효율적인 한글 탐색을 위한 CB 트라이의 재구성

정 규 철*

요 약

본 논문에서는 CB 트라이의 단점을 보완한 RCB 트라이를 제안한다. 먼저 CB 트라이의 경우 처음으로 축약된 구조를 시도하였으나 데이터의 양이 증가함에 따라 트리의 균형을 맞추기 위해 사용되는 더미노드들로 인해 삽입에 상당한 어려움을 가지고 있다. 반면 계층적으로 표현한 HCB 트라이는 map이 오른쪽으로 증가하는 것을 막기 위해 일정 깊이를 주어 깊이에 다다르면 새로운 트리를 만들어 연결시키는 방법을 이용하였다. 결과적으로 입력과 검색 속도를 상당히 빠르게 진전시킬 수 있었으나 CB 트라이와 마찬가지로 더미노드를 사용하고 여러 트리의 링크를 사용하기 때문에 저장 공간이 커지는 단점을 안고 있다. 본 논문에서 제안한 RCB 트라이는 더미노드를 완전히 없애 성능이 60% 향상되었다.

Reconstitution of CB Trie for the Efficient Hangul Retrieval

Kyu-cheol Jung*

ABSTRACT

This paper proposes RCB(Reduced Compact Binary) trie to correct faults of CB(Compact Binary) trie. First, in the case of CB trie, a compact structure was tried for the first time, but as the amount of data was increasing, that of inputted data gained and much difficulty was experienced in insertion due to the dummy nods used in balancing trees.

On the other hand, if the HCB trie realized hierarchically, given certain depth to prevent the map from increasing onthe right, reached the depth, the method for making new trees and connecting to them was used. Eventually, fast progress could be made in the inputting and searching speed, but this had a disadvantage of the storage space becoming bigger because of the use of dummy nods like CB trie and of many tree links. In the case of RCB trie in this thesis, a capacity is increased by about 60% by completely cutting down dummy nods.

Key words : Compact Binary Trie, HCB Trie, Dictionary Retrieval

* 군산대학교 컴퓨터 정보과학과

1. 서 론

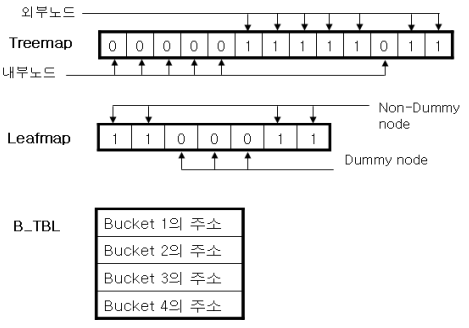
모바일 데이터베이스는 기존의 데이터베이스를 기준으로 변형된 것이기 때문에 용량이 상대적으로 크고 속도 또한 느린 단점이 있다[1]. 그리고 이러한 데이터베이스는 모바일 웹을 지원하기 위해 설계되어 서버 클라이언트 환경 하에서 작동하게 된다. 그러므로 파일관리를 위해서는 기존의 알고리즘을 이용하여 직접 레코드 주소와 키를 매핑을 하여야 하는데 모바일환경에 맞도록 필요한 많은 디스크 공간을 가능한 한 적게 사용하여야 한다. 이러한 매핑을 수행하기 위한 기존의 방법으로 해싱(Hashing), 확장성해싱(Extensible hashing), 동적해싱(Dynamic hashing)등이 있다. 특히 사전탐색이 잦은 자연어 처리 분야에서는 입력문자에 의해 찾는 트라이(trie)를 채택하고 있다[6]. 최근에는 트라이의 단점을 보완한 CB 트라이(Compact Binary Trie)와 HCB 트라이(Hierarchical Compact Binary Trie) 채택하기도 하였다. 여기서 HCB 트라이는 map이 오른쪽으로 증가하는 것을 막기 위해 일정 깊이를 주어 깊이에 다다르면 새로운 트리를 만들어 연결시키는 방법을 이용하였다. 결과적으로 입력과 검색 속도를 상당히 빠르게 진전시킬 수 있었으나 CB 트라이와 마찬가지로 더미노드를 사용하고 여러 트리의 링크를 사용하기 때문에 저장공간이 커지는 단점을 안고 있다[2, 3].

본 논문의 목적은 CB 트라이를 이용하여 모바일 환경에 적합한 탐색이 빠르고 색인 용량이 적은 알고리즘으로 재설계하는 것이다. 논문의 구성은 다음과 같다. 제 2장에서는 기존에 연구된 이진트라이와 CB 트라이에 대해 살펴본다. 제 3장에서는 CB 트라이의 더미노드를 제거하여 트리 길이를 크게 줄이기 위해 제안한 RCB 트라이를 설명한다. 제 4장에서 CB 트라이와 제안한 RCB 트라이와 비교 실험, 마지막 제 5장에서 결론 및 향후 발전방향에 대해 기술한다.

