구조와 내용 유사도에 기반한 XML 웹문서 검색시스템 구축☆

An Implementation of XML Document Searching System based on Structure and Semantics Similarity

박 우 창* 서 여 진**
Uchang Park Yeojin Seo

요 약

XML은 인터넷상에서 데이터 표현과 변환을 위한 표준이며, 이러한 XML 문서에서 필요한 정보를 찾아내기 위해 XML문서 검색 시스템이 필요하다. 본 연구는 이러한 필요성에 기반을 두어 XML 구조를 최대한 활용하여, 주어진 XML 문서에 대해 구조와 내용이 가장 가까운 문서들을 찾아내는 검색 시스템을 개발하였다. 검색 메트릭은 XML 문서들 중 각 태그의 이름에 대한 유사도, 각 태그가 포함하는 값의 유사도와 태그 간 구조에 대한 유사도를 모두 고려하며, 검색 후 유사도의 결과 값에 따라 검색 결과를 순위화 하여 보여준다. 검색 방법은 전통적인 키워드 검색 방식, 태그와 값을 입력하여 검색할 수 있는 방식, XML 문서를 입력하여 검색하는 세 가지 질의 방식을 제공함으로서 사용자들의 기호에 따라 원하는 방식을 골라 검색할 수 있도록 구성하여 시스템의 유용성을 높였다. 개발된 XML 문서 검색 시스템은 INEX에서 제공된 XML 문서들을 대상으로 하여 테스트하였다.

Abstract

Extensible Markup Language (XML) is an Internet standard that is used to express and convert data. In order to find the necessary information out of XML documents, you need a search system for XML documents. In this research, we have developed a search system that can find documents that matches the structure and content of a given XML document, making the best use of XML structure. Search metrics take account of the similarity in tag names, tag values, and the structure of tags. After a search, the system displays the ranked results in the order of aggregate similarity. Three methods of query are provided: keyword search which is conventional; search with tag names and their values; and search with XML documents. These three methods enable users to choose the method that best suits their preference, resulting in the increase of the usefulness of the system.

☞ Keyword : XML, search similarity, semantic

1. 서 론

XML(Extensible Markup Language)은 구조적 데이터를 표현하는 산업 표준이며, 어떤 플랫폼에서도 읽을 수 있는 시스템 독립적인 데이터 포맷, 내용과 구조 그리고 표현의 분리 등의 장점으로, 많이 사용되어가고 있다. 이러한 XML 문서들에 대한 검색 필요성의 증가는 정보검색(IR) 기술과 데이터베이스 질의 언어의 확장에서 뚜렷하게 보이고 있으며, 더불어 XML 문서를 대상으로 하는 검색의 결과에 대한 정확성을 요구하게 되었다.

예로서 다음과 같은 경우이다. 사용자는 웹 검색엔진에서 "Albert Einstein"라는 작가의 문서에 관해 검색하기 원한다. 그러나 일반 검색 엔진의 경우 그의 이름을 딴 대학교, 그에 대해 언급하는

* 정 회 원 : 덕성여자대학교 정보공학대학 교수
ucpark@duksung.ac.kr(제 1저자)
** 준 회 원 : 덕성여자대학교 대학원 전산학과 석사
yjseo@duksung.ac.kr(공동저자)
☆ 본 연구는 2004학년도 덕성여자대학교 연구비지원으로 이루어졌음.
[2004년/09/25 투고 - 2004/10/11 심사 - 2004/12/06 심사완료]

대학 강의, 그에 관한 사이트 링크, 그의 동료의 전기등 관계없는 검색결과를 가져올 수 있다. 작가로서 Albert Einstein에 대해 요청했다면 모든 관계없는 문서들을 피할 수 있었을 것이다. 즉 키워드와 관련하여 특정화한 XML 태그들을 사용하면 검색 결과를 향상시키고 관계성이 없는 문서가 검색되는 것을 피하며, 유용하고 더 정확한 정보 목록을 제공해 준다. 로컬 사이트와 인트라넷 검색에서도 데이터의 필드를 가지는 것은 더 유용한 아이템을 찾도록 편리성을 제공한다.

XML 문서의 구조를 비교하여 유사도를 계산하는 연구로는 Mong[1]의 XClust와 XML 문서 일부분을 이용하여 구조를 비교한 David Carmel [2]의 연구가 있다. XML 문서에서 DTD를 자동으로 추출하는 방법은 [3,4]에서 연구된 바 있으며, 그 외 XML 스키마 검색, passage retrieval 검색[9,10]이 연구되어졌다. Boris[5]은 XML 문서로부터 XML Schema를 추출하는 알고리즘을 제시한 바 있으며, Behrens[6]은 XML 스키마를 표현하는 Xtree를 제시하였다.

XML 문서들의 질의방식으로는 INEX[8]에서 제시되고 있는 CO(Content Only)와 CAS(Content and Structure) 질의방식이 있다. CO는 검색되어질 XML 문서들의 구조에 대한 사전지식이 없을 때, 사용자들이 값을 넣거나, well-formed된 자신의 XML 문서를 넣어 검색하는 질의방식이며, CAS는 검색되어질 대상문서들의 구조를 알고 태그와 값을 비교할 수 있는 질의 방식으로 XPath를 이용한다.

본 연구는 Mong[1]에서 제시된 유사도식을 변형하여, XML 문서에 포함된 태그와 값을 이용하여 문서유사도를 계산할 것이며, 이를 기반으로 문서들의 유사도가 높은 순으로 검색해 주는 XML 문서 검색 시스템을 구현하였다. 검색의 효율성을 높이기 위해 XML 대상문서들의 구조를 모르는 사용자들이 유사한 문서를 검색할 수 있도록 CO질의 방식을 선택하되 키워드만 입력하여 검색하는 방식, 태그와 값을 입력하여 검색하는 방식과 XML 문서를 입력하는 방식 등 3가지 질의방식으로 확장하였다.

본 논문의 구성은 다음과 같다. 2장에서는 XML 문서 검색 시스템 설계와 사용되는 유사도 검색 알고리즘에 대하여 소개한 후, 3장에서는 2장의 유사도를 기반으로 구축된 XML 문서 검색시스템에 대하여 논하며, 4장에서는 XML 문서 검색시스템의 구축 방법, 5장에서는 기타 문서 검색기와 XML 문서 검색시스템의 기능을 비교분석, 6장에서는 결론 및 향후 방향에 대하여 제시한다.

2. XML 문서 검색 시스템 설계

2.1 XML 문서의 유사도

XML 문서의 검색은 검색 XML 문서와 검색 대상 XML 문서간의 구조와 내용이 가장 유사한 문서를 찾는 과정이다. 즉, 노드(Element)(=태그(tag)+값(value))로 이루어진 문서 간의 구조와 내용에 대한 유사도를 계산하여 가장 유사한 문서를 검색하는 과정이다.

XML 문서의 유사도를 계산하는 일반적인 척도는 없지만 두 문서에 공통으로 포함된 문자열을 이용하여 계산을 하는 코사인 척도[14]를 사용하여 유사도를 계산하는 방법은 Carmel[2]에 의하여 실험되었다. XML 문서 검색은 단순한 문자열을 대상으로 유사도를 계산하는 것보다는 XML 문서의 구조를 고려하여 구조의 유사성도 계산한다.

