GDIT기반의 순수 구조 질의 처리를 위한 색인 구조에 대한 분석

정채영[U], 김영자, 김현주, 배종민
경상대학교 컴퓨터과학과 / 정보통신연구센터

Analysis of Index Structures for Structure-based Retrieval on GDIT

Chae-Young Jung[U], Young-Ja Kim, Hyun-Ju Kim, Jong-Min Bae
Department of Computer Science, Kyeongsang National University
/ Information and Communication Research Center

요 약

구조적 문서는 문서로의 다양한 접근 경로를 제공하므로, 구조적 문서들에 대한 정보 검색 시스템들은 문서의 구조를 사용한 구조 검색 질의를 지원하여 검색의 신뢰도를 높일 수 있다. 그러므로, 구조적 문서가 가지는 엘리먼트간의 구조적 포함관계나 순서등 문서 구조에 바탕을 둔 다양한 유형의 사용자 질의를 지원할 수 있는 색인 구조가 필요하다.

본 논문에서는 엘리먼트와 엘리먼트 사이의 구조적 상관관계나 엘리먼트의 발생순서에 관련된 질의등 여러 유형의 순수 구조 질의를 처리할 수 있는 세가지 색인구조를 제시하고 그 성능을 평가한다. 제안된 색인 알고리즘은 GDIT 개념[5]에 바탕을 두고, GDIT기반의 색인기법을 사용한다.

1. 서론

정보검색시스템에서 문서의 구조를 활용한 검색 질의를 처리하기 위해서는 문서의 내용은 물론이고, 문서의 구조를 나타내는 엘리먼트에 대해서도 색인을 해야 한다. 이때 문서의 구조를 나타내는 모든 엘리먼트를 색인어와 연관시켜서 색인하면 색인에 필요한 공간이 크게 늘어난다. 색인공간을 줄이는 대표적인 기법은 색인어를 포함하고 있는 엘리먼트, 즉 텍스트 레벨 엘리먼트만을 색인하는 것이다[4, 5]. 그러나 이 색인기법만으로 모든 구조기반 검색 질의를 효과적으로 처리하기는 힘들다. 왜냐하면, 위와 같은 색인구조는 색인어와 엘리먼트와의 관계를 나타낼 뿐, 엘리먼트와 엘리먼트 사이의 구조적 포함관계에 대한 정보가 포함되어 있지 않다. 따라서 엘리먼트 사이의 관계에 대한 질의를 효과적으로 처리하기 어렵다.

본 논문에서는 엘리먼트와 엘리먼트 사이의 구조적 상관관계를 제공하고, 엘리먼트의 발생순서에 관련된 순수 구조 질의를 처리할 수 있는 색인구조를 제시하고 그 성능을 평가한다. 본 논문에서 제시한 색인알고리즘은 GDIT[5]의 개념에 바탕을 두고 있다. GDIT는 문서 인스턴스의 구조를 트리로 표현했을 때, 모든 문서 인스턴스 트리의 합집합이다[5]. 본 논문에서는 GDIT기반의 색인기법을 사용해서, 순수 구조 질의를 처리하기 위한 문서 구조의 세가지 색인 기법을 제시한다. 제시된 세가지 색인기법은, 첫째, 문서에 존재하는 모든 엘리먼트들을 색인하는 색인구조 둘째, 문서에 존재하는 모든 단말 엘리먼트들만을 색인하는 색인구조 셋째, 문서에서 임의의 엘리먼트의 하위 문서 구조가 GDIT에 있는 구조와 동일할 경우에는 문서번호를 임의의 엘리먼트에만 기억시켜 색인하는 색인구조이다. 그리고 제시된 세가지 색인 구조들을 색인공간을 기준으로 분석한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구에 대하여 논하고 3장에서는 GDIT 구성방법과 그 의미에 대해서 논한다. 4장에서는 GDIT를 기반으로 하여 문서의 구조에 대한 세가지 색인구조들을 제안한다. 5장에서는 제안된 색인방법들을 색인공간을 기준으로 평가하고 6장에서 결론을 보인다.

2. 관련 연구

구조기반검색의 기반이 되는 색인 구조로서 전통적인 방법은, 문서에서 발생하는 모든 엘리먼트들을 색인하는 것이다[1]. 이 방법은 추출된 색인어가 발생된 엘리먼트의 모든 상위 엘리먼트에 대해서도 색인을 하게 되어 공간상의 중복이 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 모든 엘리먼트의 색인을 피하므로서, 색인에 필요한 메모리 오버헤드를 줄일 수 있으나, 내용색인만 고려함으로서, 구조에 관련된 질의를 처리하는데 제약이 있다. 다음으로 문서에 포함되어 있는 색인어와 엘리먼트들의 발생 위치를 사용하여, 내용색인 외에 각 문서의 구조들을 색인하는 방법이 있다[2]. 이 방법은 문서 구조 색인으로 순수 구조 질의들은 처리할 수 있으나 내용색인시에 색인어마다 위치정보만을 저장함으로서 내용과 구조의 혼합질의를 효율적으로 처리할 수 없으며, 각 문서마다 구조 정보를 저장함으로 메모리 오버헤드가 있다.