2. 이진 트라이와 CB 트라이

트라이(Trie)는 탐색 트리이므로 여기서는 키를 버킷주소로 맵핑 하기위해 이진 트라이를 사용한다. 이진코드의 순서 열은 문자들의 변환 값으로 되어 있어 키 값처럼 사용되어 왼쪽 간선(edge)은 0값으로 레이블(label) 되고 오른쪽 간선은 1로 레이블 된다. 키 집합 K에 대한 이진트라이이다. 이진트리에 관한 알려진 속성은 외부노드의 수가 내부노드 수보다 하나 더 많다는 것이다.

이진트라이가 구현될 때 등록된 키의 많은 수보다 더 많이 트리의 노드보다 훨씬 더 많은 저장 공간을 요구한다. 그래서 Jonge은 집적한 비트 스트림(bit stream)으로 이진트라이를 압축하는 방법인 CB 트라이를 제안하였다[9]. 이 트라이는 3개의 요소로 구성된다. treemap, leafmap과 B_TBL이다. treemap은 트리의 상태를 표현하고 전위 순회(preorder traversing)로 얻어지는데, 모든 내부노드 방문에 대해서는 0을 보내고 모든 외부노드방문에 대해 1을 내보낸다. leafmap은 각 외부노드의 상태 즉 더미인지 아닌지를 표현하고 전위 순회함으로 얻어진다.



(그림 1) CB trie 표현

만일 외부노드가 더미이면 일치하는 비트를 “0”으로 바꾸고 그렇지 않으면 “1”로 설정한다. B_TBL은 버켓주소를 저장한다. (그림 1)은 CB 트라

이를 보여주고 있다. 이진 트라이에서 저장 공간은 버켓에 대한 포인터를 제외하고 24byte = 192bit가 요구한다. 왜냐하면 한 포인터 당 2byte 소요되는데 이 트리는 12포인터를 가지고 있기 때문이다. 그러나 CB 트라이를 사용할 경우 단지 20bit만 요구된다. 왜냐하면 treemap과 leafmap의 bit길이가 20bit이기 때문이다.

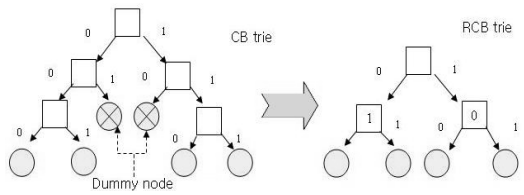
3. RCB 트라이의 구성

3.1 RCB 트라이의 개요

CB 트라이는 더미노드를 효율적으로 이용한 집약적인 이진 탐색 트리의 표현법이다. 이 더미노드의 역할은 트리의 균형을 맞추고 트리 순회를 자연스럽게 하기 위해 제안된 방법이다. 삽입 시 더미노드에 도달할 경우 버킷이 비어있는 것으로 간주하여 트리의 구조에는 영향을 주기 않고 외부메모리에만 공간을 확보하여 삽입이 쉽게 하도록 만들어 준다[3].

그러나 이러한 장점에도 불구하고 단점으로는 대량의 데이터가 다루어야할 경우 균형을 위해 과도하게 생성된 더미노드로 인해 탐색 시 많은 더미노드를 순회하여야 하며, 삽입·삭제 시 더미노드로 인해 과다한 Shift 연산이 발생하게 된다. 이러한 문제를해결하기 위해 제안된 HCB 트라이는 트리를 분리하여 treemap과 leafmap이 길어지는 것을 방지하였다. 그러나 주기억장치에 로드 되는 데이터를 줄이기 위해 버킷테이블을 주기억장치에 두지 않고 보조기억장치에 둘 경우 링크된 트리 주소를 탐색하기 위한 잦은 입출력으로 오히려 더 많은 시간을 요구하게 된다[2].

또한 위 뒤 방법 모두 한글과 같이 16비트의 이진코드를 가진 유니코드들의 경우 더미노드가 방대하게 늘어나게 된다. 본 장에서는 더미노드를 별도로 관리하기 위한 맵의 사용으로 treemap의 길이를 짧게 하고 보조기억장치를 한 번의 접근으로 탐색하여 잦은 입출력으로 인한 시간 증가를 회비하기 위한 방법으로 RCB(Reduced CB) 트라이라고 명명한 새로운 트리구조를 제안하고자 한다.



