소들을 B+트리 구조로 색인하고 IL(Indexed Lookup eager) 알고리즘에 의하여 질의 결과를 산출하는 방법을 제안하였다. IL 알고리즘은 적은 빈도수를 가지는 키워드를 포함한 노드들을 기준으로 최소 공통 선조를 찾음으로써 많은 빈도수를 가지는 키워드를 포함한 노드들 전체가 읽히는 것을 방지하였다. 또한 B+ 트리의 내부 노드들에 해당하는 원소들을 메모리에 효율적으로 캐시(cache)하여 디스크 접근 회수를 줄이는 방법을 기술하였다. [5]의 분할인덱스는 검색 대상이 되는 XML 문서에서 검색 결과로 반환 가능한 원소들의 최소 깊이가 주어지거나 관리자가 최소 깊이를 일정 수준으로 제한하는 경우(검색 결과 원소의 깊이는 주어진 최소 깊이보다 크다), 인덱스를 다수의 파티션(partition)들로 나누고 동일 파티션에 속한 원소들만의 조합으로 결과 검색을 가능하도록 하였다. 하지만 검색 결과를 의미와 상관없이 깊이에 따라 제한하므로 원소들의 깊이가 다양한 복잡한 구조의 데이타에는 적용하기 어려운 측면이 있다.

개념적 모델이나 온톨로지 등의 의미 정보를 XML 데이타에 적용함으로써 검색 성능을 높이려는 연구도 진행되어 왔다. 개념적 모델링은 자료 구조 및 관계 분

석, 제약 조건 분석 등을 통해 데이타베이스의 품질 향상을 위해 널리 사용된다. 이러한 개념적 모델링 기법을 XML 데이타베이스에 대한 설계에 적용하려는 연구나 역으로 XML 데이타의 DTD 정보를 바탕으로 개념적 모델을 이끌어 내는 연구들이 수행되었다. UML[11]이나 ER 다이어그램과 같은 개념적 설계 기반과 XML 데이타 간의 변환 시 모델의 차이로 인해 발생하는 문제점들을 해결하는데 초점이 맞추어져 있다[12-15]. 또한 온톨로지 정보를 이용한 XML 데이타의 통합 분야에서도 온톨로지와 XML 데이타의 대응에 관한 연구가 행해졌으며 세부적인 대응 방법에서는 개념적 모델 방식과 차이가 있다[7,16]. 이러한 연구들을 통해 검색 결과로 반환 가능한 개념들이 사전에 정의될 수 있다. 예를 들어, DBLP 사이트[11]에서는 질의 키워드를 포함한 논문과 저자에 대한 검색 서비스를 제공하고 있으며 사용자가 검색하고자 하는 개념(논문 또는 저자)은 질의 키워드가 입력되는 웹 문서에 따라 암묵적으로 결정된다. XIRQL[6]은 본 논문의 검색 개념과 유사한 색인과 질의 결과의 단위가 되는 목표 객체(Target Object)를 가정하여 단어 가중치(weighting) 처리, 유사도 기반 검색, 모호한 질의 조건의 평가 등의 개념을 넣어 정보검색 시스템에 적합한 XML 질의어와 경로 대수(path algebra)를 제안하였으며, 검색하고자 하는 개념이 질의 언어 상에 명시적으로 주어진다. [6,8,10]은 동일 개념의 다양한 용어 표현들에 대한 일치를 위하여 온톨로지나 시소러스의 필요성이 제기되었다. [8]은 XML 키워드 검색에서 원소의 레이블 정보를 이용하여 무의미한 원소가 질의 결과로 반환되는 것을 제한하였다.

3. 예비 연구

본 장에서는 연구 범위와 의미 기반 계층 인덱스의 사용 환경을 기술한다. 의미 기반 XML 키워드 검색은 “검색 개념 선정”, “키워드 색인”, “질의 개념 해석”, “검색 결과 산출”의 네 단계로 구성되며 그림 1과 같다. “검색 개념 선정” 단계에서는 온톨로지 등의 의미 정보와 경험적 지식 등의 부가 정보를 이용하여 검색에 사용될 개념을 결정한다. 이때 관리자는 검색서비스가 특정 개념에 제한되도록 검색에 사용될 개념을 전략적으로 결정할 수 있다. “키워드 색인” 단계에서는 색인될 키워드를 포함하고 있는 원소들이 속한 개념들에 의거하여 한 키워드의 포스팅을 다수의 포스팅으로 나누어 저장한다. “질의 개념 해석” 단계에서는 사용자가 검색하고자 하는 개념을 파악한다. “검색 결과 산출” 단계에서는 계층 인덱스에 색인된 원소들 간의 한정된 조합으로 산출된 최소 공통 선조들로부터 검색 결과를 산출한다. 본 논문에서는 첫 번째 단계와 세 번째 단계에서 행하여지는 검색 개념 선정과 질의 개념 해석의 효과성 측면은 다루지 않으며 두 번째 단계와 네 번째 단계에서 행하여지는 계층 인덱스의 생성과 검색 결과 산출의 효율성을 다룬다.

XML 문서는 순서있고(ordered) 레이블있는(labeled) 노드의 트리로 모델링된다. XML 원소는 트리의 노드이고, 원소 이름은 노드의 레이블이다. 부모 노드와 자식 노드는 직선으로 연결된다. 데이타는 둥근 사각형 안에 나타나 있고 해당 데이타를 포함한 원소가 연결되어 있다. 속성(attribute)은 속성을 포함하는 원소의 자식 원소가 된다. 그림 2는 XML 문서로부터 모델링된 XML 트리의 예이다. 노드들은 사각형으로 표기 되고 레이블이 사각형안에 표기되었다. 노드 번호가 앰퍼센드(“&”) 기호와 함께 노드 상단에 표기되었다. 본 논문에서 노드들의 집합은 N으로 표기되고 각 노드는 n_i로(i는 노드 번호) 표기된다. 예를 들어 24번 노드는 n_{24}로 표기된다. 루트(root) 노드의 깊이는 0이다. 부모 노드의 깊이가 d일 때 자식 노드의 깊이는 $(d+1)$이다.

정의 1. (ancestors(), descendants()) 노드 $n \in N$을 고려하자. “ancestors(n)”은 노드 n의 모든 선조 노드들

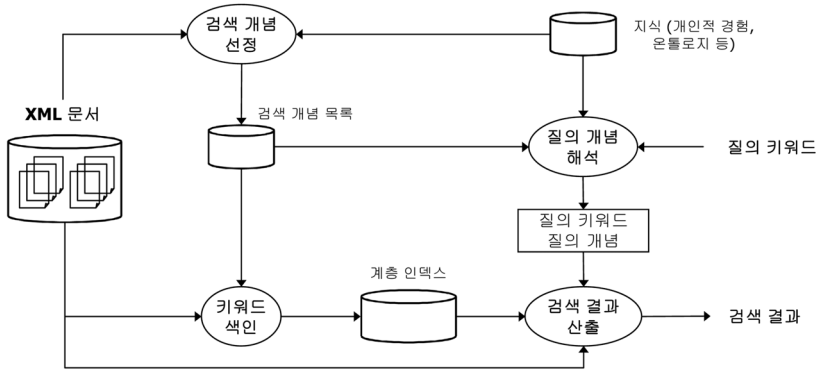


그림 1 의미 기반 키워드 검색 개요

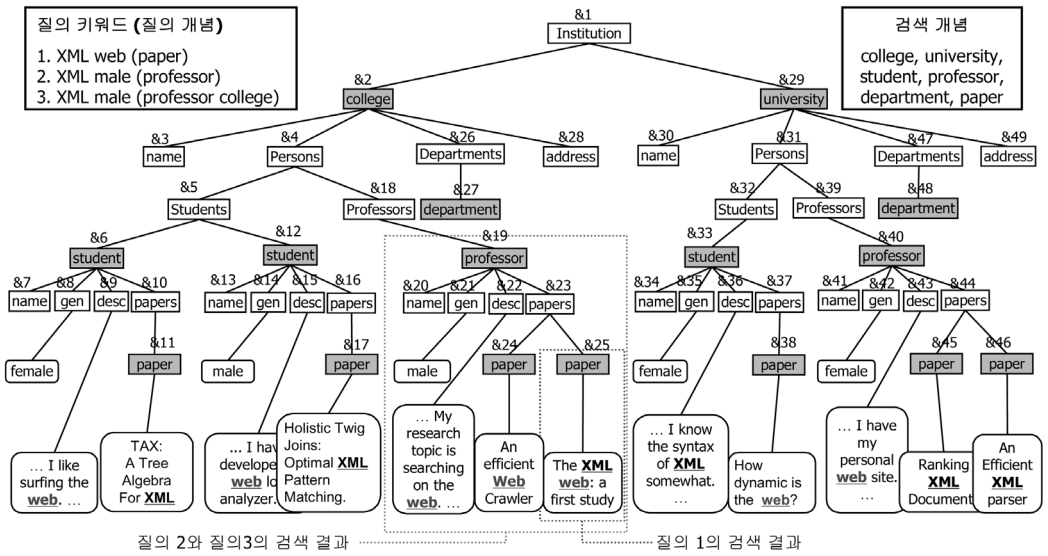


그림 2 예제 XML 트리

의 집합을 반환하는 함수이다. “descendants(n)”은 노드 n의 모든 후손 노드들의 집합을 반환하는 함수이다.

