Journal of Korea Multimedia Society Vol. 7, No. 5, May 2004(pp. 601-616)

구조 기반 검색을 위한 색인 구조에 대한 분석

김영자[†], 김현주[††], 배종민[†††]

요 약

구조적 문서들에 대한 검색 시스템은 구조 기반 검색 질의를 지원하여 다양한 수준의 검색 기능을 제공한다. 완전한 구조 기반 질의를 처리하기 위해서는 구조적 문서가 가지는 엘리먼트 간의 구조적 포함관계나 순서에 관한 정보를 유지되어야 한다. 본 논문에서는 엘리먼트와 엘리먼트 사이의 구조적 상관관계와 엘리먼트의 발생순서에 관련된 질의 등 여러 유형의 순수 구조 질의를 처리할 수 있는 네 가지 색인구조를 제시하고 그 성능을 평가한다. 제안된 색인 알고리즘은 전체 문서 인스턴스 트리 개념에 바탕을 두고 있다.

Analysis of Indexing Schemes for Structure-Based Retrieval

Young-Ja Kim[†], Hyun-Ju Kim[††], Jong-Min Bae[†††]

ABSTRACT

Information retrieval systems for structured documents provide multiple levels of retrieval capability by supporting structure-based queries. In order to process structure-based queries for structured documents, information for structural nesting relationship between elements and for element sequence must be maintained. This paper presents four index structures that can process various query types about structures such as structural relationships between elements or element occurrence order. The proposed algorithms are based on the concept of Global Document Instance Tree.

Key words: Information retrieval(정보검색), Structured Document(구조 문서), SGML, XML, GDIT (Global Document Instance Tree)

1. 서 론

SGML이나 XML은 문서가 가지는 계층적 구조정보를 명시적으로 표현한다. 이러한 구조적 문서가 가지는 구조정보는 문서로의 다양한 접근경로를 제공하므로, 문서의 구조를 활용한 검색 시스템은 사용자에게 보다 정밀한 검색 결과를 제공할 수 있다. 본 논문에서는 구조적 문서가 가지는 계층적 구조정보를 활용한 질의를 처리할 수 있는 검색 기능을 가지는 것에 주안점을 둔다.

검색 시스템이 구조적 문서가 가지는 구조정보를 활용한 질의를 처리할 수 있는 검색 기능을 가지기 위해서는 색인 과정에서부터 기존 검색방법과는 다르게 진행되어야 한다. 색인이 엘리먼트 수준에서 수행되는 것 이외에 문서의 구조가 저장되어야 엘리먼트간의 관련성에 대한 정보를 추출할 수 있다. 결국 색인에 필요한 공간이 커진다. 그러므로 질의처리 범위는 넓히면서 색인공간 오버헤드는 줄여야 한다. 색인공간을 줄이기 위해서는 모든 엘리먼트들을 색인하지 않고 문서구조의 일부분의 엘리먼트만을 색인하게 된다. 이때 색인된 엘리먼트에서 색인되지 않은 엘리먼트에 관한 정보를 추출할 수 있어야 한다. 기

※ 교신저자(Corresponding Author) : 김현주, 주소 : 경상남도 진주시 칠암동(660-758), 전화 : 055)751-3328, FAX : 055)751-3329, E-mail : khj@jinju.ac.kr
접수일 : 2003년 5월 30일, 완료일 : 2003년 7월 11일
[†] 동주여자상업고등학교 교사
(E-mail : cybercpa@chollian.net)
[††] 정회원, 진주산업대학교 컴퓨터공학부 조교수
[†††] 정회원, 경상대학교 컴퓨터과학과 교수
(E-mail : jmbae@nongae.gsnu.ac.kr)
※ 이 논문은 2003년도 기성회 연구비 지원에 의하여 연구되었음.

존의 구조기반 검색 알고리즘은 내용색인에 대한 색인 오버헤드를 줄이는 방법을 제시했지만, 제시된 모든 방법들은 순수 구조 기반 질의를 효과적으로 처리할 수 없다.

본 논문에서는 순수 구조기반 질의를 처리할 때에도 검색의 효율성을 유지하면서 색인공간 오버헤드를 줄일 수 있는 새로운 색인 구조들을 제시한다. 구조정보를 활용한 질의들은 엘리먼트간의 관련성에 대한 정보를 포함하는 질의일수도 있고 엘리먼트의 순서에 대한 정보를 포함하는 질의일 수도 있다. 본 논문에서는 내용 색인 구조[2]에 덧붙여서 순수 구조에 관련된 질의들을 처리할 수 있는 4가지 구조 색인 구조를 제시한다. 제시하는 구조 색인 구조들은 전체 문서 인스턴스 트리의 개념에 바탕을 두고 있다. GDIT(Global Document Instance Tree)는 문서 인스턴스의 구조를 트리로 표현했을 때, 모든 문서 인스턴스 트리의 합집합이다[5]. 본 논문에서는 GDIT 기반의 색인기법을 사용해서, 순수 구조 질의를 처리하기 위한 문서 구조의 네 가지 색인 기법을 제시하고 각각에 대한 성능을 분석한다. 제시된 네 가지 색인기법은, 첫째, 엘리먼트 타입별로 문서에 존재하는 모든 엘리먼트들을 색인하는 색인구조, 둘째, 엘리먼트 위치별로 문서에 존재하는 모든 엘리먼트들을 색인하는 색인구조 셋째, 엘리먼트 위치별로 문서에 존재하는 모든 단말 엘리먼트들만을 색인하는 색인구조 넷째, 엘리먼트 위치별로 문서에서 임의의 엘리먼트의 하위 문서 구조가 GDIT에 있는 구조와 동일할 경우에는 동일한 하위 문서 구조를 가지는 가장 상위 엘리먼트만을 색인하는 구조이다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구에 대하여 논하고 3장에서는 질의어의 유형을 분석하며 4장에서는 GDIT의 구성 방법과 그 의미에 대해서 논한다. 5장에서는 GDIT를 기반으로 하여 문서의 구조에 대한 색인구조들과 구조에 관련된 질의를 처리하기 위해 필요한 보조 자료구조들을 제안한다. 6장에서는 제안된 색인방법들을 색인 공간과 검색 시간을 기준으로 평가하고 7장에서는 결론을 보인다.

2. 관련 연구

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

기존의 DB 모델을 사용하여 구조적 문서를 각 모델에 맞게 변환하여 표현하는 방법이다[4,5,7-9]. 이 방법은 구조적 문서의 각 엘리먼트 타입을 관계형 데이터베이스의 테이블이나 객체지향 데이터베이스의 클래스 등으로 SGML DTD를 각 모델의 스키마로 매핑하는 과정을 수행한다. 그런데 이 과정에서 DTD가 가지고 있는 모든 정보를 표현하기가 어려우며, DTD가 가지고 있는 제약 조건 등을 무시하거나, 혹은 표현하지 못하는 경우가 있으며, DTD가 가지고 있는 모든 정보를 표현하더라도 힘든 변환과정을 수행하거나 구조적 문서를 처리하기 위해서는 시스템의 확장이 불가피하다. 또한 이로 인하여, 질의 처리 시에 단순 질의에도 많은 연산과정을 수행하게 된다.

다음으로, 엘리먼트 기반 모델을 사용하는 방법은 DTD를 변환하는 과정을 수행하지 않고 직접 엘리먼트 단위로 저장하는 방법이다[1,3,6,10-12]. 이에 대한 연구로는 첫째, 색인어가 나오는 엘리먼트에서 루트까지의 경로에 존재하는 모든 엘리먼트들을 색인하는 방법이다[1,12]. 이 방법은 추출된 색인어가 발생된 엘리먼트의 모든 상위 엘리먼트에 대해서도 색인을 하게 되어 공간상의 중복이 발생하여 색인에 필요한 메모리 오버헤드가 크다는 문제점이 있다. 둘째, 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 피하므로서, 색인에 필요한 메모리 오버헤드를 줄일 수 있으나, 내용색인만 고려함으로서, 구조에 관련된 질의를 처리하는데 제약이 있다. 셋째, 문서에 포함되어 있는 색인어와 엘리먼트들의 발생 위치를 사용하여, 내용색인 외에 각 문서의 구조들을 색인하는 방법이 있다[10]. 이 방법은 문서 구조 색인으로 순수 구조 질의들은 처리할 수 있으나 내용 색인 시에 색인어마다 위치정보만을 저장함으로서 내용과 구조의 혼합질의를 효율적으로 처리할 수 없으며, 각 문서마다 구조 정보를 저장함으로 메모리 오버헤드가 있다. 다음으로 내용 색인 시에 색인어마다 루트까지의 경로에 존재하는 엘리

먼트들의 정보를 나타내는, 경로 정보를 저장하는 방법이다[11]. 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 하여 문서의 구조에 대한 네 가지 색인 방법을 제시하고, 구조적 상관관계나 순서에 관련된 질의를 처리하기 위해 필요한 자료 구조를 제시한 다음, 이 색인방법들을 색인에 필요한 공간과 검색시간을 기준으로 평가한다.

