효율적인 부울 질의 연산에 관한 연구

An Efficient Boolean Query Processing in Information Retrieval

채승기(Seung-Gi Chae)* 남영광(Young-Kwang Nam)**
이준호(Joon-Ho Lee)*** 박현주(Hyun-Joo Park)****

목 차

초 록

본 논문에서는 부울검색시스템에서 사용자로부터 입력되는 부울 질의를 효율적으로 연산하기 위한 부울 질의 최적화 방법 4가지를 기술한다. 첫째, 프로그래밍 언어에서 논리식의 계산에 사용되는 단거리계산 방법을 적용한다. 둘째, AND, NOT과 같은 특정 연산자를 효율적으로 연산하기 위하여 색인어 출현 빈도의 차이를 이용한다. 세째, 분배법칙이 적용된 질의를 원래의 식으로 변환하여 연산의 수를 감소시킨다. 마지막으로, 반복되는 식을 포함하는 질의에 대하여 중복 연산을 회피한다. 또한 위의 4가지 방법들을 UNIX 환경에서 개발된 KRISTAL-Ⅱ 시스템에 구현하여, 제시된 방법들이 특정 경우에 검색 속도를 향상시킬 수 있음을 검증하였다.

ABSTRACT

In this paper, we propose four optimizing methods for effectively processing queries in the Boolean information retrieval system ; (i) the short-circuit evaluation scheme used for optimizing logical expressions in programming languages is applied to Boolean queries. (Ⅱ) use the difference of the number of index word frequencies appearing in the related documents. (Ⅱ i) reduce the number of operators in the queries by applying the distribution law in the set theory. (iv) evaluate only once for the repeated expressions in the query. These methods have been implemented and tested in KRISTAL-Ⅱ system on the UNIX workstation environment.

※ 키워드 : 부울 질의, 질의 최적화, 단거리계산, 출현빈도, 검색 속도

*연세대학교 문리대학 전산학과
**한국과학기술연구원 연구개발정보센터 연구개발부
■ 논문 접수일 : 1996년 5월 3일

1. 서 론

지난 30년 동안 과학과 기술 분야에서의 급속한 발전은 수많은 주제들에 대해 방대한 양의 정보가 생성되는 정보화 사회를 탄생시켰다. 원하는 정보에 대한 정확하고 빠른 접근은 정보화 사회를 살아가는 현대인들에게 성공의 여부를 결정짓는 중요한 요소가 되었다. 그러나 대용량의 데이터로부터 주어진 시간내에 원하는 정보를 발견하는 것은 매우 어려운 일이다. 이러한 문제점을 해결하기 위해 1960년대 초에 컴퓨터를 이용하여 지정된 정보를 검색하는 정보검색(Information Retrieval)이라는 연구 분야가 확립되었다.

지금까지 컴퓨터를 이용하여 대용량의 문서를 효율적으로 검색할 수 있는 정보검색시스템에 관한 많은 연구가 이루어져 왔다. 정보검색시스템의 사용은 원하는 정보에 대한 접근을 용이하게 함으로써 여러 분야에 있어서 정보 수집에 대한 시간과 노력을 단축시키게 된다. 특히 관리할 정보의 양이 기하급수적으로 증가하고 있는 정보화 시대인 오늘날에는 효율적인 정보검색시스템에 대한 요구는 더욱 절실하다.

부울검색시스템은 시스템의 구현이 용이하고 구현된 시스템이 짧은 검색 시간을 제공하기 때문에, 오늘날 정보검색 분야에서 가장 널리 사용되고 있다. 부울검색시스템(Boolean Retrieval System)에서 문서는 색인어들의 집합으로 표현되고, 질의는 색인어들을 부울연산자 AND, OR, NOT으로 연결한 부울 수식이며, 검색되는 문서는 질의로서 주어진 부울 수식을 만족하는 문서들이다.

부울검색시스템에서 일반적으로 사용되는 부울 질의 연산 과정은 다음과 같다. 먼저 부울 질의를 구성하는 각각의 색인어에 대하여 그 색인어를 포함하는 문서집합들을 선택한 후, 이 집합들에 AND, OR, NOT에 해당하는 집합연산을 수행함으로써 최종적으로 검색될 문서집합을 생성한다. 이때 두 개 이상의 연산자를 포함하는 부울 질의는 가장 안쪽에 위치하는 절부터 순환적으로 계산된다. 그러나 이러한 일반적인 부울 질의 연산 방법은 많은 경우에 검색 속도를 감소시키는 요인들을 지니고 있다.

본 논문에서는 부울검색시스템의 성능 개선을 위한 방법으로 부울 질의를 보다 효율적으로 연산하기 위한 다음과 같은 4가지 부울 질의 최적화 방법을 제시한다 : (ⅰ) 프로그래밍 언어에서 논리식을 효율적으로 계산하기 위해 사용되는 단거리계산법(Short-Circuit Evaluation)을 적용한다. (ⅱ) 연산에 참여하는 문서집합의 크기가 현저한 차이를 보일 경우, 문서식별자를 비교하여 집합연산을 수행하는 대신에 색인어가 문서에 포함되어 있는지를 확인한다. (ⅲ) 분배법칙을 이용하여 부울 질의를 간소화함으로써, 연산의 수를 감소시킨다. (ⅳ) 질의가 반복되는 식을 포함할 경우, 그 식에 대한 중복연산을 수행하지 않는다.