정의 2. (lca()) 두 노드 n_a, $n_b \in N$을 고려하자. N_a와 n_b의 최소 공통 선조 노드는 두 노드의 공통 선조 노드 중 깊이가 가장 큰 노드이다. “lca(n_a, n_b)”는 n_a, n_b의 최소 공통 선조 노드를 반환하는 함수이다.

정의 3. (속함, $\rightarrow$, $\Rightarrow$) 노드 n, 개념 c, 개념 집합 C를 고려하자. 노드 n의 개념(또는 레이블)이 c이거나, 노드 n의 선조 노드 중 개념이 c인 선조 노드가 존재하면, “노드 n이 개념 c에 속한다”고 하고, $n \nrightarrow c$로 표기한다. 노드 n이 개념 c에 속하지 않을 경우 $n \not\rightarrow c$로 표기한다. 노드 n이 개념 집합 C의 모든 개념에 각각 속할 경우 n이 개념 집합 C에 속한다고 하고 $n \Rightarrow C$로 표기한다.

예를 들어, 그림 2에서 n_{17}의 레이블이 “paper” 이므로 n_{17}이 “paper” 개념에 속한다고 한다($n_{17} \rightarrow$ “paper”). 또한, n_{17}의 선조 노드인 n_{12}의 레이블이 “student”이므로, n_{17}은 “student” 개념에도 속한다($n_{17} \rightarrow$ “student”). N_{17}은 “student”와 “paper” 개념에 속하므로 n_{17}은 개념 집합 {“student”, “paper”}에 속한다($n_{17} \Rightarrow$ {“student”, “paper”}). N_{17}의 어떠한 선조 노드도 “university”을 레이블로 가지지 않으므로 n_{17}은 “university” 개념에 속하지 않는다($n_{17} \rightarrow$ “university”). 또한 $n_{17} \Rightarrow$ {“student”, “paper”, “university”}가 성립하지 않는다.

성질 1. 노드 $n \in N$, n의 선조 노드 n_a(즉 $n_a \in$ ancestors(n)), 개념 c $\in SC$를 고려하자. n $\nrightarrow$ c이면, $\forall n_a \nrightarrow c$ 이다.

성질 2. 노드 $n \in N$, n의 후손 노드 n_d(즉 $n_d \in$ descendants(n)), 개념 c $\in SC$를 고려하자. n $\rightarrow$ c이면, $\forall n_d \rightarrow c$이다.

성질 1은 원소 n이 개념 c에 속하지 않으면, n의 모든 선조 노드가 개념 c에 속하지 않음을 나타낸다. 성질 2는 노드 n이 개념 c에 속할 경우 노드 n의 모든 후손 노드가 개념 c에 속함을 나타낸다.

4. 의미 기반 계층 인덱스

4.1 구성 개념 및 원리

본 장에서는 개념 간의 관계의 종류를 정의하고 의미 기반 계층 인덱스의 구성 원리를 이론적으로 기술한다.

정의 4 (검색 개념, 질의 개념). 개념 집합 SC는 검색 결과로 반환 가능한 개념들의 집합이다. 검색 결과로 반환되는 XML 원소의 레이블은 SC의 원소 중 하나이다. 질의 개념 집합 QC는 사용자가 검색하고자 하는 개념들의 집합이다.

검색 결과로 반환하려는 개념이 없거나 사용자가 검색하려는 개념 또는 키워드가 없으면 검색 자체가 성립하지 않으므로 SC와 QC는 공집합이 아니다. 또한 QC는 SC의 부분 집합이 다. 다수의 키워드들이 하나의 질의로 입력될 경우 접속적(conjunctive) 의미와 분리적(disjunctive) 의미로 해석 가능하다. 접속적 의미의 해석에서는 질의 키워드를 모두 포함한 원소가 검색 결과로 반환되고, 분리적 의미의 해석에서는 질의 키워드들 중 하나 이상의 키워드를 포함한 원소가 검색 결과로 반환된다. 본 논문에서는 접속적 의미로 질의를 해석한

다. 두개 이상의 질의 개념에 대해서도 접속적 의미로 해석한다. 즉 검색 결과 노드는 모든 검색 개념에 동시에 속하여야 한다.

주어진 질의 키워드와 질의 개념에서, 검색 결과는 다음 네 가지 조건을 모두 만족하는 검색 결과 노드들의 집합이다.

(1) 검색 결과 노드는 질의 키워드를 모두 포함한다.
(2) 검색 결과 노드의 개념은 QC의 개념 중 하나이다.
(3) 검색 결과 노드는 QC의 모든 개념에 속한다
(4) 위의 조건 (1), (2), (3)을 만족하는 검색 결과 노드의 하위 노드가 없다.

예를 들어, 그림 2에서 SC = {"college", "university", "student", "professor", "department", "paper"}일 때, 세 가지 질의를 보자. 질의 1(K={"XML", "web"}, QC={"paper"})은 키워드 "XML"과 "web"을 포함한 논문("paper")을 찾는 질의이다. N_{25}는 키워드 "XML"과 "web"을 포함하고 있다. N_{25}는 "paper"개념에 속하며 ($n_{25} \rightarrow$ "paper"), n_{25}의 개념은 "paper"이다. 또한 n_{25}의 후손 노드 중 "XML"과 "web" 키워드를 포함하고, "paper"에 속하며 개념이 "paper"인 원소가 없다. 따라서 n_{25}는 질의 1의 결과 원소가 된다. 질의 2(K={"XML", "male"}, QC={"professor"}), 질의 3(K={"XML", "male"}, QC={"college", "professor"})의 검색 결과는 n_{19}가 된다. 원소 n_{12}는 질의 2와 질의 3의 키워드를 모두 포함하고 있으나 $n_{12} \Rightarrow\!\!\!\!\!\!\rightarrow$ QC를 만족하지 않으므로 검색 결과가 되지 못한다.

서로 다른 두 개념 c_i, c_j를 고려하자. 본 논문에서는 한 XML 트리에서 개념이 c_i이면서 c_j에 속하는 노드 n_a와 개념이 c_j이면서 c_i에 속하는 노드 n_b가 동시에 존재하지 않는 것으로 간주한다. 또한 개념 c에 속하지만 레이블이 c가 아닌 노드가 있을 때, 그 후손 노드 중 레이블이 c인 노드가 없는 것으로 간주한다.

정의 5 (독립관계, $\perp$). 서로 다른 두 개념 c_i, $c_j \in SC$, $n \in N$을 고려하자. 두 개념에 동시에 속하는 노드가 존재하지 않으면, c_i과 c_j의 관계를 독립 관계라고 하며, $c_i \perp c_j$로 표기한다.

정의 6 (상하관계, $\vdash$ 또는 $\dashv$) 서로 다른 두 개념 c_i, $c_j \in SC$, $n \in N$을 고려하자. $N \rightarrow c_i$, $n \rightarrow c_j$ 일 때, $\exists n_a \in ancestors(n)$, $n_a \rightarrow c_i$, $n_a \not\rightarrow c_j$이면, c_i을 c_j의 상위 개념이라고 하고 c_j를 c_i의 하위 개념이라고 한다. 이때 c_i와 c_j의 관계를 $c_i \dashv c_j$ 또는 $c_j \vdash c_i$로 표기한다.