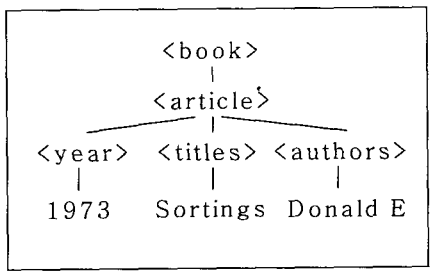
본 논문은 이러한 점을 고려하여 Lee[1]에서 사용된 유사도 기법을 본 논문에 맞게 적용하여 두 개의 XML 문서 d_1, d_2간의 문서유사도(Document Similarity, DS)를 그림 1과 같이 계산하였다. 문서 d_1, d_2에서 유사도 계산을 시작하는 root 노드를 e_1, e_2라고 할 때, XML원소간의 유사도를 계산하는 의미유사도(Semantic Similarity, SS), root 노드 e_1, e_2의 자식 노드들을 비교하는 자손 유사도(Immediate Descendent Similarity, IDS), 그리고 root 노드 e_1, e_2의 말단 노드들을 비교하

$$DS(e_1, e_2) = a * SS(e_1, e_2) + b * IDS(e_1, e_2) + c * LCS(e_1, e_2)$$
$$(단, a + b + c = 1이며, a, b, c >= 0)$$

〈그림 1〉 문서 유사도 계산식

```
<book>
<article>
<year>1973</year>
<titles>Sortings</titles>
<authors>Donald E</authors>
</article>
</book>
```

〈그림 2〉 입력 문서



〈그림 3〉 Xtree

는 잎 노드 유사도(Leaf Context Similarity, LCS)의 3개의 유사도를 가지고 문서 유사도(DS)를 계산하였다. 또한 그림 1에서와 같이 의미유사도(SS), 자손유사도(IDS), 잎노드유사도(LCS)의 각각에 대해 가중치 값으로 a, b, c를 줌으로서 사용자가 유사도에 대한 비중의 정도를 선택하여 문서유사도(DS)를 계산할 수 있도록 하였다.

검색은 먼저 그림 2와 같은 XML 문서에서 태그와 값 정보를 추출하여 그림 3과 같이 Xtree를 만든다. 만들어진 Xtree를 이용하여 위의 3가지 유사도를 구한다.

2.2 유사도 계산

2.2.1 유사도 척도

두 XML 문서로부터 만들어진 Xtree에 대한 유사성을 계산하기 위해서는 노드와 노드 구조 간의 유사성을 계산하기 위한 유사도 식을 계산한다. 먼저 Xtree의 노드 간의 유사성을 계산한다. 노드간의 유사도를 나타내는 유사도인 노드 유사도(Node Similarity)를 정의하고 노드 유사도를 이용하여 두 XML 문서 간의 문서유사도(DS)를 그림 1과 같이 구한다.

2.2.2 노드 유사도(Node Similarity)

이 절에서는 두 XML 문서 노드 사이의 유사도를 결정하는 노드 유사도(Node Similarity)에 대해 설명한다. 노드 유사도는 태그와 값이 해당 배열에 저장된 상태에서, 두 노드 간의 태그와 값이 얼마나 유사한지에 대하여 체크한다.

노드의 유사성은 그림 4의 식과 같이 계산한다. 두 노드 e_1과 e_2에 대하여 TagSim과 ValueSim은 표 1과 같이 값을 정하여 여기에 대한 가중치 w_1, w_2를 적용한다. 보통은 노드의 값(value)이 검색 시 중요하므로 w_2의 값은 크게 정하며 실험에서는 w_1=0.4, w_2=0.6을 주었다. 본 논문에서 주어진 가중치 값은 XML 문서 검색 시스템에서 사용자들의 기호에 맞게 조정할 수 있도록 구현하

$$NodeSim(e_1, e_2) = w_1 * TagSim(e_1, e_2) + w_2 * ValueSim(e_1, e_2)$$
$$(단, w_1 + w_2 = 1)$$

〈그림 4〉 노드 유사도(Node Similarity) 계산식

였으므로 고정된 가중치 값이 아니다.

본 논문에서는 문자열을 비교하여 계산하는 유사도 TagSim과 ValueSim 값의 계산시 기본적인 문자열 검색의 정확성을 높이기 위하여 WordNet [15]을 이용하였다. WordNet은 Princeton 대학에서 단어의 사전적인 의미 연관을 계산하기 위한 시스템으로 어휘의 명사, 동사, 형용사적인 동의어 집합을 찾아주는 시스템이다. WordNet에서는 자체적으로 JavaAPI를 제공하고 있으며, 본 논문에서는 WordNet의 dictionary database를 생성한 뒤에 Wordnet에서 제공되는 API중, SynSet 클래스의 함수를 이용하여 대상 문서들의 Tag나 Value 값에 대한 동의어를 검색하여 List화 한 후, 검색 문서의 Tag나 Value의 값과 비교함으로써 문자열의 동일성과 의미유사성까지 고려하여 유사도를 계산할 수 있었다.

〈표 1〉 노드 유사도(Node Similarity) 규칙

TagSim	값	ValueSim	값
일치	1	일치	1
불일치	SynSet 호출	불일치	SynSet 호출

표 1의 규칙은 TagSim과 ValueSim의 규칙으로 태그나 값이 어느 한 쪽의 substring일 경우 1이며, 불일치 할 경우 WordNet의 SynSet 클래스의 함수를 호출함을 나타낸다.

2.2.2.1 XML의 태그 유사도 계산 알고리즘 (TagSim)

TagSim은 대부분 명사형으로 단일 문자열인 경우로 가정하고 그림 5와 같은 알고리즘을 적용하였다. 두 Tag가 일치하거나 어느 한쪽의 subString이라면 1을 리턴하지만, 그 밖의 경우라면 WordNet의 SynSet을 이용하여 대상문서 e_2의 동의어 집합을 구한 뒤에 검색 문서 e_1의 문자열과 비교하여 유사도를 검색해주는 그림 6의 SynStrength 함수를 호출함으로써 문자열의 동일성도 고려하여 유사도를 계산하였다.

2.2.2.2 XML의 값 유사도 계산 알고리즘 (ValueSim)

ValueSim은 검색문자열 s_1은 단일 문자로 가정하고 s_2는 문장으로 가정하였으며 그림 7와 같은 알고리즘을 적용하였다. 이 경우 s_2 문장을 Tokenizing함으로써 TagSim의 유사도 검색 방식을 적

```
Algorithm TagSim
{
Input : 엘리먼트 이름 e1, e2, MaxDepth = 3
Output : TagSim value(최소 0, 최대 1)
if subString(e1, e2) or subString(e2, e1) return 1;
else { S = Union(SynSet(e')); return SynStrength(e1, {e2}, 1) };
}
```

〈그림 5〉 태그 유사도 계산 알고리즘

```
function SynStrength( e, S, depth)
{
Input : 엘리먼트 이름 e, Synset S, depth 검색 깊이
Output : synonym strength
if ( depth > maxDepth ) then return 0;
if e ∈ S then return 0.8^depth;
else { S = Union(SynSet(e')); return SynStrength(e, S, depth+1); }
}
```

〈그림 6〉 SynStrength 알고리즘

```
Algorithm ValueSim
{
Input : 엘리먼트 이름 e1, e2, MaxDepth = 3
Output : ValueSim value ( 최소 0, 최대 1)

S = e2의 문자 Tokenize;
v = 0;
for( S의 각 원소 e2에 대하여)
{ v1 = ValueSimComp(e1, e2);
if( v1 > v) v = v1;
}
return v;
}
```

〈그림 7〉 값 유사도 계산 알고리즘

용하되 각 단어에 대하여 문자열 비교를 시도하였다. 두 Value가 일치하거나 어느 한쪽의 subString이라면 1을 리턴하지만, 그 밖의 경우라면 WordNet의 SynSet을 이용하여 대상문서 s_2의 동의어 집합을 구한 뒤에 검색 문서 s_1의 문자열과 비교하여 유사도를 검색해주는 그림 8의 ValueSimComp 함수를 호출함으로써 문자열의 동일성도 고려하여 유사도를 계산하였다.