본 연구는 연구개발정보센터에서 개발중인 정보검색시스템 KRISTAL-Ⅱ의 부울 질의 처리기의 성능을 개선하기 위한 목적으로 수행되었다. 본 논문의 구성은 2장에서 KRISTAL-Ⅱ 시스템에 대하여 설명하고, 3장부터 6장에 걸쳐 위에서 언급한 4가지 최적화 방법에 대하여 기술한다. 7장에서 4가지

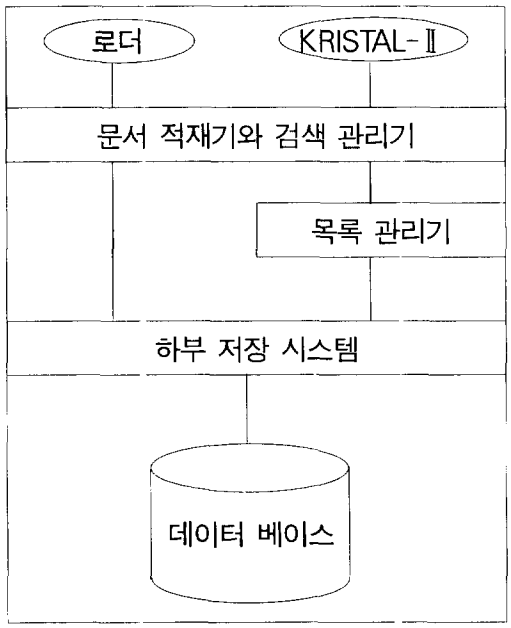
최적화 방법을 통합한 알고리즘을 제시하고, 그 적용 예를 보여준다.

2. KRISTAL-Ⅱ 시스템

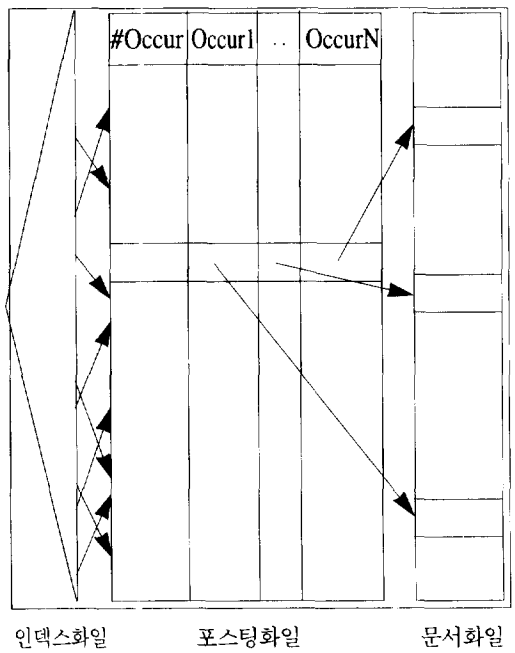
KRISTAL-Ⅱ 시스템은 저장된 문서에 대한 효율적인 접근을 지원하는 하부저장시스템, 데이터베이스에 대한 정보를 관리하는 목록관리기, UNIX 화일로부터 문서를 읽어서 데이터베이스를 구축하는 문서적재기, 질의를 연산하여 문서를 검색하는 검색관리기 등으로 구성되어 있다. 〈그림 1〉은 KRISTAL-Ⅱ 시스템의 전체 구성도이다.

하부저장시스템은 KRISTAL-Ⅱ 시스템의 최하위에 위치하며, 문서와 색인의 생성 및 접근에 대한 기본적인 함수들을 제공한다. 색인 작업은 문서식별자들을 포함하고 있는 포스팅 화일과 포스팅 화일에 대한 접근 경로를 제공하는 인덱스 화일의 생성으로 이루어진다. 인덱스 화일은 Prefix B+ 트리 구조로 구성되며, 포스팅 화일의 레코드는 문서에서의 해당 색인어에 대한 위치 정보를 포함하고 있다. 〈그림 2〉는 문서와 색인을 위한 화일들의 구조를 보여준다.

정보검색시스템의 사용시, 사용자는 질의어를 사용하여 데이터베이스로부터 정보를 추출하게 되므로, 정보검색시스템이 제공하는 질의어의 형태는 정보검색시스템 사용의 편리성을 결정한다. National Information Standards Organization(NISO)에서 부울 질의에 대한 표준화 작업을 수행하였으며,

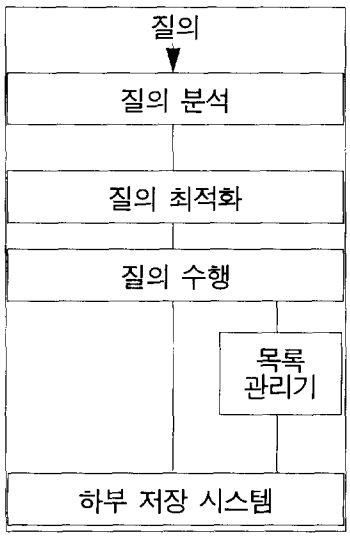


〈그림 1〉 KRISTAL-Ⅱ 시스템 구성도



〈그림 2〉 문서와 색인을 위한 화일 구조

KRISTAL-Ⅱ 시스템의 검색관리기는 NISO에서 제정한 질의어를 기반으로 하여 구현되었다. 검색관리기는 입력된 질의를 분석하여 파싱트리를 생성하고, 질의최적화 단계에서 결정된 접근계획에 따라 하부구조의 기능을 호출함으로써 질의연산을 수행한다. 검색관리기의 전체 구성은 〈그림 3〉과 같다.