그림 2에서 "paper"와 "professor" 개념에 동시에 속한 모든 원소들은 그 상위 원소 중에 "professor"에 속하고 "paper"에 속하지 않는 원소들이 존재한다. 예를 들어, n_{25}는 "paper"와 "professor" 개념에 동시에 속하고, n_{25}의 선조 노드 중 n_{23}은 "professor" 개념에 속하나 "paper" 개념에 속하지 않는다. 따라서 "paper"는 "professor"의 하위 개념이고, "paper" $\vdash$ "professor" 관계가 성립한다. "student"와 "professor"에 동시에 속하는 노드가 존재하지 않는다. 따라서 두 개념 사이에는 "student" $\perp$ "professor" 관계가 성립한다.

보조정리 1. $n_a \rightarrow c_i$, $n_b \rightarrow c_j$, $n = lca(n_a, n_b)$, $c_i \perp c_j$ 이면, $n \not\rightarrow c_i$, $n \not\rightarrow c_j$ 이다.

증명. $n \rightarrow c_i$ 라면, $n_b \rightarrow c_i$ 성립한다(성질 2). 즉, $n_b \rightarrow c_i$, $n_b \rightarrow c_j$가 성립되므로, 주어진 조건 $c_i \perp c_j$에 위배된다. $N \rightarrow c_j$ 라면, $n_a \rightarrow c_j$ 성립한다(성질 2). 즉, $n_a \rightarrow c_i$, $n_a \rightarrow c_j$가 성립되므로, 주어진 조건 $c_i \perp c_j$에 위배된다. 따라서, $n \not\rightarrow c_i$이고 $n \not\rightarrow c_j$이다.

정의 7. (관련 검색 개념 집합, RSC). 노드 n의 관련 검색 개념 집합 RSC는 노드 n이 속한 모든 개념들의 집합이다(즉, $RSC = \{c \mid c \in SC, n \rightarrow c\}$). 이때, $n \Rightarrow RSC$로 표기하고 노드 n은 RSC의 멤버 노드라고 한다.

성질 3. $n \Rightarrow RSC$이면, $RSC \neq \varnothing$이고 $n \Rightarrow\!\!\!\!\!\!\rightarrow RSC$가 성립한다.

예제. XML 트리에서 총 12개의 RSC가 존재한다. RSC_1 = {"college"}, RSC_2 = {"college", "student"}, RSC_3 = {"college", "student", "paper"}, RSC_4 = {"college", "professor"}, RSC_5 = {"college", "professor", "paper"}, RSC_6 = {"college", "department"}, RSC_7 = {"university"}, RSC_8 = {"university", "student"}, RSC_9 = {"university", "student", "paper"} RSC_{10} = {"university", "professor"}, RSC_{11} = {"university", "professor", "paper"}, RSC_{12} = {"university", "department"}가 있다. N_{24}는 RSC_4의 개념 집합에 나타나지 않는 "paper" 원소에 속하므로 $n_{24} \Rightarrow\!\!\!\!\!\!\rightarrow RSC_4$이지만 $n_{24} \Rightarrow RSC_4$가 성립하지 않는다. 노드 n_{23}는 RSC_4에 나타난 개념들에만 속하므로 $n_{23} \Rightarrow\!\!\!\!\!\!\rightarrow RSC_4$이고 $n_{23} \Rightarrow RSC_4$가 성립한다. 따라서 노드 n_{23}는 RSC_4의 멤버이다.

정리 1. $QC \subset SC$, $n_a \Rightarrow RSC_i$, $n_b \Rightarrow RSC_j$ $(i \neq j)$, $n = lca(n_a, n_b)$을 고려하자. $QC \not\subset RSC_i$ 또는 $QC \not\subset RSC_j$ 이면, $n \Rightarrow\!\!\!\!\!\!\rightarrow QC$가 아니다.

증명. (1) $QC \not\subset RSC_i$이므로, $c_q \in QC$, $c_q \notin RSC$를 만족하는 c_q가 존재한다.
(2) n_a는 RSC_i의 개념들에만 속하고 RSC_i의 원소가 아닌 개념에는 속하지 않으므로(정의 7), $n_a \not\rightarrow c_q$이다.
(3) $n_a \not\rightarrow c_q$이고 n ∈ $ancestors(n_a)$이므로, $n \not\rightarrow c_q$이다(성질 1).
(4) $n \not\rightarrow c_q$, $c_q \in QC$ 이므로, $n \Rightarrow\!\!\!\!\!\!\rightarrow QC$가 아니다.

정리 1에 따르면, 질의 개념 QC가 주어졌을 때, QC를 부분집합으로 가지지 않는 두 관련 검색 개념 집합

RSC_i와 RSC_j (즉, $i \neq j$, $QC \not\subset RSC_i$ 또는 $QC \not\subset RSC_j$)의 멤버 노드들($n_a \Rightarrow RSC_i$, $n_b \Rightarrow RSC_j$)의 조합으로 산출된 최소 공통 선조 노드($n = lca(n_a, n_b)$)는 검색 개념 모두에 동시에 속하지 않으므로($n \not\Rightarrow QC$가 아님) 검색 결과가 될 수 없다. 따라서 검색 결과 산출을 위하여 QC를 부분집합으로 가지지 않는 RSC의 멤버 노드들을 인덱스에서 불필요하게 읽을 필요가 없다.

정리 2. $QC \subset SC$, $n_a \Rightarrow RSC_i$, $n_b \Rightarrow RSC_j$ $(i \neq j)$, $n = lca(n_a, n_b)$, $RSC_i \supset QC$, $RSC_j \supset QC$을 고려하자. $\exists c_x \in (RSC_i - RSC_j)$, $\exists c_y \in (RSC_j - RSC_i)$, $\exists c_q \in QC$, $c_x \perp c_y$, $c_q \vdash c_x$, $c_q \vdash c_y$ 이면, $n \not\Rightarrow QC$가 아니다.

증명. (1) $n_a \Rightarrow RSC_i$이므로, $n_a \not\Rightarrow RSC_i$ 이다. (성질 3)

(2) $n_b \Rightarrow RSC_j$이므로, $n_b \not\Rightarrow RSC_j$ 이다. (성질 3)

(3) $n_a \not\Rightarrow RSC_i$이고 $c_x \in (RSC_i - RSC_j)$이므로, $n_a \rightarrow c_x$이다. (정의 1)

(4) $n_b \not\Rightarrow RSC_j$이고 $c_y \in (RSC_j - RSC_i)$이므로, $n_b \rightarrow c_y$이다. (정의 1)

(5) $\exists c_q \in QC$, $RSC_i \supset QC$, $RSC_j \supset QC$이므로, $n_a \rightarrow c_q$이고 $n_b \rightarrow c_q$이다.

(6) $c_x \perp c_y$이므로, $n \not\rightarrow c_x$이고 $n \not\rightarrow c_y$이다. (정의 5)

(7) $c_q \vdash c_x$, $c_q \vdash c_y$이므로, $n \not\rightarrow c_q$이다. 만약 $n \rightarrow c_q$라고 가정하면,

(가) $n_a \rightarrow c_x$, $n_a \rightarrow c_q$, $n \not\rightarrow c_x$이므로, $c_q \vdash c_x$가 아니다(대신 $c_q \dashv c_x$가 성립). (정의 6)

(나) $n_b \rightarrow c_y$, $n_b \rightarrow c_q$, $n \not\rightarrow c_y$이므로, $c_q \vdash c_y$가 아니다(대신 $c_q \dashv c_y$가 성립). (정의 6)

(8) 마지막으로, $n \not\rightarrow c_q$이므로 $n \not\Rightarrow QC$가 아니다.

