구조적 문서의 효율적인 검색을 위한 자료 구조와 알고리즘 설계

김영자, 정채영, 김현주, 배종민
경상대학교 전자계산학과

Design of Data Structures and Algorithms for Efficient Retrieval of Structured Documents

Young-Ja Kim, Chae-Young Jung, Hyon-Ju Kim, Jong-Min Bae
Department of Computer Science, Kyeongsang National University

요 약

SGML이나 XML과 같은 마크업 언어를 사용하여 생성된 구조적 문서에 대한 검색 시스템은 문서의 임의의 부분에 대한 검색을 지원한다. 문서의 구조에 바탕을 둔 다양한 유형의 사용자 질의를 처리하기 위해서는 색인에 필요한 메모리양이 커지게 된다. 색인에 필요한 메모리양을 줄이기 위해 문서구조의 일부만을 색인할 경우에는, 찾고자 하는 노드에 대한 정보를 알기 위해, 색인된 노드의 ID에서 찾고자 하는 노드의 ID를 계산할 수 있어야 한다. 그러나 이 경우 각 노드에 ID가 고정되기 때문에 문서의 갱신이 발생할 때, 많은 부분이 수정되어야 하기 때문에 갱신에 필요한 오버헤드가 커지게 된다.

본 논문에서는 전체문서인스턴스트리 구조를 제안하고, 이를 기반으로 하여 노드의 ID를 구성함으로서, 색인과 검색의 효율성을 유지하면서 자료의 추가나 삭제등의 갱신이 발생할 때, 갱신의 파장을 최소화시킬 수 있는 색인구조와 질의처리 알고리즘을 제시한다.

1. 서론

일반적으로 문서는 묵시적이든 명시적이든 다양한 구조 정보를 가지고 있다. 명시적으로 텍스트에 문서의 구조를 넣은 문서를 구조적 문서(structured document)라고 한다. SGML(Standard Generalized Markup Language)이나 XML(eXtensible Markup Language)등의 마크 업 언어들은 문서가 가지는 계층적 구조를 기술하여 엘리먼트들 사이의 구조적 연관성을 표현하고 이러한 구조정보는 텍스트 브라우징이나 검색에 사용할 때 문서전체가 아닌 부분항목들로 다룸으로서 사용자 질의에서 원하는 특정 영역에 바로 접근할 수 있어 이와 같은 구조기반 정보검색시스템들에 대한 연구가 진행되고 있다[1, 2, 3, 4].

문서의 구조에 바탕을 두어서 문서의 일부를 검색하기 위해서는 문서의 구조에 대한 색인과 그 색인 구조에 바탕을 둔 질의 처리 알고리즘을 개발해야 한다. 문서 구조에 대한 색인을 할 때 주요 고려 사항은 첫째, 텍스트 속의 단어뿐 아니라 텍스트를 구성하고 있는 엘리먼트에 대하여 색인이 필요하기 때문에 색인에 관련된 메모리 오버헤드를 줄여야 하고, 둘째, 질의에 대한 응답시간이 빨라야 하고, 셋째, 문서 구조에 바탕을 둔 다양한 유형의 사용자 질의를 지원해야 하며, 넷째, 문서 구조에 대한 변경이 발생했을 때 색인 구조에 대한 변경사항을 최소화하여야 한다. 질의에 대한 응답시간을 빠르게 하기 위해서는 문서의 구조에 대하여 가능한 많은 정보를 색인 해야하므로 색인을 위한 메모리 오버헤드가 늘어난다.