(그림 2) 트리 구조의 비교

CB 트라이에서는 초기상태가 treemap이 '011', leafmap이 '00'이었으나 RCB 트라이에서는 모두 '0'으로 초기화하였다. 왜냐하면 CB 트라이는 더미노드를 허용하면서 이진트리의 구조를 유지해야 하기 때문인데 RCB 트라이에서는 둘이상이 삽입이 되어야 이진트리의 구조를 이룰 수 있기 때문이다.

3.2 RCB 트라이의 탐색

RCB 트라이는 더미노드를 가지고 있지 않은 treemap을 표현하기 때문에 더미노드의 상태를 표현한 leafmap은 더 이상 필요하지 않다. 반면 내부노드에 생략된 비트를 표현하기 위한 새로운 방식의 맵이 필요한데 이 맵의 이름을 innermap이라고 부르기로 한다.

그리고 모아진 비트들의 정보를 가지고 있는 skipmap을 새로 만들었다. 이 innermap은 얼마나 모았는지에 관한 내용을 가지며 skipmap은 모아진 비트들의 내용을 담고 있다.

그리고 삽입키의 물리적 위치를 가지고 있는 B_TBL을 포함한다. (그림 3)에서 보듯이 treemap과 B_TBL는 일반적인 CB 트라이와 같음을 볼 수 있다.

그러나 innermap의 경우 treemap을 전위순회방식으로 트리를 순회하면서 입력 키값을 비교하여

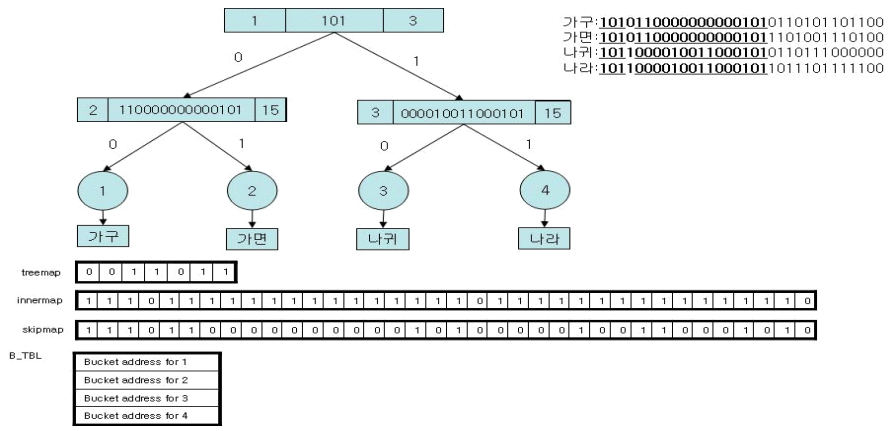
얻어지는데 만일 내부노드에 모아진 비트가 있다면 그 개수만큼 '1'로 표현하고 그 다음에 '0'을 추가시킨다. 그리고 모아진 비트가 없는 내부노드라면 그냥 '0'을 추가시킨다. 그리고 skipmap은 innermap과 마찬가지나 모아진 비트가 있으면 그 비트를 그대로 표현하고 다음에 '0'을 추가 시키고 모아진 비트가 없으면 그냥 '0'을 추가시킨다. 그리고 B_TBL의 주소는 treemap의 '1'의 수가 B_TBL의 주소가 된다. 왜냐하면 RCB 트라이에서는 외부노드에는 반드시 키값이 들어가기 때문이다. 먼저 RCB 트라이의 탐색은 innermap을 검사하면서 진행된다. treemap에서 비트 값이 '0'이 선택될 경우 innermap을 확인하여 '1'비트인지 '0'비트인지를 검사한다. 만일 '1'비트이면 '0'비트가 나올 때까지 삽입키의 비트위치를 계속 증가시켜 나간다. 그 후 treemap에서 '1'비트를 가리킬 경우 그 위치까지 해당하는 '1'의 개수를 계수하여 그 수에 해당하는 B_TBL의 주소를 검사한다. 만일 그 주소에 입력키 값과 같은 키가 존재할 경우 탐색을 완료하고 그렇지 못할 경우 실패를 발생시킨다.