정리 2에 따르면, 질의 개념 QC가 주어졌을 때, RSC_i와 RSC_j($i \neq j$) 모두 QC를 부분집합으로 가진다 하더라도(다시 말하면 정리 1을 만족하지 않기 때문에 RSC_i와 RSC_j의 멤버 노드간 최소 공통 선조 노드가 질의 결과 후보로 고려될 수 있으나), RSC_i와 RSC_j 둘 중 하나의 관련 검색 개념 집합에만 속하는 개념들이 서로 존재하고 서로 독립이며($\exists c_x \in RSC_i - RSC_j$, $\exists c_y \in (RSC_j - RSC_i)$, $c_x \perp c_y$), 독립된 두 개념이 QC의 어느 한 개념보다 상위 개념이면($\exists c_q \in QC$, $c_q \vdash c_x$, $c_q \vdash c_y$), RSC_i의 멤버 노드($n_a \Rightarrow RSC_i$)와 RSC_j의 멤버 노드($n_b \Rightarrow RSC_j$)의 조합으로 산출된 최소 공통 선조 노드는 모든 검색 개념들에 동시에 속하지 않으므로 검색 결과가 될 수 없다.

정리 1과 정리 2는 주어진 질의 개념에 대해 결과 생성이 불가능한 관련 개념 집합 간의 관계를 나타낸다. 즉, 정리 1 또는 정리 2를 만족하는 두 관련 개념 집합 각각에 속하는 원소들의 조합은 검색 결과가 될 수 없다. 그러므로 우리는 이를 이용하여 결과 생성에 소요되는 비교 연산 비용을 줄일 수 있다.

정의 8 (비교 유닛, CU). QC, RSC_i, RSC_j를 고려하자. RSC_i와 RSC_j가 다음 두 조건 중 하나를 만족할 때, RSC_i와 RSC_j 사이의 관계를 주어진 질의 개념들에 대해 "합병 가능" 관계라고 한다. 이 때 비교 유닛 CU는 "합병 가능" 관계에 있는 RSC들에 속한 노드들의 집합이다.

- 조건 1 : $RSC_i \supset RSC_j \supset QC \vee QC \subset RSC_i \subset RSC_j$
- 조건 2 : $\exists c_x \in (RSC_i - RSC_j)$, $\exists c_y \in (RSC_j - RSC_i)$, $\forall c_q \in QC$ $(c_x \vdash c_q \wedge c_y \vdash c_q)$

주어진 질의 개념에 대해 관련 개념 집합들로 구성되는 비교 유닛 각각은 결과 가능성을 갖는 노드 들의 집합이라고 할 수 있다. 예를 들어, 질의 1(QC = {"paper"})에 대해서, 4개의 비교 유닛 CU_1 = {n | n $\Rightarrow RSC_3$}, CU_2 = {n | n $\Rightarrow RSC_5$}, CU_3 = {n | n $\Rightarrow RSC_9$}, CU_4 = {n | n $\Rightarrow RSC_{11}$}가 존재한다. 질의 2(QC = {"professor"})에 대해서는, 2개의 비교 유닛 CU_1 = {n | n $\Rightarrow RSC_4$ 또는 n $\Rightarrow RSC_5$}, CU_2 = {n | n $\Rightarrow RSC_{10}$ 또는 n $\Rightarrow RSC_{11}$}가 존재하며 질의 3(QC = {"college", "professor"})에 대해서는 하나의 비교 유닛 CU_1 = {n | n $\Rightarrow RSC_4$ 또는 n $\Rightarrow RSC_5$}가 존재한다.

그림 3은 그림 2의 예제 XML 트리에 대한 키워드 "XML", "web", "male"의 포스팅을 저장하고 있는 의미 기반 계층 인덱스, 분할 인덱스, 기존 인덱스 구조를 개념적으로 나타낸다. 의미 기반 계층 인덱스에서 한 키워드의 포스팅은 해당 키워드를 포함한 원소들이 속한 개념들에 따라 구분되어 저장된다. 그림 3(a)는 계층 인덱스를 개념적으로 나타내고 있다. "paper" 개념에 속하는 노드 n_{11}, n_{17}, n_{24}, n_{25}, n_{38}, n_{45}, n_{46}은 "student"와 "professor" 개념 별로 나뉜다. 그 중 "student" 개념(또는 "professor" 개념)에 속한 노드 n_{11}, n_{17}, n_{38}(또는 n_{24}, n_{25}, n_{45}, n_{46})은 "college"와 "university" 개념 별로 나뉘어 저장된다. "paper" 개념에는 속하지 않으면서 "professor" 개념에 속한 노드들(n_{24}, n_{25}, n_{45}, n_{46})은 "college"와 "university" 개념의 종속 여부에 따라 나뉘어 저장된다. 그림 3(b)는 분할 인덱스[5]의 개념적 구성을 나타낸다. 분할 인덱스는 최하위 검색 결과 깊이의 원소들을 분할 요소 δ개 파티션들로 분할하여 저장하고 나머지 원소들을 최하위 깊이에서 조상 노드가 저장된 파티션과 동일한 파티션에 저장한다. 예제 XML 트리에서 검색 결과의 최하위 깊이는 1이다. 파티션 요소 δ가 2라고 하면 깊이 1의 원소들을 두개의 파티션들로 나누어 저장한다. 깊이 1의 첫번째 원소 n_2와 그 후손 노드들은 첫번째 파티션에 저장되고 깊이 1의 두 번째 원소는 n_{29}와 그 후손 노드들은 두 번째 파티션에 저

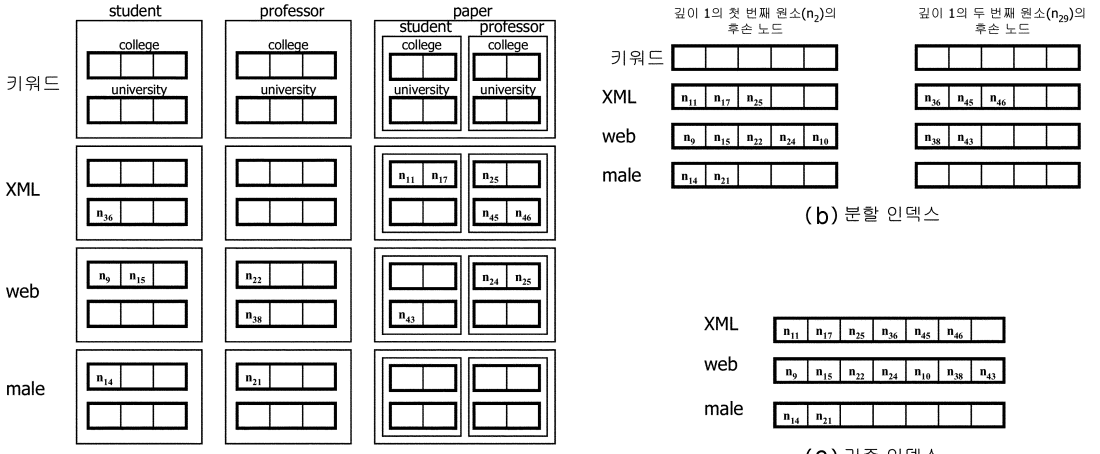


그림 3 의미 기반 계층 인덱스, 분할 인덱스, 기존 인덱스

장된다. 그림 3(c)는 한 키워드의 포스팅이 나뉘지 않고 순차적으로 XML 원소들을 저장하고 있는 일반적인 인덱스를 나타낸다.