본 논문에서는 하나의 DTD(Document Type Definition)를 문서 구조로 가지는 모든 문서 인스턴스들에 대한 전체문서인스턴스트리(GDIT:Global Document Instance Tree)라는 새로운 구조를 제안한다. GDIT란 문서 인스턴스의 구조를 트리로 표현했을 때, 모든 문서 인스턴스 트리의 합집합을 말한다. 그리고 문서 인스턴스에서 색인어를 포함하고 있는 텍스트 레벨 엘리먼트에 대해서만 색인하여 GDIT기반의 구조기반 검색 알고리즘을 제시한다. 이 방법은 모든 엘리먼트에 색인하는 구조기반 검색방법에 비하여 색인에 필요한 메모리와 시간 비용이 적다. 또한 기존의 텍스트 레벨 엘리먼트만 색인하는 방식에 비하여 색인에 필요한 메모리와 시간비용이 뒤떨어지지 않으면서 질의에서 찾는 엘리먼트의 레벨과 관계없이 질의처리시간이 빠르면서 일정하다. 특히 제안된 알고리즘은 문서의 갱신이 발생할 때, 갱신될 엘리먼트와 형제관계에 있는 엘리먼트들 중에서, 갱신될 엘리먼트 뒤에 위치한 동일한 이름을 가지는 엘리먼트들과 그 엘리먼트의 서브 엘리먼트에 대해서만 갱신이 발생하게 함으로서, 엘리먼트 갱신에 따른 갱신의 파장을 최소화시킬 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구에 대하여 논하고 3장에서는 GDIT 구성방법과 그 의미에 대해서 논한다. 4장에서는 색인구조를 제안한다. 5장에서는 검색과 갱신 알고리즘을 제시한다. 6장에서는 본 논문에서 제시된 색인 구조의 성능을 분석하고 7장에서는 결론을 보인다.

2. 관련 연구

구조기반검색의 기반이 되는 색인 구조로서 전통적인 방법은, SGML 문서를 트리 형태로 표현하여, 문서에서 발생하는 모든 엘리먼트를 대상으로 색인하는 것이다[1, 2]. 이 방법은 추출된 색인어가 발생된 엘리먼트의 모든 상위 엘리먼트에 대해서도 색인을 하게 되어 공간상의 중

복이 발생하여 색인 오버헤드가 크다.

문서에서 발생되는 모든 엘리먼트의 색인을 피하는 방법으로서, 문서를 k-ary 완전트리로 표현하여, 임의의 엘리먼트의 하위엘리먼트 모두에 특정 색인어가 공통적으로 존재할 경우 그 색인어를 임의의 엘리먼트에만 기억시켜 색인하는 방법과[3], 문서구조에서 색인어를 포함하고 있는 엘리먼트 중에서 색인어와 가장 가까운 레벨에 있는 엘리먼트만을 색인하는 방법이 있다[4]. 이 방법들은 자식 엘리먼트의 최대 수가 고정되어 있기 때문에, 엘리먼트 추가 시에 추가할 엘리먼트 이후의 모든 형제 엘리먼트에 대하여 그 위치 정보를 변경해야 하고, 자식 엘리먼트 수가 미리 정해진 한계를 넘을 경우, 모든 문서 인스턴스에서 발생되는 모든 엘리먼트들의 위치 정보를 변경해야 한다.

본 논문에서는 각 DTD 기반의 문서 인스턴스의 집합에 대하여 하나의 GDIT를 구성하고, 문서 인스턴스에서 색인어를 포함하고 있는 텍스트 레벨 엘리먼트만을 색인하여, GDIT기반의 구조 기반 정보 검색 알고리즘을 제시한다.

3. Global Document instance Tree (GDIT)

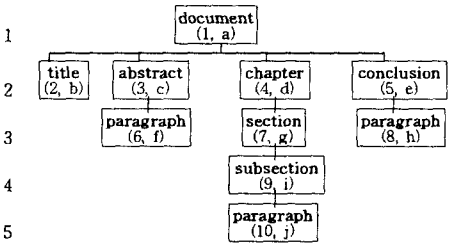
제안된 색인 구조는 GDIT에 기반을 두고 있다. 여기서는 GDIT의 구성 방법과 그것이 내포하고 있는 의미에 대하여 논한다.

3.1 GDIT 구성

GDIT는 문서 인스턴스의 구조를 트리로 표현했을 때 모든 문서 인스턴스 트리의 합집합을 말한다. GDIT를 구성하기 위해서 DTD 트리가 필요하다.

3.1.1 DTD 트리

<그림 1>과 같은 DTD 트리가 있다고 가정하자.



<그림 1> DTD 트리의 예

<그림 1>의 각 엘리먼트에는 순서쌍 (i, j)가 부여되어 있다. 여기서 i는 트리를 너비우선순회 규칙에 따라서 차례로 부여된 번호이다. 이 번호를 앞으로 DEN(Dtd Element Number)라 칭한다. j값은 문서 인스턴스에서 대응되는 엘리먼트가 실제로 발생된 횟수를 말한다.

