# 중첩 집합과 포함 질의를 이용한 고급 검색 시스템

염준호, 최규태, 이기훈[1)]
광운대학교 컴퓨터공학과
e-mail: kihoonlee@kw.ac.kr

# Advanced Search System Using Nested Sets and Containment Queries

Joon-Ho Yeom, Kyu-Tae Choi, and Ki-Hoon Lee
Dept. of Computer Engineering, Kwangwoon University

요 약

포함 질의는 자주 사용되는 기본적인 질의 패턴 중에 하나이고 많은 연구가 진행되어 왔다. 하지만 중첩 집합에 대한 포함 질의는 아직 많은 연구가 이루어지지 않고 있다. 본 연구에서는 중첩 집합에 대한 포함 질의를 이용하여 사용자에게 더 편리하고 정교한 검색 기능을 제공하는 검색 시스템을 제안한다. 그리고 중첩 집합에 대한 역 색인을 $B^+$-Tree와 해시 테이블로 각각 구현하였을 때의 성능을 비교 실험한다.

## 1. 서론

과학, 산업 분야에는 중첩 집합으로 모델링 될 수 있는 많은 자료들이 존재한다. 예를 들어 유전자계보도나 JSON 데이터 등 많은 것들이 중첩 집합으로 표현되고 있다. 하지만 중첩 집합을 효율적으로 처리할 수 있는 방법에 대한 연구는 많이 이루어지지 않고 있다[1].

본 논문에서는 중첩 집합에 대한 포함 질의를 이용한 고급 검색 시스템을 제안한다. 제 2절에서는 중첩 집합, 중첩 집합에 대한 포함 질의의 정의, 질의 처리를 위한 역색인(inverted-index)의 구조를 소개한다. 제 3절에서는 고급 검색 시스템을 제안한다. 제 4절에서는 역 색인을 $B^+$-Tree와 해시 테이블로 구현했을 때 각 구조의 효율성을 실험을 통해 비교한다. 마지막으로 제 5절에서는 결론을 내린다.

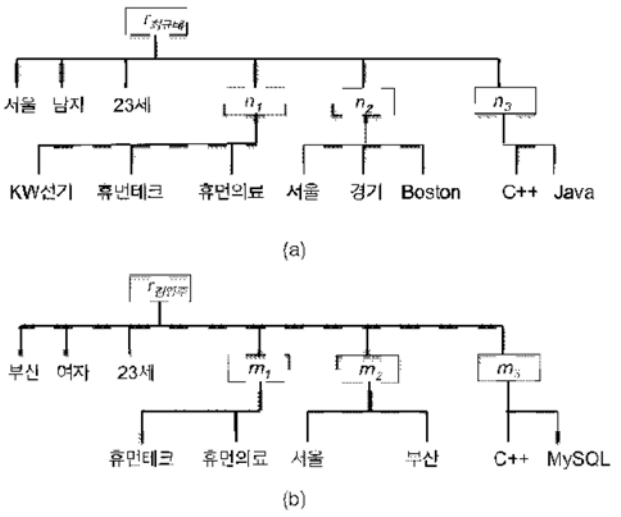
## 2. 예비 지식

### 2.1 중첩 집합

중첩 집합은 집합을 원소로 갖는 집합을 의미한다. 표 1은 중첩 집합으로 이루어진 데이터의 예를 보여주고 있다. key 컬럼은 각 중첩 집합을 식별하기 위해 사용되고 value 컬럼은 중첩 집합을 저장하고 있다. 중첩 집합은 그림 1과 같이 트리 구조로 표현할 수 있다. 이때 각 집합에는 임의의 정수 id를 부여하며 그림 1에서는 id를 $r_{최규태}$, $n_1$, $n_2$, $n_3$로 표기하였다. 최규태에 대한 중첩 집합의 첫 번째 레벨에는 거주지와 성별, 나이와 같은 개인 정보와 중첩된 집합 $n_1$, $n_2$, $n_3$가 있다. $n_1$, $n_2$, $n_3$ 집합은 각각 근무 경력, 근무지의 도시, 보유 기술을 의미한다.

<표 1> 중첩 집합의 예

| key | value |
|---|---|
| 최규태 | {서울, 남자, 23세, {KW전기, 휴먼테크, 휴먼의료}<br>, {서울, 경기, Boston}<br>, {C++, Java}} |
| 김영주 | {부산, 여자, 23세, {휴먼테크, 휴먼의료}<br>, {서울, 부산}<br>, {C++, MySQL}} |




(그림 1) 중첩 집합의 트리 표현

1) 교신저자

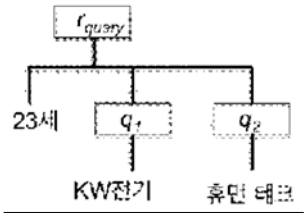
### 2.2 중첩 집합에 대한 포함 질의

중첩 집합에 대한 포함은 [1]의 subtree homomorphism에 따라 정의 한다.

