동적 연산을 위한 집약 이진(CB) 트리

Compact Binary Tree for Dynamic Operations

김 성 완
삼육대학교

Sung Wan Kim
Sahmyook University

요약

정보 검색 분야에서 키 탐색을 빠르게 하기 위한 인덱스 구조로 이진 트라이가 대표적으로 사용된다. CB 트리는 이진 트라이 구조를 구현할 경우 발생하는 저장 공간의 부담을 축소하기 위해 이진 시퀀스를 사용하여 저장한다. 이는 저장 공간 측면에서 상당한 우수성을 보여주나 키의 잦은 삽입 및 삭제 요구가 있을 경우 이진 비트열에 대한 시프트 연산이 요구되는 부담이 있다. 본 논문에서는 완전 이진 트라이 구조를 사용하여 CB 트리를 표현하는 방법을 제시하였다. 저장 공간의 크기가 증가되기는 하지만 키가 삽입되거나 삭제되어도 이진 비트열에 대한 시프트 연산이 필요하지 않은 장점이 있다.

1. 서론

효율적인 정보 검색 시스템 구축을 위해서는 빠른 키 검색 및 동적 환경에 적합한 자료 구조가 요구 된다. 이를 위해 이진 트라이 (binary trie) 구조의 집약적 표현을 위해 CB 트리[2]가 제안되었다. CB 트리는 이진 비트열 형태로 표현되므로 공간 측면에서 우수성이 있으나 키의 삽입 및 삭제 요구가 있을 경우 이진 비트열에 대한 시프트 연산이 필요하다. 본 논문에서는 키의 잦은 삭제 및 삽입이 요구되는 동적인 환경에 효과적으로 대처할 수 있도록 하기 위해 완전 이진 트라이 구조를 사용하여 CB 트리를 표현하는 방법을 제시하였다. 제안 방법은 키가 삽입되거나 삭제되어도 이진 비트열에 대한 시프트 연산이 필요하지 않은 장점이 있다. 2장에서는 기존의 CB 트리의 구성 및 특징에 대해 설명하고 3장에서는 동적 환경에 효과적으로 대처할 수 있는 방안을 서술하며 4장에서 결론을 맺는다.

2. 집약 이진 트리

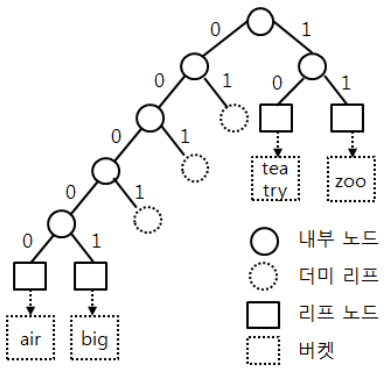
이진 트라이[1]에서 각 키는 이진 시퀀스 형태로 표현된다. 이를 위해 키를 구성하는 각 문자는 이진 코드로 매핑 된다. 예를 들어 문자 'a'는 5비트의 이진수 코드 '00000'으로 표현할 수 있으며 키 'air'는 3개의 이진 코드를 합한 '00000 01000 10001' 형태의 이진 시퀀스로 변환된다. 〈표 1〉은 키 집합 K = {air, big, tea, try zoo}에 대한 이진 시퀀스를 나타낸 것이다.

표 1. 키 집합 K에 대한 이진 시퀀스

키 값	내부 코드	이진 시퀀스
air	0 8 17	00000 01000 10001
big	1 8 6	00001 01000 00110
tea	19 4 0	10011 00100 00000
try	19 17 24	10011 10001 11000
zoo	25 14 14	11001 01110 01110

〈그림 1〉은 위 이진 시퀀스들을 이진 트라이 구조에 모두 삽입한 것이다. 내부 노드는 분기를 위해 사용되며 리프 노드는 키가 저장된 버켓을 참조한다. 이진 트라이를 효과적으로 압축하기 위해 버켓을 참조하지 않는 더미 리프를 추가하며 이는 내부 노드의 개수 보다 리프 노드의 개수가 하나 더 많다는 이진 트리의 특성을 만족하게 된다. 이진 트라이를 구현 할 경우 키 집합이 커지면 저장 공간도 커지는 부담이 있다.

이진 트라이 구현 시 저장 공간의 문제를 해결하고자 제안된 집약 이진 트리 (compact binary tree ; 이하 CB 트리)는 이진 트라이를 이진 비트 시퀀스를 이용한 형태로 표현한다[2, 3].



▶▶ 그림 1. 키 집합 K에 대한 이진 트라이

CB 트리는 Tmap, Lmap, BA로 구성 된다. Tmap은 이진 트라이를 전위 순회 하여 얻는 비트 시퀀스이며 내부 노드와 리프 노드를 방문할 때 각각 '1'과 '0'으로 방문 순서를 표현한다. Lmap은 리프 노드들에 대한 전위 순회하여 얻은 비트 시퀀스로, 더미 리프는 '0'으로 일반 리프 노드는 '1'로 방문 순서를 표현한다. BA는 키를 저장

하는 버켓을 참조한다. 〈그림 1〉의 이진 트라이를 CB 트리로 변환할 경우 〈그림 2〉와 같다. BA를 제외할 경우 Tmap과 Lmap은 각각 13 비트와 7 비트를 사용하여 표현된다.