3. 질의 유형

구조적 문서를 검색하는데 필요한 대표적인 질의 유형을 분류하면 크게 다음과 같은 네 가지로 분류할 수 있다.

(1) 내용 검색 질의

① 전문 검색

질의 1 : '환경보호'나 '리사이클링'이라는 단어가 나오는 잡지를 찾으시오.

② 문서의 부분 검색

질의 2 : '정보'라는 단어가 있는 잡지를 찾으시오.

(2) 구조 검색 질의

문서의 논리적인 구조에 대한 질의로서 시작 엘리먼트와 주어진 관계에 있는 문서나 엘리먼트를 검색하는 질의등 문서의 구조정보에 내재되어 있는 의미정보를 사용한 질의이다.

① 문서 구조만으로 구성된 질의

질의 3 : 문화면에서 인물란에 게재된 기사의 제목을 찾으시오.

② 순서가 포함된 질의

질의 4 : 경제면에서 첫 번째 기사의 두 번째 소제목을 찾으시오.

(3) 내용과 구조의 혼합질의

질의 1+ 3 : 문화면의 인물란에 게재된 '박인경'에 관한 기사를 찾으시오.

질의 1+4 : 경제면에서 '전자화폐'에 관한 첫 번째 기사의 두 번째 도표그림을 찾으시오.

질의 3에서 문화, 인물, 제목등은 모두 엘리먼트이다. 따라서 이 질의는 문서에 속한 단어를 검색하는 것이 아니라 문서의 구조만으로 구성된 질의어이다. 아울러 질의 1+3에서 문화, 인물은 문서 구조를 나타내는 엘리먼트이고, '박인경'은 문서에 포함된 색인어이다. 본 논문에서는 질의 3, 질의 4를 처리하기 위한 색인 구조에 대하여 논한다.

(4) 서로 다른 DTD에 대한 질의

질의) 제목에 '색인어'라는 단어가 있는 잡지나 문서를 찾아라.

본문에서 제시된 색인 구조는 다수의 DTD를 지원하기 때문에 질의 유형 (4)는 저절로 지원된다. 따라서 질의 유형 (4)에 대해서는 더 이상 논하지 않는다.

4. GDIT 개요

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

4.1 GDIT 구성

GDIT는 문서 인스턴스의 구조를 트리로 표현했을 때 모든 문서 인스턴스 트리의 합집합을 말한다. GDIT를 구성하기 위해서 DTD 트리가 필요하다. GDIT를 구성하기 위해서는 문서 인스턴스 트리를 합치고, 합쳐진 각 노드의 ID로서 사용될 (DEN, IEN)값을 결정하는 두 가지 작업이 필요하다. 여기서 IEN(Instance Element Number)이란 문서 인스턴스에서 해당 엘리먼트가 발생된 순서를 의미한다. 임의의 첫 번째 문서 인스턴스의 각 엘리먼트에 대하여는 DTD트리로부터 DEN값을 알 수 있고, DTD노드에 부여된 순서쌍의 두 번째 값 즉, 엘리먼트가 실제로 발생된 횟수를 1증가시키고 그 값을 IEN으로 부여한다. 결국, 임의의 첫 번째 문서 인스턴스 구조와 동일한 구조의 GDIT가 생성된다. 다음으로, 임의의 두 번째 문서 인스턴스 구조를 GDIT와 비교하여, 구조상 중복되는 엘리먼트들은 하나로 인식하고, 두 번째 문서 인스턴스에는 있으나 GDIT에 없는 엘리먼트들은 GDIT에 그 엘리먼트를 추가하고, 위와 같은 방식으로 (DEN, IEN)을 부여한다.

4.2 엘리먼트 식별자(EID:Element IDentifier)

DTD트리의 구조는 문서에 무관하게 변하지 않는 구조이기 때문에, 각 엘리먼트에 할당된 DEN에 그

엘리먼트의 트리상의 조상의 경로정보라는 의미를 부여한다. DEN에는 발생지시자에 대한 정보가 포함되어 있지 않다. 따라서 실제 문서에서, 예를 들어 다수의 chapter가 있고, 각 chapter에서는 다수의 section이 있을 때, 특정 section이 속한 chapter를 DEN만으로는 결정할 수 없다. 이 문제를 해결하는 요소가 IEN이다. 결국 IEN에 문서상의 조상의 경로에 대한 의미를 부여한다.

GDIT는 모든 문서 인스턴스에서 발생된 엘리먼트의 갯수가 모두 반영되므로, GDIT의 각 엘리먼트에 부여된 값의 쌍 (DEN, IEN)은 문서내에서 엘리먼트의 위치를 나타내게 되어, 문서에 존재하는 엘리먼트를 식별하는데 사용할 수 있다. 엘리먼트 식별자의 구성요소는 (DEN, IEN)쌍에 문서 인스턴스 번호(DIN:Document Instance Number)를 추가하여 <DIN, (DEN, IEN)>로 표현한다.

4.3 내용 색인 구조

제안된 GDIT기반의 검색을 위한 내용 색인 구조는 그림 1, 그림 2와 같다. 색인 화일의 색인어에 대해 포스팅 파일의 구성요소는 색인어를 포함하고 있는 EID테이블에 대한 포인터, 색인어 빈도수로 구성된다. EID테이블은 색인어를 포함하고 있는 EID를 저장하고 있는 테이블이다. 이때 문서 엘리먼트 중에서, 색인어를 포함하고 있는 텍스트 레벨 엘리먼트만 색인함으로서 메모리 오버헤드를 줄인다. 또한 포스팅 파일에서의 EID에 대한 중복 저장을 줄이고, 문서구조의 수정이 발생할 때 한 텍스트 레벨 엘리먼트내에서 수정될 색인어의 개수와 무관하게 갱신을 처리하기 위하여 EID테이블을 따로 둔다.

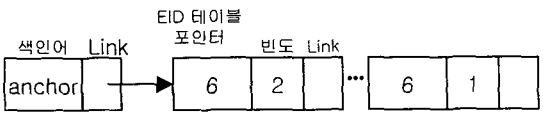


그림 1. An Indexing structure

번 호	EID <DIN,(DEN,IEN)>
:	:
6	<1, (6, 1)>
:	:

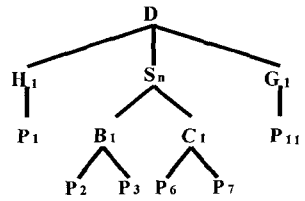
그림 2. EID table

5. 문서 구조에 대한 색인 구조

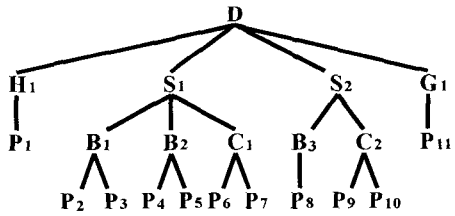
4.3에서 제시한 내용 색인 구조는 문서에 존재하는 엘리먼트와 엘리먼트 사이의 관계를 포함하는 질의를 처리할 수 없다. 여기서는 이와 같은 구조정보를 사용한 질의를 처리하기 위한 네 가지 색인방법들을 제안하고, 각 색인 방법들이 여러 유형의 구조 질의를 처리하기 위해서 필요한 자료구조를 제시한다.

5.1 색인 유형

그림 3과 같은 임의의 문서 인스턴스 구조와 GDIT가 있다고 가정하자.



(a) 임의의 문서 인스턴스 구조.



(b) GDIT

그림 3. A document instance structure and GDIT

(1) 엘리먼트 타입별로 문서에 존재하는 모든 엘리먼트들을 색인하는 색인 구조

엘리먼트 타입별로 문서에 존재하는 모든 엘리먼트들을 색인하는 방법이다. 엘리먼트 타입별로, 역리스트에 해당 엘리먼트 타입이 존재하는 모든 문서의 관련 엘리먼트들의 EID를 가진다. 임의의 DTD에 대한 GDIT가 그림 3의 (b)라고 가정하고 그림 3의 (a)의 문서구조를 색인할 경우, 각 엘리먼트 타입별로 색인파일을 구성할 엘리먼트들은 다음과 같다.

index(H)={H}, index(S)={S}, index(B)={B}, index(C)={C}, index(P)={P}, index(G)={G},

그림 3의 (a)의 문서에 대한 역리스트는 다음과

같다.

invert-list(H)={H_1}, invert-list(S)={S_1}, invert-list(B)={B_1}, invert-list(C)={C_1}, invert-list(G)={G_1}, invert-list(P)={P_1, P_2, P_3, P_6, P_7, P_{11}}

색인구조는 그림 4과 같다.