〈그림 3〉 검색관리기 구성도

질의최적화기는 데이터베이스의 상태에 관한 정보를 이용하여 가장 효율적인 접근계획을 수립하고, 질의연산기는 결정된 접근계획에 따라 질의를 연산한다. 검색관리기가 수행하는 질의처리 과정은 다음과 같다. Rop를 연산자 op의 오른쪽(R) 피연산자라 하고, Lop를 연산자 op의 왼쪽(L) 피연산자라 하며 |Rop|와 |Lop|를 각 피연산자로부터 생성될 수 있는 최대 문서수라 하자.

(1) 입력 질의에 대한 파싱트리를 생성한다. 파싱트리는 연산자 또는 색인어 노드로 구성되는 이진 나무구조이다.

(2) 색인어 노드에 대하여 그 색인어를 포함하는 문서수를 구한다.

(3) 각 노드마다 검색될 수 있는 최대 문서수를 다음과 같이 계산한다.

AND 연산자 : min(|Rand|, |Land|)

OR 연산자 : |Ror| + |Lor|

W, N 연산자 : min(|Rw(n)|, |Lw(n)|)

NOT 연산자 : |Lnot|

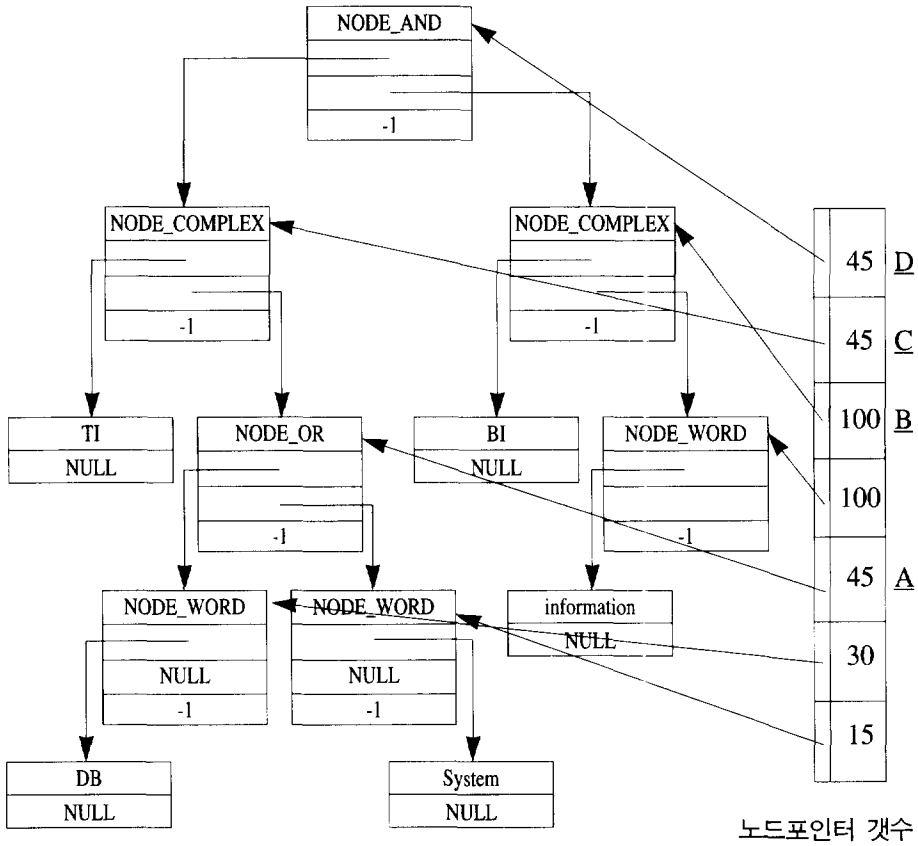
(4) 질의최적화 및 질의연산 수행

〈그림 4〉는 다음과 같은 입력 질의에 대하여 생성된 파싱트리와 각 노드가 가질 수 있는 최대 문서수이다.

입력질의 : Find(TI:DB W4 System) AND (Information)

OR 노드의 경우(레이블 A경우)는 |Lor|이 30이고 |Ror|이 15이므로 OR 노드의 최대문서수는 45이다. Complex 노드는 자식노드로 단어를 포함하는 것과 연산자를 포함하는 두 가지 경우가 있다

(레이블 B와 레이블 C경우에 각각 해당됨). 레이블 B는 Complex 노드가 단어를 포함하므로 이 노드의 최대문서수는 그 단어(Information)를 포함하는 문서수인 100이다. 레이블 C의 Complex 노드는 연산자를 자식노드로 가지고 있기 때문에 연산자의 종류에 따라서 최대문서수가 결정된다. AND 노드의 경우(레이블 D경우) |Land|(45)와 |Rand|(100)를 비교하여 작은 것을 그 노드의 문서수로 정하므로 45가 이 노드의 최대 문서수이다.