**정의 1.** 중첩 질의 트리 $q$와 $s$가 있을 때, 다음을 만족하면 $q$가 $s$에 포함 된다고 하고 $q$ $_{\text{nested-set}}$ $s$와 같이 표기한다.

$q$에 있는 부모와 자식을 연결하는 간선은 반드시 $s$에 있는 부모와 자식을 연결하는 간선에 대응 되어야 하고 $q$의 비단말 정점과 $s$의 비단말 정점 사이의 대응이 단사 함수일 필요는 없다.

그림 2는 그림 1의 $r_{최규태}$ 트리에 포함되는 트리의 예이다. $r_{query}$ 트리에서 $r_{최규태}$ 트리로 정점을 대응시킬 때 23세 노드는 23세로, 루트는 루트로, $q_1$과 $q_2$를 $n_1$으로 대응 시키면 $r_{query}$ 트리의 모든 부모-자식 연결 관계가 보존이 되므로 포함의 정의에 부합하는 트리이다. 단사 함수일 필요는 없다는 조건에 의해 $q_1$과 $q_2$는 $n_1$에 함께 대응시킬 수 있다.




(그림 2) $r_{query} \subseteq_{nested\text{-}set} r_{최규태}$

### 2.3 역 색인 구조

중첩 집합에 대한 포함질의를 처리하기 위해서는 역 색인 구조가 필요하다. Ibrahim[1]에 의해 제시된 역색인 구조는 단말 값을 key로 갖고 단말 값의 위치에 관한 정보를 역 리스트(inverted list)로 저장한다. 중첩 집합을 위한 역 색인에서는 단말이 나타난 위치뿐만 아니라 중첩 집합의 구조에 관한 정보도 저장해야한다. 부모의 id와 단말이 아닌 형제의 id들을 쌍으로 저장하여 구조에 관한 정보를 저장한다. 식 1은 단말 값 $a$에 대한 역 리스트를 나타낸 식이다. $p_i(1 \le i \le n)$는 $a$ 단말의 부모의 id를 의미하고 $C_i$는 $a$의 단말이 아닌 형제들의 id의 집합을 의미한다.

그림 1의 '서울'을 식 1의 형태로 표현하면 <($r_{최규태}$, <$n_1$, $n_2$, $n_3$>), ($n_2$, <>), ($m_2$, <>)>로 표현할 수 있다. 그림 1(a)의 첫 번째 레벨에서 서울은 부모로 $r_{최규태}$를 갖고 단말이 아닌 형제로 $n_1$, $n_2$, $n_3$를 가지므로 ($r_{최규태}$, <$n_1$, $n_2$, $n_3$>)로 표현한다. 두 번째 레벨에 있는 서울로부터 ($n_2$, <>), 그림 1(b)로부터 ($m_2$, <>)를 가지므로 '서울'의 역 리스트를 <($r_{최규태}$, <$n_1$, $n_2$, $n_3$>), ($n_2$, <>), ($m_2$, <>)>로 표기한다. 이때 <>은 공집합을 의미한다. 표 2는 표 1의 중첩 집합을 역 색인 구조로 표현한 것이다.

$$_F(a) = < (p_i, C_1), \ldots, (p_n, C_n) > \qquad (1)$$

우리는 식 2와 같이 정의되는 '역 색인 조인'[1]을 통해 주어진 질의를 포함하는 중첩 집합을 찾을 수 있다. 식 2의 $L$과 $L'$은 역 리스트를 의미한다.

$$L \bowtie_{IF} L' = \{(p, C') \mid (p, C) \in L \wedge (p', C') \in L' \wedge p' \in C\} \qquad (2)$$

'서울'에 거주하고 '휴먼테크' 근무경력을 가진 사람을 찾기 위해서 '서울'과 '휴먼테크'를 역 색인 조인하면 $n_1$을 매개로 ($r_{최규태}$,<>)를 얻을 수 있다. 역색인 조인의 결과로 질의를 포함하는 트리들의 root id를 얻을 수 있다. 분량 제한으로 인해 상세한 질의 처리 알고리즘[1]은 생략한다.