그림 3(a)에서 한 RSC의 멤버 노드들, 그림 3(b)에서 하나의 파티션에 존재하는 노드들, 그림 3(c)에서 한 키워드의 포스팅을 구성하는 노드들이 DIL 구조로 구성되는 경우를 가정하여 결과 산출에 필요한 노드간 조합 회수를 비교하면 다음과 같다. 그림 2에서 질의 1을 고려하자(QC = {“paper”}, K={”XML”, “web”}). 정리 1에 따라 QC를 부분집합으로 가지는 RSC_3, RSC_5, RSC_9, RSC_{11}를 대상으로 하여, 정리 2에 따라 4개의 비교 유닛 CU_1 = {n | n⇒RSC_3}, CU_2 = {n | n⇒RSC_5}, CU_3 = {n | n⇒RSC_9}, CU_4 = {n | n⇒RSC_{11}}을 정의한다. CU_1의 멤버이며 “XML” 키워드를 포함하는 노드들의 집합은 {n_{11}, n_{17}}이다. CU_1의 멤버이며 “web” 키워드를 포함하는 노드들의 집합이 ∅이므로, n_{11}과 n_{17}은 “web” 키워드를 포함하는 어떠한 노드와의 조합에서도 검색 결과가 되는 최소 공통 선조가 반환되지 않는다. 따라서 “XML” 키워드를 포함하는 n_{11}과 n_{17}은 읽힐 필요가 없다. CU_2의 멤버이며 “XML” 키워드를 포함하는 노드들의 집합은 {n_{25}}이다. CU_2의 멤버이며 “web” 키워드를 포함하는 노드들의 집합은 {n_{24}, n_{25}}이다. 두 번의 최소 공통 선조를 산출하게 된다. lca(n_{25}, n_{24}) = n_{23}이고 n_{23} ⇏ QC가 아니다. lca(n_{25}, n_{25}) = n_{25} 이고, n_{25} ⇒ QC가 ●이므로, n_{25}가 검색 결과로 반환된다. CU_3의 멤버이면서 키워드 “XML”을 포함하는 노드가 없으므로, CU_3의 멤버이면서 키워드 “web”을 포함하는 노드를 읽을 필요가 없다. CU_4의 멤버이면서 키워드 “web”을 포함하는 노드가 없으므로 CU_4의 멤버이면서 키워드 “XML”을 포함하는 노드를 읽을 필요가 없다. 결론적으로 의미 기반 계층 인덱스를 사용하는 경우에 질의 1을 처리하기 위하여 RSC_5에 해당하는 키워드 “XML”의 포스팅과 키워드 “web”의 포스팅만 읽히면 되며, 2번의 최소 공통 선조 연산만을 수행하면 된다.

분할 인덱스에서는 질의 개념에 상관없이 포스팅들의 분할 관계에 의거해서 검색결과를 찾는다. 그림 3(b)에서 동일 분할에 속하고 키워드 “XML”를 포함한 노드와 키워드 “web”을 포함한 노드들간의 최소 공통 선조 산출 연산을 수행한다. 즉 11번의 최소 공통 선조 산출 연산을 수행한다. 첫 번째 분할에서 7번의 연산 lca(n_{11}, n_9), lca(n_{11}, n_{15}), lca(n_{17}, n_{15}), lca(n_{17}, n_{22}), lca(n_{25}, n_{22}), lca(n_{25}, n_{24}), lca(n_{25}, n_{25}), 두 번째 분할에서 4번의 최소 공통 선조 산출 연산 lca(n_{36}, n_{38}), lca(n_{45}, n_{38}), lca(n_{45}, n_{43}), lca(n_{46}, n_{43})을 수행한다. 기존 인덱스에서는 하나의 키워드가 하나의 포스팅을 가지고 있으며, 그림 3(c)에서 총 12번의 최소 공통 선조 산출 연산이 발생한다.

3.2 인덱스 생성 및 검색 과정

색인 과정에서는 XML 문서를 순차적으로 읽으면서 색인될 키워드를 포함한 노드가 발견되면 동일 RSC의 멤버가 되는 노드들로 포스팅을 구성한다. 노드들이 발견되는 순서는 XML 트리를 프리오더(preorder)로 탐색할 때 노드가 조회되는 순서와 같다. 그림 2의 예제 XML 트리에 나타난 노드 번호는 각 노드가 발견되는 순서를 의미한다.

그림 4는 계층 인덱스의 색인 과정을 나타낸다. 불용어(stopword) 처리나 스테밍(stemming)과 같이 색인의

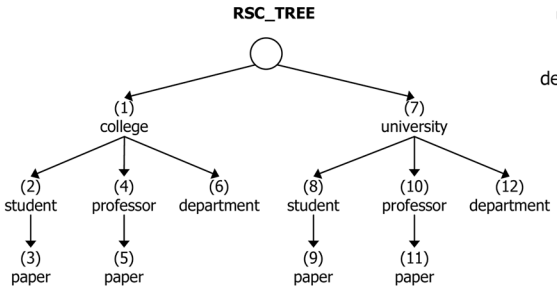
```
while (read(doc)) {
    start_element() {
        node_id = get_node_identifier();
        term = get_term();
        LABEL_STACK.push(term);
        if (SC_LOOKUP.isSearchConcept(term)) {
            RSC_TREE.put(term);
            rsc_id = RSC_TREE.getRscIdentifier();
        }
    }
    characters() {
        term = get_term();
        if (term is keyword?)
            if (IDX_HEADER.getAddress(rsc_id) is NULL)
                IDX_ELIST.createNewPosting(term, rsc_id, node_id);
                IDX_HEADER.setAddress(term, rsc_id, IDX_ELIST.getAddress(term, rsc_id));
            } else {
                IDX_ELIST.putTerm(term, rsc_id, node_id);
            }
    }
    end_element() {
        term = LABEL_STACK.currentLable();
        if (SC_LOOKUP.isSearchConcept(term))
            RSC_TREE.pop();
        LABEL_STACK.pop();
    }
}
```

그림 4 계층 인덱스 생성 과정

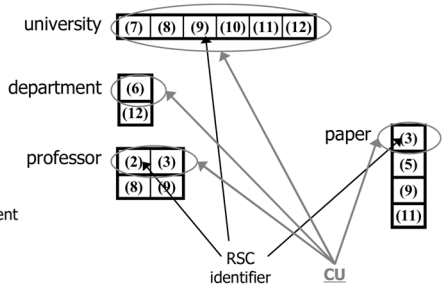
일반적 내용은 생략되었다. 색인을 위한 입력은 검색 개념 목록(SC_LOOKUP)과 색인될 문서들이다. 색인 결과로는 계층 인덱스, 계층 인덱스 헤더, 관련 개념 집합 트리가 생성된다. IDX_ELIST, IDX_HEADERS, RSC_TREE는 각각 계층 인덱스, 계층 인덱스 헤더, 관련 개념 집합 트리를 나타낸다. IDX_LIST에는 키워드들의 포스팅들이 저장된다. 동일 키워드를 포함한 노드들은 다수의 포스팅으로 구성되는데 동일 RSC의 멤버 노드들끼리 하나의 포스팅을 구성한다. 동일 RSC의 멤버 노드들은 필요에 따라 다양한 구조로 구성될 수 있다(예를 들어, DIL 구조[3]). IDX_HEADERS는 IDX_ELIST에서 각 키워드의 포스팅들의 시작 위치를 저장하고 있다. 색인된 모든 키워드들은 IDX_HEADERS에서 그림 5(c)와 같은 헤더를 보유하고 있으며 IDX_LIST에서 해당 포스팅들의 시작 위치를 가리키고 있다. RSC_TREE는 RSC들의 관계를 트리 구조로 나타내는 그래프이다. 예제 XML에 대한 RSC_TREE는 그림 5(a)와 같이 생성된다.

"start_element()", "characters()", "end_element()"는 "원소 시작", "단어 시작", "원소 끝" 사건(event)에 의해 트리거되는(triggered) 함수이다. "get_node_identifier()"는 현재 원소의 노드 식별자를 반환한다. "get_term()" 함수는 XML 문서에서 한 단어를 읽고 반환한다. "SC_LOOKUP.isSearchConcept()" 함수는 현재 트리 노드의 레이블이 검색 개념 중 하나이면, "참"을 반환한다. RSC_TREE.put() 함수는 입력되는 개념을 포함하는 RSC의 존재 여부를 RSC 트리에서 확인하고 존재하지 않을 경우 새로운 RSC 노드와 RSC 번호를 생성한다. IDX_ELIST.createNewPosting(term, rsc_id, node_id)은 키워드 term의 RSC 번호 rsc_id에 대한 새로운 포스팅을 생성하고 새로 생성된 포스팅에 node_id를 추가한다. IDX_ELIST.getAddress(term, rsc_id)는 term의 rsc_id에 해당하는 포스팅의 시작 주소를 반환한다. IDX_HEADER.setAddress(term, rsc_id, address)는 term의 포스팅 헤더의 rsc_id 번째에 해당 포스팅의 주소 address를 저장한다.