2.2.2.3 TagSim과 ValueSim의 예제

본 논문에서는 각 XML 문서들의 태그와 값을 비교하기 위해 SAX Parser를 이용하여 DOM 트리로 구성한 후 배열에 저장하였으며, 그림 4와 같이 입력 문서의 태그인 e_1과 비교문서의 태그인 e_2가 유사한지 계산하는 TagSim(e_1, e_2)과 각각의 노드에 해당되는 값이 얼마나 유사한지 비교한 ValueSim(e_1, e_2)를 이용하여 두 XML 문서의 NodeSim(e_1, e_2)를 구했다.

예를 들어, 주어진 가중치 값이 w_1=0.4, w_2=0.6일 때, "<animal> dog </animal>"의 구조를 가진 검색 문서와 "<beast> cad and tiger </beast>"의 구조를 가진 대상문서의 NodeSim을 구해보자. TagSim의 경우, 태그가 불일치하므로 SynSet(beast)를 호출하여 beast의 동의어 집합을 구하게 되고 여기서 animal이라는 동의어를 depth=1에서 찾게 되면 SynStrength('animal', {'beast'}, 1)의 값은 0.8이 되고 TagSim은 0.8의 값을 갖게 된다. ValueSim의 경우, 값이 일치한다면 1이라는 값이 리턴되지만 위의 경우 'dog'와 'cad and tiger'로 값이 다르다. 이 경우 대상문서의 문장 'cad and tiger'를 Tokenize하여 단어로 분리한 뒤에 ValueSimComp를 Tokenize한 수만큼 각각의 단어들의 SynStrength를 구한 뒤, 최대 값을 ValueSim으로 저장한다. 위의 경우 'cad', 'and', 'tiger'로 대상문서가 분리 되었으며, 'cad'의 SynStrength depth가 2가 되는 'cad'의 동일어 집합에서 'dog'를 검색할 수 있었다. 그리하여 0.8^2인 0.64로 ValueSim을 구할 수 있었다. 결과적으로 w_1=0.4, w_2=0.6이라고 가정하였을 경우, TagSim=0.8, ValueSim=0.64이므로 NodeSim = 0.4*0.8 +0.6*0.64 = 0.704가 나왔다.

2.2.3 의미 유사도

문서 전체의 유사도를 구하기에 앞서 그림 10과 같이 입력 문서의 트리 구조와 가장 유사한 트리 구조를 찾기 위한 작업이 필요하다. 입력 문서

```
function ValueSimComp
{
Input : 엘리먼트 이름 e1, e2, MaxDepth = 3
Output : ValueSim value ( 최소 0, 최대 1)
if subString(e1, e2) or subString(e2, e1) return 1;
else { S = Union(SynSet(e')); return SynStrength(e1, {e2}, 1) };
}
```

〈그림 8〉 ValueSimComp 알고리즘

와 비슷한 구조를 가진 문서라면 입력 문서의 root와 동일하거나 유사한 노드를 root로 하는 트리 구조를 찾아 유사도를 계산하지 않아도 유사도의 정확성에 그다지 큰 영향을 주지는 않을 것이다. 그러나 입력문서와 유사한 구조를 단지 포함하는 복잡한 문서의 경우 모든 트리 구조를 검색하여 유사도를 계산하는 것은 불필요한 검색 결과를 포함시킴으로서 유사도에 대한 정확도를 떨어뜨리게 된다.

예를 들어, 입력 문서 d_1의 root 노드가 "book"일 경우, 검색 대상 문서 d_2의 노드 중에서 "book"이라는 이름을 가지고 있거나, 부분적으로 같은 이름을 가진 노드가 있는지 검색하여, 그림 10과 같이 입력 문서 d_1의 root노드와 d_2의 가장 유사한 노드 e_{22}를 찾은 후, e_{22}를 root노드로 하는 선택된 트리 구조를 가지고 입력 문서와 비교하는 것이 검색 대상 문서 전체를 비교한 것보다 더 정확한 유사도를 계산할 수 있다.

입력 문서의 root와 가장 유사한 노드를 구하기 위해서는 그림 4의 NodeSim(e_1.root, e_2)를 구하여 얻어진 NodeSim테이블에서 가장 큰 값을 가진 노드를 검색 대상 문서의 root노드로 선정하게 되며 그 값을 의미 유사도, SS(e_1, e_2)라 하며 그림 9과 같이 표현한다.

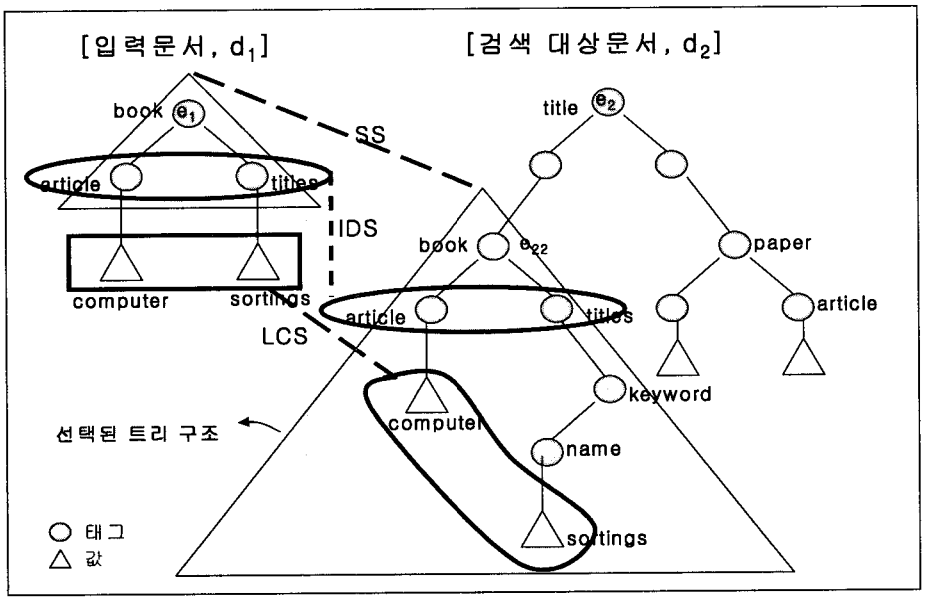
$$SS(e_1, e_2) = Max(NodeSim(e_1, e_2))$$

〈그림 9〉 의미 유사도(SS) 계산식

그림 10의 경우 입력 문서와 가장 유사한 root 노드로 e_{22}가 선정되었으며, Max(NodeSim(book, book))=1이므로 SS=1이 된다.

2.2.4 자손 유사도

자손유사도(IDS)는 두 문서에서 노드 간의 인접한 자손 노드들의 유사도를 계산하여, 두 문서 간의 구조 유사성을 측정하기 위한 식으로 그림 11과 같으며, descendants(e_1)은 e_1의 자손 노드를 말하며, descendants(e_2)은 e_2의 자손 노드를 말한다. 그림 10의 경우 e_1의 자손 2개와 e_{22}의 자손 2개에 대하여 NodeSim을 계산하여 기준 값을 넘는 경우 matchlist에 포함한다. 입력문서 e_1의 자손 수는 descendants(e_1)=2이며, 검색대상문서 e_{22}



〈그림 10〉 두 문서 간의 유사도 매칭

$$IDS(e1, e2) = \sum NodeSim \in Matchlist / Max(|descendants(e1)|, |descendants(e2)|)$$

〈그림 11〉 자손 유사도(IDS) 계산식

의 자손 수는$descendants(e_{22})$=2이므로 descendants의 Max값인 2와 Matchlist (article, article), (title, titles)의 NodeSim을 계산하여 IDS를 구한다.