본 논문에서는 GDIT를 기반으로 하여 순수 구조기반 검색을 할 수 있는 문서 구조에 대한 세가지 색인 방법을 제시하고 이 색인 방법들을 색인에 필요한 공간을 기준으로 평가한다.

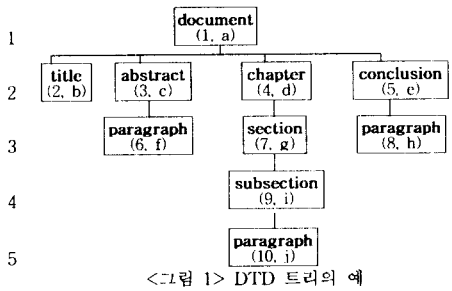
3. GDIT

3.1 GDIT 구성

GDIT[5]는 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)이라 칭한다. j값은 문서 인스턴스에서 대응되는 엘리먼트가 실제로 발생된 횟수를 말한다.

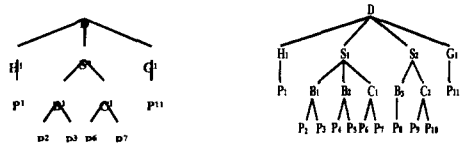
GDIT를 구성하기 위해서는 모든 문서 인스턴스 트리를 합치고, 합쳐진 각 노드에 ID로서 사용될 (DEN, IEN)값을 결정하는 두가지 작업이 필요하다. 여기서 DEN이란 DTD 엘리먼트 발생 순서를 말하고, IEN(Instance Element Number)이란 문서 인스턴스에서 해당 엘리먼트가 발생된 순서를 의미한다. 임의의 첫 번째 문서 인스턴스의 각 엘리먼트에 대하여, <그림 1>의 DTD트리로부터 DEN값을 알 수 있고, DTD노드에 부여된 순서쌍의 두 번째 값 즉, 엘리먼트가 실제로 발생된 횟수를 1증가시키고 그 값을 IEN으로 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. GDIT의 각 엘리먼트에 부여된 DEN값은 그 엘리먼트의 DTD트리 상의 조상의 경로정보를, IEN은 문서 인스턴스상의 조상의 경로에 관한 정보라는 의미를 부여하여 엘리먼트 식별자의 구성요소는 (DEN, IEN)쌍에 문서 인스턴스 번호(DIN:Document Instance Number)를 추가하여 <DIN, (DEN, IEN)>로 표현한다.

4. 문서 구조에 대한 색인 구조

3장에서 제안한 GDIT를 기반으로 하여 문서의 구조와 구조사이의 관계를 포함하는 질의를 처리하기 위한 세가지 색인방법들을 제안한다.

4.1 색인 유형

<그림 2>과 같은 임의의 문서 인스턴스 구조와 GDIT가 있다고 가정하자.



(a) 임의의 문서 인스턴스 구조 (b) GDIT

<그림 2> 임의의 문서 인스턴스 구조와 GDIT

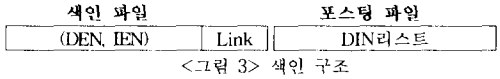
(1) 엘리먼트 위치별로 문서에 존재하는 모든 엘리먼트들을 색인하는 색인 구조

엘리먼트 위치별로 문서에 존재하는 모든 엘리먼트들을 색인하는 방법이다. 엘리먼트 위치별로, 역리스트에 해당 엘리먼트가 존재하는 모든 문서 인스턴스의 문서 인스턴스 번호 DIN을 가진다. 임의의 DTD에 대한 GDIT가 <그림 2>의 (b)라고 가정하고, 엘리먼트 위치별로 색인할 경우, 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 2)의 (a)의 문서 인스턴스에 대하여 역리스트에 문서 인스턴스 번호 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}}



<그림 3> 색인 구조

(2) 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 2>의 (b)라고 가정하고, 엘리먼트 위치별로 색인할 경우, 색인파일을 구성할 엘리먼트들은 다음과 같다.

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 2>의 (a)의 문서 인스턴스에 대하여 역리스트에 문서 인스턴스 번호 DIN이 나타날 엘리먼트들은 다음과 같다.

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

색인구조는 <그림 3>의 색인구조와 동일하다.