색인 파일

Indexed element	Link

포스팅 파일

EID <DIN, (DEN, IEN)>	Link

그림 4. Index structure

이 색인방법은 질의에서 원하는 엘리먼트를 찾기 위해, 포스팅 파일에서 엘리먼트들의 EID 구성 항목인 (DEN, IEN)을 매번 비교해야 하며, 엘리먼트들간의 부모 자식관계나 형제 관계 등의 엘리먼트간의 관련성을 고려하지 않고 모든 엘리먼트들을 색인하므로, 역리스트에 많은 중복이 발생한다.

(2) 엘리먼트 위치별로 문서에 존재하는 모든 엘리먼트들을 색인하는 색인 구조

엘리먼트 위치별로 문서에 존재하는 모든 엘리먼트들을 색인하는 방법이다. 엘리먼트 위치별로, 역리스트에 해당 엘리먼트가 존재하는 모든 문서들의 문서번호 DIN을 가진다. 임의의 DTD에 대한 GDIT가 그림 3의 (b)라고 가정하고, 엘리먼트 위치별로 색인할 경우, 각 엘리먼트 위치별로 색인파일을 구성할 엘리먼트들은 다음과 같다.

index(H)={H_1}, index(S)={S_1, S_2}, index(B)={B_1}, index(C)={C_1, C_2}, index(G)={G_1}

index(P)={P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}}

색인파일을 구성하는 엘리먼트들 중에서 그림 3의 (a)의 문서에 대하여 역리스트에 문서번호 DIN이 나타날 엘리먼트들은 다음과 같다

index-element(H)={H_1}, index-element(S)={S_1}, index-element(B)={B_1}, index-element(C)={C_1}, index-element(G)={G_1}, index-element(P)={P_1, P_2, P_3, P_6, P_7, P_{11}}

색인구조는 그림 5와 같다.

색인 파일

(DEN, IEN)	Link

포스팅 파일

DIN리스트	Link

그림 5. Index structure

(1)에서 제안한 색인구조와 같이 역리스트에 엘리먼트의 EID를 색인하지 않고, 문서번호 DIN을 색인하므로 메모리 오버헤드가 줄어들며, 문서구조 트리의 어떤 레벨의 엘리먼트에도 접근할 수 있다. 그러나 (1)의 색인구조와 같이 문서의 모든 엘리먼트들을 색인하므로 역 리스트에 많은 중복이 발생한다.

(3) 엘리먼트 위치별로 문서에 존재하는 모든 단말 엘리먼트들만을 색인하는 색인 구조

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 색인리스트의 합이다.

INDEX[leaf $node_1$] ∪ INDEX[leaf $node_2$] ∪...∪ INDEX[leaf $node_n$]

임의의 DTD에 대한 GDIT가 그림 3의 (b)라고 가정하고, 엘리먼트 위치별로 색인할 경우, 각 엘리먼트 타입별로 색인파일을 구성할 엘리먼트들은 다음과 같다.

index(H)={}, index(S)={}, index(B)={}, index(C)={}, index(G)={}

index(P)={P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}}

색인파일을 구성하는 엘리먼트들 중에서 그림 3의 (a)의 문서에 대하여 역리스트에 문서번호 DIN이 나타날 엘리먼트들은 다음과 같다.

index-element(P)={P_1, P_2, P_3, P_6, P_7, P_{11}}

색인구조는 그림 5의 색인구조와 동일하다.

(2)에서 제안한 색인구조와 같이 문서의 모든 엘리먼트를 색인하지 않고 텍스트 레벨 엘리먼트만을 색인하므로 메모리 오버헤드가 줄어든다. 이 방법은 검색 엘리먼트를 색인된 텍스트 레벨 엘리먼트에서 찾을 때에 검색 엘리먼트의 하위 문서구조에 존재하

는 텍스트 레벨 엘리먼트가 하나 이상이다. 그러므로 각 텍스트 레벨 엘리먼트의 색인 리스트에 있는 문서번호 중 중복되는 문서번호를 조사해야 한다. 결국 질의에서 찾고자 하는 엘리먼트의 텍스트 레벨 엘리먼트 개수가 많을수록 검색시간이 길어지게 된다.

(4) 문서구조에서 특정 엘리먼트의 하위 문서구조가 GDIT에 있는 구조와 동일할 경우에는 동일한 하위 문서구조를 가지는 가장 상위 엘리먼트만을 색인하는 색인구조

문서에 존재하는 엘리먼트들의 하위 문서구조를 GDIT구조와 비교하여, 엘리먼트의 하위 문서구조가 GDIT의 해당 하위 문서구조와 동일할 경우에는, 동일한 하위 문서구조를 가지는 가장 상위 엘리먼트만을 색인하고 그 하위 엘리먼트들은 색인하지 않는다. 결국 임의의 엘리먼트가 있는 문서를 찾을 경우에는 임의의 엘리먼트의 색인리스트와 임의의 엘리먼트의 조상 엘리먼트들의 색인리스트 그리고 임의의 엘리먼트의 하위 엘리먼트들의 색인리스트의 합이다.

INDEX[node] ∪ INDEX[ancestors] ∪ INDEX[descendents]

임의의 DTD에 대한 GDIT가 그림 3의 (b)라고 가정하고, 엘리먼트 위치별로 색인할 경우, 각 엘리먼트 타입별로 색인파일을 구성할 엘리먼트들은 (2)의 색인방법에서의 색인파일과 같이 GDIT에 존재하는 모든 엘리먼트들이다.

index(H)={H_1}, index(S)={S_1, S_2}, index(B)={B_1}, index(C)={C_1, C_2}, index(G)={G_1}

index(P)={P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}}

그림 3의 (a)의 문서에 대하여 역리스트에 문서번호 DIN이 나타날 엘리먼트들은 다음과 같다.

index-element(H)={H_1}, index-element(B)={B_1}, index-element(C)={C_1}, index-element(G)={G_1}

색인구조는 그림 5의 색인구조와 동일하다.

본 논문에서의 내용 색인 구조와 같이 색인어가 존재하는 텍스트 레벨 엘리먼트에 대해서만 색인하는 또 다른 방법은 문서를 차수가 k'인 완전트리라고 가정하고 레벨 우선 순회 규칙으로 엘리먼트의 ID를 할당하는 방법이다(BUS[6-8]). 이 방법은 엘리먼트 EID를 <문서번호, 엘리먼트 타입, 엘리먼트 레벨, ID>로 구성한다. 이 방법은 자식 엘리먼트의 ID에서 부모 엘리먼트의 ID를 공식으로 구할 수 있으므로 텍스트 레벨 엘리먼트에 대해서만 색인한다. 이 방법의 문제점은 구조 색인 구조 구성시에 문서의 구조에 대한 정보를 효율적으로 관리할 수 있는 방법이 없다는 것이다. 그러므로 본 논문에서는 다음의 규칙을 적용한 다음 레벨 우선 순회 규칙을 사용하여 ID를 할당하는 방법과 비교하면서 분석한다.

• ID할당시에 사용한 차수 k'의 완전트리를 구성한 다음 EID구성 항목중 엘리먼트 타입정보를 DEN처럼 루트에서 해당 엘리먼트까지의 조상의 경로 정보를 나타내고, ID를 IEN처럼 문서상의 조상의 경로 정보로 사용한다. 결국 GDIT는 차수가 k'인 완전트리라고 생각할 수 있다.

그러므로 위에서 제안한 색인 방법들을 적용할 경우, (1)부터 (3)까지의 색인 방법에서는 본 논문에서 제안한 색인 결과와 같다. 레벨 우선 순회규칙으로 ID를 할당하는 경우 문서에 엘리먼트 추가등의 갱신이 발생할 수 있으므로, 차수 k'는 최대한 큰 수로 정해야 한다. 따라서 문서의 엘리먼트 중에 하위 문서 구조에 존재하는 엘리먼트들의 차수가 k'인 경우는 존재하지 않는다. 그러므로 (4)의 색인 방법을 사용하여 색인한 결과는 (2)의 색인 방법을 사용한 결과와 같다.

5.2 자료구조

5.1에서 제시한 각 색인 구조들이 엘리먼트 간의 관련성에 관한 질의, 순서가 추가된 질의 등 여러 유형의 구조 질의를 처리하기 위해 다음과 같은 자료구조들이 필요하다.

5.2.1 DTD트리에서의 각 엘리먼트에 대한 정보

엘리먼트별로 조상의 경로, 조상의 DEN리스트, 그러한 조상 엘리먼트를 가지는 엘리먼트의 DEN과 레벨의 엘리먼트에 관한 정보가 들어가게 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
- GDIT기반으로 ID를 할당할 경우 : <DEN, level>
- 레벨우선순회규칙으로 ID를 할당할 경우 : <level, element type number>

그림 6. DTD table

5.2.2 GDIT에서의 각 엘리먼트에 대한 정보

DTD의 각 엘리먼트별로, 실제 문서에서 엘리먼트들의 조상의 순서에 따라, 조상 엘리먼트의 IEN정보, 엘리먼트의 ID정보, 최하위 엘리먼트들의 ID정보등, 각 색인 구조들에 따라 필요한 정보가 들어가게 된다.

