論文
2-8-9

구조적 문서의 효율적인 구조 질의 처리 및 검색을 위한 알고리즘의 설계

(Design of Algorithm for Efficient Retrieve Pure Structure-Based Query Processing and Retrieve in Structured Document)

김 현 주*
(Hyun-Ju Kim)

요 약

구조적 문서가 가지는 구조 정보는 문서로의 다양한 접근경로를 나타내는데 사용될 수 있다. 이러한 구조적 문서가 가지는 구조 정보를 활용하기 위해서는 문서의 구조에 대해서 색인을 해야 한다. 이때 내용색인뿐만 아니라 문서마다 구조정보를 저장하므로 색인에 필요한 공간이 커진다. 그러므로, 색인공간 오버헤드를 최소화시키면서도 엘리먼트간의 포함관계나 순서등 문서의 순수 구조에 바탕을 둔 순수 구조 질의를 처리할 수 있어야 한다. 본 논문에서는 색인공간 오버헤드를 최소화하면서도 여러 유형의 구조 관련 질의를 효율적으로 처리할 수 있는 구조 색인 구조와 GDIT자료구조를 제시한다.

제안하는 구조 색인 구조는 문서에 존재하는 가장 하위 엘리먼트만을 색인대상으로 하며, 검색엘리먼트가 존재하는 문서개수에 영향을 받지 않는다. 그리고 이 색인구조를 바탕으로 순수 구조에 대한 질의 처리과정을 보이고 색인공간에 대해 그 성능을 평가한다. 제안된 색인 구조는 GDIT개념[2]에 바탕을 두고, GDIT기반의 색인기법을 사용한다.

ABSTRACT

Structure information contained in a structured document supports various access paths to document. In order to use structure information contained in a structured document, it is required to construct an index structural on document structures. Content indexing and structure indexing per document require high memory overhead. Therefore, processing of pure structure queries based on document structure like relationship between elements or element orders, low memory overhead for indexing are required.

This paper suggests the GDIT(Global Document Instance Tree) data structure and indexing scheme about structure of document which supports low memory overhead for indexing and powerful types of user queries. The structure indexing scheme only index the lowest level element of document and does not effect number of document having retrieval element. Based on the index structure, we propose an query processing algorithm about pure structure, proof the indexing schemes keeps up indexing efficient in terms of space. The proposed index structure bases GDIT concept and uses index technique based on GDIT.

* 정회원 : 경남정보대학 컴퓨터정보시스템계열 전임강사

논문접수 : 2001. 8. 3.
심사완료 : 2001. 8. 21.

1. 서론

정보 검색 시스템이 구조적 문서가 가지는 구조 정보를 활용한 질의를 처리할 수 있는 검색 기능을 가지기 위해서는 색인이 엘리먼트 수준에서 수행되는 것 이외에 문서의 구조가 저장되어야 한다. 그러므로, 구조 정보를 활용한 여러 유형의 질의를 처리하기 위해서는 그만큼 더 많은 구조 정보가 색인되어야 하고, 색인되는 내용이 많아지면 색인 공간이 커지는 문제점이 있다. 결국 질의 처리 범위가 커짐으로서 문제가 되는 것은 색인에 필요한 공간이다. 그러므로 질의 처리 범위는 넓히면서 색인에 필요한 공간 오버헤드는 줄여야 한다. 내용 색인에서 색인 공간을 줄이는 대표적인 기법은 색인어가 존재하는 가장 하위 엘리먼트 만을 색인하는 것이다[4, 5]. 문제는 여기에 문서의 트리 구조를 저장하지 않으면서, 색인되는 엘리먼트에 문서가 가지는 구조 정보를 효율적으로 함축적으로 저장하는 것이 어렵다는 것이다. 지금까지의 구조 색인 방법들은[1, 2] 색인 공간에 대한 오버헤드가 크고, 질의 처리시에 검색 엘리먼트가 있는 문서 구조 트리마다 검색하므로, 검색 엘리먼트가 존재하는 문서 개수에 영향을 받는다.

본 논문에서는 색인 공간 오버헤드를 최소화하면서도 다양한 구조 질의 유형들을 처리할 수 있는 구조색인 방법을 제시한다. 그리고 구조 색인 방법을 바탕으로 순수 구조에 관련된 질의 처리 알고리즘을 제시한다. 제안하는 구조 색인 방법은 순수 구조 질의 처리 시에 검색 엘리먼트가 존재하는 문서 개수에 상관없이 질의 처리 시간이 일정하다. 제안하는 구조 색인 방법은 GDIT[2]의 개념에 바탕을 두고 있다. GDIT는 문서 인스턴스 구조를 트리로 표현했을 때, 모든 문서 인스턴스 트리의 합집합이다. GDIT에는 모든 문서 인스턴스에 존재하는 엘리먼트와 엘리먼트 간의 구조 정보를 포함하게 된다. 이러한 GDIT가 가지는 구조 정보를 문서 색인 시에 사용함으로서 각 문서의 구조 트리 정보를 저장하지 않으면서 구조적 문서에 대해 발생할 수 있는 여러 유형의 구조 질의를 효율적으로 처리할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대하여 논하고 3장에서는 GDIT 구성 방법과 그 의미에 대해서 논한다. 4장에서는 구조적 문서의 구조 정보를 재구성하여 GDIT를 기반으로 한 구조적 문서의 검색 모델과 색인 구조에 대하여 논한다. 5장에서는 구조 관련 질의를 효율적으로 처리하는데 필요한 GDIT 자료 구조를 제시한다. 6장에서는 4장에서 제시한 색인 구조를 기반으로 순수 구조에 대한 질의 처리 과정을 보인다. 7장에서는 본 논문에서 제시한 색인 구조에 대해 색인 비용에 대해서 분석하고 8장에서 결론을 보인다.

