韓國 컴퓨터情報學會 論文誌
第12卷 第4號, 2007. 9.

2007-12-4-1-2

RCB트라이를 이용한 빠른 검색과 소용량 색인 구조에 관한 연구

정 규 철*

A Study on Small-sized Index Structure and Fast Retrieval Method Using The RCB trie

Jung Kyu-cheol*

요 약

본 논문에서는 CB 트라이와 HCB 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 HCB트라이는 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 CB트라이와 마찬가지로 더미노드를 사용하고 여러 트리의 링크를 사용하기 때문에 저장공간이 커지는 단점을 안고 있다. 본 논문에서 제안한 RCB트라이는 더미노드를 완전히 없애 treemap을 약 35%정도 줄일 수 있었고 HCB트라이에 비해 전체 색인의 크기를 절반으로 줄였다.

Abstract

This paper proposes RCB(Reduced Compact Binary) trie to correct faults of both CB(Compact Binary) trie and HCB(Hierarchical 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 on the 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, the tree-map could be reduced by about 35% by completely cutting down dummy nods and the whole size by half, compared with the HCB trie.

▸ Keyword : CB트라이(Compact Binary Trie), HCB트라이(Hierarchical Compact Binary Trie), 이진 트라이(Binary Trie), 사전검색(Dictionary Retrieval), 색인 구조(index structure)

• 제1저자 : 정규철
• 접수일 : 2007. 6.21, 심사일 : 2007. 8.4, 심사완료일 : 2007. 9.20
* 군산대학교 시간강사

Ⅰ. 서 론

최근 유비쿼터스와 텔레매틱스 같은 모바일 환경에서 휴대전화 및 PDA(Personal Digital Assistants)나 PMP (Portable Multimedia Player)등의 휴대용 컴퓨터의 사용이 증가하면서 기존의 데스크탑 환경에서 사용하던 검색 방법을 그대로 모바일 환경에 적용하는 것이 힘들어지게 되었다. 물론 시스템의 발달로 인해 빠른 처리속도와 대용량의 휴대장치들이 나오기는 하지만 아직 가격 면에서 많은 부담을 주고 있다. 또한 이러한 모바일 장치의 운영체제에서는 모바일 데이터베이스를 지원한다[1][2]. 그러나 모바일 데이터베이스는 기존의 데이터베이스를 기준으로 변형된 것이기 때문에 용량이 상대적으로 크고 속도 또한 느린 단점이 있다[3]. 그리고 이러한 데이터베이스는 모바일 웹을 지원하기 위해 설계되어 서버 클라이언트 환경 하에서 작동하게 된다[4]. 그러므로 파일관리를 위해서는 기존의 알고리즘을 이용하여 직접 레코드 주소와 키를 매핑을 하여야 하는데 모바일 환경에 맞도록 필요한 많은 디스크 공간을 가능한 한 적게 사용하여야 한다[3][5]. 이러한 매핑을 수행하기 위한 기존의 방법으로 트라이(Trie), 해싱(Hashing), 확장성 해싱(Extensible hashing), 동적해싱(Dynamic hashing)등이 있다. 그중에서 사전탐색이 잦은 자연어 처리 분야에서는 입력문자에 의해 찾는 트라이(trie) 탐색을 많이 채택하고 있다 [6][7][8].

본 논문의 목적은 유비쿼터스와 텔레매틱스 환경에서 PDA나 휴대전화등과 같은 모바일 환경에서 대량의 데이터를 기존의 데스크 탑에서 적용되는 방식이 아닌 모바일 환경에 적합한 탐색 알고리즘을 설계하는 것이다. 그리고 데스크 탑에서 주로 사용되고 있는 트라이(trie)알고리즘을 작은 맵(map)으로 표현 가능한 CB(Compact Binary)트라이를 활용하는 것이다. 그러나 이 CB트라이는 대량의 데이터를 색인할 경우 많은 더미노드로 인해 트리의 깊이가 깊어지고 더불어 탐색 속도 또한 저하되는 단점을 가지고 있다. 이를 보완하기 위해 더미노드를 별도로 관리하여 트리의 깊이를 줄이는 알고리즘을 설계하는 것이다. 이를 이용하여 소규모 모바일 환경에서도 대용량 사전에 있는 원하는 데이터를 적은 색인으로 파일로 빠르게 탐색이 가능하도록 하는 것이다.