예를 들어, 그림 10에서 book의 자손 유사도(IDS)를 구해보자. 먼저 각 노드유사도를 구하면 다음과 같다.

NodeSim(article, article) = 1;
NodeSim(article, titles) = 0;
NodeSim(titles, article) = 0;
NodeSim(titles, titles) = 1;

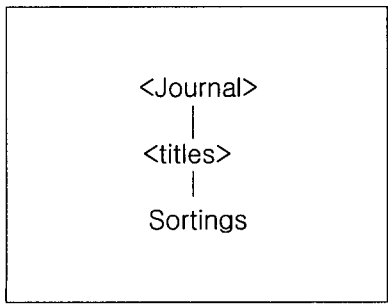
이 때 Matchlist={NodeSim(article, article), NodeSim(titles, titles)}이 되고, 자손유사도 IDS (book, book)=(1+1)/max(2, 2)=2/2=1이 된다.

2.2.5 경로상수 PCC(Path Context Coefficient)

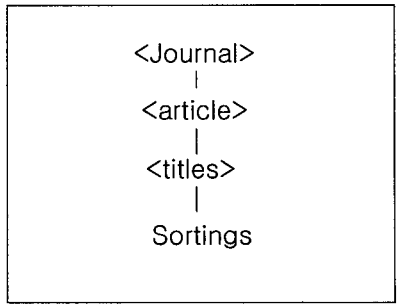
2.2.2절에서 단순히 노드의 태그와 값을 비교하였다면 이번 절에서는 트리의 잎 노드의 문맥 경로에 대한 유사도를 파악하기 위해, 경로상수(Path Context Coefficient)의 개념을 설명한다. PCC는 잎 노드 유사도를 계산할 때 잎 노드까지의 경로에 포함된 노드에 대한 유사도를 측정하기 위하여 필요하다. 예를 들어, 사용자가 "Sortings"라는 단어를 검색하기를 원할 때, root로부터 Sortings를 가진 노드까지의 문맥 경로를 따져보는 것이다. 그림 12와 그림 13의 두 문서를 비교해 보자.

"<titles>Sortings</titles>"라는 XML 문서를 검색하기 위해, 그림 12의 문서 1은 그림 13의 문서 2에 비해 적은 경로를 거쳐 <title> 태그를 검색할 수 있으므로 문서 1과 같은 구조를 가진 문서를 입력할 경우 문서 2에 비해 구조적인 면에서 문서 1이 검색 대상 문서에 더 유사하다.

많은 노드를 가진 문서와 비교한다고 생각해 보자. 단지 "Sortings"라는 단어만을 검색한다면 PCC를 사용하는 것은 비효율적이다. 하지만, 가장 유사한 문서를 검색하고자 할 때 문서 간의 구조, 즉 노드의 값에 이르는 문맥은 유사한 문서를 검색할 때, 값만큼 중요한 비중을 차지한다. 그러므로 긴 문장을 가진 문서들의 경우, "Sortings"라는 값이 문서의 어디에 위치하고 있는지 그 비중을 계산하는 것은 가장 유사한 문서를 검색해 내기 위해 반드시 필요하며, 문서 간 유사도의 정확성을 높이는 효과가 있다. 그림 14와 같이 PCC를 계산하기 위해서는 위의 절에서 제시된 NodeSim을 이용하여 대응리스트(matchlist)를 구해야 한다.



〈그림 12〉 문서 1



〈그림 13〉 문서 2

$$PCC(l_1, e_1, l_2, e_2, threshold) = \sum NodeSim \in Matchlist \,/\, Max(|dest_1.path(source_1)|, |dest_2.path(source_2)|)$$

〈그림 14〉 PCC 계산식

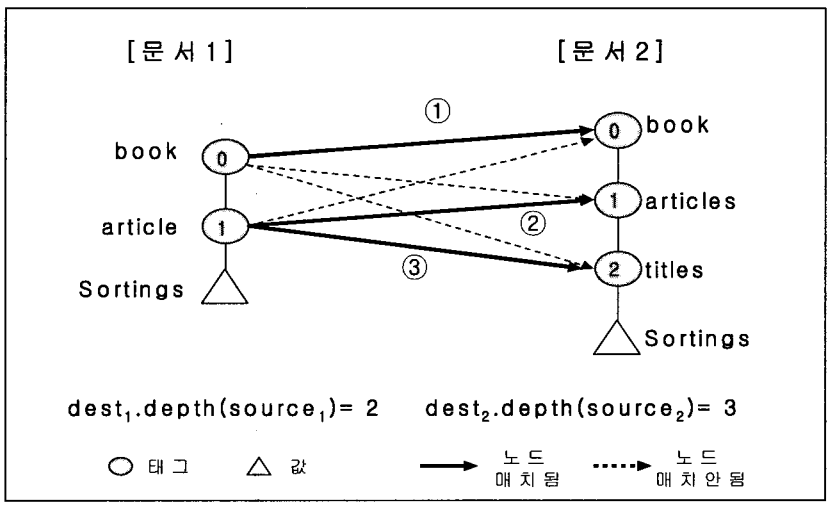
각 잎 노드에 대한 문맥 경로를 나타내는 PCC는 그림 14와 같이 계산되어 진다. value의 값이 null인 tag의 경우는 ValueSim을 적용하지 않은 TagSim값이 NodeSim이 된다.

여기서 제시되고 있는 Matchlist는 태그와 값을 유사 정도를 비교하여, 표 1과 같은 규칙에 따라 태그와 값의 가중치를 부여하여 생성된 NodeSim에서, NodeSim들의 값 중 threshold이상의 값만 뽑아내어 불필요하게 작은 값들을 제거한 배열 리스트를 말한다. d.path(source)는 d.path(source)={source, element$_1$, ... ,dest(d)}로 나타낼 수 있으며, 문서의 root 노드를 source, 찾고자 하는 "Sortings"를 가진 노드를 dest(d)라하고, "Sortings"를 찾기 위한 노드 path를 d.path라 한다.

본 논문에서는 입력문서의 d_1.path를 구하고, 검색 대상 문서들의 d_2.path를 구하여 두 문서의 부분적인 path도 고려하여 유사도를 계산하였다. 예를 들어, 그림 14의 계산식에서 제시된 Matchlist는 2.2.2절에서 설명되어진 노드유사도(NodeSim) 테이블 값 중 threshold 이하 값은 버림으로서 두 유사도의 정확성을 높이는 Matchlist를 구한다.

예를 들어, 그림 15을 보자. 그림 15에서 실선은 두 문서의 모든 태그 간 비교를 계산하고 노드에 포함된 값을 비교하여 같거나 비슷한 노드의 태그나 값을 표시한 것이며, 비교하였지만 매치되지 않는 노드에 대해서는 점선으로 표시하였다. ①의 경우NodeSim=1, ②의 경우 NodeSim=0.5, ③의 경우 TagSim=0.4*0=0이고 ValueSim=0.6*1=0.6이므로 NodeSim=0.6이 된다. threshold=0.3인 경우 ①, ②, ③의 값이 모두 threshold 이상이므로 Matchlist테이블에 저장된다. 그러므로 Matchlist={NodeSim(book, book), Node(article, article), NodeSim(article, titles)}={1, 0.5, 0.6}이 된다.