2. 관련 연구

구조 질의 처리를 위하여 구조적 문서를 표현하는 데이터 모델에 관한 연구에는 크게 기존의 DB 모델을 사용하는 방법과 직접 엘리먼트 단위로 저장하는 엘리먼트 기반 모델을 사용하는 방법이 있다.

기존의 DB 모델을 사용하는 방법은 관계형, nested 관계형, 객체지향 모델 등을 사용하여, 구조적 문서를 각 모델에 맞게 변환하여 표현하는 방법이다[4, 5, 7]. 이 방법은 구조적 문서의 SGML DTD를 각 모델의 스키마로 매핑하는 과정을 수행한다. 그런데 이 과정에서 DTD가 가지고 있는 모든 정보를 표현하기가 어려우며, DTD가 가지고 있는 모든 정보를 표현하더라도 힘든 변환 과정을 수행하거나 구조적 문서를 처리하기 위해서는 시스템의 확장이 불가피하다. 그러므로 질의 처리 시에 단순 질의에도 많은 연산 과정을 수행해야 하며 순수 구조에 관련된 질의들은 처리하기 어렵다.

다음으로, 엘리먼트 기반 모델을 사용하는 방법은 DTD를 변환하는 과정을 수행하지 않고 직접 엘리먼트 단위로 저장하는 방법이다[1, 3, 6, 8, 9]. 이에 대한 연구로서 전통적인 방법은 문서에 나타나는 모든 엘리먼트들을 색인 대상으로 하는 방법이다.[1, 9]. 이 방법은 추출된 색인어가 발생된 엘리먼트의 모든 상위 엘리먼트에 대해서도 색인을 하게 되어 공간상의 중복이 발생하여 색인에 필요한 메모리 오버헤드가 크다는 문제점이 있다.

문서에서 발생되는 모든 엘리먼트의 색인을 피하는 방법으로서, 색인어가 존재하는 텍스트 레벨 엘리먼트만 색인하거나 임의의 엘리먼트의 하위 엘리먼트 모두에 특정 색인어가 공통적으로 존재할 경우 하위 엘리먼트들은 색인하지 않고 그 색인어를 임의의 엘리먼트에만 기억시켜 색인하는 방법이다[3, 6].

이 방법은 모든 엘리먼트의 색인을 피함으로서, 색인에 필요한 메모리 오버헤드를 줄일 수 있으나, 내용색인만 고려함으로서, 구조에 관련된 질의를 처리하는데 제약이 있다. 다음으로 내용 색인 시에 색인어마다 루트까지의 경로에 존재하는 엘리먼트들의 정보를 나타내는, 경로 정보를 저장하는 방법이다[8]. 이 방법은 하나의 색인 구조만으로 구조에 관련된 여러 질의를 처리할 수 있으나, 이로 인하여, 색인어마다 조상 엘리먼트들의 경로 정보를 저장하므로 색인에 필요한 메모리 오버헤드가 있으며, 순수 구조에 관련된 질의들은 처리하기 어렵다.

본 논문에서는 GDIT를 기반으로 한 구조적 문서에 대한 구조 색인 구조를 제시하고 이를 기반으로 하여 순수 구조에 대한 질의와 내용과 구조가 혼합된 질의에 대한 처리 알고리즘을 제시한다. 제시한 모델은 검색의 효율성을 유지하면서 색인 공간 오버헤드를 최소화한다. 제시한 검색 모델을 색인 공간을 기준으로 평가한다.

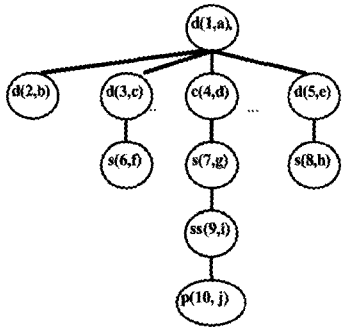
3. GDIT 개요

제안된 색인 구조들은 GDIT[2]에 기반을 두고 있다. 여기서는 GDIT의 구성 방법과 그것이 내포하고 있는 의미를 소개한다.