3.1.2 GDIT 구성 알고리즘

<Algorithm 3.1>은 GDIT를 구성하는 알고리즘을 개략적으로 보인 것이다.

```
GDIT(DTDN, targetElement, N, ANCESTORS){
  If (The Nth targetElement having ANCESOTRS as its
      ancestor does not exist in GDIT){
    get the DEN of targetElement from the DTD tree;
    increment j of (i, j) attached to the DTD node;
    assign j to IEN of targetElement;
    insert a targetElement with (DEN, IEN) into GDIT;
  }
  else
      return;
}
```

<Algorithm 3.1> GDIT 구성 알고리즘

GDIT를 구성하기 위해서는 모든 문서 인스턴스 트리를 합치고, 합쳐진 각 노드에 ID로서 사용될 (DEN, IEN)값을 결정하는 두가지 작업이 필요하다. 여기서 DEN이란 3.1.1에서 제시한 DTD 엘리먼트 발생 순서를 말하고, IEN(Instance Element Number)이란 문서 인스턴스에서 해당 엘리먼트가 발생된 순서를 의미한다. 임의의 첫 번째 문서 인스턴스의 각 엘리먼트에 대하여, <그림 1>의 DTD트리로부터 DEN값을 알 수 있고, DTD노드에 부여된 순서쌍의 두 번째 값 즉, 엘리먼트가 실제로 발생된 횟수를 1증가시키고 그 값을 IEN으로 부여한다. 모든 문서 인스턴스들을 대상으로 위와 같은 과정을 반복하면, 문서 인스턴스 구조의 루트에서 텍스트 레벨 엘리먼트까지의 각 경로별로 발생하는 가능한 경로 정보들을 모두 포함하는 트리인 GDIT가 구성된다.

3.2 GDIT의 노드에 대한 의미 부여

3.2.1 DEN(Dtd Element Number)의 의미

DTD트리의 구조는 인스턴스에 무관하게 변하지 않는 구조이기 때문에, 각 엘리먼트에 할당된 DEN에 일정한 의미를 부여할 수 있다. 여기서는 각 엘리먼트에 부여된 DEN값은, 그 엘리먼트의 트리 상의 조상의 경로정보라는 의미를 부여한다. 예를 들어 <그림 1>에서 DEN 10은 조상의 경로가 document/chapter/section/subsection인 paragraph을 의미한다. 이 경로정보는 비록 DTD트리에서 유도된 정보이지만, 모든 문서 인스턴스에 대하여 변함없이 적용된다.

3.2.2 IEN(Instance Element Number)의 의미

DEN에는 발생지시자에 대한 정보가 포함되어 있지 않다. 따라서 실제 문서 인스턴스에서, 예를 들어 다수의 chapter가 있고, 각 chapter에서는 다수의 section이 있을 때, 특정 section이 속한 chapter를 DEN만으로는 결정할 수 없다. 이 문제를 해결하는 요소가 IEN이다.

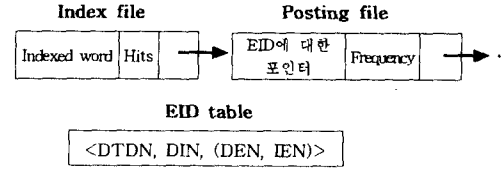
GDIT는 모든 문서 인스턴스에서 발생된 모든 엘리먼트의 발생횟수가 반영된 구조이기 때문에 IEN은 문서 인스턴스상의 조상의 경로에 관한 정보로 활용된다.

3.2.3 엘리먼트 식별자(EID : Element IDentifier)

엘리먼트 식별자는 문서 인스턴스에서 발생되는 각각의 엘리먼트를 유일하게 식별하는데 사용된다. 엘리먼트 식별자의 구성요소는 (DEN, IEN)쌍에 각 DTD문서번호(DTDN : DTD Number)와 문서 인스턴스 번호(DIN : Document Instance Number)를 추가하여 <DTDN, DIN, (DEN, IEN)>로 표현된다.

4. 색인 구조

제안된 GDIT기반의 검색을 위한 색인구조는 <그림 2>와 같다.