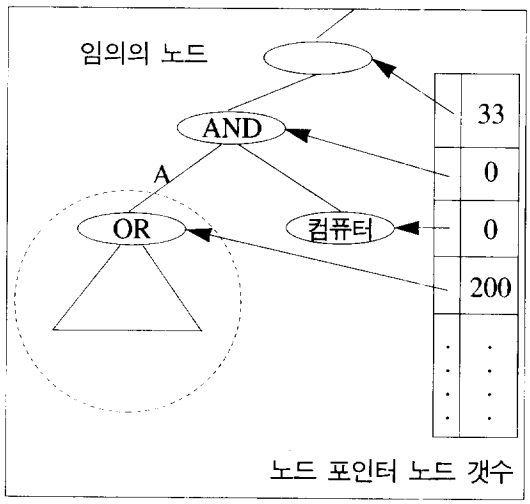


〈그림 4〉 파싱트리 및 각 노드의 최대문서수 생성 예

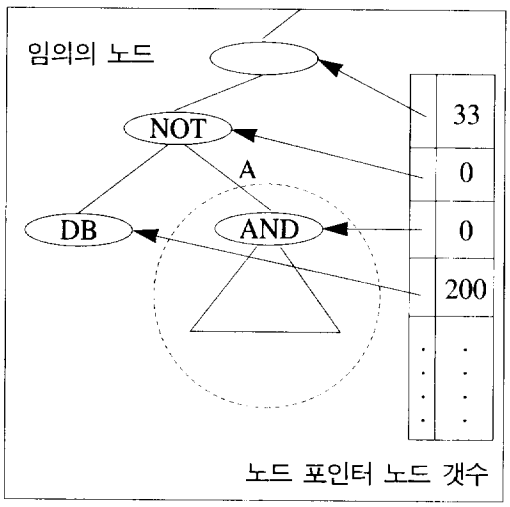
3. 단거리계산 방법

NISO에서 제정된 부울 질의어를 기본으로 하여 구현된 KRISTAL-Ⅱ 시스템은 사용자로부터 입력된 질의를 이진 나무구조로 변환하여, 이를 상향식 방식으로 연산하는 방법을 사용하고 있다. 이때 연산자의 특성 및 피연산자의 값에 따라 불필요한 연산을 피할 수 있다. 예를 들어 두개의 색인어를 피연산자로 갖는 AND 연산자의 연산시, 한쪽 색인어를 포함하는 문서가 없을 경우를 가정하자. 이때 다른 한쪽의 색인어를 포함하는 문서가 매우 많다고 할지라도 두 색인어를 공통적으로 포함하는 문서가 존재하지 않기 때문에, 파싱트리에서 이러한 AND 연산자의 밑에 있는 연산자에 대해서는 더 이상 연산을 수행할 필요가 없다. 이러한 특성을 지닌 연산자로서는 AND, NOT, W, N 등이 있다. 단, AND, N, W 연산자의 경우는 피연산자의 문서수가 어느 쪽이 0이든지 관계 없이 이 방법을 적용할 수 있으나 NOT 연산자는 왼쪽 피연산자의 문서수가 0일 경우에만 적용할 수 있다.

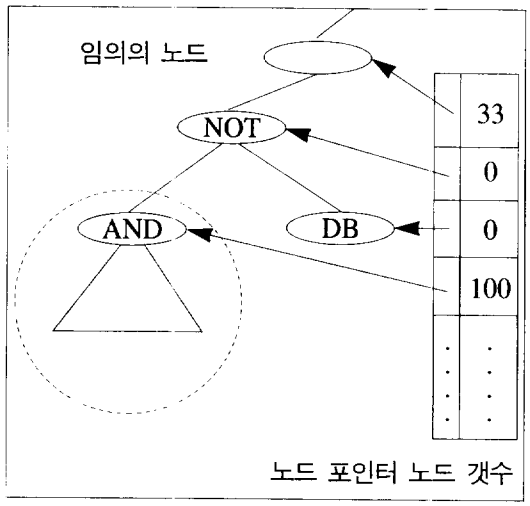
이러한 연산자들에 대한 단거리계산 최적화 수행과정을 살펴보면 다음과 같다 ; 생성된 파싱트리의 각 색인어에 대해 그 색인어를 포함하고 있는 문서수를 인덱스 화일에서 구한 다음, 각 노드마다 가질 수 있는 최대 문서수를 계산한다. 각 노드마다 검색될 수 있는 최대 문서수는 연산자의 성질에 따라 다르다(2.2 참조). 만약 검색 가능한 최대문서수가 0이고 해당되는 연산자가 위의 4가지 중의 하나이면 단거리계산 방법이 적용될 수 있으므로 해당



〈그림 5〉 AND 노드에서 단거리계산 방법 적용



〈그림 6〉 NOT 연산자에서 단거리계산 방법이 적용되는 경우



〈그림 7〉 NOT 연산자에서 단거리계산 방법이 적용되지 않는 경우예

노드를 자식노드로 가지고 있는 노드의 포인터를 NULL 값으로 바꾸어서 하부트리를 삭제하도록 한다.