3.1 DTD 트리

GDIT는 문서 인스턴스 구조를 트리로 표현했을 때 모든 문서 인스턴스 트리의 합집합을 말한다. GDIT를 구성하기 위해서 DTD 트리가 필요하다. [그림 1]과 같은 DTD 트리가 있다고 가정하자. [그림 1]에서 d는 Document, c는 Chapter, s는 Section, ss는 Subsection, p는 Paragraph 엘리먼트를 나타낸다고 가정한다.

[그림 1]의 각 엘리먼트에는 순서쌍 (i, j)가 부여되어 있다. 여기서 i는 트리를 너비 우선 순회 규칙에 따라서 차례로 부여된 번호이다. 이 번호를 앞으로 DEN(Dtd Element Number)라 칭한다. DEN은 루트에서부터 각 엘리먼트까지의 경로 정보를 나타내는 것으로 활용된다. j는 문서 인스턴스에서 대응되는 엘리먼트가 실제로 발생된 횟수를 말한다.

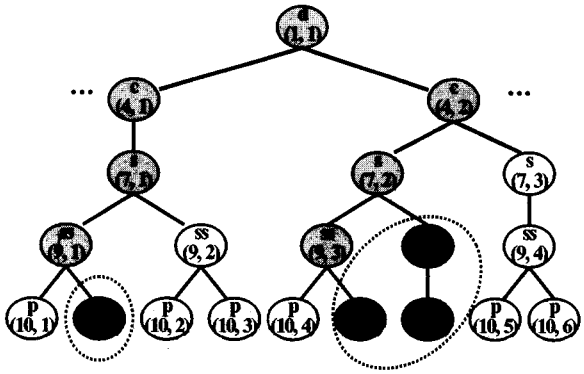


[그림 1] DTD 트리의 예
[Fig. 1] example of DTD tree

3.2 GDIT 구성

GDIT를 구성하기 위해서는 문서 트리를 합치고, 합쳐진 각 노드에 ID로 사용될 (DEN, IEN)값을 결정하는 두 가지 작업이 필요하다. [그림 2]는 임의의 두개의 문서에 대한 GDIT이다. 각 엘리먼트에는 순서쌍 (i, j)가 부여되어 있다. [그림 2]의 GDIT에서 j는 IEN(Instance Element Number)으로 GDIT에서 해당 엘리트가 발생된 순서대로 부여한 번호이며, 앞으로 조상의 경로에 대한 순서 정보를 나타내는 것으로 활용된다. 예를 들어, (10, 7)을 가지는 p엘리먼트의 IEN 7의 의미는 조상의 경로가 2-1-1-1임을 의미하는 것으로 고정된다.

[그림 2]에서 점선으로 묶인 부분을 제외한 부분이 임의의 첫 번째 문서이고, 점선으로 묶인 부분 (9, 5), (10, 7), (10, 8), (10, 9)는 두 번째 문서에는 있으나 첫 번째 문서에는 없는 구조이다. 남아 있는 모든 문서들을 대상으로 위와 같은 과정을 반복하면, 문서 구조의 루트에서 텍스트 레벨 엘리먼트 까지의 각 경로별로 가능한 경로 정보들을 모두 포함하는 트리인 GDIT가 구성된다. 즉 GDIT는 모든 문서에서 발생된 엘리먼트들의 개수가 모두 반영된다. 그러므로, GDIT의 각 엘리먼트에 부여된 값의 쌍 (DEN, IEN)은 문서 내에서 엘리먼트의 위치를 나타내게 되어, 문서에 존재하는 엘리먼트를 식별하는데 사용할 수 있다. GDIT의 각 엘리먼트에 부여된 DEN값은 그 엘리먼트의 DTD트리 상의 조상의 경로정보를, IEN은 문서 인스턴스상의 조상의 경로에 관한 정보라는 의미를 부여하여 엘리먼트 식별자의



[그림 2] GDIT의 일부분

[Fig. 2] part of GDIT

구성요소는 (DEN, IEN)쌍에 문서 인스턴스 번호(DIN: Document Instance Number)를 추가하여 <DIN, (DEN, IEN)>로 표현한다.

4. 구조적 문서에 대한 검색 모델

4.1 구조적 문서의 검색 모델

일반적으로 문서의 구조 정보를 활용할 때에 구조 정보를 활용하는 질의 처리 범위가 클수록 그만큼 더 많은 구조 정보를 색인해야 함으로 색인 공간 비용이 커진다. 또한 순수 구조에 관한 질의 처리 시에 문서의 특정 구조에 관한 질의일수록, 즉 질의를 만족하는 문서 개수가 적을수록, 검색 엘리먼트가 존재하는 문서마다 구조 검색을 하는 것은 비효율적이다. 그러므로, 구조적 문서의 구조 정보를 활용하면서도 색인 공간의 오버헤드는 최소화하고 검색 시간은 빠른 색인 구조가 필요하다.

본 논문에서는 GDIT를 기반으로 하여 다음과 같은 구조 색인 구조와 질의 처리 방법을 사용하여 이러한 문제점을 해결한다.