PCC를 계산하기 위해서는 matchlist에서 구해



〈그림 15〉 두 문서 간 노드의 태그와 값 비교

$$LCS(e_1, e_2) = \sum LeafSim \;/\; Max(|leaves(e_1)|, |leaves(e_2)|)$$

〈그림 16〉 잎 노드 유사도(LCS)

$$LeafSim = PCC(l_1, e_1, l_2, e_2, threshold) * NodeSim(l_1, l_2)$$

〈그림 17〉 LeafSim 계산식

진 값들의 합을 두 트리의 태그의 깊이 중 max값으로 나눠줘야 한다. 여기서 깊이의 max값은 그림 15에서 보여 지듯이$dest_1$.path($source_1$)=2이고 $dest_2$.path($source_2$)=3 인 경우 문서 2가 더 크므로 3의 값이 선택된다. 그러므로 그림 15의 두 문서 간 PCC(article, book, titles, book, 0.3)= (1+ 0.5+0.6)/3= 0.7이라는 값을 가지게 된다.

2.2.6 잎노드 유사도

노드의 값은 주로 DTD 트리의 말단의 leaf-nodes에 포함되어진다. 그러므로 선택되어진 태그로 부터 잎 노드까지의 문맥을 고려하면 더 근접한 유사도를 측정할 수 있다. 그림 16에서 제시된 LeafSim은 잎 노드로부터 root 노드까지의 문맥에 대한 유사도를 가진 테이블을 말하며 이 LeafSim 테이블들의 합과 잎 노드들의 개수를 나타내는 leaves(e)을 가지고 그림 16과 같은 잎 노드 유사도(LCS)를 구한다.

2.2.5절에서 설명하였듯이 잎 노드의 문맥 경로를 고려하기 위하여 입력 문서의 잎 노드 l_1으로부터 root 노드 e_1까지의 노드들과, 검색 대상 문서의 잎 노드 l_2로부터 입력 문서의 시작 노드 e_2까지의 노드들 사이의 PCC를 구한다. 여기에 잎 노드 l_1과 l_2의 NodeSim을 곱하면 그림 17과 같이 LeafSim을 구할 수 있다. LeafSim의 값들의 합에 두 문서의 잎 노드의 개수 중 Max값을 나누어 그림 16과 같이 LCS를 구한다.

2.2.3절의 그림 10의 예에서 "computer"와 "sortings"를 가지는 잎 노드에 대한 LCS를 구해보자. 여기서 PCC 계산시 0이 되는 NodeSim은 값을 빼고 계산하였다. LeafSim을 구하기 위한 잎 노드의 NodeSim은 다음과 같다.

NodeSim(article, article) = 1

NodeSim(title, name) = 0.6

PCC(article, book, article, book, 0.3) = (1 + 1)/ 2 = 1

PCC(titles, book, name, book, 0.3) = (1 + 1)/4 = 0.5

이 때, LeafSim(article, book, 0.3) = 1 * 1 = 1

LeafSim(titles, book, 0.3) = 0.5 * 0.6 = 0.3

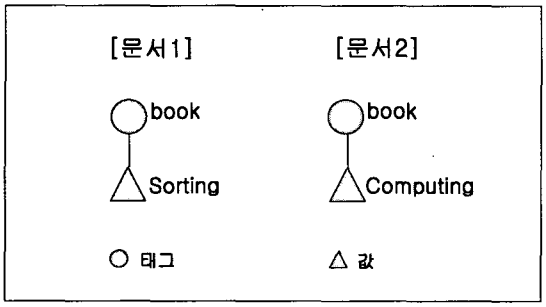
그러므로, LCS(boo, book) = (1 + 0.3)/ max(2, 2) = 0.65가 된다.

2.2.7 문서 유사도 계산 알고리즘

이 절에서는 그림 18의 유사도 계산 알고리즘에 대하여 설명한다. 문서 1과 문서 2가 그림 19와 같이 root 노드 밑에 잎 노드만 가지고 있는 단순 트리 구조라면 의미 유사도인 SS(e1, e2)의 유사도 식만으로도 두 문서의 유사도 값을 구할 수 있지만, 검색 대상 문서의 트리 구조가 앞에서 예로든 2.2.3절의 그림 10에 제시된 문서 2와 같이 복잡한 구조를 가진 문서일 경우, 2.2.4절에서 설명했듯이 e_1과 같거나 유사한 노드 e_{22}를 root로 하는 트리 구조와 비교하여 SS, IDS LCS를 구한다.

```
Algorithm
{
입력 : 입력문서의 e1, 대상문서의 e2; threshold; 가중치 a, b, c
출력 : 문서 유사도(Document Similarity, DS)
if e1 or e2 is leaf node then return SS(e1, e2);
else
{ 1. 대상 문서에서 입력 문서의 root 노드 e1과 가장 유사한 시작
노드 e2(이를 e22라 부름)를 검색;
2. SS(e1, e22) 계산;
3. IDS(e1, e22) 계산;
4. LCS(e1, e22) 계산;
5. DS= a*SS + b*IDS + c*LCS 계산;
}
return DS;
}
```

〈그림 18〉 유사도 계산 알고리즘



〈그림 19〉 잎 노드만 가진 단순한 트리 구조

즉, 그림 19와 같은 root와 잎 노드로 구성된 문서들의 경우라면 SS만으로도 유사도를 검색할 수 있으며, 그 외의 복잡한 문서의 경우 문서 2의 노드 중, 문서 1의 root 와 가장 유사한 태그인 e_{22}를 찾아 그림 18의 알고리즘과 같이 e_{22}를 조상 노드로 하는 모든 부속 트리만을 비교하여 계산하도록 하였다. 알고리즘은 이제까지 설명한 내용을 기반으로 2개의 XML 문서에 대한 문서유사도(Document Similarity)를 계산한다.

2.3 간단한 XML 문서들의 유사도 검증 실험

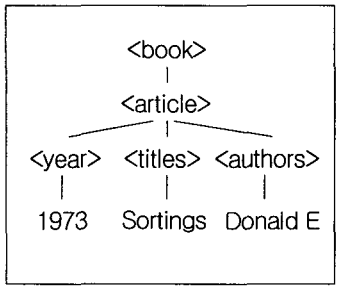
간단한 입력 XML 문서와 5개의 XML 대상 문서들의 유사도를 검색한 결과는 그림 21부터 그림 25와 같다.

그림들은 그림 20 입력문서와 비교하였을 때 유사도가 큰 순으로 나열한 것이다. 그림 22 Doc2.xml은 그림 20 입력 문서와 비교하였을 때, 태그와 문서의 구조는 일치하나 값이 불일치 할 경우로 유사도는 0.96이며, 그림 23 Doc3.xml은 문서의 구조는 일치하나 태그와 값이 부분 일치할 경우로 유사도는 0.938이다. 그림 24의 Doc4.xml은 태그와 문서의 구조는 일치하나 값이 모두 불일치할 경우로 유사도는 0.88이고, 그림 25 Doc5.xml은 태그와 문서의 구조가 부분 일치하나 값이 불일치할 경우로 유사도는 0.826이다.

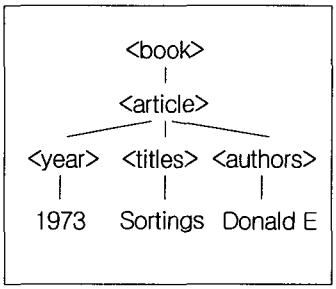
표 2는 각 XML 문서들을 입력문서와 비교하여 얻어진 유사도를 나타낸다.