본 논문의 구성은 다음과 같다. 제 Ⅱ장에서는 기존에 연구된 CB트라이, Hierarchical CB(HCB)트라이에 대해 살펴본다. 제 Ⅲ장에서는 CB트라이의 더미노드를 제거하여 트리 길이를 크게 줄이기 위해 제안한 RCB트라이를 설명한다.제 Ⅳ장에서 CB트라이와 HCB트라이, 제안한 RCB트라이와 비교 실험 마지막 제 Ⅴ장에서 결론 및 향후 발전방향에 대해 기술한다.

Ⅱ. 관련 연구

2.1 이진 트라이(Binary Trie)

트라이는 용어 retrieval(탐색)의 중간 글자의 'trie'를 따온 것인데 트리와 구별하기 위해 일반적으로 트라이라 부른다[7]. 즉 트라이는 탐색 트리라 보면 된다. 여기서는 키를 버킷주소로 맵핑 하기위해 이진 트라이를 사용한다.

이진코드의 순서 열은 문자들의 변환 값으로 되어있어 키 값처럼 사용되어 왼쪽 간선(edge)은 0값으로 레이블(label) 되고 오른쪽 간선은 1로 레이블 된다.

예를 들어 내부코드 a~z가 해싱함수 H(a~z)를 통해 각각 0~25으로 변환하였다고 하자. 그리고 그들은 5비트의 이진코드로 전이 되어진다고 하자.

표1 은 키 집합으로 일치하는 이진 순서를 가리킨다.

표 1 키 집합 K의 이진코드열
Table. 1 Binary sequences of a key set K

K = {air, bag, eat, tea, zoo}		
키	내부 코드	이진 표현
air	0 / 8 / 17	00000 01000 10001
bag	1 / 0 / 6	00001 00000 00110
eat	14/ 0 / 19	01110 00000 10011
tea	19 / 4 / 0	10011 00100 00000
zoo	25 / 14 / 14	11001 01110 01110

키 집합 K에 대한 이진트라이는 〈그림 1〉과 같다. 이진 트리에 관한 알려진 속성은 외부노드의 수가 내부노드 수보다 하나 더 많다는 것이다.

〈그림 1〉에 주목할 것은 간선들의 레이블이다. 왼쪽 간선은 '0'으로 레이블 되고 오른쪽 간선은 '1'로 레이블 되었다. 이진 숫자를 사용함으로 모든 버켓은 Root로부터 패스에 따라 레이블 될 수 있다. 〈그림 1〉에서는 이진 트라이에서 버켓에 대해 어떠한 주소도 가지고 있지 않는 특별한 외부노드를 효율적인 트리의 압축을 위해 사용하고 있다.

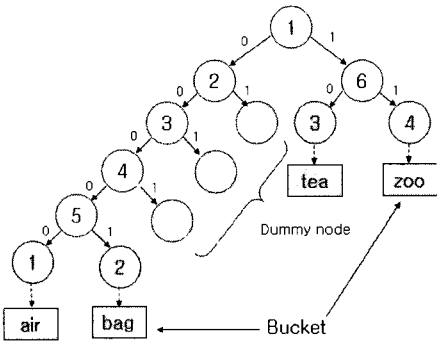

그림 1 이진 트라이
Fig. 1 Binary Trie

이 외부노드들을 더미노드라고 한다. 더미노드를 사용하므로 부가적인 이익들이 파생되는데 먼저 외부노드의 수가 내부노드의 수보다 하나 더 많은 이진 트리속성을 만족하게 된다.