- 문서에 존재하는 엘리먼트들 중에서 단말 엘리먼트들 만을 색인한다. 단말 엘리먼트들의 (DEN, IEN)에서 조상 엘리먼트들의 (DEN, IEN)들을 알 수 있으므로, 임의의 엘리먼트가 있는 문서들은 임의의 엘리먼트의 하위 문서 구조에 존재하는 단말 엘리먼트들이 가지는 색인 리스트의 합이다.

  INDEX[leaf node1] ∪ INDEX[leaf node2] ∪...∪ INDEX[leaf noden]

- 색인 시에 구조 정보의 중복 색인을 최대한으로 줄이기 위해 GDIT를 사용하여 문서 구조에 나타날 수 있는 엘리먼트의 구조 정보별로 문서 번호를 색인한다.
- 질의 처리는 다음의 방법을 사용한다.
- 내용과 구조의 혼합 질의의 경우 내용에 관한 질의를 먼저 처리하여 처리 영역을 줄인 다음, 구조에 관련된 질의를 처리한다.
- 순수 구조 질의의 경우 GDIT를 사용하여 질의에서 찾는 검색 엘리먼트의 (DEN, IEN)정보를 추출한 다음, 색인 구조에서 해당 정보에 관한 문서 번호를 바로 추출한다. 그러므로, 검색 엘리먼트가 존재하는 문서별 검색을 할 필요가 없으므로 문서개수에 영향을 받지 않고 검색시간이 빠르며 질의 처리 시간이 일정하다.

4.2 색인 구조

[그림 3]은 본 논문에서 제안하는 구조 색인 구조이다. 색인 파일은 GDIT에 존재하는 엘리먼트의 (DEN, IEN)으로 구성되며, 포스팅 파일에는 해당 엘리먼트가 있는 문서 번호가 색인된다.

[그림 3] 구조적 색인
[Fig. 3] Structed index

이러한 색인구조 구성은 다음과 같은 장점을 가진다.

- 색인 구조에서 (DEN, IEN) 정보에 해당하는 문서 번호를 바로 추출할 수 있으므로 질의 처리 시간이 빠르다.
- 질의 처리 시에 검색 엘리먼트가 존재하는 문서별 검색을 하지 않고, 질의에서 찾는 구조에 대한 문서를 바로 추출하므로 검색 엘리먼트가 존재하는 문서 개수에 상관없이 질의 처리 시간이 일정하다.
- 질의에서 찾는 구조가 특정 문서에 존재하는 세부적인 구조 질의일수록 효과가 있다.

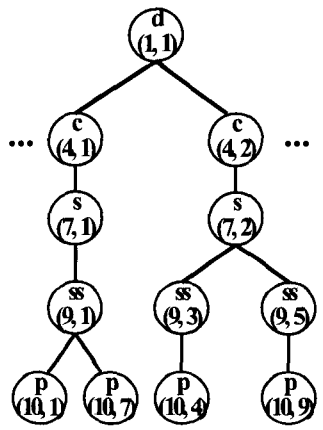
[그림 4]는 [그림 2]의 GDIT를 따르는 문서이고, 문서번호 DIN이 1이라고 가정하자. 각 엘리먼트 타입별로 색인파일을 구성할 엘리먼트들은 다음과 같다.

index(H) = {}
index(S) = {}
index(B) = {}
index(C) = {}
index(G) = {}
index(P) = {P1, P2, P3, P4, P5, P6, P7, P8, P9}

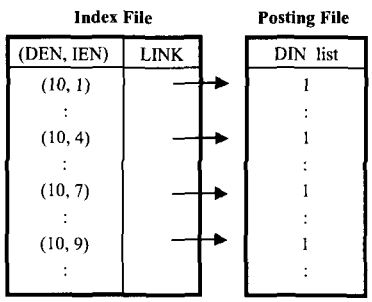
색인 파일을 구성하는 엘리먼트들 중에서 [그림 4]의 문서에 대하여 역 리스트에 문서 번호 DIN이 나타날 엘리먼트들은 다음과 같다.

index-element(P) = {P1, P4, P7, P9}

결국 [그림 4]의 문서 구조에 대한 구조 색인 구조는 [그림 5]와 같다. [그림 5]의 구조 색인 구조에서 색인 파일은 GDIT에 존재하는 엘리먼트의 (DEN, IEN)으로 구성되며, 포스팅 파일은 (DEN, IEN)이 존재하는 문서 번호 DIN 리스트로 구성된다. 순수 구조에 관한 질의 처리시에 구조에 해당하는 GDIT에서 관련 (DEN, IEN) 정보를 가지고 색인 구조에서 해당 문서 번호를 추출한다.