Tmap	0	0	0	0	0	1	1	1	1	1	0	1	1

Lmap	1	1	0	0	0	1	1

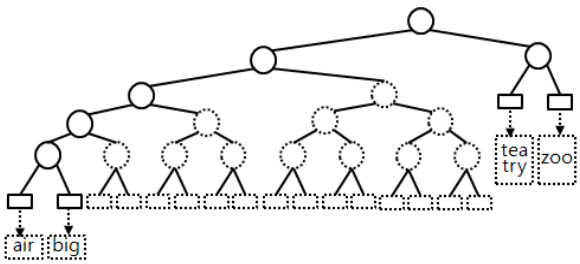
BA	1	버켓 1의 주소
	2	버켓 2의 주소
	3	버켓 3의 주소
	4	버켓 4의 주소

▶▶ 그림 2. CB 트리 1

CB 트리를 사용한 탐색은 키에 대한 이진 시퀀스와 Tmap를 좌측에서 우측방향으로 비트별로 각각 비교하면서 진행된다. 다만 이진 시퀀스의 특정 비트가 0일 경우 좌측 서브 트리 탐색을 생략하고 우측 서브 트리를 탐색해야 하는데 이진 트리의 특성을 이용하여 Tmap에서 '1'로 세팅된 연속된 비트들의 수가 '0'으로 세팅된 연속된 비트들의 수보다 하나 클 때 까지 우측으로 이동하여 우측 서브 트리로 접근할 수 있다. 그러나 키가 새로 삽입되거나 삭제될 경우 Tmap과 Lmap의 비트 시퀀스에 대한 시프트 연산이 필요하게 되어 동적인 환경에서의 부담이 크다[3].

3. 제안 구조

Tmap과 Lmap의 잦은 시프트 연산의 부담을 줄이기 위해 본 논문에서는 이진 시퀀스를 〈그림 3〉과 같이 완전 이진 트라이(complete binray trie) 구조에 표현한다. 더미 노드는 내부 및 리프를 포함하여 25개가 필요하다. 〈그림 4〉는 〈그림 3〉의 완전 이진 트라이를 CB 트리로 변환한 것이다. Tmap과 Lamp은 각각 35 비트와 18 비트를 사용한다. BA 구조는 기존과 동일하다.



▶▶ 그림 3. 완전 이진 트라이

Tmap	0	0	0	0	0	1	1	1	…	1	0	1	1

Lmap	1	1	0	…	0	1	1

▶▶ 그림 4. CB 트리 2

제안된 트리를 이용한 특징은 다음과 같다. 첫째, 키 탐색에서 Tmap의 i번째 비트에 대한 좌측 서브 트리에 대한 탐색 생략을 생략하고 우측 서브 트리를 접근할 경우 $(i + 2^{d-1})$ 째 비트를 계산하여 바로 찾을 수 있다(단, d는 i번째 비트를 루트로 하는 서브 트리의 높이). 즉, 기존의 탐색 방법에서는 '1' 과 '0'으로 세팅된 연속된 비트들의 개수를 비교하여야 하므로 그 개수에 따라 탐색시간의 차이가 발생할 수 있으나 제안 방법에서는 수식을 이용하므로 항상 상수 시간 내에 우측 서브트리를 접근할 수 있다. 예를 들어, 루트 노드(즉, i = 1)에 대한 우측 서브 트리는 $i + 2^{d-1} = 1 + 2^{6-1} = 33$ 번째 비트에 해당한다.

둘째, 키를 새로 삽입하거나 삭제할 경우 이미 내부 및 리프에 더미 노드를 포함하여 Tmap과 Lmap을 표현하였으므로 트리의 높이가 증가되거나 축소되지 않는 이상 비트열의 시프트 연산은 필요하지 않다. 예를 들어 〈그림 1〉에서 키 'big'과 'art'를 연속하여 삭제할 경우 각 키는 해당 버켓의 유일한 키이므로 이를 참조하는 단말 노드를 더미 리프로 각각 변경한다. 두 더미 리프의 부모 노드는 더 이상 유효 자식을 갖지 않으므로 두 더미 리프와 부모 노드를 하나의 더미 리프로 합병해야 한다. 결국 Tmap의 5~7 번째 비트 시퀀스 '011'을 '1'로, Lmap의 1~2번째 비트 시퀀스 '11'을 '0'으로 각각 합병하는 시프트 연산이 요구된다. 그러나 제안 방법은 완전 이진 트라이 구조를 유지하기 때문에 노드의 합병이 발생하지 않으므로 Tmap의 비트 시퀀스 '011'는 변경이 없으며 Lmap의 비트 시퀀스 '11'을 '00'으로 변경하기만 하면 된다.

또한, 새로운 키 삽입 시 기존 방법은 노드 생성에 따른 비트 시퀀스의 시프트 연산이 요구되나 제안 방법은 더미 노드를 표현하기 위한 비트 필드를 소유하고 있으므로 트라이 높이가 늘어나지 않는 이상 해당 비트 시퀀스의 필드 값을 수정하기만 하면 된다.

4. 결론

정보 검색 분야에서 중요하게 사용되는 이진 트라이를 비트 시퀀스 형태의 집약 표현으로 대체하기 위한 CB 트리에 대해 동적인 환경에서도 효과적으로 대처하기 위한 방법을 제시하였다. 향후 저장 공간을 최소화하기 위한 후행 연구가 필요 된다.

■ 참 고 문 헌 ■

[1] 이석호 역, "C로 쓴 자료구조론", 교보문고, 서울, 2008
[2] Jonge, D., et. al., "Two access methods using compact binary trees", IEEE Trans. Software Eng., Vol. 13, No. 7, pp. 799-810. 1987
[3] Masami Shishibori, et. al, "Two improved access methods on compact binary (CB) trees", Information Processing & Management, Vol. 36, No. 3, 2000