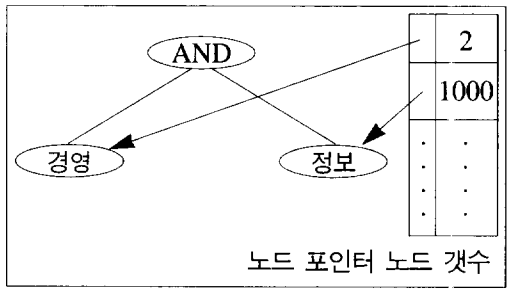
예를 들어, ... (... OR ... AND 컴퓨터 ...)와 같은 질의에 대한 파싱트리와 검색 가능한 최대문서수가 〈그림 5와 같다고 가정하자. |컴퓨터and|는 0이고 적용되는 연산자가 AND이므로 AND 연산자를 자식노드로 포함하고 있는 노드 B에 대한 포인터를 NULL 값으로 바꾸어 준다. 그러면 AND 노드 아래에 있는 연산자에 대한 집합연산만큼 수행 시간을 줄일 수 있다 (〈그림 5〉에서 빗금으로 둘러싸인 부분에 속해 있는 연산자). NOT 연산자의 경우 단거리계산 방법이 적용되는 경우와 적용 안되는 경우가 그림 6과 7에 각각 나타나 있다.

4. 색인어 출현 빈도수의 차에 의한 최적화

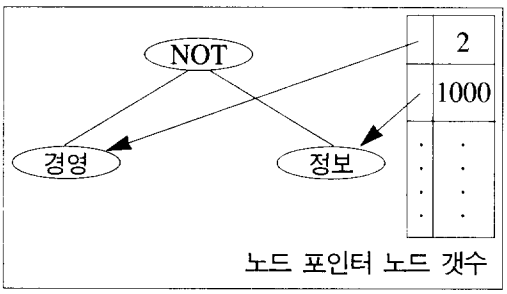
일반적으로 데이터베이스 내에서 색인어의 출현빈도는 일정치 않으며, 비전문적인 색인어의 출현빈도가 전문적인 색인어의 출현빈도보다 크다. 사용자는 널리 사용되는 색인어부터 시작하여 좀더 구체적인 전문적인 단어를 이용함으로써, 원하는 문서 혹은 정보를 검색하기 위한 대상 범위를 좁혀 나간다. 이러한 특성에 근거하여 연산에 참여하는 두 색인어의 출현빈도차가 클 경우 해당 연산자에 대한 집합연산을 수행하는 대신에 문서검색을 통해 수행시간을 줄일 수 있다. 이러한 수행 최적화의 대상이 되는 연산자로 AND, NOT, W, N이 있다.

임의의 연산자 OP에 대한 집합연산은 Rop를 포함하는 문서에 대한 문서식별자와 Lop를 포함하는 문서에 대한 문서식별자를 비교해야 하므로 |Rop| * |Lop| 만큼의 비교연산이 필요하다. 그러나 ||Rop| - |Lop|| 〉 C (C의 값은 데이터를 분석하여 시스템에서 지정한다.) 경우, (ⅰ) |Rop| 〈 |Lop|이면, Rop를 포함하는 문서가 Lop를 포함하고 있는지를 검색하고, (ⅱ) |Rop| 〉 |Lop|이면, Lop를 포함하는 문서가 Rop를 포함하고 있는지를 검색함으로써 수행시간을 축소시킬 수 있다.

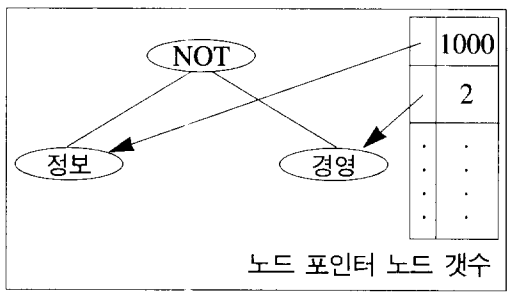
예를 들어, ... (경영 AND 정보) ... 라는 부분을 포함하는 질의에 대하여, OP = AND', RAND = 정보, LAND = 경영이며, 이때 |RAND| = 1000, |LAND| = 2이라 가정하자. 이 경우에 집합비교연산을 수행하려면 RAND와 LAND를 포함하는 문서에 대한 문서식별자를 메모리에 적재한 후 2000번의 비교연산이 필요하다. 그러나 만약 LAND를 포함하고 있는 문서의 길이가 짧을 경우 그 문서 속에 RAND가 속해 있는지를 검사하면 효율적일 수 있다. 이때 걸리는 시간은 |LAND| * (LAND를 포함하고 있는 문서의 평균단어수)이다. 따라서 LAND를 포함하고 있는 문서의 평균 단어수가 1000보다 작으면 이 방법은 집합연산보다 많이 걸리지 않는다. 그러므로 ||Rop|-|Lop|| 〉 C에서의 C값은 문서의 평균단어수와 min(|Rop|, |Lop|)의 곱으로 계산할 수 있다. 단, NOT 연산자의 경우는 연산자의 특성에 의해서 |Rop| 〉 |Lop|일 경우만 적용된다.