<그림 2> 색인구조

색인화일의 구성요소는 색인어, 색인어를 포함하고 있는 텍스트레벨 엘리먼트 갯수이고, 포스팅 파일의 구성요소는 색인어를 포함하고 있는 EID테이블에 대한 포인터, 색인어 빈도수로 구성된다. EID테이블은 색인어를 포함하고 있는 EID를 저장하고 있는 테이블이다. 이때 유의할

사항은 문서 인스턴스 엘리먼트 중에서, 색인어를 포함하고 있는 텍스트 레벨 엘리먼트만 색인함으로서 메모리 오버헤드를 줄인다. 또한 포스팅파일에서의 EID에 대한 중복 저장을 줄이고, 문서구조의 수정이 발생할 때 한 텍스트레벨 엘리먼트내에서 수정될 색인어의 개수와 무관하게 갱신을 처리하기 위하여 EID테이블을 따로 두었다.

5. 질의어 처리

5.1 검색

검색하고자 하는 색인어와 엘리먼트가 주어지면, GDIT를 사용하여, 역리스트에서 해당 색인어의 엘리먼트가 질의에서 찾는 엘리먼트의 하위 엘리먼트인지를 조사한다. 하위 엘리먼트이면 색인어 빈도수를 누적시킨다. <Algorithm 5.1>은 검색 알고리즘의 일부이다.

```
retrieval(queryWord, queryElement){
    get <keyword, frequency, <DTDN, DIN, (DEN, IEN)> from the
      queryWord in the inverted list;
    For each <keyword, frequency, <DTDN, DIN, (DEN, IEN)>>{
        get a ancestors' path of the (DEN, IEN) from the GDIT;
        store the path into a record array PATH;
        get a triple (DTDN' , DEN' , Level) for the queryElement;
        If (DTDN == DTDN' )
            If(DEN' == PATH[Level].DEN){
                weight[PATH[Level]] += frequency;
                break;
            }
    }
}
```

<Algorithm 5.1> 검색 알고리즘의 일부

5.2 삭제

삭제하고자 하는 문서번호, DTD번호, 엘리먼트및 엘리먼트까지의 조상의 경로가 주어지면, 먼저 조상의 경로를 이용해서 GDIT로부터 삭제할 엘리먼트의 (DEN, IEN)을 구한다. 삭제할 엘리먼트가 텍스트 레벨 엘리먼트가 아니면, 문서에서 삭제할 엘리먼트의 하위 문서 구조에서 텍스트 레벨 엘리먼트들을 찾고, 그 엘리먼트들의 (DEN, IEN)을 찾아서 <DTD번호, 문서번호, (DEN, IEN)>을 결정한다. 역리스트에서 이 순서쌍과 같은 값을 가지는 색인어에 대하여 해당 포스팅 엔트리를 삭제한다. 본 논문에서는 텍스트레벨 엘리먼트에 대해서만 색인을 하기 때문에, 텍스트레벨 엘리먼트가 아닌 것은 삭제 대상에서 제외된다.

5.3 추가

DTD와 문서번호, 추가할 엘리먼트, 추가할 엘리먼트까지의 조상의 경로가 주어지면, 추가할 엘리먼트의 텍스트 레벨 엘리먼트를 색인어와 함께 역리스트에 추가한다. 추가할 엘리먼트 뒤에 위치하는 엘리먼트 중에서 이름이 같은 엘리먼트들과 그 하위 엘리먼트들은 문서에서 위치가 변경되는 엘리먼트들이므로 EID가 수정되어야 한다. GDIT를 이용해서 수정될 엘리먼트의 새로운 EID값을 결정하고, <그림 2>의 EID 테이블에서 엘리먼트의 EID를 수정한다.

6. 평가

6.1. 색인비용

전통적인 구조 기반 검색에서는 문서에서 색인어가 나타난 텍스트 레벨 엘리먼트의 모든 조상 엘리먼트들에 대해서도 색인하므로 중복색인이 된다. 그러므로 색인에 드는 메모리는 문서에 있는 엘리먼트의 갯수에 영향을 받는다. 본 논문에서는 색인어를 포함하는 텍스트 레벨 엘리먼트에 대해서만 색인하므로 색인에 드는 메모리양은 텍스트 레벨 엘리먼트의 갯수에만 영향을 받는다. 한편 색인시간에 관해서는, 문서가 완전트리라고 가정하면 문서의 모든 레벨에서 색인하는 전통적인 검색방법보다 1/문서의 깊이(d)로 색인시간이 줄어든다.