(3) 문서 인스턴스 구조에서 특정 엘리먼트의 하위 문서 구조가 GDIT에 있는 구조와 동일할 경우에는 문서번호를 특정 엘리먼트에만 기억시켜 색인하는 색인구조

문서 인스턴스에 존재하는 엘리먼트들의 하위 문서 구조를 GDIT 구조와 비교하여, 엘리먼트의 하위 문서 구조가 GDIT의 해당 하위 문서 구조와 동일할 경우에는, 동일한 하위 문서 구조를 가지는 엘리먼트만 색인하고 그 하위 엘리먼트들은 색인하지 않는다. 결국 임의의 엘리먼트가 있는 문서를 찾을 경우에는 임의의 엘리먼트의 색인리스트와 임의의 엘리먼트의 조상엘리먼트들의 색인리스트 그리고 임의의 엘리먼트의 하위 엘리먼트들의 색인리스트의 합이다.

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

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

<그림 2>의 (a)의 문서 인스턴스에 대한 문서 인스턴스 번호 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 3>의 색인구조와 동일하다.

5. 평가

본 장에서는 본 논문에서 제시한 엘리먼트 색인 구조의 성능을 분석한다.

5.1 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 경로 갯수
- m : DTD트리의 각 단말 엘리먼트마다 GDIT에서 발생하는 노드 갯수
- 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 DTD내에 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.

5.2 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.

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

이와 같은 n_{doc}개의 전체 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.

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

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.

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

5.3 색인공간 분석

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.

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

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 색인될 정보의 전체 개수 n_{index}에 색인정보의 평균 크기 S_{id}를 곱한 것이다.

$$S_{posting\_f} = n_{index} * S_{id}$$

<표 1> 색인에 필요한 공간의 비교(I)

A : 문서 인스턴스에 존재하는 DEN별로 색인하는 색인 구조
B : 문서 인스턴스 구조에서 모든 단말 엘리먼트들만을 색인하는 색인구조
C : 문서 인스턴스 구조에서 특정 엘리먼트의 하위 문서 구조가 GDIT에 있는 구조와 동일할 경우에는 문서 번호를 특정 엘리먼트에만 기억시켜 색인하는 색인구조

	색인어로 사용되는 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	(DEN, IEN)	$(S_{key\_element} + S_{ptr}) * \sum_{i=1}^{h} k^{i-1}$

A와 C색인구조는 GDIT에 존재하는 모든 엘리먼트를 색인하고 B색인구조는 단말 엘리먼트만을 사용하므로, B색인구조가 색인파일 크기가 A와 C색인구조보다 작다.

<표 2> 색인에 필요한 공간의 비교(II)

A, B, C : <표1>과 동일

	포스팅 파일 공간($S_{posting\_f}$)
A	S_{DIN} * 식(3)
B	S_{DIN} * 식(2)
C	$\{식(2) - \sum_{i=1}^{h-1}(n_{l\_element} * h^{h-i}) + \sum_{i=1}^{h-1} n_{l\_element}\} * S_{DIN}$

B색인구조는 문서 인스턴스에 존재하는 모든 단말 엘리먼트만을 색인하므로 A색인구조보다 포스팅파일 크기가 작다. C색인구조는 문서 인스턴스의 각 엘리먼트의 하위 문서 구조를 GDIT의 해당 하위 문서 구조와 비교하여 같을 경우에는 하위 문서 구조가 같은 가장 상위 엘리먼트만을 색인한다. 그러므로, A색인구조보다 포스팅 파일 크기가 작다. <표 1>에서 C색인구조가 A색인구조보다 색인파일 크기는 더 크나 이것은 GDIT에 존재하는 엘리먼트들에 대한 크기이다. 그러므로, C색인구조가 색인공간이 제일 작게 필요하다.

7. 결론

구조적 문서는 문서로의 다양한 접근 경로를 제공하므로, 문서의 구조를 사용한 구조 검색 질의는 검색의 신뢰도를 높일 수 있다. 그러므로, 구조적 문서에 관련된 순수 구조 질의를 처리하기 위한 색인 구조가 필요하다. 본 논문에서는 GDIT기반의 색인기법을 사용해서 순수 구조에 관련된 질의들을 처리할 수 있는 세가지 색인 구조를 제시한 다음, 색인에 필요한 공간에 대하여 그 성능을 분석하였다. 그 결과 문서 인스턴스의 각 엘리먼트의 하위 문서 구조를 GDIT의 해당 하위 문서 구조와 비교하여 같을 경우에는 하위 문서 구조가 같은 가장 상위 엘리먼트만을 색인하는 색인 방법이 가장 우월하였다.

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] 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.

[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

[5] 배종민, 김영자 "GDIT를 기반으로 한 구조적 문서의 효율적 검색과 갱신을 위한 인덱스 설계", 정보처리학회지