[그림 4] 임의의 문서 인스턴스 구조
[Fig. 4] Instance structure of any document



[그림 5] 구조적 색인구조
[Fig. 5] Structured index sturecture

5. 색인 구조들에 대한 GDIT 구성 정보

[그림 3]에서 제안한 문서 구조에 대한 색인 구조들은 색인 파일에 GDIT에 존재하는 엘리먼트의 (DEN, IEN) 정보를 사용한다. 그러므로 구조색인 구조에 접근하기 이전에 질의에서 찾고자 하는 구조와 관련된 엘리먼트의 (DEN, IEN) 정보를 추출하는 것이 중요하다. 이 정보를 가지고 있는 것이 GDIT이므로, 효율적인 질의 처리가 되기 위해서 GDIT를 재구성할 필요가 있다. 본 장에서는 3장에서 제안한 GDIT를 기반으로 하여 [3]에서 제안한 색인 구조들이 문서의 구조와 구조사이의 관계를 포함하는 질의 등 문서의 순수 구조에 관련된 질의를 처리하기 위해 필요한 GDIT 정보를 재구성한 GDIT 자료 구조를 제안한다.

5.1 GDIT 자료 구조

[그림 6]은 GDIT를 테이블로 재구성한 것이다. [그림 6]에서 (a)는 GDIT에서 DEN에 관한 정보를 중심으로 추출한 것이다. 이것은 DTD 트리에 존재하는 엘리먼트에 대한 정보가 된다.

Element_name	Ancestor's path	Ancestor's DEN list	Element_inf

Element_name : 엘리먼트명
Ancestor's path : 조상의 경로
Ancestor's DEN list : 조상의 DEN 리스트
Element_inf : 엘리먼트 관련 정보 <DEN, level>

(a) DEN 정보

DEN	Ancestor's Order	Ancestors' IEN order	(DEN, IEN) list

DEN : 엘리먼트의 DEN
Ancestor's Order : 루트부터 엘리먼트까지의 조상의 구체적인 순서
Ancestor's IEN order : 조상 엘리먼트들의 IEN 리스트
(DEN, IEN) list : 하위 문서 구조에 존재하는 단말 엘리먼트들의 (DEN, IEN)리스트

(b) IEN 정보

[그림 6] GDIT 테이블
[Fig. 6] GDIT Table

엘리먼트 별로 조상의 경로, 조상의 DEN 리스트, 그러한 조상 엘리먼트를 가지는 엘리먼트의 DEN과 레벨 정보가 들어가게 된다. [그림 6]의 (b)는 (a)에서 나타난 각 엘리먼트에 대해 GDIT에서의 IEN 정보를 중심으로 나타낸 것이다. DTD의 각 엘리먼트별로, 실제 문서 인스턴스에서 엘리먼트들이 가지는 조상의 순서에 따라, 엘리먼트의 ID 정보나 최하위 엘리먼트들의 ID 정보 등 각 색인 구조들에 따라 필요한 정보가 들어가게 된다.

4.2의 구조색인 구조는 문서 구조에서 가장 하위 엘리먼트 만을 색인한다. 그러므로 (b)는 DTD의 엘리먼트 별로 실제 문서에서 나타날 수 있는 단말 엘리먼트들의 (DEN, IEN) 정보를 나타낸 것이다.

6. 질의 평가

구조적 문서에 대해서 문서가 가지는 논리적인 구조에 대한 질의가 발생할 수 있다. 여기서는 엘리먼트 간의 순서가 포함된 순수 구조에 대한 질의 처리 과정을 보인다.

6.1 순서가 있는 순수 구조 질의 처리

예를 들어, "두 번째 <chapter>의 첫 번째 <section>을 검색하라" 라는 질의가 있다고 가정하자. 질의 처리는 다음과 같은 3단계로 구성된다.

단계 1 : DTD 트리에서 검색 엘리먼트까지의 조상 엘리먼트의 DEN 리스트를 추출한다. 그림 1의 DTD에서 조상이 <chapter>인 <section> 엘리먼트를 찾은 다음 <section>까지의 조상 엘리먼트들의 DEN 리스트 (1, 4, 7)를 추출한다. 조상 엘리먼트의 DEN 리스트 (1, 4, 7)이므로 document의 DEN은 1번, chapter의 DEN은 4번, section 엘리먼트의 DEN은 7번이고 레벨 3에 위치한다.

단계 2 : GDIT에서 검색 엘리먼트의 하위 문서 구조에 존재하는 단말 엘리먼트의 구조 색인 리스트에 질의에서 찾는 문서 구조가 존재한다. GDIT에서 검색 엘리먼트의 하위 문

서 구조에 존재하는 단말 엘리먼트들의 (DEN, IEN) (7,2)를 추출한다.

단계 3 : 단계 2에서 추출한 단말 엘리먼트의 (DEN, IEN)을 가지고 GDIT테이블에서 관련 문서 번호를 추출한다. 검색 엘리먼트의 (DEN, IEN)이 (7, 2)이므로, [그림 2]의 GDIT에서 (7, 2)의 하위 문서 구조에 존재하는 단말 엘리먼트(10, 4), (10, 8), (10, 9)을 추출한 다음 구조 색인 구조에서 (10, 4), (10, 8), (10, 9)의 문서 번호 DIN을 추출한다.

6.2 엘리먼트간의 관계가 포함된 순수 구조 질의

예를 들어, "부모가 <chapter>인 <section>엘리먼트를 찾아라."라는 질의가 있다고 가정하자. 질의 처리는 다음과 같은 3단계로 구성된다.