예를 들어 노드 n_1부터 n_{16}까지의 색인 과정을 설명하면 다음과 같다. n_1은 어떠한 개념에도 속하지 않고 RSC도 가지지 않는다. 따라서 n_1에 포함된 키워드들은 색인되지 않는다. n_2는 "college" 개념에 속하고 RSC_1 = {"college"}의 멤버이다. RSC_TREE에는 레이블이 "college"인 노드가 생성된다. N_5까지 새로 발견되는 개념이 없으므로, RSC_TREE에 추가되는 개념 노드가 없고, n_2부터 n_5까지의 노드들은 RSC_1={"college"}의 멤버가 된다. N_2부터 n_5까지 발견되는 키워드들의 포스팅은 동일한 영역에 저장된다. N_6의 레이블은 "student"이



(a) RSC 트리



(b) 질의 개념에 따른 CU 구성 예

(1) {college}	(2) {college, student}	(3) {college, student, paper}	(4) {college, professor}	(5) {college, professor, paper}	(6) {college, department}	(7) {university}	(8) {university, student}	(9) {university, student, paper}	(10) {university, professor}	(11) {university, professor, paper}	(12) {university, department}

(c) 키워드 별 인덱스 헤더

그림 5 자료 구조

고 검색 개념 중 하나이므로 RSC 트리에 개념 노드가 추가된다. 즉 n_6은 “student”에도 속하고 “college”에도 속하여 RSC_2={“college”, “student”}의 멤버노드이다. “student”는 “college”를 레이블로 가지는 노드의 하위 노드에서 발견된 개념이므로 RSC_TREE에서 “college” 개념 노드의 자식 노드로 생성된다. N_{10}까지 새로 발견된 검색 개념이 없으므로, n_6부터 n_{10}까지의 노드들은 RSC_2={“college”, “student”}의 멤버가 되고 n_6부터 n_{10}에서 발견된 키워드들의 포스팅은 동일 영역에 저장된다. N_{11}은 RSC_3={“college”, “student”, “paper”}의 멤버가 된다. N_{12}부터 n_{16}까지의 노드들은 n_6부터 n_{10}까지의 노드들과 마찬가지로 RSC_2의 멤버가 된다. 따라서 노드에서 발견된 키워드 n_{11}을 제외하고 n_6부터 n_{16}까지의 포스팅은 모두 동일 영역에 저장된다.

검색 과정은 비교 유닛을 구성하는 단계와 동일 비교 유닛에 속한 원소들간의 최소 공통 선조를 산출하여 검색 결과를 산출하는 두 단계로 구성된다. 그림 5(b)는 질의 개념이 “university”, “department”, “professor”, “paper”일 때 CU 구성 예를 개념적으로 나타낸다. 질의 개념이 “university”일 경우 RSC_7, RSC_8, RSC_9, RSC_{10}, RSC_{11}, RSC_{12}의 멤버 노드들로 구성되는 하나의 비교 유닛이 고려된다. 질의 개념이 “department”일 경우 RSC_6의 멤버 노드들로 구성되는 비교 유닛과 RSC_{12}의 멤버 노드들로 구성되는 비교 유닛이 고려된다.

의미 기반 계층 인덱스를 이용한 검색 과정은 그림 6과 같다. 검색은 색인 단계에서 생성된 RSC_TREE, IDX_HEADERS, IDX_ELIST가 이용된다. CU 변수는 주어진 질의 개념에 대해 비교 유닛을 구성하는 노드가 저장된 포스팅의 물리적 주소가 저장된 3차원 배열이다. CU[cu_num][k][cu_rsc_num]는 cu_num번째 비교 유닛에서 k번째 키워드의 cu_rsc_num번째 RSC의 멤버 원소들이 저장된 물리적 주소 값을 가진다. 예를 들어, 질의 2 (QC={“professor”}, K = {“XML”, “male”})에서 비교 유닛 CU_0 = {n | n ⇒ RSC_4 또는 n ⇒ RSC_5}를 고려하자. 첫번째 키워드 “XML”에 대해 CU는 CU[0][0][0] = IDX_HEADER.getAddress(“XML”, 4), CU[0][0][1] = IDX_HEADER.getAddress(“XML”, 5)로 구성된다. 두 번째 키워드 “male”에 대해서는 CU[0][1][0] = IDX_HEADER.getAddress(“male”, 4), CU[0][1][1] = IDX_HEADER.getAddress(“male”, 5)로 구성된다. k번째 키워드의 cu_rsc_num번째 RSC의 멤버 원소들이 존재하지 않을 경우 널(NULL)이 할당된다. “rsc_cnt” 변수는 현재 비교 유닛을 구성하는 RSC의 개수이다. RSC_TREE.traverse()는 프리오더(pre-order) 순서로 RSC_TREE를 순회하는 함수이다. RSC_TREE.getRSCIdentifier()는 RSC_TREE의 현재 포인터가 가리키고 있는 RSC의 번호를 반환한다. LoadRscMembers() 함수는 각 키워드별로 비교 유닛을 구성하게 될 RSC의 멤버들이 저장되어 있는 물리적 주소를 CU 변수에 적재한다. RSC_TREE의 현재 RSC 노드가 QC가 아니면 더 이상 현재 CU에 추가될 RSC가 없음을 의미한다. CU에 속한 RSC가 존재하는 경우에 CU에 포함된 원소들로부터 최소 공통 선조를 산출하는 연산을 수행한다.

RSC_TREE에서 반환된 현재 RSC가 QC에 포함되지 않으면 CU 구성이 끝난 것이므로, 현재까지 CU를 구성하던 RSC들의 멤버 원소들로 질의 결과를 산출한다. 비교 유닛을 구성하는 한 키워드의 모든 RSC에서 멤버

```
// CU_CNT: 비교 유닛 개수,  K: 질의 키워드 개수
// RSC_MAX_CNT: 하나의 비교 유닛을 구성하는 RSC의 최대 개수

Declare CU[CU_CNT][K][RSC_MAX_CNT];   // 비교 유닛을 구성하는 RSC 멤버 원소들의 저장
                                      // 위치를 가리키는 3차원 포인터 배열
Declare cu_num = 0;                   // 현재 비교 유닛의 번호
Delcare cu_rsc_num = 0;               // 비교 유닛을 구성하고 있는 RSC의 개수
Declare rsc=NULL;                     // RSC_TREE의 노드 하나를 가리키는 포인터

While (RSC_TREE.traverse()) {

    rsc = RSC_TREE.getCurrentRSC();
    if rsc ⊃ QC
        rsc_id = RSC_TREE.getRSCIdentifier ();
        for k = 0 to K
            loadRSCMembers(CU[cu_num][k][cu_rsc_num], IDX_HEADER[rsc_id];
        cu_rsc_num++;
    else
        // CU 구성이 끝났으면, LCA 산출 시작
        if ((rsc_cnt <> 0) && !check_empty_CU(CU))
            LCAs = find_LCA_in_CU (CU[cu_num], IDX_ELIST)
        cu_rsc_num = 0;
        cu_num++;
}
```

그림 6 검색 과정

원소가 없을 경우, 다른 키워드의 동일 비교 유닛을 구성하는 RSC의 멤버 원소들을 읽을 필요가 없다. “check_empty_CU” 함수는 각 키워드의 CU에 속한 모든 RSC의 IDX_ELIST의 주소가 널이면 “참”을 반환한다. 반대로 모든 질의 키워드들의 비교 유닛이 하나 이상의 멤버 원소들을 포함할 경우 각 비교 유닛으로부터 검색 결과가 되는 최소 공통 선조를 찾는다. “find_LCA_in_CU”는 각 질의 키워드들의 비교 유닛에서 질의 결과가 되는 최소 공통 선조를 산출하는 함수이다. 각 비교 유닛을 구성하는 RSC의 멤버 원소들의 목록들은 하나의 포스팅으로 간주가 가능하며, 최소 공통 선조를 검색하는 알고리즘들[3,4]이 사용될 수 있다.