<표 2> 중첩 집합을 위한 역 색인 모델

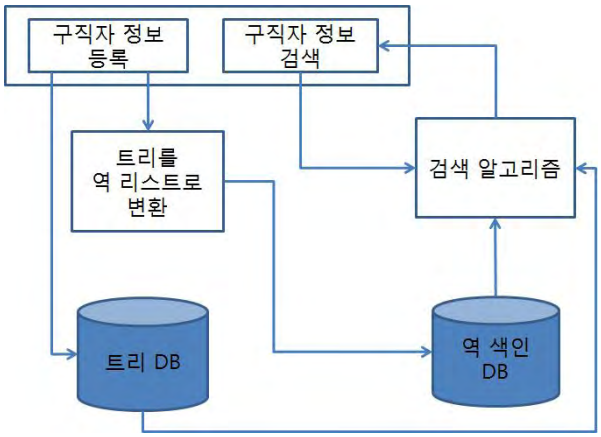
| 단말 값 | 역 리스트(inverted list) |
|---|---|
| 서울 | <($r_{최규태}$, <$n_1$, $n_2$, $n_3$>), ($n_2$, <>), ($m_2$, <>)> |
| 남자 | <($r_{최규태}$, <$n_1$, $n_2$, $n_3$>)> |
| 23세 | <($r_{최규태}$, <$n_1$, $n_2$, $n_3$>), ($r_{김영주}$, <$m_1$, $m_2$, $m_3$>)> |
| 여자 | <($r_{김영주}$, <$m_1$, $m_2$, $m_3$>)> |
| 경기 | <($n_2$, <>)> |
| Boston | <($n_2$, <>)> |
| 부산 | <($r_{김영주}$, <$m_1$, $m_2$, $m_3$>), ($m_2$, <>)> |
| KW전기 | <($n_1$, <>)> |
| 휴먼테크 | <($n_1$, <>), ($m_1$, <>)> |
| 휴먼의료 | <($n_1$, <>), ($m_1$, <>)> |
| C++ | <($n_3$, <>), ($m_3$, <>)> |
| MySQL | <($m_3$, <>)> |
| Java | <($n_3$, <>)> |

## 3. 고급 검색 시스템의 설계 및 구현

구직자의 정보를 중첩 집합 형태로 입력 받아 데이터베이스를 구축하고 질의 또한 중첩 집합 형태로 받아 구직자를 검색할 수 있는 검색 시스템을 만들었다.

### 3.1 검색 시스템의 설계

그림 3은 제안하는 검색 시스템의 전체 구조를 보여준다. 검색시스템에는 구직자 정보 등록과 구직자 정보 검색 두 가지 기능이 있다. 구직자 정보 등록은 입력된 구직자 정보 트리를 중첩 집합을 위한 역 색인의 구조에 알맞은 형태로 쪼개어 저장한다. 검색 결과의 출력을 위해 전체 트리 또한 데이터베이스에 따로 저장하였다. 구직자 정보 검색은 사용자가 원하는 조건 트리를 입력받는다. 질의 처리 알고리즘은 트리를 입력으로 받아 역 색인 조인을 통해 질의를 포함하는 root id들을 결과로 반환한다. 얻어진 root id들로 각 구직자 전체 트리를 얻어와 최종 결과로 출력한다.



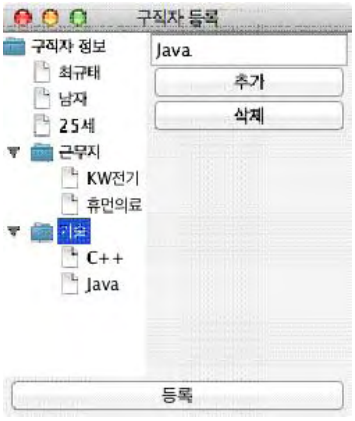

(그림 3) 검색 시스템 흐름도

### 3.2 메인 화면 및 검색 결과 화면

그림 4의 메인 화면에는 구직자를 등록하고 구직자를 검색할 수 있게 하였다. 사용자로부터 구직자 정보를 트리 형태로 입력받는다.


(그림 4) 메인 화면

구직자 등록을 클릭하면 그림 5의 화면을 볼 수 있다. 구직자가 근무했던 회사들과 보유한 기술들을 하위 집합으로 등록하였다. 집합의 중첩은 사용자가 원하는 만큼 할 수 있도록 하였다.




(그림 5) 구직자 등록 화면

구직자 검색을 클릭하면 사용자가 원하는 질의 조건을 트리로 표현하여 검색할 수 있다. 그림 6은 남자이면서 Java를 사용할 수 있는 구직자를 검색하는 화면이다. 그림 7에서는 그림 5에서 저장했던 구직자의 정보를 검색 결과로 출력한다.


(그림 6) 검색 입력 화면

(그림 7) 검색 결과 화면

## 4. 실험 및 평가

Tokyo Cabinet[4]을 저장 시스템으로 사용하고 Ibrahim[1]의 알고리즘을 이용하여 검색을 구현하였다. Tokyo Cabinet이 제공하는 B$^+$-Tree와 해시 테이블을 이용해 역 색인을 구현하였고, B$^+$-Tree와 해시 테이블을 이용 했을 때 성능 차이를 비교해본다.