단계 1 : DTD트리에서 검색 엘리먼트까지의 조상 엘리먼트의 DEN리스트를 추출한다. 그림 1의 DTD에서 조상이 <chapter>인 <section>엘리먼트를 찾은 다음 <section>까지의 조상 엘리먼트들의 DEN 리스트 (1, 4, 7)를 추출한다. 조상 엘리먼트의 DEN 리스트 (1, 4, 7)이므로 document의 DEN은 1번, chapter의 DEN은 4번, section엘리먼트의 DEN은 7번이고 레벨 3에 위치한다.

단계 2 : 단계 1에서 구한 부모가 <chapter>인 <section>엘리먼트의 DEN은 7번이므로, 구조 색인 구조에서 DEN번호 7의 문서 번호 DIN을 추출한다.

7. 평가

본 장에서는 본 논문에서 제시한 엘리먼트 색인 구조의 성능을 분석한다.

7.1 평가조건

다음은 본 논문에서 제시된 색인구조들의 색인에 필요한 기억공간을 분석하기 위하여 사용되는 기호의 일부이다.

h : GDIT의 높이 k : GDIT의 차수

d : DTD트리의 루트부터 단말엘리먼트까지를 하나의 경로로 가정할때 DTD트리가 가지는 개수

m : DTD트리의 각 단말 엘리먼트마다 GDIT에서 발생하는 노드 개수

n_{doc} : 문서 인스턴스의 총 개수

$n_{element}$: n_{doc}개의 전체 문서 인스턴스에 존재하는 노드들의 전체 개수

$key\_element$: 색인어로 사용되는 엘리먼트 정보

$n_{key\_element}$: 색인어로 사용될 엘리먼트 정보의 총 갯수

S_{EID} : EID(Element IDentifier)의 크기

$S_{key\_element}$: 색인어로 사용될 엘리먼트 정보의 평균 크기

S_{ptr} : 포인터 크기

색인에 필요한 기억공간을 분석하기 위하여 다음과 같은 가정을 한다.

① DTD 내에 DEN이 같은 엘리먼트는 존재하지 않는다.

② GDIT는 차수가 k인 완전트리이다.

③ 임의의 문서 인스턴스의 레벨 i의 임의의 노드가 차수가 m일 확률이 $\frac{1}{k}(1 \le m \le k)$이다.

④ 문서인스턴스의 총개수(n_{doc})는 GDIT에서 발생할 수 있는 모든 가능한 문서 인스턴스 경우들의 합이다.

⑤ 문서인스턴스에서 임의의 엘리먼트가 나타날 경우 DTD트리에서 임의의 엘리먼트가 존재하는 경로보다 앞에 위치하는 경로에 존재하는 엘리먼트들이 모두 나타난다.

7.2 전체문서 개수와 전체 노드 개수에 대한 평가

k와 d의 관계에 따라, 레벨 h에는 각 $k^{h-2}*\frac{k}{d}$ 개의 노드들은 서로 같은 조상을 가지는 같은 종류의 엘리먼트들이다. 즉, DTD트리의 각 단말 엘리먼트마다 GDIT에서 발생하는 노드 개수 m은 $k^{h-2}*\frac{k}{d}$이다. 그러므로, 발생할 수 있는 문서 인스턴스의 총개수 n_{doc}는 다음과 같다.

$$n_{doc} = (k^{h-2}*\frac{k}{d})^{d} + (k^{h-2}*\frac{k}{d})^{d-1} + \cdots + (k^{h-2}*\frac{k}{d})^{1} = m^{d} + m^{d-1} + \cdots + m^{1} = \sum_{i=1}^{d} m^{i} \quad (1)$$

이와 같은 n_{doc}개의 전체 문서 인스턴스에 존재하는 노드 개수의 합을 구하면 다음과 같다.

레벨 h 에 m개의 노드가 d개 존재할 경우, 발생할 수 있는 노드 개수는 다음과 같다.

$$\sum_{p=1}^{m}(p*m^{d-1} + \sum_{n=1}^{m}(n*m^{d-2} + \cdots + \sum_{l=1}^{m}(l*m^{2} + \sum_{j=1}^{m}(j*m + \sum_{i=1}^{m} i))\cdots))$$

이와 같은 m개의 노드가 1개 존재하는 경우부터 d개 존재하는 경우까지 발생하는 노드 개수를 모두 합하면, n_{doc}개의 전체 문서 인스턴스의 레벨 h에 나타날 수 있는 노드의 전체 개수 $n_{leaf_element}$가 된다.

$$n_{leaf\_element} = \sum_{i=1}^{m} i + \sum_{j=1}^{m}(j*m + \sum_{i=1}^{m} i) + \sum_{l=1}^{m}(l*m^{2} + \sum_{j=1}^{m}(j*m + \sum_{i=1}^{m} i)) + \cdots + \sum_{q=1}^{m}(q*m^{d-1} + \sum_{p=1}^{m}(p*m^{d-2} + \cdots + \sum_{n=1}^{m}(n*m^{2} + \sum_{j=1}^{m}(j*m + \sum_{i=1}^{m} i))\cdots)) \quad (2)$$

n_{doc}개의 문서 인스턴스들의 1레벨부터 h레벨까지 나타날 수 있는 노드 개수들의 전체 합 $n_{element}$는 다음과 같다.