3.3 RCB 트라이의 삽입·삭제

키의 삽입은 CB 트라이와 비슷한 방법으로 삽입된다. 단, 다른 점이 있다면 중복되는 키 값이 있을 경우 CB 트라이는 서로 다른 비트가 나올 때 까지 더미 노드를 계속해서 만들어나가는 것이지만 RCB 트라이는 같을 경우 내부노드에 모아둔다는 것이다. 그런데 CB 트라이에서 삽입은 매우 간단하였다. 왜냐하면 무조건 삽입하고자하는 곳이 더미노드이면 더미노드에 해당하는 버킷에 키를 삽입하고 버킷이 차있을 경우에만 새로운 기본 트리를 삽입하면 되기 때문이다. 하지만 RCB 트라이에서는 다음과 같은 사항을 고려하여야 한다.

① 모아진 내부노드 안에서 분기가 일어날 경우
② 기존 버킷에 다른 키 값이 존재하여 서브트리를 생성해야할 경우

먼저 내부노드에 모아진 비트에서 분기가 일어나는 경우는 삽입되는 키의 분기 위치 비트가 '0'이냐 '1'이냐에 따라 다르게 설정하여야 하고 다음 기존 버킷에 다른 키값이 존재하는 경우 서브트리를 생성한 후 기존 키의 현재 위치 비트와 삽입하고자 하는 키값의 현재 위치비트를 비교하여 중복되는 비트를생성하는 서브트리의 내부노드에 모아두고 다른 비트들의 크기를 비교하여 '0'이면 왼쪽, '1'이면 오른쪽으로 이동 시킨다.



(그림 3) RCB 트라이 구조

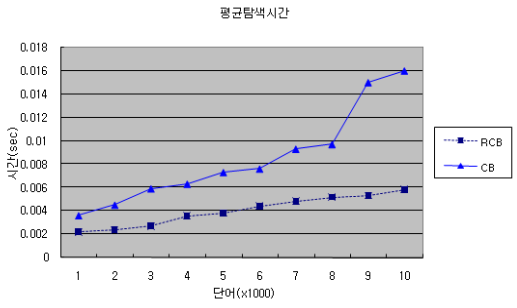
4. 실험 및 평가

탐색 실험을 위해 형태소 분석기용 명사사전[13]에 수록된 1만 단어를 이용하여 기본데이터를 만들어 CB 트라이와 RCB 트라이를 탐색 시간과 생성되는 맵의 크기를 실험 하였다. 개발 언어는 범용성을 고려하여 Java 1.5를 이용하였다.

우선 실험에 앞서 전제가 되어야 하는 것은 CB 트라이는 참고 자료를 이용하여 직접 프로그램을 작성하였고 실험 데이터는 참고 저자들이 제시한 데이터가 아닌 RCB 트라이를 구성하면서 만든 데이터를 사용하였기 때문에 실제 참고 저자들의 실험과결과가 다를 수 있다는 것을 전제로 두고자 한다. 또한 실험은 빠른 결과를 위해 모바일용으로 직접 구현하지 않고 데스크 탑에서 속도와 테이블의 크기를 계산하였다.

4.1 탐색 시간 비교 실험

비교실험은 색인파일의 크기를 1000개씩 늘리면서 그중 무작위로 100개의 단어를 선정한후 탐색시간을 측정하였다. 그리고 측정단위가 미세한 밀리 초가 되어 시스템의 상태에 따라 다른 결과가 나올 수 있으므로 정확한 측정을 위해 같은 조건에서 3번의 탐색을 더하여 평균탐색 시간을 계산하였다. (그림 4)에서 보듯이 CB 트라이에 비해서는 RCB 트라이 속도가 평균 60% 정도 향상되었다. 이

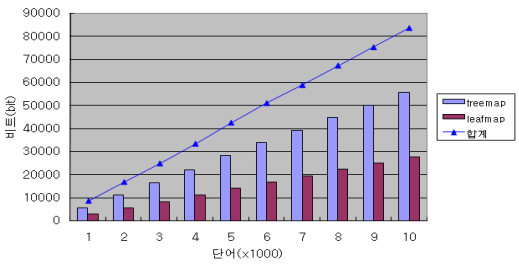


(그림 4) 평균탐색 시간 비교

결과는 더미노드가 없어지면서 트리의 크기가 전체적으로 줄어든 것에 따른 효과라고 볼 수 있다.

4.2 색인 용량의 비교 실험

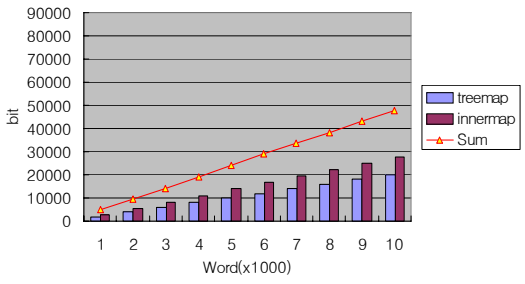
(그림 5)는 CB 트라이에서 실험한 데이터를 가지고 맵의 크기를 살펴본 결과이다. treemap이 leafmap의 2배 정도 많은 공간을 차지하는 것을 볼 수 있다. 더미노드의 비율은 leafmap의 64%에 해당할 정도로 많은 부분을 차지하고 있다.