〈표 2〉 입력 문서와 검색 대상 문서들과의 유사도

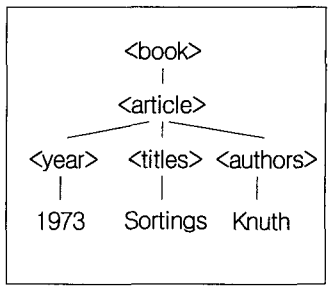
	SS	IDS	LCS	유사도
Doc1.xml	1.0	1.0	1.0	1.0
Doc2.xml	1.0	1.0	0.8	0.96
Doc3.xml	1.0	1.0	0.693	0.938
Doc4.xml	1.0	1.0	0.4	0.88
Doc5.xml	1.0	1.0	0.13	0.826



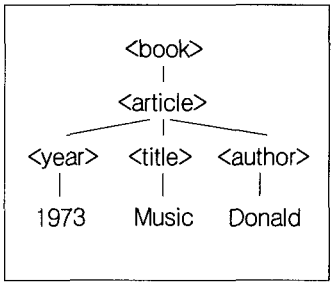
〈그림 20〉 입력 문서



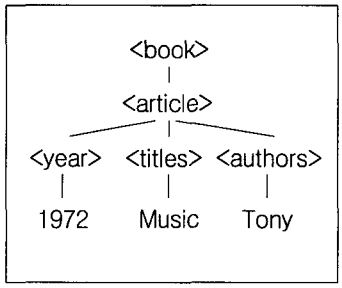
〈그림 21〉 Doc1.xml 유사도 : 1.0



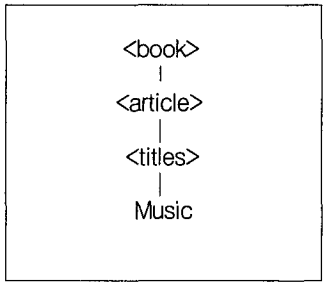
〈그림 22〉 Doc2.xml 유사도 : 0.96



〈그림 23〉 Doc3.xml 유사도 : 0.938



〈그림 24〉 Doc4.xml 유사도 : 0.88



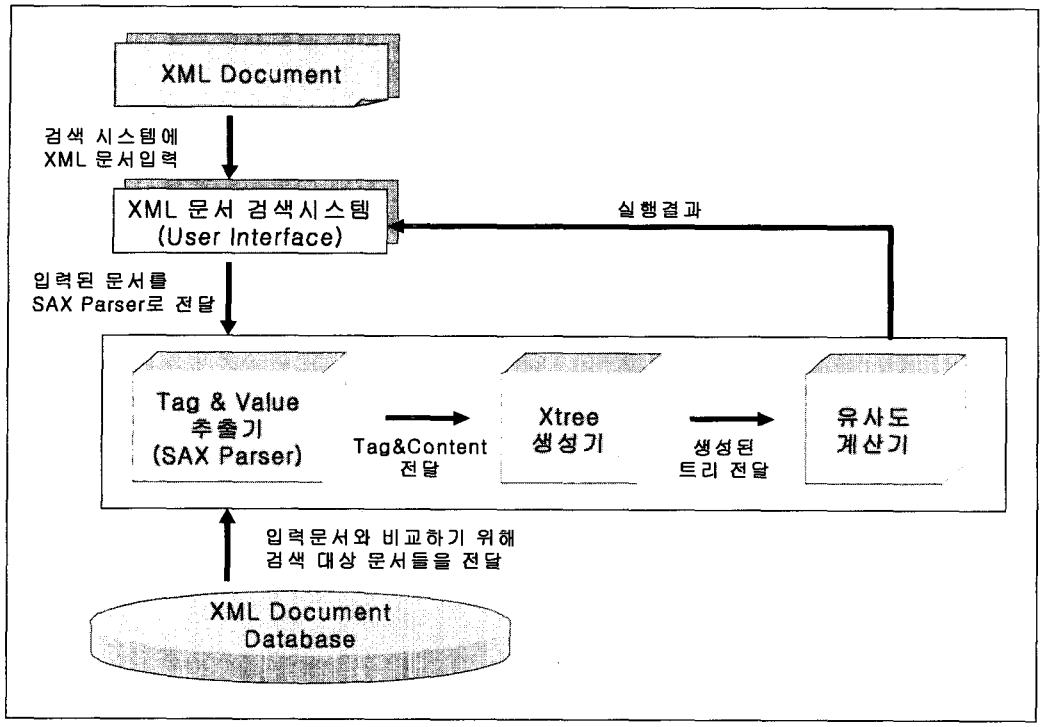
〈그림 25〉 Doc5.xml 유사도 : 0.826

본 논문의 검색 시스템은 SS, IDS, LCS의 유사도 가중치를 각각 0.5, 0.3, 0.2로 두었을 경우이며, 사용자의 선택 기호에 따라 유사도의 가중치를 조정할 수 있도록 구현하였다. 또한 유사도를 구할 때 Matchlist 계산시에 적용되는 threshold 값에 따라서 유사도의 결과가 다르게 나올 수 있다. PCC의 threshold를 0.6로 주었을 경우, Doc4.xml의 유사도 값이 0.826으로 Doc1.xml보다 낮고 Doc5.xml과는 동일한 결과가 나오는 것을 볼 수 있었다. 이것은 PCC 계산 시 NodeSim 값들 중, 0.6을 넘지 못하는 NodeSim은 포함되지 않았기 때문이다. threshold를 높인다는 것은 두 문서의 유사도를 계산할 때, 두 문서간의 문맥 경로의 노드에 대한 유사도의 정확성을 따진다는 것이므로 Doc4.xml이 Doc1.xml에 비해 잎 노드의 값이 유사하더라도 Doc1.xml에 비해 노드의 구조적인 면에서 정확도가 떨어지므로 위와 같은 결과를 갖는 것이다. 이처럼 사용자는 각 유사도의 가중치, Matchlist 계산 시에 불필요한 값을 제거하기 위해 사용되는 threshold, 표 1의 규칙 값을 조정함에 따라 다양한 유사도 결과를 가질 수 있음을 볼 수 있었다.

2.4 XML 문서 검색시스템 구성도

그림 26는 XML 문서 검색 시스템 흐름도이다. 입력 받은 XML 문서와 유사한 것을 XML DB로부터 검색하기 위해서 입력 문서와 DB상에 다음의 3가지 단계가 필요하게 된다. 첫째는 XML 태그와 값의 노드를 추출하기 위한 태그&값 추출기이고, 둘째는 Xtree 생성기이며, 셋째는 유사도 계산기이다. 검색을 위한 XML Document DB에서 추출한 Xtree가 저장되어 있는 Xtree DBMS가 존재하게 된다. 검색 단계는 다음과 같다.

첫째, 검색할 XML 문서를 입력 받는다. 둘째, 입력받은 문서가 XML 태그&값 추출기에서 XML 태그와 값이 추출된다. 셋째, 추출된 태그와 값은 Xtree 생성기에 들어가서 Xtree가 생성된다. 넷째, DB에 저장된 검색 대상 문서들이 XML 태그&값 추출기에서 호출되어 각각의 XML



〈그림 26〉 XML 문서 검색 시스템 흐름도

태그와 값이 추출되고 Xtree 생성기에서 Xtree가 생성되면 입력 받은 XML 문서의 태그와 값을 가지고 유사도 계산기에서 유사도를 계산한다. 마지막으로 DBMS의 Xtree와 유사도의 결과를 보여준다.

3. XML 문서 검색 시스템의 개요

3.1 XML 문서 검색 시스템 소개

본 논문의 XML 문서 검색 시스템은 검색되어질 문서들의 XML 구조를 모르는 사용자들을 위하여 구현되었으며, INEX[8]에서 제시되어진 Content Only(CO) 질의 방식이다. 키워드 검색 방식(CO1), 키워드와 태그 검색방식(CO2), XML 문서 검색 방식(CO3)의 3가지 질의 방식을 구현하여, 시스템의 다양성과 유용성을 높이고, 위에서 제시된 유사도식을 이용하여 검색의 정확성을 높였다.