4. 실험 및 평가

본 실험은 제안된 의미 기반 계층 인덱스의 구현 가능성과 계층 인덱스 사용에 따른 검색 성능 향상 정도를 보이는 것에 목적을 둔다. 본 실험에서는 의미 기반 계층 인덱스, 분할 인덱스[5], DIL 구조의 기존 인덱스[3]가 비교된다. 검색 대상으로는 DBLP 사이트의 XML 문서와 INEX2003 테스팅 셋[17]에서 제공되는 XML 문서들이 사용되었다. 본 실험은 펜티엄-III 993MHz CPU와 1기가바이트(gigabytes) 메인 메모리를 지닌 기계에서 진행되었다. 우리는 C++로 XML 키워드 검색 시스템을 구현하였으며, 버클리(Berkeley) 데이타베이스[18]를 사용하여 인덱스를 저장하였다. 하나의 질의에 대한 응답 시간은 동일 질의를 다섯번 수행한 결과에 대해 최대값과 최소값을 제외한 평균 값이다.

실험에 사용된 DBLP XML 문서는 약 50만개의 논문 요약 정보 목록을 보유한 약 210 메가바이트(megabytes) 크기의 문서이다. DBLP XML 문서는 루트 노드에서 팬-아웃(fan-out) 되는 노드 각각에 하나의 논문 요약 정보가 기록된 구조를 지닌다. 즉 깊이가 1인 노드의 개수가 약 50만개에 이른다. 문서의 최대 깊이가 얕고 깊이 1에 존재하는 노드 하나는 하나의 논문 요약 정보를 포함하고 있다. 그림 7은 DBLP XML 문서의 DTD에서 일부를 발췌한 것이다.

검색 개념은 “article”, “inproceedings”, “proceedings”, “phdthesis”, “book”, “author”이고, 질의 개념은 “proceedings”일 때, 표 1에 나타난 4개의 질의를 고려하자. 검색 개념들 간의 관계를 살펴보면 “author”를 제외한 나머지 개념들은 모두 서로 독립이며, “author” 개념은 나머지 개념들의 하위 개념이므로 RSC의 개수는 총 10개(RSC_1={“inproceedings”}, RSC_2={“proceedings”}, …, RSC_5={“book”}, RSC_6={“inproceedings”, “author”}, RSC_7={“proceedings”, “author”}, …, RSC_{10}={“book”, “author”})가 된다.

그림 8은 DIL 구조의 일반 인덱스, 분할 인덱스, 계층 인덱스의 검색 소요 시간을 나타낸다. 6개의 검색 개념을 고려하면 검색 결과의 최소 결과 깊이는 1이다. 분할 인덱스는 질의 개념에 상관없이 단순히 동일 파티션에 존재하는 노드들 간의 최소 공통 선조를 산출하게 된다. 두 종류의 분할 인덱스가 비교된다. 우선 계층 인

```
<! ELEMENT dblp (article | inproceedings | proceedings | book | incollection |
        phdthesis | mastersthesis | www)*>
<!ENTITY % field
        author|editor|title|booktitle|pages|year|address|journal|volume|number|month|url|
        ee|cdrom|cite|publisher|note|crossref|isbn|series|school|chapter>
<!ELEMENT article  (%field;)*>
                    …
<!ELEMENT www (%field;)*>
```

그림 7 DBLP XML 문서의 DTD

표 1 DBLP XML 문서에 대한 질의

질의번호	질의 키워드	질의번호	질의 키워드
Q1	Won, Kim, Relational, Query	Q3	Michael, David, Architecture, 2002
Q2	Database, System, 1996	Q4	Computer, System, Architecture, 2002

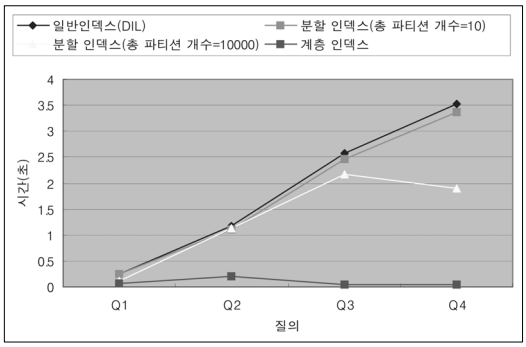


그림 8 DBLP XML 문서에 대한 검색 소요 시간

덱스의 RSC 개수와 동일하게 10개의 파티션을 가진 분할 인덱스가 비교된다. 10개의 파티션을 가진 분할 인덱스는 일반 인덱스와 비슷한 질의 응답 시간을 나타내었다. 파티션 개수가 약 10,000개일 때 분할 인덱스 사용에 따른 질의 응답 시간 감소가 크게 나타났으며 질의 4에서 약 46%의 응답 시간 감소를 나타냈다. 계층 인덱스를 사용할 경우에 "proceedings" 개념에 속하지 않는 노드들은 검색 결과 산출을 위하여 읽히지 않고 오직 "proceedings" 개념에 속하는 원소들 간의 비교로만 검색 결과 산출이 가능하다. 즉 "proceedings" 개념을 원소로 가지는 RSC(RSC_2, RSC_7)의 멤버 노드들만 읽히면 된다. 비교 유닛은 1개이며, RSC_2와 RSC_7의 멤버 노드들간의 최소 공통 선조를 산출하여 검색 결과를 산출할 수가 있다. 계층 인덱스를 사용하는 경우 질의 6에서 최대 99%의 응답 시간 감소를 보였다.

INEX 2003 테스팅 셋은 XML 검색의 성능 평가를 위한 XML 문서들과 질의들이 정의되어 있다. INEX-2003에는 125개의 XML 문서들이 있으며 각 XML 문서는 IEEE 저널들에 게재된 논문 제목과 내용을 포함하고 있다. XML 문서 하나의 크기는 평균 4.5 메가 바이트(mega bytes)이다. XML 문서 하나는 하나의 XML 트리가 되며 125개 XML 트리의 루트 노드를 자식 노드로 가지는 논리적 XML 트리를 구성하였다. 그림 9는 INEX2003 XML 문서들의 DTD에서 일부를 발췌한 것이다.

시험 질의는 크게 CO(Content Only)와 CAS(Content And Structure)로 구분된다. 본 논문에서는 XML 문서의 구조를 고려하지 않으므로 CO에 속하는 36개의 질

```
<!ELEMENT books        (journal*)>
<!ELEMENT journal      (title, issue, publisher, graphicc?, (sec1|article|sbt)*)>
<!ELEMENT sec1         (title)>
<!ELEMENT issue        (#PCDATA)>
<!ELEMENT publisher    (#PCDATA)>
<!ELEMENT article      (fno, doi?, fm, bdy, bm?)>
<!ELEMENT fno          (#PCDATA)>
<!ELEMENT doi          (#PCDATA)>
<!ELEMENT fm           (hdr?, (edinfo | au | tig | pubfm | abs | edintro | kwd | fig | figw)*)>
<!ELEMENT hdr          (fig?, hdr1, hdr2)>
<!ELEMENT hdr1         (#PCDATA | crt | obi | pdt | pp | ti)*>
<!ELEMENT hdr2         (#PCDATA | crt | obi | pdt | pp | ti)*>
<!ELEMENT bdy          (sec*)>
<!ELEMENT sec          (st, (p | ss1)*)>
```

그림 9 INEX XML 문서들의 DTD

의 중에서 6개의 질의를 선택하여 실험하였다. 6개 질의에 대한 키워드의 구성은 표 2에 나타나 있다. 6개의 선택된 질의 중 앞의 3개(Q1 ~ Q3)는 세 개의 질의 키워드로 구성된 비교적 간단한 질의들이고, 뒤의 질의 세 개(Q4 ~ Q6)는 최소 5개 이상의 키워드로 구성되는 상대적으로 복잡한 질의들이다.

검색 개념은 “journal”, “article”, “fm”, “body”일 때, XML 문서집합으로부터 각 개념간의 관계는 다음과 같다. “journal” ┤ “article”, “article” ┤ “fm”, “article” ┤ “body”, “fm” ⊥ “body”. RSC의 개수는 총 4개이다. 최소 검색 결과 깊이는 “journal” 개념에 속하는 원소들의 깊이인 2가 된다. 질의 개념이 “journal”과 “fm”일 때, 각 인덱스를 사용하여 검색 결과를 산출하는 응답 시간은 그림 10과 같다. 파티션 요소를 2로 설정할 경우 총 4개의 파티션이 존재한다. 분할 인덱스의 파티션 개수는 계층 인덱스의 RSC 개수와 동일한 4개와 분할 인덱스의 성능을 위하여 25개의 파티션으로 구성된 두 가지 경우로 하였다. 분할 인덱스를 사용하는 경우 파티션 개수가 4개인 분할 인덱스에서 평균 22%정도의 응답 시간 감소가 있었고 파티션 개수가 25개인 분할 인덱스에서 평균 70%의 응답 시간 감소가 있었다. 계층 인덱스를 사용하는 경우 “journal”과 “fm”에 속하는 원소들간의 조합으로 질의 결과를 찾게 되며, 평균적으로 87%의 응답시간 감소가 일어 났으며 질의 1에서 최대 92%의 응답 시간 감소가 일어났다.