### 4.1 실험 환경 및 실험 방법

집합을 임의로 발생시켜 역 색인과 질의를 만들어 실험하였고 단말 값들은 1-10,000,000의 범위에서 균등 분포로 발생 시켰다. 집합의 발생은 각 집합에서 재귀적으로 발생 시켰고 집합 발생 시킬 때 중지 확률을 적용하여 확률에 따라 집합 발생을 중지시켜 집합의 중첩도를 조절하였다.

집합을 발생 시킬 때 집합의 중첩도는 높으나 각 집합 안에 단말 값이 적은 경우(deep)와 중첩도는 낮으나 각 집합 안에 단말 값이 많이 포함되는 경우(wide)로 나누어 B$^+$-Tree와 해시 테이블 간의 성능을 비교 했다. deep과 wide의 기준은 [1]에 따라 표 3과 같이 설정하였다.

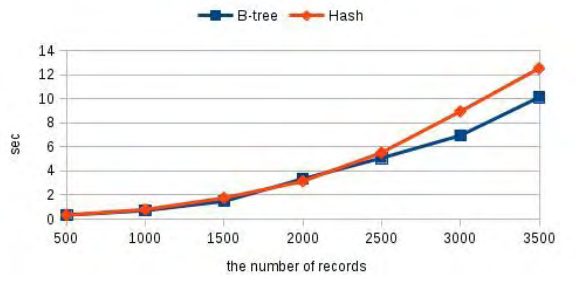
<표 3> 중첩 집합 발생 기준

| parameter | wide | deep |
|---|---|---|
| 한 집합의 단말 값의 최대 갯수 | 12 | 2 |
| 한 집합의 비 단말 값의 최대 갯수 | 6 | 3 |
| 중지 확률 | 0.8 | 0.2 |

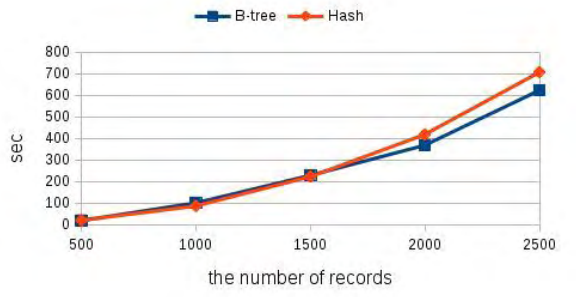
실험은 인텔 i5-4690 CPU (3.5 GHz, quad-core), 4GB 주기억 장치, 삼성 840 PRO SSD 256GB를 가진 리눅스 PC에서 수행하였다.

### 4.2 실험 결과 및 분석

그림 8과 9에서 *x*축은 데이터베이스에 저장되어있는 중첩 집합의 수를 의미하고 *y*축은 질의처리 시간을 의미한다. 데이터가 커짐에 따라 $B^+$-Tree가 해시 테이블 보다 좋은 성능을 보이고 있다. 이는 크기가 커질수록 해시 테이블에 비해 $B^+$-Tree의 디스크 페이지 접근이 적기 때문이다. 주기억 장치에 충분히 올려놓을 수 있는 크기인 경우에는 검색 성능이 크게 차이 나지 않지만, 데이터 개수가 많아지면 성능차이가 점점 커진다.




(그림 8) 중첩도가 낮은 집합(wide)의 성능비교




(그림 9) 중첩도가 높은 집합(deep)의 성능비교

그림 8과 9를 비교해 보았을 때 집합의 중첩도가 높은 경우(deep)는 중첩도가 낮은 경우에 비해 $B^+$-Tree와 해시 테이블 모두 성능이 나쁘다. 트리가 계층 별로 조각나 산란되어 분포하기 때문에 중첩도가 높을수록 랜덤 액세스의 발생 횟수가 증가하기 때문이다.

## 5. 결론

중첩 집합을 이용한 검색 시스템을 통해 사용자는 더 정교한 질의를 편리하게 할 수 있다. 집합의 개념이 적용되는 데이터를 집합 형태 그대로 저장하기 때문에 데이터를 직관적으로 처리할 수 있게 되었다. 하지만 지금의 역색인 구조는 트리를 계층별로 나누어 저장하기 때문에 중첩도가 높아질수록 랜덤 액세스가 증가하는 문제점이 있다. 이 문제를 해결하기 위해서 새로운 역 색인 구조를 제안하거나 랜덤 액세스를 줄일 수 있는 알고리즘의 연구가 필요하다.

## 참고문헌

[1] A.Ibrahim and G.H.I. Fletcher, "Efficient Processing of Containment Queries on Nested Sets," EDBT 2013.
[2] N. Mamoulis, "Efficient Processing of Joins on Set-Valued Attributes," SIGMOD 2003.
[3] M. Terrovitis et al,. "Efficient Answering of Set Containment Queries for Skewed Item Distributions," EDBT 2011.
[4] Tokyo Cabinet, fallabs.com