(그림 5) CB 트라이에서 맵크기 변화

이 때문에 탐색과 삽입 삭제 시 많은 시스템의 부담을 초래하고 있다. RCB 트라이의 경우 탐색을 위해서는 treemap과 innermap만 필요하기 때문에 주 기억장치에 로드되는 두 트리의 합만 계산하였다. 만약 여기에 skipmap의 크기를 같이 포함하더라도 innermap 만큼의 용량이 더 필요한 것이기 때문에 실험에서는 제외를 시켰다. 위의 CB트라이와 RCB 트라이의 생성 크기를 비교해보면 평균 57% 정도 적게 생성된다. 그리고 깊이가 11인 HCB 트라이의 생성 크기를 보면 CB 트라이와 많은 차이는 없는 것을 볼 수 있다. 이유는 실제로 CB 트라이에서 추가되는 것은 다음 트리의 포인터 정보를 가지고 있는 링크가 더 추가되기 때문에 실제로 많은 비트가 추가되지는 않는다.

하지만 기억장치에는 treemap과 leafmap 이외에 버킷테이블이 함께 올라간다. 왜냐하면 다음 트리를 탐색해나가기 위해 버킷테이블을 찾아야하기 때문이다. 이 때문에 treemap과 leafmap의 크기만으



(그림 6) RCB 트라이의 treemap과 innermap 크기 변화

로는 의미가 없다. 그리고 CB 트라이는 아래 그림들에서 보듯이 83,000비트 즉, 10Kbyte면 트리를 다 표현할 수 있고, RCB 트라이는 47,000비트 즉, 5Kbyte면 트리를 전부 표현이 가능하다. 그리고 CB 트라이의 leafmap에 존재하는 더미노드의 비중을 살펴보면 약 50% 정도의 공간을 절약된 것으로 나타났다.

5. 결론 및 향후 과제

본 논문에서는 이진트라이를 축약된 배열 구조로 표현한 CB 트라이의 단점을 보완하여 한 번의 보조 기억장치의 접근으로 빠르게 탐색할 수 있고 트리 길이에 부담을 주는 더미노드를 없애고자하여 본 논문에서는 RCB 트라이를 제안하였다. 이 RCB 트라이는 더미노드를 직접 외부노드에 표현하지 않고 내부노드에 표현하는 방식을 채택하였다. 기존의 트리와는 달리 leafmap이 아닌 innermap을 사용하여 내부노드에 더미노드에 해당하는 중복되는 비트를 보관할 경우 '1'로 설정하고 내부노드는 '0'으로 설정한다. 이렇게 하면 더미노드는 만들어지지 않고 트리의 깊이도 상당히 줄어듦을 볼 수 있다. 또한 treemap과 innermap이 생성될 경우 CB 트라이보다 50% 감소됨을 실험을 통해 증명하였고 탐색 시간도 CB 트라이에 비해 60% 향상되어 소규모의 모바일 환경이나 임베디드 환경에 적용이 가능 하게 되었다.

참 고 문 헌

[1] 정규철, 이진관, 장혜숙, 박기홍, "CBDS 트리를이용한 모바일 기기용 저용량사전 구축에 관한 연구", 한국컴퓨터정보학회, 제10권, 제5호, 2005.

[2] 정규철, "RCB 트라이를 이용한 빠른 검색과 소용량 색인 구조에 관한 연구", 한국컴퓨터정보학회, 제12권, 제4호, 2007.

[3] J. Aoe, K. Morimoto, M. Shishibori, and K. Park, "A Trie Compaction Algorithm for a Large Set of Keys", IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 3, pp. 476-491, 1996.

[4] M. Jung, M. Shishibori, and A. J. Tanaka, Aoe : "A Dynamic Construction Algorithm for the Compact Patricia Trie Using the Hierarchical Structure", Information Processing & Management, Vol. 38, No. 2, pp. 221-236.

[5] W. D. Jonge, "Two access methods using compact binary tree", IEEE Trans. on Software Engineering, Vol. 13, No. 7, pp. 7999-809, 1987.

[6] 고려대학교 자연어처리 연구실, nlp.korea.ac.kr, 2002.

정 규 철

2006년 군산대학교 컴퓨터과학과 (이학박사)

1999년~현재 군산대학교 컴퓨터과학과 강사