이 속성은 집적된 자료구조를 사용하는 검색 알고리즘에 아주 중요한 속성이다. 다음은 더미노드에서 검색이 종료되면 검색키는 이진트라이에 속하지 않는 걸로 간주되고 디스크접근이 하지 않는다. 이 더미노드는 2차 메모리에 할당한 버켓이 없기 때문에 보조 메모리의 공간 효율성에 영향을 끼치지 않는다.

2.2 Compact Binary Trie

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 방법을 제안하였고 우리는 이 방법을 이용하고자 한다[9][10][11]. 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. 만일 외부노드가 더미이면 일치하는 비트를 "0"으로 바꾸고 그렇지 않으면 "1"로 설정한다. B_TBL은 버켓주소를 저장한다. <그림 2>는 <그림 1>의 집적한 이진트라이를 보여주고 있다.

예를 들어 전위 순회에서 루트 노드1부터 시작하여 노드 2, 3, 4, 5를 지나간다.

treemap의 처음부터 5번째 비트까지의 모든 비트 값은 '0'으로 세팅한다. 다음 외부노드 1, 2 그리고 3개의 더미 외부노드들을 지나간다. 그리고 6번째 비트부터 10번째 비트까지의 모든 비트 값을 1로 세팅 한다. 그런 후에 내부노드 6, 외부노드 3, 4를 순서대로 지나간다.

그래서 11번째부터 13번째까지의 3비트를 각각 '011'로 세팅되어 있다.

leafmap의 경우 외부노드 1, 2, 3 더미 외부노드 3과 4는 순서에 의해 지나가고 leafmap은 '1100011'로 세팅된다.

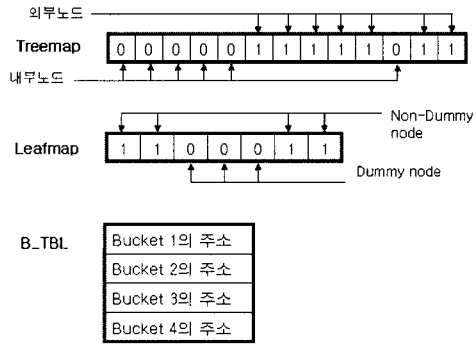

그림 2 <그림1>에 대해 Compact Binary 트라이
Fig. 2 Compact binary trie of Fig 1

이런 경우 이진트라이는 <그림 1>처럼 모두 한 노드에서 두 포인터를 갖는 일반적인 트리주소처럼 구현된다. 저장공간은 버켓에 대한 포인터를 제외하고 24byte = 192bit가 요구한다. 왜냐하면 한 포인터 당 2byte소요되는데 이 트리는 12포인터를 가지고 있기 때문이다.

그러나 집적한 이진트라이를 사용할 경우 단지 20bit만 요구된다. 왜냐하면 treemap과 leafmap의 bit길이가 20bit이기 때문이다.

CB트라이를 이용한 탐색알고리즘은 오른쪽으로 진행하는 treemap의 첫 번째 비트로부터 비트단위로 전위순회 탐색을 하며 처리하는데 treemap의 비트위치가 '0'일 경우 비트를 전진시키면서 왼쪽 서브트리를 처리한다. 만일 오른쪽으로 서브 트리로 나아가고자 할 경우 왼쪽 서브트리를 반드시 건너뛰어야 한다. 이럴 경우 왼쪽 서브트리에서 외부노드의 수가 내부노드의 수보다 하나 더 많다는 이진트리 속성을 이용하게 된다. 이 처리는 1비트의 수가 0비트의 수보다 하나 더 많은 때까지 즉 '1' 비트들의 합과 '0'비트들의 합이 같은 다음 위치로 treemap의 비트위치를 전진시키면서 오른쪽 서브트리의 첫 번째 위치를 발견한다.

leafmap의 '1'비트 값은 더미노드가 아닐(non-dummy) 경우 leafmap에서 '1'비트들의 개수는 B_TBL에서 요구되는 버킷주소를 포함하는 슬롯을 가리킨다.

〈그림 2〉를 바탕으로 "eat(00100 00000 10011)"을 삽입하는 과정을 〈그림 3〉에서 보이고 있다.