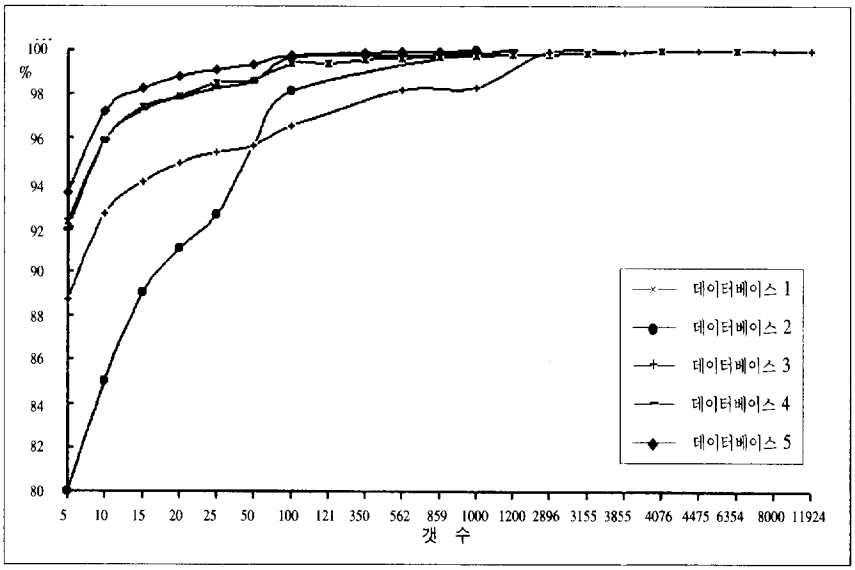
〈그림 8〉 AND 노드에서 빈도수 차에 의한 최적화 적용 예



〈그림 9〉 NOT 노드에서 빈도수 차에 의한 최적화 적용의 예



〈그림 10〉 NOT 노드에서 빈도수 차에 의한 최적화가 적용되지 않는 경우



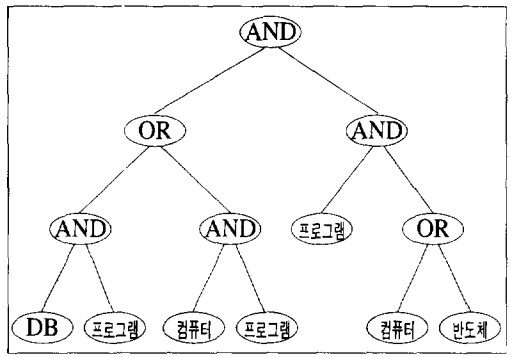
〈그림 11〉 데이터베이스의 색인어 출현빈도 분포도

〈그림 11〉은 현재 사용중인 20개의 데이터베이스 중에서 대표적인 것의 색인어 출현빈도를 백분율로 표시한 그림이다. 위의 표에서 알 수 있듯이 약 90% 이상의 단어의 출현빈도가 5 이하이며, 나머지 약 10% 내에 출현빈도가 높은 단어가 있다는 것을 알 수 있다. 이러한 분석은 피연산자들의 출현빈도에 큰 차이가 발생할 가능성이 높음을 암시하며, 따라서 출현빈도차를 이용한 최적화 방법이 검색 효율 개선에 적용될 수 있다.

5 분배 법칙을 이용한 최적화

분배 법칙의 적용에 의한 최적화는 집합연산의 기본원칙인 분배 법칙을 이용하여 수행시간을 줄이는 데 있다. 부울수식(A AND B) OR C에 분배 법칙을 적용하면 부울수식(A OR C) AND (B OR C)을 얻을 수 있다. 앞의 두 부울수식 중 뒤의 식이 앞의 식보다 연산자가 많기 때문에 해당되는 연산자에 대한 문서 검색을 위하여 비교해야 하는 횟수가 많다. 그러나 부울검색에 있어서 두 개의 식의 계산 결과가 같기 때문에, 주어진 질의가 이미 분배법칙이 적용된 질의라면 이를 자동적으로 변환하여 비교 횟수를 줄이는 것이 필요하다. 예를 들어, 다음과 같은 질의가 입력되었을 때 〈그림 12〉과 같은 파싱트리가 만들어진다.

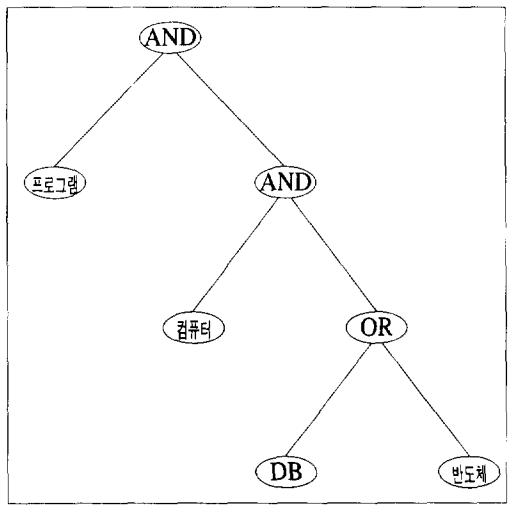
Find ((DB AND 프로그램) OR (컴퓨터 AND 프로그램)) AND (프로그램 AND (컴퓨터 OR 반도체))



〈그림 12〉 분배 법칙이 적용되기 전의 질의 파싱트리

위의 질의를 분배법칙이 적용되기 전으로 환원하여 원래의 식으로 만들어 보면 다음과 같이 연산자의 수가 줄어드는 것을 알 수 있다. 따라서 〈그림 13〉과 같이 6개의 연산자를 포함하는 질의에서 3개의 연산자를 포함하는 파싱트리가 생성된다.