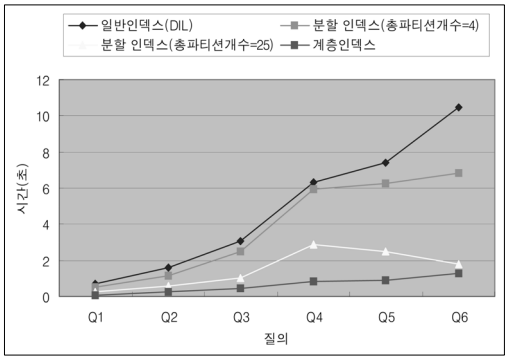


그림 10 INEX XML 문서들에 대한 검색 소요 시간

5. 결론 및 향후 계획

본 논문에서는 의미 정보를 이용하여 검색 개념과 사용자가 질의하는 개념이 정의되는 경우 원소들이 속한 개념간의 관계에 의거한 계층 인덱스를 이용하여 XML 키워드 검색을 효율적으로 처리하는 방법을 기술하였다. 기존 인덱스 구조 하에서는 의미 정보와 상관없이 색인된 모든 원소들간의 조합으로부터 최소 공통 선조를 산출하고 검색 결과 여부를 결정하였으나, 계층 인덱스는 질의 키워드의 포스팅 전체를 읽지 않으면서 최소 공통 선조 산출 연산 회수를 줄임으로써 타 인덱스에 비해 검색 시간을 상당히 감소시킬 수 있다. 계층 인덱스는 질의 개념에 속한 원소들 간의 제한된 조합으로 검색 결과를 산출하기 때문에 질의 개념에 속하는 노드 개수가 적을수록 응답 시간이 큰 폭으로 감소하였다. 또한 감소 폭이 크게 나타났다. 기존 인덱스에서는 질의를 구성하는 키워드가 많아질수록 고려해야 하는 원소들 간의 조합이 급격히 많아 지고 응답시간이 크게 증가하였으나, 계층 인덱스를 사용하는 경우 응답 시간의 증가 정도가 상대적으로 작게 나타났다.

분할 인덱스는 원소가 속한 개념에 상관 없이 단지 원소의 위치 정보만을 이용하여 다수의 파티션으로 나누었기 때문에 사용자 질의 개념에 상관없이 동일 파티션에 속한 모든 원소들간의 조합 연산이 수행된다. 분할 인덱스는 파티션 개수를 적당히 조절할 경우 질의 키워드 개수가 많아지거나 복잡해 질수록 질의 응답 시간이 증가하는 것을 막을 수 있다. 분할 인덱스에서는 질의 키워드 개수가 많아질수록 동일 파티션에 속한 노드들의 개수가 줄어들어 최소 공통 선조로 고려할 원소 간의 조합이 감소한다. 이 경우 분할 인덱스를 사용하는 것이 계층 인덱스를 사용하는 것보다 더 빠른 검색 시간을 보일 수 있다. 노드가 속한 개념 정보와 원소의 위상(topological) 정보를 동시에 사용하여 XML 키워드 검색을 효율적으로 수행하는 연구가 현재 진행 중이다. 즉 계층 인덱스에서 한 RSC의 멤버 노드들의 개수가 일정 수 이상일 때 그 노드가 속한 위상 정보를 사용하여 다수의 파티션으로 나누어 구성한다. 향후 계획은 다음과 같다. 온톨로지 등의 의미 정보를 효과적으로 이용

표 2 INEX XML 문서들에 대한 질의

질의번호	질의 키워드	질의번호	질의 키워드
Q1	singular, value, decomposition	Q4	information, data, visualization, technique, hierarchy, space
Q2	wireless, security, application	Q5	concurrency, control, semantic, transaction, management, application, performance, benefit
Q3	Recommender, system, agent	Q6	machine, learning, adaptive, algorithm, probabilistic, model, neural network, support, vector, machine

하여 질의 결과로 반환 가능한 후보를 얻는 방법에 관한 연구가 필요하다. 또한 사용자 질의 개념과 검색 개념에 대한 관련도가 랭킹에 관여되도록 하는 것에 대한 연구가 필요하다.

참 고 문 헌

[1] F. Daniela, K. Donald and M. Ioana: "Integrating keyword search into XML query processing," Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking, 2000.

[2] C. David, S. M. Yoelle, M. Matan, M. Yosi and S. Aya: "Searching XML documents via XML fragments," Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, 2003.

[3] L. Guo, F. Shao, C. Botev and J. Shanmugasundaram: "XRANK: Ranked keyword search over XML documents," SIGMOD, 2003.

[4] Y. Xu and Y. Papakonstantinou: "Efficient keyword search for smallest LCAs in XML databases," SIGMOD, 2005.

[5] S. J. Kim, Lee, H., and Kim, H-J., "Adaptive Partitioned Index for Efficient XML Keyword Search," submitted for publication (Journal of Research and Practice in Information Technology) 2005.

[6] F. Norbert, G. Kai and johann: "XIRQL: a query language for information retrieval in XML documents," Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, 2001.

[7] M. C. A. Klein: "Interpreting XML Documents via an RDF Schema Ontology," DEXA, 2002.

[8] S. Cohen: "XSEarch: A semantic search engine for XML," VLDB, 2003.

[9] Y. Li, C. Yu and H. V. Jagadish: "Schema-Free XQuery," VLDB, 2004.

[10] F. Norbert, G. Kai and johann, "XIRQL: An XML query language based on information retrieval concepts," ACM Trans. Inf. Syst. 2004.

[11] http://www.uml.org/, The Unified Modeling Language.

[12] D. Carlson, Modeling XML Applications with UML: Practical e-Business Applications, Addison-Wesley, 2001.

[13] P. Giuseppe: "ERX: a conceptual model for XML documents," Proceedings of the 2000 ACM symposium on Applied computing - Volume 2, 2000.

[14] L. Bernadette Farias, sio, S. Ana Carolina, R. Luciano do and G. go: "Conceptual modeling of XML schemas," Proceedings of the 5th ACM international workshop on Web information and data management, 2003.

[15] M. Ronaldo dos Santos and A. H. Carlos: "A Rule-Based Conversion of a DTD to a Conceptual Schema," Proceedings of the 20th International Conference on Conceptual Modeling: Conceptual Modeling, 2001.

[16] I. F. Cruz, H. Xiao and F. Hsu: "An Ontology-Based Framework for XML Semantic Integration", IDEAS, 2004.

[17] http://inex.is.informatik.uni-duisburg.de:2003/, INEX (2003): INitiative for the Evaluation of Xml retrieval.

[18] http://www.sleepycat.com/, BerkeleyDB.

이 형 동

1997년 홍익대 컴퓨터공학과(학사). 1999년 서울대 컴퓨터공학과(석사). 2006년 서울대 컴퓨터공학과(박사). 2006년~현재 삼성전자 연구원. 관심분야는 데이타베이스, 정보검색

김 성 진

1998년 숭실대학교 소프트웨어 공학과(학사). 2000년 숭실대학교 대학원 컴퓨터학과(석사). 2004년 숭실대학교 대학원 컴퓨터학과(박사). 2004년~현재 서울대학교 전기컴퓨터공학부, 박사후과정연구원. 관심분야는 인터넷 데이타베이스, 데이타베이스 시스템 성능평가

김 형 주

1982년 서울대학교 전산학과(학사). 1985년 미국 텍사스 대학교 대학원 전산학(석사). 1988년 미국 텍사스 대학교 대학원 전산학(박사). 1988년 5월~1988년 9월 미국 텍사스 대학교 POST-DOC 1988년 9월~1990년 12월 미국 조지아 공과대학 조교수. 1991년~현재 서울대학교 컴퓨터공학부 교수