그림 7의 (a)는 GDIT에 있는 엘리먼트의 DEN별로, 문서에서 루트에서부터 엘리먼트까지의 경로에 나타나는 엘리먼트의 순서에 따라, 각 색인 구조에서 필요한 엘리먼트들의 (DEN, IEN)리스트를 나타낸 것이다. 예를 들어, 문서에서 모든 단말 엘리먼트들만을 색인하는 구조일 경우에는 (DEN, IEN)리스트에는 모든 단말 엘리먼트들의 (DEN, IEN)리스트가 들어가게 된다. (b)는 레벨우선순회규칙으로 ID를 할당할 경우에 엘리먼트의 타입별로, 루트에서부터 엘리먼트까지의 경로에 나타나는 엘리먼트의 순서에 따라, 각 색인 구조에서 필요한 엘리먼트들의 (타입정보, ID)리스트가 들어가게 된다.

(a) GDIT 기반으로 ID를 할당한 경우

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

- DEN : 엘리먼트의 DEN
- Ancestor's Order : 엘리먼트까지의 조상의 순서
- Ancestor's IEN order : 조상 엘리먼트들의 IEN 리스트
- (DEN, IEN) List : 관련 엘리먼트들의 (DEN, IEN) 리스트

(b) 레벨우선순회규칙으로 ID를 할당할 경우

E_type_num	Ancestor's Order	Ancestor's ID List	(Element type number, ID List)

- E_type_num : 엘리먼트 type 번호
- Ancestor's Order, Ancestor's ID List는 (a)의 항목의미와 동일
- Element type number, ID List : 관련 엘리먼트들의 (타입 정보, ID) 리스트

그림 7. GDIT table

6. 평 가

본 장에서는 5장에서 제시한 구조 색인 구조들의 성능을 분석한다.

6.1 기본 가정

다음은 본 논문에서 제시한 구조 색인 구조들의 색인에 필요한 기억공간을 분석하기 위하여 사용되는 기호의 일부이다.

h : GDIT의 높이

k : GDIT의 차수

k' : 레벨 우선 순회규칙으로 ID를 할당할 때 문서의 차수

d : DTD트리의 루트부터 단말 엘리먼트까지를 하나의 경로로 가정할 때 DTD트리가 가지는 경로 갯수

m : DTD트리의 각 단말 엘리먼트마다 GDIT에서 발생하는 노드 갯수

n_{doc} : 전체 문서 개수

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

$key\_element$: 색인어로 사용될 엘리먼트 정보

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

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

S_{EID} : EID(Element IDentifier)의 크기

S_{ptr} : 포인터 크기

S_{id} : 색인 정보의 평균적인 포스팅 크기

색인에 필요한 기억공간을 분석하기 위하여 다음과 같은 가정을 한다.

가정 1 : DTD내에 DEN이 같은 엘리먼트는 존재하지 않는다.

가정 2 : GDIT는 차수가 k인 완전트리이다.

가정 3 : 임의의 문서의 레벨 i의 임의의 노드의 차수가 m'일 확률이 $\frac{1}{k}$ ($1 \le m' \le k$)이다.

가정 4 : 전체 문서 개수(n_{doc})는 GDIT에서 발생할 수 있는 모든 가능한 문서 경우들의 합이다.

가정 5 : 문서에서 임의의 엘리먼트가 나타날 경우 DTD트리에서 임의의 엘리먼트보다 앞에 위치하는 엘리먼트들이 모두 나타난다. 즉 DTD트리의 루트에서 단말 엘리먼트까지의 각 경로에 대해 문서에서 임의의 경로가 존재할 경우에는 DTD트리에서 앞에 위치하는 경로들도 발생한다.

GDIT의 레벨 h에는 k^{h-1}개의 노드가 존재하는데, 만약 k와 d가 같다면, 레벨 h에는 k^{h-2}개의 노드들이 d개 존재하며, 각 k^{h-2}개의 노드들은 서로 같은 조상을 가지는 같은 종류의 엘리먼트들이다.

k가 d의 2배라면, 레벨 h에는 $k^{h-2}*2$개의 노드들이 d개 존재하며, 각 $k^{h-2}*2$개의 노드들은 서로 같은 조상을 가지는 같은 종류의 엘리먼트들이다. 결국, 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 다음과 같다.

$$m = k^{h-2} * \frac{k}{d}$$

그러므로, 발생할 수 있는 전체 문서 개수 n_{doc}는 다음과 같다.

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

이와 같은 n_{doc}개의 전체 문서에 존재하는 노드 개수의 합을 구하면 다음과 같다. GDIT의 레벨 h에는 m개의 노드들이 d개 존재하는데, 레벨 h 에 m개의 노드가 1개 존재할 경우, 발생할 수 있는 노드 개수는 1부터 m까지를 모두 더한 $\sum_{i=1}^{m} i$이다. 레벨 h 에 m개의 노드가 2개 존재할 경우, 발생할 수 있는 노드 개수는 $\sum_{j=1}^{m}(j*m + \sum_{i=1}^{m} i)$이다. 레벨 h 에 m개의 노드가 3개 존재할 경우, 발생할 수 있는 노드 개수는 $\sum_{l=1}^{m}(l*m^{2} + \sum_{j=1}^{m}(j*m + \sum_{i=1}^{m} i))$이다. 레벨 h에 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 각 문서 인스턴스의 레벨 h에 나타날 수 있는 노드의 전체 개수 $n_{leaf_element}$가 된다.

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

레벨 $h-1$에는 $\frac{m}{k}$개의 노드가 d개 존재하므로, n_{doc}개의 전체 문서의 레벨 h-1에 나타날 수 있는 노드의 전체 개수는 다음과 같다.

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

n_{doc}개의 전체 문서들의 레벨 1부터 레벨 h까지 나타날 수 있는 노드 개수들의 전체 합 $n_{element}$는 다음과 같다.

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

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

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

6.2 색인 비용

색인에 필요한 기억공간에 대해서 분석한다. 색인된 결과가 색인 파일과 포스팅 파일을 사용하여 역리스트로 저장될 때, 색인에 필요한 공간 S_{index}은 색인 파일에 필요한 공간 $S_{index-f}$ 와 포스팅 파일에 필요한 공간 $S_{posting-f}$ 의 합이다.

$S_{index} = S_{index-f} + S_{posting-f}$

색인파일에 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.

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

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

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

(1) 엘리먼트 타입별로 모든 엘리먼트들을 색인하는 색인 구조

색인어로 사용될 엘리먼트 정보는 엘리먼트 타입이므로, $S_{key_element}$는 엘리먼트의 평균 크기, $n_{key_element}$는 DTD트리에 존재하는 엘리먼트의 총 개수인 $d*(h-1)+1$이다. 그러므로, 색인 파일에 필요한 공간 $S_{index-f}$는 엘리먼트의 평균크기 $S_{key_element}$와 포인터 크기 S_{ptr}을 더한 다음, DTD 트리에 존재하는 엘리먼트의 개수 $d*(h-1)+1$을 곱한 것이다.

$S_{index-f} = (S_{key\_element} + S_{ptr}) * (d*(h-1)+1)$

($S_{key\_element}$: 엘리먼트의 평균크기)

포스팅 파일에 저장되는 색인정보는 엘리먼트의 EID이므로, 색인정보의 평균 크기 S_{id}는 엘리먼트의 EID크기 S_{EID}이고, 모든 엘리먼트들을 색인하므로 색인정보의 전체 개수 n_{index}는 전체 문서의 엘리먼트의 개수 $n_{element}$이다. 포스팅 파일에 필요한 공간 $S_{posting-f}$는 엘리먼트의 EID크기 S_{EID}에 엘리먼트의 총 개수인 $n_{element}$(식(3))를 곱한 것이다.

$S_{posting-f} = S_{EID} * n_{element} = S_{EID} *$ 식(3)

(2) 엘리먼트 위치별로 문서에 존재하는 모든 엘리먼트들을 색인하는 색인구조

색인어로 사용될 엘리먼트 정보는 엘리먼트의 (DEN, IEN)이므로, $S_{key_element}$는 (DEN, IEN)의 평균크기이다. 문서의 전체 엘리먼트들을 색인하므로 $n_{key_element}$는 GDIT에 존재하는 엘리먼트의 총 개수 $\sum_{i=1}^{h} k^{i-1}$ 이다. 그러므로, 색인파일에 필요한 공간 $S_{index-f}$는 (DEN, IEN)의 평균크기 $S_{key_element}$와 포인터 크기 S_{ptr}을 더한 다음, GDIT에 존재하는 엘리먼트의 총 개수 $\sum_{i=1}^{h} k^{i-1}$을 곱한 것이다.