$$n_{element} = (m^{1} + m^{2} + \cdots + m^{d-1} + m^{d}) + \sum_{l=2}^{h}\{\sum_{i=1}^{\frac{m}{k^{h-l}}} i + \sum_{j=1}^{\frac{m}{k^{h-l}}}(j*m + \sum_{i=1}^{\frac{m}{k^{h-l}}} i) + \sum_{n=1}^{\frac{m}{k^{h-l}}}(n*m^{2} + \sum_{j=1}^{\frac{m}{k^{h-l}}}(j*m + \sum_{i=1}^{\frac{m}{k^{h-l}}} i)) + \cdots + \sum_{q=1}^{\frac{m}{k^{h-l}}}(q*m^{d-1} + \sum_{p=1}^{\frac{m}{k^{h-l}}}(p*m^{d-2} + \cdots + \sum_{n=1}^{\frac{m}{k^{h-l}}}(n*m^{2} + \sum_{j=1}^{\frac{m}{k^{h-l}}}(j*m + \sum_{i=1}^{\frac{m}{k^{h-l}}} i))\cdots)))\} \quad (3)$$

그러므로, 임의의 문서 인스턴스에 존재하는 평균적인 노드 개수는 n_{doc}개의 문서 인스턴스에 존재하는 노드 개수의 합 $n_{element}$를 전체 문서 인스턴스 갯수 n_{doc}로 나눈 것이다.

$$\frac{n_{element}}{n_{doc}} = \frac{(3)}{n_{doc}} \quad (4)$$

7.3 색인공간 분석

다음은 구조 색인 구조의 색인에 필요한 기억공간을 분석하기 위하여 사용되는 기호의 일부이다.

S_{id_e} : 색인 정보의 평균적인 포스팅 크기
$key\_element$: 색인어로 사용될 엘리먼트 정보
$S_{key\_element}$: 색인어로 사용될 엘리먼트 정보의 평균 크기
$n_{key\_element}$: 색인어로 사용될 엘리먼트 정보의 전체 개수

색인 파일에 필요한 공간 $S_{index-f}$는 색인어로 사용될 엘리먼트 정보의 평균 크기 $S_{key_element}$와 포인터 크기 S_{ptr}를 더한 다음, 색인어로 사용될 엘리먼트 정보의 전체 개수 $n_{key_element}$를 곱한 것이다.

$$S_{index-f} = (S_{key\_element} + S_{ptr}) * n_{key\_element}$$

포스팅 파일에 필요한 공간 $S_{posting-f}$는 색인 정보의 전체 개수 n_{index}에 색인정보의 평균 크기 S_{id_e}를 곱한 것이다.

$$S_{posting-f} = n_{index} * S_{id\_e}$$

색인에 관련된 공간에 대한 분석을 요약하면 <표 1>, <표 2>와 같다.

<표 1> 색인파일 공간의 비교

<Table 1> Compare of index file space

색인 유형	색인 엘리먼트 정보 (*key_element*)	색인파일 공간(S_{index_f})
A	(DEN, IEN)	$(S_{key\_element} + S_{ptr}) * \sum_{i=1}^{h} k^{i-1}$
B	(DEN, IEN)	$(S_{key\_element} + S_{ptr}) * k^{h-1}$
C	(Element type , ID)	$(S_{key\_element} + S_{ptr}) * k'^{h-1}$ k' : ID할당시에 정한 차수

A : 엘리먼트 위치별로 문서에 존재하는 모든 엘리먼트들을 색인하는 색인구조

B : 엘리먼트 위치별로 문서에 존재하는 단말 엘리먼트만을 색인하는 색인 구조

C : 레벨 우선 순회규칙으로 엘리먼트의 ID를 할당하여 GDIT를 구성한 후 색인 유형B 방법으로 색인하는 색인 구조

<표 1>은 앞의 세 가지 구조 색인 방법에 대해 색인 파일 공간에 대해 비교한 것이다. B와 C색인 구조는 GDIT의 단말 엘리먼트가 색인 파일의 구성 요소가 될 수 있으므로 색인 파일의 색인어 개수는 GDIT에 존재하는 단말 엘리먼트 개수 k^{h-1}로 동일하다. A색인 구조는 문서에 존재하는 모든 엘리먼트들을 색인하므로 색인파일의 색인어 개수는 GDIT에 존재하는 엘리먼트 개수 $\sum_{i=1}^{h} k^{i-1}$이다. 그러므로 B, C색인 구조는 A색인 구조보다 색인파일의 공간이 작게 필요하다. C색인 구조는 노드 추가 등으로 차수 k를 큰 수로 정해야 하므로, B색인구조보다 색인 파일 공간이 크다.