이 검색 시스템의 목적은 질의 개념과 가장 관련된 XML 컴포넌트를 가진 XML 문서를 검색해 주는 것이며, INEX04에서 제공한 문서들을 기반으로 사용자가 선택한 CO방식에 따라 검색되어진 문서들을 유사도가 가장 높은 순으로 반환해 주도록 설계되었다.

3.2 XML 문서 검색 시스템의 3가지 질의방식

CO1 질의방식(Input Keyword)은 일반 정보검색시스템에서 사용자들이 검색하고자 하는 키워드만 입력하면 검색 결과를 가져오듯이, XML 문서 구조의 지식이 없는 사용자들이 검색하고자 하는 키워드를 입력하여 원하는 문서를 검색하도록 구현하였다.

CO2 질의방식(Input Tag and Value)은 XML 문서의 구조에 대해 약간의 지식이 있는 사용자들이 사용할 수 있도록 만들었으며, 사용자가 찾고

자 하는 태그와 값을 직접 입력하여, 문서안의 태그까지 고려하여 유사도를 계산하여 문서를 검색해 줌으로서 CO1 질의방식보다 정확한 검색 결과를 가져다주는 장점이 있다.

CO3 질의방식(Input well-formed XML File)은 XML의 구조를 알고, 검색하고자 하는 well-formed된 XML 문서를 가지고 있는 사용자들이 검색할 수 있도록 구현하였다. 단지 노드의 태그와 값만을 비교하여 유사한 문서를 검색해주는 다른 질의 방식과는 달리, CO3 질의방식은 문서의 트리 구조까지도 고려함으로서 두 문서간의 유사도를 계산하여 가장 유사도가 높은 문서 순으로 문서들을 검색해 주는 CO1, CO2보다 확장된 질의방식이다.

4. XML 문서 검색 시스템 구현

본 절에서는 XML 문서 검색 시스템의 구현을 간략히 소개한다. XML 문서 검색 시스템은 Java로 구현되었으며, JSP와 Apache웹 서버를 이용하여 구현되었다.

4.1 XML 문서 검색 사용자 인터페이스

그림 27은 XML 문서 검색 시스템의 첫 화면이다. 각 질의 방식에서 데이터를 입력받기 위하여 CO1과 CO2 질의방식은 Java Beans를 사용하였으며, CO3 질의방식은 MultipartRequest방식을 이용하여 입력받은 파일을 데이터베이스에 저장하였다.

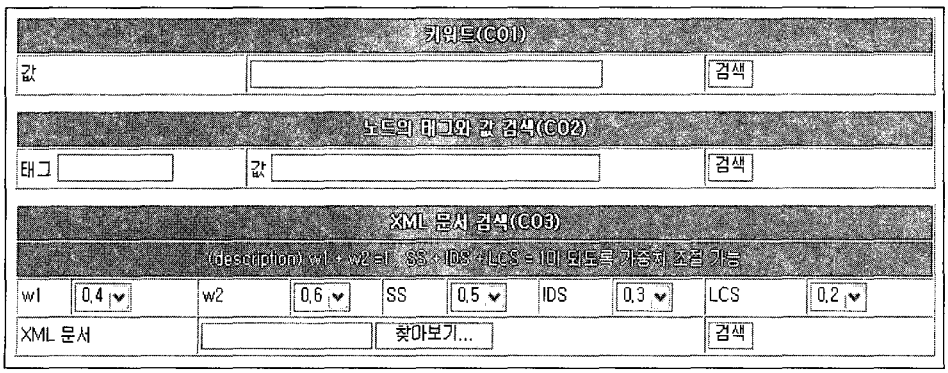
4.2 XML 문서 검색 실험

본 논문의 검색 실험을 위한 XML 문서는 INEX에서 제공되는 8,000여 개의 XML문서 파일들 중 50개의 파일을 추출하여 사용하였으며 실험 결과는 다음과 같이 나왔다.

CO1 질의방식은 키워드를 입력하고 검색을 하면 유사도에 상관없이 검색 대상 문서 안에서 유사한 값을 포함하는 문서를 찾아주며, 검색 시 문서의 태그나 문서 트리의 구조는 고려하지 않고, 단지 키워드만을 고려하여 검색 결과를 추출해 준다.

CO2 질의 방식을 사용하려면, 사용자가 검색하고자 하는 태그와 값을 입력하여 검색한다. 검색되어질 대상 문서들의 태그와 값을 비교하여 검색 결과를 유사도가 높은 순으로 출력하도록 하였다.

CO2 질의 방식은 CO1 질의방식에 비해 본 시스템에 저장되어진 XML 문서들의 구조를 알고 있는 사용자들이 값뿐 아니라 XML 문서의 태그까지도 고려하여 검색하고 싶을 때 사용하면 좋은 질의 방식이다. CO2 질의방식은 문서의 태그도 비교하여 검색된 대상 문서들을 유사도가 높은 순으로 나열해 줌으로서 보다 정확성을 가진 문서를



〈그림 27〉 XML 문서 검색 화면

검색해 주는 유용성을 가졌다.

CO3 질의 방식은 XML 문서에 대한 지식을 가지고 있는 사용자가 well-formed된 XML 문서를 입력함으로서 두 문서의 노드 간 태그와 값뿐만 아니라 문서의 트리 구조까지도 고려하여 유사도가 높은 순으로 문서를 검색하여 준다.

예를 들어, 대상 문서의 구조가 book.article.title.computer와 book.title.computer의 구조를 가지고 있고 사용자가 'title'이 'computer'인 문서를 검색하고자 할 경우, 트리의 구조면에서 후자에 더 높은 유사도를 부여할 것이다.

그림 29는 CO3의 실행 결과 화면으로 그림 28의 구조를 가진 문서를 입력하였을 경우 검색 결과 화면이며, 태그를 'ti'로 값을 'computer'로 가진 r2097.xml 문서가 가장 유사도가 높은 것으로 검색된 것을 볼 수 있었다.

```
<article>
<ti>computer</ti>
</article>
```

〈그림 28〉 입력 문서의 구조

5. XML 문서 검색기 기능 비교

본 절에서는 기존의 XML 문서 검색기의 기능을 본 논문에서 제안한 검색 시스템을 여러 유형별로 비교 평가한다. 질의 시 노드의 구조 제한 여부, 태그의 구조 검색 여부, 값 검색 여부, 문서 간 구조 검색 여부, 검색의 유용성 측면을 고려하여 표 3과 같이 비교하였다.

Yosi[7]은 성능은 좋지만 검색 대상 문서의 구조를 아는 사용자들이 XML의 태그들을 구조에 적합하게 조합하여 검색해야 하는 조건이 있으며, 문서 간 검색에도 조건이 따르므로 사용자 입장에서는 유용성이 떨어진다.

Zobel[9], Salton[10]은 절 안에서 사용자의 질의와 가장 유사한 문장을 찾아주는데 목적이 있으며, 검색되는 문장들은 일관성이 없더라도 조합되어져 결과로 검색되어지기 때문에 XML 컴포넌트 검색에는 적당하지 않으며, 원하지 않는 검색결과가 나올 수 있어 검색의 정확성과 유용성이 떨어진다.

Richmond[11]에서 묘사되는 작업은 단지 주제와 관련된 단어들을 문서와 비교함으로서 그 단어가 나온 비율을 뽑아주는 키워드 검색만 제공해 주므로 본 연구 시스템과 같은 문서 검색 시스템을 제공해 주지 않으므로 사용자 인터페이스면에서 검색의 유용성이 떨어진다.