$S_{index-f} = (S_{key\_element} + S_{ptr}) * \sum_{i=1}^{h} k^{i-1}$

($S_{key\_element}$: (DEN, IEN)의 평균크기)

포스팅 파일에 저장되는 색인정보는 문서번호 DIN이므로, 색인정보의 평균 크기 S_{id}는 문서번호 크기 S_{DIN}이다. 문서의 모든 엘리먼트들을 색인하므로 색인정보의 전체 개수 n_{index}는 전체 문서 인스턴스의 엘리먼트의 총 개수인 $n_{element}$이다. 그러므로, 포스팅 파일에 필요한 공간 $S_{posting-f}$는 문서번호크기 S_{DIN}에 엘리먼트의 총 개수 $n_{element}$(식(3))를 곱한 것이다.

$S_{posting-f} = S_{DIN} * n_{element} = S_{DIN} *$ 식(3)

(3) 엘리먼트 위치별로 문서에 존재하는 모든 단말 엘리먼트들만을 색인하는 색인 구조

색인어로 사용되는 정보는 엘리먼트의 (DEN, IEN)이므로, $S_{key_element}$는 (DEN, IEN)의 평균크기이다. 문서의 단말 엘리먼트만을 색인하므로, $n_{key_element}$는 GDIT에 존재하는 단말 엘리먼트의 총 개수인 k^{h-1}이다. 그러므로, 색인파일에 필요한 공간 $S_{index-f}$은 다음과 같다.

$S_{index-f} = (S_{key\_element} + S_{ptr}) * k^{h-1}$

포스팅 파일에 저장되는 정보는 문서번호 DIN이므로, 색인정보의 평균 크기 S_{id_e}는 문서번호크기 S_{DIN}이다. 단말 엘리먼트들만을 색인하므로 색인정보의 전체 개수 n_{index}는 전체 문서 인스턴스의 단말 엘리먼트의 총 개수인 $n_{element}$이다. 그러므로, 포스팅 파일에 필요한 공간 $S_{posting-f}$는 문서번호크기 S_{DIN}에 엘리먼트의 총 개수 $n_{element}$(식(2))를 곱한 것이다.

$S_{posting-f} = S_{DIN} * n_{element} = S_{DIN} *$ 식(2)

(4) 문서내의 엘리먼트 위치별로, 문서 구조에서 특정 엘리먼트의 하위 문서구조가 GDIT의 해당 위치의 하위 문서구조와 동일할 경우에는 동일한 하위 문서 구조를 가지는 가장 상위 엘리먼트만을 색인하는 색인 구조

엘리먼트 위치별로 색인하므로, 색인파일에 필요한 공간 $S_{index-f}$는 앞의 (2)에서 제안한 색인구조에서 필요한 색인공간과 같다. 포스팅 파일에 저장되는 색인정보는 문서번호 DIN이므로, 색인정보의 평균 크기 S_{id}는 문서번호 크기 S_{DIN}이다.

레벨 h에 m개의 노드가 1개 존재할 경우, 발생할 수 있는 문서 인스턴스 개수는 m개다. 이 m개의 문서 인스턴스의 $h-1$레벨의 노드가 색인되는 개수는 다음과 같다.

$h-1$레벨의 $\frac{m}{k}$개의 노드가 색인되는 문서 - 1개

$\frac{m}{k}-1$개의 노드가 색인되는 문서 - k개

$\frac{m}{k}-2$개의 노드가 색인되는 문서 - k개

:

1개의 노드가 색인되는 경우 - k개

0개의 노드가 색인되는 경우 - $k-1$개(하나도 색인되지 않는 경우)

레벨 h에 m개의 노드가 1개 존재할 경우 $h-1$레벨의 노드가 색인되는 갯수를 행렬 A라고 가정한다면, 레벨 h에 m개의 노드가 2개 존재할 경우 발생할 수 있는 문서 인스턴스 개수는 m^2개이고, 이 m^2개의 문서 인스턴스에서 $h-1$레벨의 노드가 색인되는 개수는 행렬 $A\times A$이다. 이때 노드 개수끼리는 서로 더하고, 문서 개수끼리는 서로 곱한다. 예를 들어, 행렬 A가 $\begin{pmatrix} 0개\ 2번 \\ 1개\ 3번 \end{pmatrix}$일 경우, $A\times A$는 다음과 같다.

$$\begin{bmatrix} 0개\ 2번 \\ 1개\ 3번 \end{bmatrix}\times\begin{bmatrix} 0개\ 2번 \\ 1개\ 3번 \end{bmatrix}=\begin{bmatrix} 0개4번,1개6번 \\ 1개6번,2개9번 \end{bmatrix}=\begin{bmatrix} 0개\ 4번 \\ 1개\ 12번 \\ 2개\ 9번 \end{bmatrix}=30개$$

이와 같은 과정을 반복하여 h레벨에 m개의 노드가 1개 존재하는 경우부터 d개 존재하는 경우까지의 각 경우의 노드 개수를 모두 더하면, n_{doc}개의 전체 문서 인스턴스의 $h-1$레벨의 엘리먼트 중에서 자식 노드의 차수가 k인 노드 개수 $n_{h-1_element}$가 되며, 이 개수에 k를 곱한 수만큼 단말 노드의 색인 개수가 줄어들게 된다.

$n_{h-1\_element}=\sum_{i=1}^{d}\{$ h레벨에 m개의 노드가 i개 있을 경우 $h-1$레벨의 노드중에서 자식 노드 차수가 k인 노드 개수$\}$

결국 단말 엘리먼트 중에서 색인되는 정보 개수는 전체 단말 엘리먼트 개수에서 $n_{h-1_element}$에 k를 곱한 것을 뺀 것이다.

$$n_{leaf\_element}-(n_{h-1\_element}*k)$$

다음으로, $h-1$레벨의 엘리먼트 중에서 색인 정보 개수는 다음과 같다.

$h-2$레벨에는 m개의 단말노드마다 $\frac{m}{k^2}$개의 노드가 존재하며, $\frac{m}{k^2}$개의 노드의 d개 존재한다. 그러므로, $h-2$레벨의 노드 개수 $n_{h-2_element}$를 구할 수 있고, 이 개수에 k을 곱한 수 만큼 h-1레벨의 색인 노드 개수가 줄어들게 된다. 결국 $h-1$레벨의 색인 정보 개수는 $h-1$레벨의 노드 개수 $n_{h-1_element}$에서 h-2레벨의 노드 개수 $n_{h-2_element}$에 k를 곱한 것을 뺀 것이다.

$$n_{h\_1\_element}-(n_{h-2\_element}*k)$$

루트노드가 색인되는 경우는 1번이고, 이 경우 k^{h-1}개의 단말노드를 색인하지 않는다. 이 과정을 모든 레벨에 걸쳐 수행하면, n_{doc}개의 전체 문서 인스턴스의 각 레벨의 색인되는 엘리먼트 개수를 구할 수 있다. 그러므로, 색인되는 문서의 전체 개수 n_{index}는 각 레벨의 색인되는 노드 개수를 더한 것이다.

$$\begin{aligned} n_{index} &= \{n_{leaf\_element}-(n_{h-1\_element}*k)\}+\sum_{l=1}^{h}\{n_{l\_element}-l-1\_element*k)\} \\ &= \sum_{l=1}^{h}\{n_{l\_element}-(n_{l-1\_element}*k)\} \end{aligned} \quad (5)$$

포스팅 파일에 필요한 공간 $S_{posting-f}$는 문서번호 크기 S_{DIN}에 색인되는 문서 개수 n_{index}를 곱한 것이다.

$$S_{posting-f}=[\sum_{l=1}^{h}\{n_{l\_element}-(n_{l\ 1\_element}*k)\}]*S_{DIN}$$

(5) 레벨 우선 순회규칙으로 엘리먼트의 ID를 할당하여 색인하는 경우

문서에 존재하는 엘리먼트 타입별로 모든 엘리먼트들을 색인하는 경우, 엘리먼트 타입별로 색인하므로 색인파일에 필요한 공간은 앞의 (1)에서의 색인파

일에 필요한 공간과 같으며 엘리먼트의 ID크기가 본 논문에서의 EID크기와 같다고 가정하면, 포스팅 파일에 필요한 공간도 앞의 (1)에서의 포스팅 파일에 필요한 공간과 같다. 나머지 색인 방법의 경우, 색인 파일에 필요한 공간은 앞의 (2), (3), (4)에서의 색인 파일에 필요한 공간 식에서 차수 k대신에 레벨 우선 순회규칙으로 엘리먼트의 ID를 할당할 때에 사용한 차수가 k'을 사용한 것이다. 문서구조에 엘리먼트의 추가가 발생하더라도 자식 엘리먼트의 최대 수가 k'를 넘지 않도록 k'은 큰 수로 정해야 한다. 포스팅 파일에 필요한 공간은 앞의 (2), (3)에서 나타낸 포스팅 파일에 필요한 공간과 같은 크기의 공간이 필요하며, (4)의 색인 방법을 사용하여 색인하는 경우에 포스팅 파일에 필요한 공간은, 차수가 k'인 문서는 존재하지 않을 것이므로, 문서 인스턴스의 가장 하위 레벨에 존재하는 엘리먼트만을 색인하는 경우인 (3)의 색인방법 필요한 포스팅 파일 공간과 같은 크기의 공간이 필요하다.