6.2. 검색비용

레벨우선순회규칙으로 ID를 할당하여 텍스트 레벨 엘리먼트만 색인하는 방법은 텍스트 레벨에서부터 질의에서 찾고자 하는 엘리먼트 레벨까지 부모 노드의 ID값을 계산하는 과정을 반복하므로, 색인어가 존재하는 텍스트 레벨과 질의에서 찾고자 하는 엘리먼트 레벨과의 차이에 영향을 받는다.

본 논문에서는 모든 문서 인스턴스에 발생된 엘리먼트의 위치를 기억하고 있는 GDIT의 각 엘리먼트에 부여된 값의 쌍 (DEN, IEN)이 조상 엘리먼트들의 (DEN, IEN)을 나타내도록 하고 엘리먼트의 ID로 사용함으로서, 텍스트 레벨 엘리먼트에 부여된 (DEN, IEN)에서 조상 엘리먼트들의 (DEN, IEN)을 알 수 있다. 그러므로, 질의에서 찾는 엘리먼트 레벨과 색인어를 포함하는 텍스트 레벨과의 차이에 관계없이 검색시간이 동일하게 소요된다.

6.3. 갱신비용

레벨우선순회규칙으로 ID를 할당하여 텍스트 레벨 엘리먼트만 색인하는 방식은 자식 엘리먼트의 최대 수를 고정시키므로, 엘리먼트가 추가되어 자식 엘리먼트의 최대 수를 넘게 될 경우에는 저장되어 있는 모든 문서 인스턴스의 엘리먼트들을 갱신해야 한다. 문서에서 정한 자식 엘리먼트의 최대 수를 넘지 않을 경우에는, 추가된 엘리먼트뒤에 위치하고 있는 모든 형제엘리먼트의 ID가 변경되어야 한다. 그러므로 추가된 엘리먼트 뒤에 위치하는 모든 형제 엘리먼트의 텍스트 레벨 엘리먼트에 대해 갱신이 발생한다.

본 논문에서는 모든 문서 인스턴스에서 발생된 엘리먼트의 갯수를 반영하고 있는 GDIT의 각 엘리먼트에 (DEN, IEN)을 부여한 다음 (DEN, IEN)을 엘리먼트의 ID로 사용함으로, 엘리먼트가 추가되면 추가된 엘리먼트의 뒤에 위치하는 형제엘리먼트중에서 같은 이름의 형제 엘리먼트의 텍스트 레벨 엘리먼트에 대해 갱신이 발생한다. 갱신 수행시 텍스트 레벨 엘리먼트에 존재하는 색인어마다 갱신을 수행하지 않고 텍스트 레벨 엘리먼트 개수만큼 갱신을 수행하기 위하여, EID테이블을 사용하여 색인구조에 텍스트 레벨 엘리먼트의 EID대신에 텍스트 레벨 엘리먼트의 EID에 대한 포인터를 둔다.

7. 결론

효율적인 구조기반검색을 위해서는 엘리먼트의 색인에 따르는 메모리 오버헤드를 줄이고, 검색시간을 빠르게 하면서 문서 구조의 변경도 효율적으로 처리되어야 한다. 이를 위해 본 논문에서는 GDIT기반의 새로운 색인구조와 질의처리 알고리즘을 제시하였다. 제시된 알고리즘은 색인과 검색시간의 효율성을 유지하면서, 특히 문서구조의 갱신을 효율적으로 처리한다. 앞으로 성능에 관한 다양한 실험자료가 더 필요하다.

참고 문헌

[1] 이희주, 장재우, 심부성, 주종철, "구조화 문서를 위한 정보 검색 인덱스의 설계," 정보과학회 가을 학술발표논문집 Vol. 24, No. 2, pp.337-340, 1997

[2] R. Sacks-Davis, T. Arnold-Moore and J. Zobel, "Database Systems for Structured Documents," Proceedings of the International Symposium on Advanced Database Technologies and Their Integration(ADTI '94), Nara, Japan, October 1994.

[3] Lee, Y.K. Yoo, S.J. Yoon, K. Berra, P.B. "Index Structure for Structured Documents," in Proc. Digital Library '96 pp. 91-99.

[4] Shin, D.W. Jang, H.C. Jin, H.L. "BUS:An Effective Indexing and Retrieval Scheme in Structured Documents," Proceedings of the Digitial Libraries, 1998