Carmel[2], Fuhr[12], Grabs[13]에서 제시된 작업은 XML검색 시 검색하고자 하는 구조의 경

Total:12 Articles(1/3Pages)

유사도 순위	XML문서 이름	유사도(0.5*SS + 0.3*IDS + 0.2*LeafSim)	문서보기
1	r2097.xml	0.5842105 (0.5 * 1.0 + 0.3 * 0.0 + 0.2 * 0.42105263 = 0.5842105)	문서보기
2	r2094.xml	0.56666666 (0.5 * 1.0 + 0.3 * 0.0 + 0.2 * 0.33333334 = 0.56666666)	문서보기
3	x1026.xml	0.55419356 (0.5 * 1.0 + 0.3 * 0.0 + 0.2 * 0.27096775 = 0.55419356)	문서보기
4	t0015.xml	0.53818184 (0.5 * 1.0 + 0.3 * 0.0 + 0.2 * 0.19090909 = 0.53818184)	문서보기
5	t0024.xml	0.535 (0.5 * 1.0 + 0.3 * 0.0 + 0.2 * 0.175 = 0.535)	문서보기

Go to Page 1 2 3

다른 문서 검색하기

〈그림 29〉 CO3 실행화면

〈표 3〉 타 검색기와 본 연구의 검색 시스템의 비교

	제안하는 검색 시스템	Yosi[7]	Zobel[9], Salton[10]	Richmond[11]	Carmel[2], Fuhr[12], Grabs[13]
질의시 노드의 구조 제한여부	제한없음	제한	제한없음	제한없음	제한
태그의 구조 검색 여부	○	○	×	×	○
값 검색 여부	○	○	○	○	×
문서 간 구조 검색 여부	○	×	×	×	×

로에 해당되는 노드의 태그 빈도수를 가지고 검색하므로, 유사성을 검색하기 위해서는 검색 대상 문서들의 태그의 구조에 맞게 경로를 지정해 주기 위해, 대상 문서들의 구조를 알고 질의하거나; 경험에 의해 그 문서들을 검색해 본적이 있어야만 질의가 가능하므로 주어진 문서들에 대한 구조를 모르는 사용자들에게는 적합하지 않으므로 비효율적이며 검색의 유용성이 떨어진다.

본 연구의 검색 시스템은 키워드 검색, 태그와 값 검색, XML 문서 검색의 3가지 방식을 구현함으로서 사용자들이 문서의 구조를 모르더라도 단지 키워드나 문서의 입력만으로도 자신이 원하는 결과를 검색해 줄 수 있도록 구현하였으므로 다른 문서 검색기와 비교할 때, 사용자 편리성과 유용성을 제공해 주며, 노드의 태그 구조 뿐 아니라 노드의 값도 고려하여 유사도를 계산함으로서 검색의 정확성을 한층 높였다는 장점이 있다.

6. 결론 및 향후 연구 과제

본 논문은 XML 문서의 구조에 대한 지식이 없는 사용자가 키워드만을 가지고 원하는 문서를 검색할 수 있도록 구현된 질의방식(CO1), XML의 문서구조에 대한 지식이 어느 정도 있는 사용자가 검색하고자 하는 태그와 값을 입력하여, 이 정보를 포함하고 있는 문서를 검색해 주는 질의방식(CO2), well-formed된 문서를 가진 사용자가 이 문서를 입력함으로서 태그와 값뿐 아니라 두 문서의 트리 구조까지도 비교하여 유사도가 높은 순으로 XML 문서를 검색해 주는 질의방식(CO3)의 3가지 방식을 제공하는 시스템을 구현하였다.

본 연구의 검색 시스템은 검색 대상 문서들의 구조를 모르더라도 사용자가 값만 넣어 검색하거나, well-formed된 문서만 입력하여 검색할 수 있는 3가지 질의 방식을 제공함으로서 표 3에서 제시되는 바와 같이 검색의 유용성을 제공하고 있으며, JSP와 JavaBeans를 이용하여 사용자들이 보다 쉽게 검색할 수 있는 인터페이스를 제공해 주었고 유사도 검색 시 불필요한 값을 제거하기 위해, threshold 값을 적용하여 가장 최적의 유사도 값을 가진 문서들을 찾아줌으로서 검색의 정확도를 높였다.

향후 연구과제로는 본 연구에서 더 많은 문서들에 대해 테스트함으로서 보다 정확한 유사도 결과를 얻는 것이다.

참 고 문 헌

[1] Mong Li Lee, Liang Huai Yang, Wynne Hsu, Xia Yang. XClust: Clustering XML Schemas for Effective Integration. ACM, 2002

[2] David Carmel, Yoelle S. Maarek, Matan Mandelbrod, Yosi Mass, Aya Soffer "Searching XML Documents via XML Fragments", Communication of ACM., July 2003.

[3] Minos Garafalaos, Aristides Gionis Ra-

jeev Rasogi, S.Seshadri, Kyuseo Shim, "XTRACT:A System for Extracting Document Type Descriptors from XML Documents," In Proc. ACM SIGMOD, 2000

[4] Jason San Key, Raymond K.Wong, "Structural Inference for Semistructured Data", Proc. International Conference on Information and Knowledge Management, 2001.

[5] Boris Chidlovskii, "Schema Extraction from XML Collections," Proc. ACM/ IEEE-CS Joint Conference on Digital Libraries, 2002.

[6] Ralf Behrens, "A Grammar Based Model for XML schema Integration," BNCOD, 2000.

[7] Yosi Mass, Matan Mandelbrod, "Retrieving the most relevant XML Components", In Proceedings of INEX'2003, 2003

[8] INEX'04, Retrieval task, http://inex.is.informatik.uni-duisburg.de:2004/internal/pdf/INEX04_Retrieval_Task.pdf

[9] M. Kaszkiel and J. Zobel, "Effective ranking with arbitrary passges", Journal of the American Society of Information Science, volumn 52, number 4,pg 344-364, 2001.

[10] G. Salton, J. Allan, and C. Buckley. "Approaches to passage retrieval in full text information systems." In Processings of SIGIR'93, Pittsburgh, PA, 1993.

[11] K. Richmond, A. Smith and E. Amitray, "Detecting Subject Boundaries within Text: A Language Independent Statistical Approach", Proceedings of the Second Conference on Empirical Methods in Natural Language Processing, pg 47-54, 1997

[12] N. Fuhr and K. GrossJohann, "XIRQL: A Query Language for Information Retrieval in XML Documents". In Proceedings of SIGIR'2001, New Orleans, LA, 2001

[13] T. Grabs and H. J. Scheck, "Generating Vector Spaces On-the-fly for Flexible XML Retrieval"

[14] G. Salton and M.J McGill, "Introduction to Modern Information Retrieval". McGraw-Hill, New York, 1983.

[15] WordNet, Princeton Univ., http://www.cogsci.princeton.edu/~wn/

◐ 저 자 소 개 ◑

박 우 창 (Uchang Park)
1982년 서울대학교 계산통계학과 졸업(학사)
1985년 서울대학교 대학원 계산통계학과 졸업(석사)
1993년 서울대학교 대학원 계산통계학과 전산과학전공 졸업(박사)
1988년~현재 덕성여자대학교 컴퓨터과학부 교수
관심분야 : 인터넷 응용, 데이터베이스, 알고리즘
E-mail : ucpark@duksung.ac.kr

서 여 진 (Yeojin Seo)
2002년 덕성여자대학교 전산학과 졸업(학사)
2004년~현재 덕성여자대학교 대학원 전산 및 정보통신학과(석사)
관심분야 : 인터넷 응용, 데이터베이스, 알고리즘
E-mail : yjseo@duksung.ac.kr