색인에 관련된 공간에 대한 분석을 요약하면 표 1과 표 2와 같다.

표 1. Comparison of space requirements for index file

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

A : 엘리먼트 타입별로 문서에 존재하는 모든 엘리먼트들을 색인하는 색인 구조
B : 엘리먼트 위치별로 문서에 존재하는 모든 엘리먼트들을 색인하는 색인 구조
C : 엘리먼트 위치별로 문서에 존재하는 모든 단말 엘리먼트들만을 색인하는 색인 구조
D : 엘리먼트 위치별로, 문서구조에서 엘리먼트의 하위 문서 구조가 GDIT의 해당 하위 위치의 하위 문서구조와 동일할 경우에는 동일한 하위 문서 구조를 가지는 가장 상위 엘리먼트만을 색인하는 색인구조
E : 레벨 우선 순회규칙으로 엘리먼트의 ID를 할당한 다음 D색인방법을 사용한 색인구조

표 2. Comparison of space requirements for posting file

색인 유형	포스팅 파일 공간($S_{posting\_f}$)
A	S_{EID} * 식(3)
B	S_{DIN} * 식(3)
C	S_{DIN} * 식(2)
D	$S_{DIN}*\sum_{i=1}^{h}\{n_{i\_element}-(n_{i-1\_element}*k)\}$
E	S_{DIN} * 식(2)

(A, B, C, D, E는 표 1 참조)

표 1에서 A를 제외한 나머지 색인 구조는 GDIT에 존재하는 엘리먼트들의 (DEN, IEN)별로 색인하고 A색인 구조는 엘리먼트 타입별로 색인한다. 그러므로 A색인 구조의 색인 파일의 색인어 갯수는 DTD트리에 존재하는 엘리먼트 개수 $d*(h-1)+1$개로 색인 파일 공간의 크기가 제일 작게 필요하다. A를 제외한 나머지 색인 구조에서 B와 D 그리고 E색인 구조는 모든 엘리먼트를 색인하므로 색인파일의 색인어 개수는 GDIT에 존재하는 엘리먼트 개수 $\sum_{i=1}^{h}k^{i-1}$로 동일하다. C색인 구조는 단말 엘리먼트만 색인하므로 색인파일의 색인어 개수는 GDIT에 존재하는 단말 엘리먼트 개수 k^{h-1}이다. 그러므로 C색인 구조는 B, D, E색인 구조보다 색인파일의 공간이 작게 필요하다.

표 2에서 A를 제외한 나머지 색인 구조는 포스팅 파일에 문서 번호 DIN으로 색인함으로 포스팅 파일 공간 식에서 DIN의 크기 S_{DIN}을 곱한다. 반면에 A색인 구조는 <DIN, (DEN, IEN)>으로 색인함으로 EID의 크기 S_{EID}을 곱한다. 또한 A색인 구조는 문서에 존재하는 모든 엘리먼트들(식(3))을 색인 대상으로 한다. 그러므로 A색인 구조가 포스팅 파일 공간이 제일 많이 필요하다. A를 제외한 나머지 색인 구조 중에서는 B색인 구조가 문서에 존재하는 모든 엘리먼트(식(3))를 색인한다. 그러므로 A를 제외한 나머지 색인 구조중에서 B색인 구조가 포스팅 파일 공간이 제일 많이 필요하다. C색인 구조는 문서의 단말 엘리먼트(식(2))만을 색인한다.

D와 E색인 구조는 GDIT와 비교하여 하위 문서 구조가 GDIT와 같을 경우에는 같은 하위 문서 구조를 가지는 가장 상위 엘리먼트를 색인한다.

따라서 D색인 구조는 색인되는 엘리먼트의 텍스트 레벨 엘리먼트들은 색인 대상에서 제외된다. 그러므로 색인되는 문서 갯수는 문서의 각 레벨의 노드 개수 $n_{l_element}$에서 부모 레벨의 노드 개수 $n_{l+1_element}$에 k를 곱한 것을 뺀 것이다. 결국, D색인구조는 C색인 구조보다 $n_{leaf\_element} - \sum_{l=1}^{h}\{ n_{l\_element} - (n_{l-1\_element} * k)\}$ 만큼 색인 개수가 줄어든다. 그러므로, D색인 구조는 C색인 구조보다 포스팅 파일 공간이 작게 필요하다. E색인 구조는 D와 같은 색인 구조이나 엘리먼트 추가등의 갱신때문에 레벨 우선 순회 규칙에서 정한 차수 k'은 큰 수로 정한다. 그러므로 문서에서 자식 노드 수가 k'인 엘리먼트는 존재하지 않기 때문에 C색인 구조와 같은 크기의 포스팅 공간이 필요하다.

6.3 검색 비용

5.1에서 제시한 색인 구조들에 대하여 질의 처리 시의 검색 비용을 분석한다. 다음은 구조 색인 구조들의 검색 비용을 분석하기 위하여 사용되는 기호의 일부이다.

- T_{ancest} : GDIT테이블에서 조상의 순서를 조사하여 관련 엘리먼트의 (DEN, IEN)을 찾는데 걸리는 시간
- n_{search_leaf} : GDIT에서 검색 엘리먼트의 하위에 존재하는 단말 엘리먼트 개수

색인에 필요한 검색비용을 분석하기 위하여 다음과 같은 가정을 한다.

가정 1 : 색인 파일은 B^+-트리로 구성되며 B^+-트리의 차수는 k로 GDIT의 차수와 같다.

가정 2 : 색인 파일 접근 시간은 모든 색인 구조에 대해 동일하다.

구조 질의를 처리하는데 필요한 시간 T_{search}는 DTD 테이블에서 질의에서 찾는 엘리먼트의 DEN을 찾는 시간 T_{DTD_t}, GDIT 테이블에서 해당 DEN의 관련 정보를 찾는 시간 T_{GDIT_t}, 역리스트에서 질의에서 찾는 엘리먼트의 해당 문서번호를 찾는 시간 $T_{inverted-list}$의 합이다.

$$T_{search} = T_{DTD\_t} + T_{GDIT\_t} + T_{inverted-list}$$

5장에서 제시한 색인 구조들은 DTD와 GDIT테이블을 사용하여 질의를 처리함으로 T_{DTD_t}와 T_{GDIT_t}은 모두 동일하다. 그러므로 여기서는 역리스트에서 질의에서 찾는 엘리먼트의 해당 문서번호를 찾는 시간 $T_{inverted-list}$에 대해서 비교한다. $T_{inverted-list}$는 색인 파일에서 색인어 엘리먼트를 찾는 시간 $T_{index-f}$와 포스팅 파일에서 해당 정보를 찾는 시간 $T_{posting-f}$의 합이다

$$T_{inverted-list} = T_{index-f} + T_{posting-f}$$

여기서 색인 파일은 B^+-트리로 구성되며 B^+-트리의 차수는 k로 GDIT의 차수와 같다고 가정한다. 그러므로, B^+-트리의 높이가 l이면 색인 파일에서 색인어 엘리먼트를 찾는 시간 $T_{index-f}$는 다음과 같다.

$$T_{index-f} = l * t_{random}$$

(1) 엘리먼트 타입별로 모든 엘리먼트들을 색인하는 색인 구조

색인 파일은 DTD의 엘리먼트들로 구성되며 DTD의 엘리먼트 개수는 $(d*(h-1)+1)$이므로 B^+-트리의 높이 l은 $\log_k (d*(h-1)+1)$이다. 그러므로 색인 엘리먼트를 찾는 시간 $T_{index-f}$는 DTD에 나타나는 엘리먼트 종류 중에서 원하는 엘리먼트를 찾는 시간이다.

$$T_{index-f} = \log_k (d*(h-1)+1) * t_{random}$$

포스팅 파일에서 엘리먼트를 찾는 평균시간 $T_{posting-f}$는 색인 엘리먼트의 포스팅 엔트리 개수 $N_{post-per-key}$에 포스팅 엔트리의 EID들을 조사하여 검색 엘리먼트의 (DEN, IEN)을 찾는데 걸리는 시간 t_{EID}를 곱한 것이다.

$$T_{posting-f} = (N_{post-per-key} - 1) * t_{EID}$$

색인 엘리먼트마다 평균적인 포스팅 엔트리 개수 $N_{post-per-key}$는 전체 엘리먼트 갯수 $n_{element}$를 DTD 트리의 노드 개수 $(d*(h-1)+1)$로 나눈 것이다.

$$N_{post-per-key} = \frac{n_{element}}{(d*(h-1)+1)}$$

그러므로, 검색 시간 T_{search}는 다음과 같다.