Find (프로그램 AND (컴퓨터 OR (반도체 AND DB)))



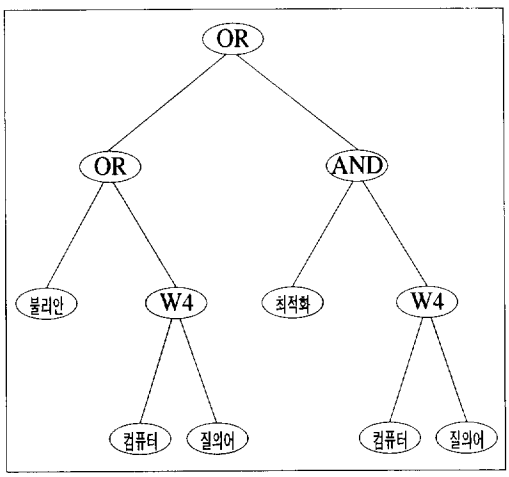
〈그림 13〉 분배 법칙 적용 후 파싱트리 예

이러한 방법의 적용은 사용자의 입장에서 보았을 때 분배법칙이 적용된 질의가 분배법칙이 적용되지 않은 질의보다 의미적으로 사용하기가 편리하기 때문에 질의 최적화에 매우 효과적이라 할 수 있다.

6 반복되는 식에 대한 최적화

입력 질의 속에 같은 식이 반복되어 들어왔을 때 이를 중복하여 계산하지 않고 1번만 수행해서 수행 시간을 줄이는 방법이다. 이 방법은 프로그래밍 혹은 데이터베이스 등에서는 물론이고 일반적으로 널리 사용되는 방법으로서 부울 질의에 적용하여도 마찬가지로 검색 시간을 줄일 수 있다. 예를 들어, 다음과 같은 질의가 입력되었다고 가정하자.

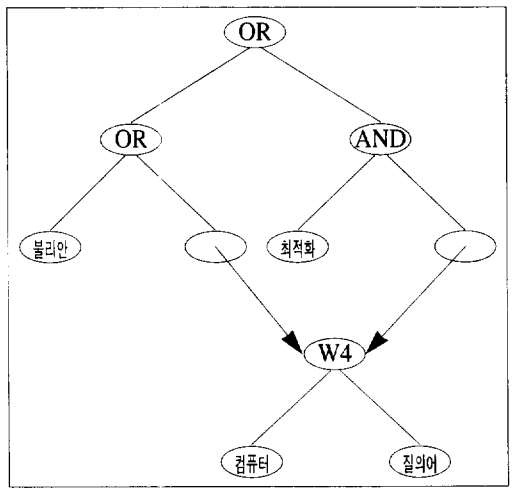
Find (불리안 OR (컴퓨터 W 4 질의어)) OR ((컴퓨터 W 4 질의어) AND 최적화)



〈그림 14〉 반복되는 식에 대한 최적화 적용 예

위의 식에 대하여 파싱트리를 만들어 보면 〈그림 14〉과 같이 된다. 위의 그림에서 "컴퓨터 W 4 질의어" 연산은 트리의 왼쪽에도 있고 오른쪽에도 있다. 이를 하향식 방법에 의해서 계산하면 "컴퓨터 W 4 질의어"에 대하여 두 번 계산하여야 한다. 그러나 "컴퓨터 W 4 질의어"에 대한 파싱트리를 따로 만들고 이 트리에 대한 포인터를 연결시키면 중복되는 식이 한번만 수행하게 된다.

이 방법은 입력 질의에서 중복되는 식이 많이 존재하면 할수록 위의 최적화의 성능은 좋아진다.



〈그림 15〉 반복되는 식에 대한 최적화 적용 후

7. 전체 최적화 알고리즘

전체 최적화 알고리즘은 앞절에서 언급한 4가지 방법을 적용하는 것으로서 다음과 같은 순서로 수행된다. 처음에 사용자로부터 질의를 입력받아서 그것이 분배법칙이 적용된 질

의일 경우 원래의 식으로 환원한다. 환원된 식이 만약 하나 이상의 같은 식을 포함하고 있다면 이를 포인터를 이용하여 한번만 계산할 수 있도록 질의를 재구성한 다음 재구성된 질의에 대해 파싱트리를 생성한다. 생성된 파싱트리를 가지고 그 파싱트리 안에 단거리계산방법이 적용될 수 있는가를 확인하여 만약 적용할 수 있으면 적용된 연산자에 대한 서브트리를 삭제한 파싱트리를 리턴한다. 리턴된 파싱트리에 대한 상향식으로 트리를 방문하면서 빈도수의 차를 이용하여 문서에서 직접 단어를 검색할 수 있는지를 결정, 수행한다.

```
최적화 방법 적용 순서 ;
begin
input_query = read(query) ; /
  * 사용자로부터 질의를 입력받는다 */
undist_query = apply_dist_law(input_
query) ; /* 분배법칙을 원래의 식으로 환
원 */
same_expr_removed_query = remove_
same_expr(undist_query) ; /
  * 같은 식을 포함하는 질의에 대한 최적
  화 */
parse_tree = generate_parse_tree(same_
expr_removed_query) ; /
  * 파싱트리 생성 */
apply_sec_rule(parse_tree) ; /
  * 생성된 파싱트리에 단거리 계산 방법
  적용 */
apply_diff_rule(parse_tree) ; /
  * 빈도차를 이용한 최적화 방법 적용 */
end ;
```