<표 2> 포스팅 파일 공간의 비교

<Table 2> Compare of posting file space

색인유형	포스팅 파일 공간($S_{posting\_f}$)
A	S_{DIN} * 식(3)
B	S_{DIN} * 식(2)
C	S_{DIN} * 식(2)

(A, B, C : 표1과 동일)

<표 2>는 앞의 세 가지 구조 색인 방법에 대해 포스팅 파일 공간에 대해 비교한 것이다. 색인 방법들은 포스팅 파일에 문서 번호 DIN으로 색인함으로 포스팅 파일 공간 식에서 DIN의 크기 S_{DIN}을 곱한다. A색인 구조는 문서의 모든 엘리먼트(식(3))를 색인한다. B와 C색인 구조는 문서에 존재하는 단말 엘리먼트들(식(2))을 색인한다. 그러므로, B와 C색인 구조는 A색인 구조보다 포스팅 파일 공간이 작게 필요하다. C색인 구조는 엘리먼트 추가 등의 갱신 때문에 레벨 우선 순회 규칙에서 정한 차수 k'은 큰 수로 정한다. 그러나 문서에서 자식 노드 수가 k'인 엘리먼트는 존재하지 않기 때문에 B색인 구조와 같은 크기의 포스팅 공간이 필요하다.

8. 결론

구조적 문서는 문서로의 다양한 접근 경로를 제공하므로, 문서의 구조를 사용한 구조 검색 질의는 검색의 신뢰도를 높일 수 있다. 문서의 구조에 기반을 둔 검색기를 개발하기 위해서는 문서의 구조 정보를 추가로 색인해야 하므로, 이에 따른 메모리 오버헤드 및 검색 비용이 증가된다.

본 논문에서는 순수 구조에 관련된 질의들을 효율적으로 처리할 수 있는 GDIT 기반의 구조 색인 구조를 제시하였다. 제시한 구조 색인 구조는 문서에 존재하는 엘리먼트들 중에서 단말 엘리먼트들만을 색인한다. 단말 엘리먼트들의 (DEN, IEN)에서 조상 엘리먼트들의 (DEN, IEN)들을 알 수 있으므로, 임의의 엘리먼트가 있는 문서들은 임의의 엘리먼트의 하위 문서 구조에 존재하는 단말 엘리먼트들이

가지는 색인 리스트의 합이다. 그리고 이를 기반으로 순수 구조에 대한 질의 처리 과정을 제시하고 색인 비용에 대하여 그 성능을 분석하였다. 분석시에 레벨 우선 순회 규칙으로 ID를 할당하는 방법에 본 논문에서의 GDIT 기법을 적용할 경우의 색인 공간과 검색 시간에 대하여도 성능을 분석하였다. 분석 결과 색인 공간에서 특정 엘리먼트의 하위 문서 구조에 존재하는 단말 엘리먼트들만을 색인하는 방법이 가장 우월하였다.

※ 참고문헌

[1] 이희주, 장재우, 심부성, 주종철, "구조화 문서를 위한 정보 검색 인덱스의 설계," 정보과학회 가을학술발표논문집 Vol. 24, No. 2, 1997.

[2] 배종민, 김영자 "GDIT를 기반으로 한 구조적 문서의 효율적 검색과 갱신을 위한 인덱스 설계", 정보처리학회 논문지, 제 7권 제 2호, 2000.

[3] Y. K. Lee, S. J. Yoo, K. Yoon, and P. B. Berra, "Index Structure for Structured Documents," in Proc. Digital Libraries, 1996.

[4] I. A. Macleod, "Storage and retrieval of structured documents. Information Processing and Management," 26(2), 1990.

[5] V. Christophides, S. Abiteboul, S. Cluet and M. Scholl, "From Structured Documents to Novel Query Facilities," ACM SIGMOD, 1994.

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

[7] G. E. Blake, M. P. Consens, P. Kilpelainen, P. Larson, T. Snider, and F. Tompa, "Text/ Relational database management systems: Harmonising SQL and SGML," In Proc. Int. Conf. on Applications of Databases, no. 819 in Lecture Notes in Computer Science, pages 267-280, 1994.

[8] S. H. Myaeng, D.-H. Jang, M.-S. Kim, Z.-C. Zhoo, "A Flexible Model for Retrieval of Structured Documents," In Proc. of ACM SIGIR 1998.

[9] B. Lowe, J. Zobel and R. Sacks-Davis, "A Formal Model for Databases of Structured Text," In Proc. DASFAA, 1995.

[10] 김영자, 배종민, "구조적 문서의 효율적인 색인 구조에 대한 분석" 정보과학회 가을학술발표논문집 Vol. 24, No. 2, 2000.

김 현 주

1988년 경상대학교 전산통계학과(이학사)
1990년 숭실대학교 전자계산학과(공학석사)
2000년 경상대학교 전자계산학과(공학박사)
1994년 ~ 1997년 제일정밀공업(주) 연구원
20000년 ~ 현재 경남정보대학 컴퓨터정보계열 전임강사
관심분야 : 정보검색, 디지털 도서관, 웹 프로그래밍