$$T_{index-f} = \log_k (d*(h-1)+1) * t_{random} + (\frac{n_{element}}{(d*(h-1)+1)} - 1) * t_{EID}$$

(2) 문서내의 엘리먼트 위치별로 문서에 존재하는 모든 엘리먼트들을 색인하는 색인구조

색인 파일은 GDIT의 모든 엘리먼트들로 구성된다. GDIT의 엘리먼트 개수는 $\sum_{i=1}^{h} k^{i-1}$이므로 B$^+$-트리의 높이 l은 $\log_k \sum_{i=1}^{h} k^{i-1}$이다. 그러므로 $T_{index-f}$는 다음과 같다.

$$T_{index-f} = \log_k \sum_{i=1}^{h} k^{i-1} * t_{random}$$

포스팅 파일에서 엘리먼트를 찾는 평균시간 $T_{posting-f}$는 포스팅 파일에서 검색 엘리먼트의 문서 번호 리스트를 추출하는 시간이다. 엘리먼트 색인 정보 (DEN, IEN)마다 포스팅 파일에 색인되는 평균적인 문서 개수 $N_{post-per-key}$는 전체 엘리먼트 개수 $n_{element}$를 GDIT의 노드 개수 $\sum_{i=1}^{h} k^{i-1}$로 나눈 것이다.

$$N_{post-per-key} = \frac{n_{element}}{\sum_{i=1}^{h} k^{i-1}}$$

하나의 문서 번호를 추출하는 시간이 t_{DIN}이라면, 포스팅 파일에서 엘리먼트를 찾는 평균시간 $T_{posting-f}$는 다음과 같다.

$$T_{posting-f} = N_{post-per-key} * t_{DIN} = \frac{n_{element}}{\sum_{i=1}^{h} k^{i-1}} * t_{DIN}$$

그러므로, 검색 시간 T_{search}는 다음과 같다.

$$T_{search} = \log_k \sum_{i=1}^{h} k^{i-1} * t_{random} + \frac{n_{element}}{\sum_{i=1}^{h} k^{h-i}} * t_{DIN}$$

(3) 엘리먼트 위치별로 문서에 존재하는 모든 단말 엘리먼트들만을 색인하는 색인 구조

색인 파일은 GDIT의 단말 엘리먼트들로 구성된다. GDIT의 단말 엘리먼트 개수는 k^{h-1}이므로 B$^+$-트리의 높이 l은 $\log_k k^{h-1}$이다. 검색 엘리먼트의 레벨이 n이라면, GDIT에서 검색 엘리먼트의 하위에 존재하는 단말 엘리먼트 개수 n_{search_leaf}는 k^{h-n}이므로, $T_{index-f}$는 B$^+$-트리의 높이 $\log_k k^{h-1}$에 t_{random}과 k^{h-n}을 곱한 것이다.

$$T_{index-f} = \log_k k^{h-1} * t_{random} * k^{h-n}$$

엘리먼트 위치별로 색인하므로 포스팅 파일에는 문서 번호가 색인된다. GDIT에서의 단말 엘리먼트에 대해서만 문서번호를 색인하므로, 레벨 n의 검색 엘리먼트가 있는 문서를 찾기 위해서는 검색 엘리먼트의 하위에 존재하는 단말 엘리먼트들의 포스팅 리스트를 검색해야 한다. 이 때 단말 엘리먼트의 관련 문서 번호를 추출할 때에 문서 번호가 이미 추출된 문서인지의 중복 여부를 조사해야 한다. 엘리먼트 위치별로 색인하므로 단말 엘리먼트의 평균적인 문서 개수는 앞의 (2)에서와 같다. 그러므로 포스팅 파일에서 엘리먼트를 찾는 평균시간 $T_{posting-f}$는 포스팅 파일에서 레벨 n의 검색 엘리먼트의 단말 엘리먼트의 포스팅 리스트에서 문서 번호를 추출하는 시간의 합이다.

$$T_{posting\_f} = N_{post\_per\_key} * t_{DIN} * t_{comp-DIN} * n_{search\_leaf} = \frac{n_{element}}{\sum_{i=1}^{h} k^{i-1}} * t_{DIN} * t_{comp\_DIN} * k^{h-n}$$

여기서 $t_{comp-DIN}$는 검색시에 임의의 문서가 이미 추출된 문서인지 조사하는 시간이다. 그러므로, 검색 시간 T_{search}는 다음과 같다.

$$T_{index-f} = \log_k k^{h-1} * t_{random} * k^{h-n} + \frac{n_{element}}{\sum_{i=1}^{h} k^{i-1}} * t_{DIN} * t_{comp-DIN} * k^{h-n} = (\log_k k^{h-1} * t_{random} + \frac{n_{element}}{\sum_{i=1}^{h} k^{i-1}} * t_{DIN} * t_{comp-DIN}) * k^{h-n}$$

(4) 엘리먼트 위치별로, 엘리먼트의 하위 문서구조가 GDIT의 하위 문서구조와 비교하여 동일할 경우에 동일한 하위 문서 구조를 가지는 가장 상위 엘리먼트만을 색인하는 색인하는 색인구조색인 파일은 GDIT의 모든 엘리먼트들로 구성된다. GDIT의 엘리먼트 개수는 $\sum_{i=1}^{h} k^{i-1}$이므로 B$^+$-트리의 높이 l은 $\log_k \sum_{i=1}^{h} k^{i-1}$이다. 검색 엘리먼트의 레벨이 n이라면, GDIT에서 검색 엘리먼트를 포함하여 검색엘

리먼트의 하위에 존재하는 엘리먼트 개수 n_{desce_e}가 $\sum_{i=0}^{h-n} k^i$이고, 검색엘리먼트의 조상 엘리먼트의 개수 n_{ances_e}가 $h-n$이므로, $T_{index-f}$는 다음과 같다.

$$T_{index-f} = \log_k \sum_{i=1}^{h} k^{i-1} * t_{random} * (\sum_{i=0}^{h-n} k^i + (h-n))$$

엘리먼트 위치별로 색인하므로 포스팅 파일에는 문서 번호가 색인된다. GDIT와 비교하여 하위 문서 구조가 같은 가장 상위 엘리먼트를 색인하므로, 레벨 n의 검색 엘리먼트가 있는 문서를 찾기 위해서는 검색 엘리먼트의 조상 엘리먼트들과 검색 엘리먼트의 하위 문서 구조에 존재하는 엘리먼트들의 포스팅 리스트를 검색해야 한다. 이 때 하위 문서 구조에 존재하는 엘리먼트들의 관련 문서 번호를 추출할 때에는 문서 번호가 이미 추출된 문서인지의 중복 여부를 조사해야 한다. 색인 엘리먼트마다 포스팅 파일에 저장되는 평균적인 문서 개수 $N_{post-per-key}$는 앞의 색인공간 비교에서의 식(5)를 GDIT에서의 노드 개수 $\sum_{i=1}^{h} k^{i-1}$로 나눈 것이다.

$$N_{post-per-key} = \frac{\text{식(5)}}{\sum_{i=1}^{h} k^{i-1}}$$

그러므로 포스팅 파일에서 엘리먼트를 찾는 평균 시간 $T_{posting-f}$는 포스팅 파일에서 레벨 n의 검색 엘리먼트의 조상 엘리먼트들과 하위 문서구조에 존재하는 엘리먼트들의 포스팅 리스트에서 문서 번호를 추출하는 시간의 합이다.

$$T_{posting-f} = N_{post-per-key} * t_{DIN} * N_{ances\_t} + N_{post-per-key} * t_{DIN} * N_{desce\_t} * t_{comp-DIN} = \frac{\text{식(5)}}{\sum_{i=1}^{h} k^{i-1}} * t_{DIN} * (h-n) + \frac{\text{식(5)}}{\sum_{i=1}^{h} k^{i-1}} * t_{DIN} * \sum_{i=0}^{h-n} k^i = \frac{\text{식(5)}}{\sum_{i=1}^{h} k^{i-1}} * t_{DIN} * [(h-n) + \sum_{i=0}^{h-n} k^i]$$

여기서 $t_{comp-DIN}$는 검색시에 임의의 문서가 이미 추출된 문서인지 조사하는 시간이다. 그러므로, 검색 시간 T_{search}는 다음과 같다.

$$T_{search} = \log_k \sum_{i=1}^{h} k^{i-1} * t_{random} * [\sum_{i=0}^{h-n} k^i + (h-n)] + \frac{\text{식(5)}}{\sum_{i=1}^{h} k^{i-1}} * t_{DIN} * [(h-n) + \sum_{i=0}^{h-n} k^i]$$

(5) 레벨 우선 순회규칙으로 엘리먼트의 ID를 할당하여 색인하는 경우

엘리먼트 타입별로 모든 엘리먼트들을 색인하는 경우, 엘리먼트 타입별로 색인하므로 검색시간은 앞의 (1)에서의 검색시간과 같다. 나머지 색인 방법의 경우, 색인파일에 필요한 검색시간은 앞의 (2), (3), (4)에서의 식에서 차수 k대신에 레벨 우선 순회규칙으로 엘리먼트의 ID를 할당할 때에 사용한 차수 k'을 사용한 것이다. k'은 엘리먼트의 추가가 발생하더라도 자식 엘리먼트의 최대 수가 k'을 넘지 않도록 큰 수로 정해야 한다. 검색시간에 관련된 분석을 요약하면 표 3과 같다.