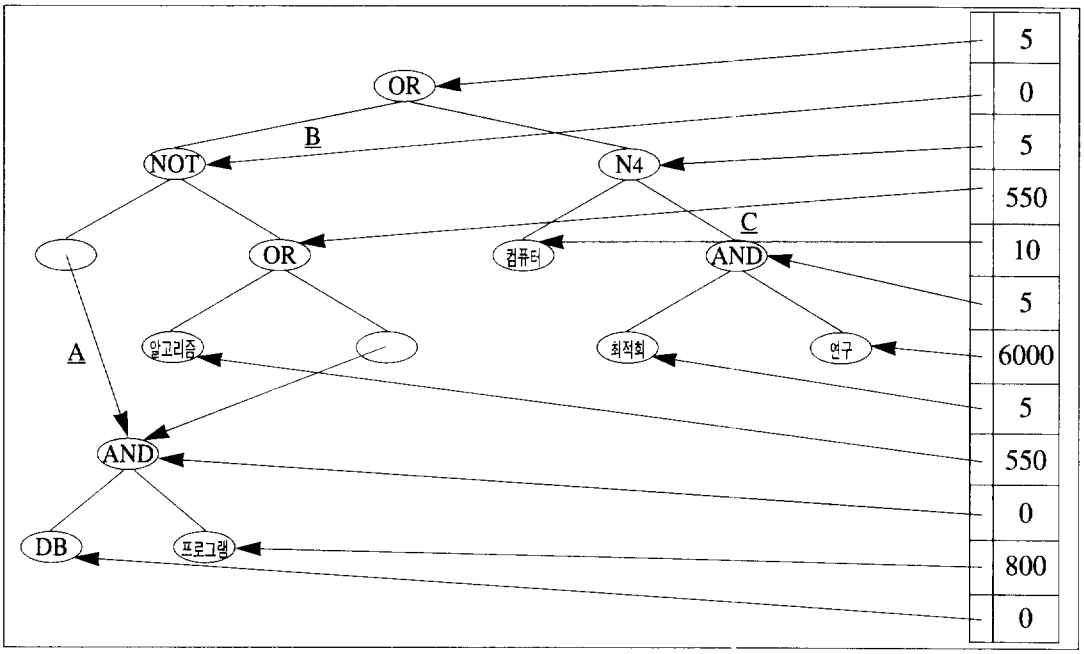
예를 들어, 다음과 같은 질의가 입력되었을 경우 어떻게 탐색공간을 줄여서 최적화를 수행해 나가는지를 살펴보자

입력질의 :
Find (DB AND 프로그램) NOT ((알고리즘 OR DB) AND (알고리즘 OR 프로그램)) OR (컴퓨터) N4 (최적화 AND 연구)

먼저, 분배법칙이 적용되었는지를 확인하여, 이를 적용되기 전의 식으로 변형시킨다.

분배법칙 최적화방법 수행후의 질의 :
Find (DB AND 프로그램) NOT (알고리즘 OR (DB AND 프로그램)) OR (컴퓨터) N4 최적화 AND 연구)

〈그림 16〉은 분배법칙 최적화방법 수행 후의 질의에 대해 생성된 파싱트리이다. A부분은 반복되는 식에 대한 최적화방법을 적용하여 만들어진 노드이다. 그러나 A노드의 왼쪽 단어 "DB"의 갯수가 0이기 때문에 NOT 노드이 특성상 왼쪽 노드의 값이 0이 된다. 따라서 OR 노드의 왼쪽 자식노드를 계산하지 않아도 된다. 그리고 C부분에서는 단어가 속해 있는 문서를 직접 메모리에 가지고 와서 그 문서 안에 "연구"라는 단어가 있는지를 조사하게 된다.



〈그림 16〉 전체 알고리즘을 적용한 예

8. 결 론

본 연구에서는 정보 검색 시스템에서 부울 질의를 사용할 경우, 이를 최적화할 수 있는 4가지 방법을 모색해 보았다. 프로그래밍 언어에서 논리식의 계산에 사용되는 단거리계산 방법과, AND, NOT과 같은 특정 연산자를 효율적으로 연산하기 위하여 색인어 출현 빈도의 차이를 이용하는 방법, 분배법칙이 적용된 질의에 대하여 중복 연산을 회피하는 방법에 관한 것이다. 또한 위의 4가지 방법들을 UNIX환경에서 개발된 KRISTAL-Ⅱ 시스템에 구현하여, 제시된 방법들이 특정 경우에 검색 속도를 향상시킬 수 있음을 검증하였다.

본 연구를 통하여 데이터의 특성 및 연산자에 따라서 검색시간을 줄일 수 있는 방법에 관한 연구가 더욱 필요함을 발견하였으며 특히 한 연산자에 대한 색인어의 빈도수가 비슷한 경우 이를 최적화하는 기술에 관한 것이 향후 해결해야 할 과제라 하겠다.

참 고 문 헌

Baeza-Yates, R. 1989. "Expected Behavior of B+-Trees under Random insertions." Scta informative, 26(5), 439-72. as Research Research Report CS-86-87, University of Waterloo, 1986.

Bayer, R., and K. Unterauef. "Prefix B-Tree." ACM transactions on database Systems, 2(1), 11-26,1977.

Gonner, G, "Efficient Searching of Text and Pictures (extended abstract)" Technical Report OED-88-02, Center for the New OED., University of Waterloo, 1988.

Sebesta, R.W. Concepts Of Programming Languages. University of Colorado, 1990.

Sparck-Jones, K. Information Retrieval Experiment. London : Butterworths, 1981.

Sedgewick, R. Algorithms in C. Reading, Mass : Addisom-Wesley, 1990.

Van Rijsbergen, C. J. Information Retrieval. London : Butterwords, 1979.

Witten, I. H., Slistari, M., and Bell, T. C. Managing Gigabytes-Compressing and Indexing Documents and Images, Van Nostrand Reinghold, New York, 1994.