표 3. Comparison of search time

색인 유형	검색 시간(T_{search})
A	$\log_k (d*(h-1)+1) * t_{random} + (\frac{n_{element}}{(d*(h-1)+1)} - 1) * t_{EID}$
B	$\log_k \sum_{i=1}^{h} k^{i-1} * t_{random} + \frac{n_{element}}{\sum_{i=1}^{h} k^{h-i}} * t_{DIN}$
C	$(\log_k k^{h-1} * t_{random} + \frac{n_{element}}{\sum_{i=1}^{h} k^{i-1}} * t_{DIN} * t_{comp-DIN}) * k^{h-n}$
D	$\log_k \sum_{i=1}^{h} k^{i-1} * t_{random} * [\sum_{i=0}^{h-n} k^i + (h-n)] + \frac{\text{식(5)}}{\sum_{i=1}^{h} k^{i-1}} * t_{DIN} * [(h-n) + \sum_{i=0}^{h-n} k^i]$
E	$(\log_k k'^{h-1} * t_{random} + \frac{n_{element}}{\sum_{i=1}^{h} k'^{i-1}} * t_{DIN} * t_{comp-DIN}) * k'^{h-n}$

(A, B, C, D, E는 표 1 참조)

표 3에서 A색인 구조는 엘리먼트 타입별로 색인하므로, 질의 발생시 $\frac{n_{element}}{(d*(h-1)+1)}$개의 엘리먼트 EID를 매번 비교해야 한다. 그러므로 검색 시간이 제일 많이 필요하다. A를 제외한 나머지 색인 구조들은 GDIT의 (DEN, IEN)별로 색인한다. B색인 구조는 모든 엘리먼트들을 색인하므로 포스팅 파일에서 검색 엘리먼트의 문서번호를 추출하면 된다. 그러므로 검색 시간이 제일 작게 필요하다. C색인 구조는 n_{search_leaf}번, D와 E색인 구조는 $n_{ances_t} + n_{desce_e}$번의 포스팅 리스트 접근이 필요한데 n_{search_leaf}는 검색 엘리먼트의 단말 엘리먼트 개수

이고, $n_{ances_t} + n_{desce_e}$는 검색 엘리먼트의 조상 엘리먼트와 하위에 존재하는 엘리먼트 개수이다. 따라서, $n_{search_leaf} < n_{ances_t} + n_{desce_e}$이다. 또한 C색인 구조는 레벨 n의 검색 엘리먼트의 단말 엘리먼트 k^{h-n}개의 포스팅 리스트의 문서번호의 중복여부을 비교하고 D색인구조는 검색 엘리먼트의 하위에 존재하는 엘리먼트 $\sum_{i=0}^{h-n} k^i$개의 포스팅 리스트의 문서번호의 중복여부를 조사해야 한다. 그러므로, $k^{h-n} < \sum_{i=0}^{h-n} k^i$이다. 그러나 C색인 구조에서 색인정보의 평균 포스팅의 개수 $N_{post-per-key}$는 $\frac{n_{element}}{\sum_{i=1}^{h} k^{i-1}}$이고, D색인 구조에서 색인정보의 평균 포스팅의 개수 $N_{post-per-key}$는 $\frac{식(5)}{\sum_{i=1}^{h} k^{i-1}}$로 $\frac{식(5)}{\sum_{i=1}^{h} k^{i-1}} < \frac{n_{element}}{\sum_{i=1}^{h} k^{i-1}}$이다. 또한 D색인구조는 검색 엘리먼트의 조상 엘리먼트에 대한 포스팅 리스트의 문서번호는 중복 여부를 비교하지 않아도 되며, 조상 엘리먼트의 포스팅 리스트의 문서 개수에 단말 엘리먼트 개수를 곱한 만큼 문서 중복 비교 횟수가 줄어든다. 그러므로 D색인 구조가 C색인 구조보다 검색 시간이 빠르다. E색인 구조는 D와 같은 색인 방법을 사용하지만 문서에서 자식 노드의 수가 k'인 경우가 없으므로, C색인 구조의 검색시간에서 차수 k대신에 k'을 사용한 것이다.

7. 결 론

구조적 문서는 문서로의 다양한 접근 경로를 제공하므로, 문서의 구조를 사용한 구조 검색 질의는 검색의 신뢰도를 높일 수 있다. 구조적 문서에 내장된 구조 정보를 검색하는 구조 기반 검색기에는 구조적 문서에 관련된 구조기반, 내용과 구조의 혼합, 순서에 관련된 질의 등 여러 유형의 구조 질의들을 처리하기 위한 색인 구조가 필요하다.

본 논문에서는 색인어가 존재하는 텍스트 레벨 엘리먼트에 대해서만 색인하는 GDIT기반의 내용기반 색인 구조에 덧붙여서 순수 구조에 관련된 질의들을 처리할 수 있는 네 가지 구조 색인 구조를 제시하였다. 제시한 순수 구조 색인 구조들은 GDIT를 기반으로 한다. 그리고 네 가지 색인 구조를 색인에 필요한 공간과 검색시간에 대하여 그 성능을 분석하였다. 또한 레벨 우선 순회규칙으로 ID를 할당하는 방법에 본 논문에서의 GDIT기법을 적용할 경우의 색인공간과 검색시간에 대하여도 성능을 분석하였다. 색인 공간에서는 엘리먼트 타입별로 문서에 존재하는 모든 엘리먼트들을 색인하는 색인 구조가 색인 공간의 크기가 제일 작게 필요하고, 포스팅 공간에서는 엘리먼트 위치별로, 문서구조에서 특정 엘리먼트의 하위 문서구조가 GDIT의 해당 위치의 하위 문서구조와 동일할 경우에는 동일한 하위 문서 구조를 가지는 가장 상위 엘리먼트만을 색인하는 색인구조가 포스팅 파일의 크기가 제일 작게 필요하다. 결국 문서구조에서 특정 엘리먼트의 하위 문서구조가 GDIT의 해당 위치의 하위 문서구조와 동일할 경우에는 동일한 하위 문서 구조를 가지는 가장 상위 엘리럼트만을 색인하는 색인방법이 색인 공간면에서 가장 우월하다. 검색시간에서는 엘리먼트 위치별로 문서에 존재하는 모든 엘리먼트들을 색인하는 색인방법이 색인공간은 많이 필요하나 검색 시간은 제일 빠르다.

참 고 문 헌

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

[2] 배종민, 김영자 "GDIT를 기반으로 한 구조적 문서의 효율적 검색과 갱신을 위한 인덱스 설계", 정 보처리학회 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] R. Sacks-Davis, A. Kent, K. Ramamohanarao, J. Thom, and J. Zobel, "Atlas: a nested relational database system for text applications," Technical Report CITRI/TR-92-52, Collaborative Information Technology Research Institute, Melbourne, Australia, 1992.

[9] J. A. Thom, A. J. Kent, and R. Sacks-Davis, "TQL: A nested relational query language," Australian Computer Journal, 23(2), 1991.

[10] K. Hirotaka, K. Hiroyuki, K. Hiroko and Y. Masatoshi, "An Efficiently Updatable Index Scheme for Structured Document," in proc. 9th International Workshop on Database and Expert Systems Application(DEXA), 1998.

[11] 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.

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

김 영 자

1991년 경상대학교 전산과(이학사)
1993년 경상대학교 전산과(공학석사)
2000년 경상대학교 전산과(공학박사)
1993년~현재 동주여자상업고 재직
관심분야 : 병렬프로그래밍 언어, 디지털 도서관, 구조정보 검색

김 현 주

1988년 경상대학교 전산과(이학사)
1990년 숭실대학교 전산과(공학석사)
2000년 경상대학교 전산과(공학박사)
1994년~1997년 제일정밀공업(주) 연구원
2000년~2002년 경남정보대학 컴퓨터정보계열 전임강사
2002년~현재 진주산업대학교 컴퓨터공학부 조교수
관심분야 : 정보검색, 디지털 도서관, 웹 프로그래밍, XML, Sematics Web

배 종 민

1980년 서울대학교 사범대학 수학과(이학사)
1983년 서울대학교 계산통계학과(이학석사)
1995년 서울대학교 계산통계학과(이학박사)
1982년~1984년 한국전자통신연구원 연구원
1984년~현재 경상대학교 컴퓨터과학과 교수
관심분야 : 병렬 프로그래밍 언어, 디지털 도서관, 정보검색