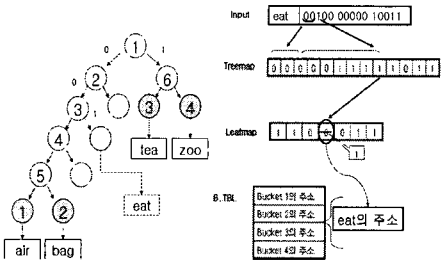

그림 3 CB 트라이에 "eat" 삽입
Fig. 3 Inserting the word of "eat" into CB Trie

2.3 Hierachical Compact Binary Trie

CB트라이의 문제점은 삽입·삭제 시 비트 스트링의 잦은 좌우 이동에 있다. 이를 해결하기 위해 [12]에서는 트리에 일정의 깊이(depth)를 두어 그 이상 늘어나게 되면 트리를 분리하는 기법을 적용하였다.

이 깊이를 분할 깊이라고 부른다. 그리고 이 작은 트리를 분할된 트리라고 부른다. 이 같은 방법으로 분리된 트리는 계층적 이진 트라이 라고 부르며, 이 트라이를 CB 트라이구조에 기반을 둔 표현한 기법을 HCB(Hierarchical Compact Binary) 트라이라고 부른다. 계층적 이진 트라이에서 i번째 분리된 트리에 일치하는 HCB트라이는 treemap-i, leafmap-i와 B_TBL-i로 구성한다. 그러나 분리된 다음 트리의 포인터가 되는 외부노드는 특별한 외부노드로 간주된다.

그리고 B_TBL-i는 외부노드에 일치하는 슬롯에 음수 기호로 처리되어 다음 분리된 트리의 주소를 포함한다.

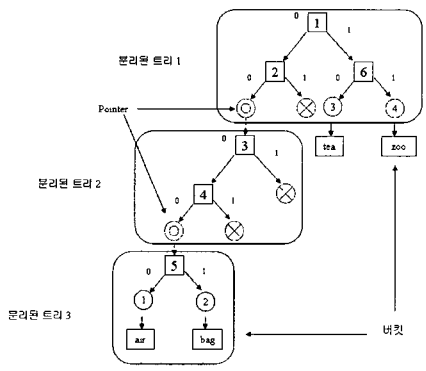

그림 4 〈그림 1〉을 기반으로 한 HCB트라이
Fig. 4 The HCB Trie based on Fig. 1

분리된 깊이 2를 가지고 〈그림 1〉의 이진 트리를 기반으로 얻어진 계층적 이진 트라이를 〈그림 4〉에서 보여주고 있고 이에 대한 HCB트라이는 〈그림 5〉에서 보인다.

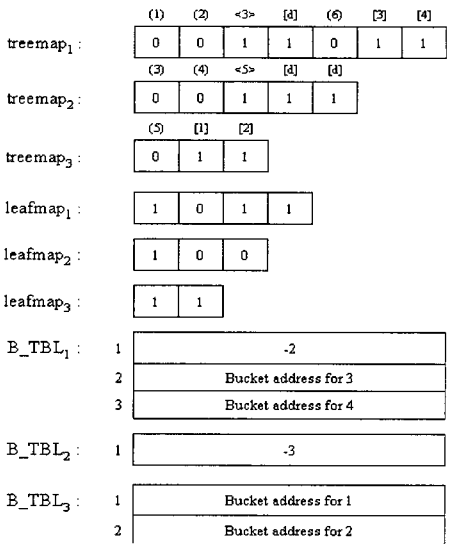

그림 5 HCB트라이 표현
Fig. 5 Representation of HCB Trie

이 향상된 방법의 사용에 의해 각 처리는 속도를 높일 수 있다. 왜냐하면 각 분리된 트리에 대한 HCB트라이의 불필요한 탐색이 생략되기 때문이다. HCB트라이에서 키를 탐색하고자 할 때 키의 이진 시퀀스가 아래의 이진 시퀀스로 나누어진다.

$$H(k) = H1(k)\ H2(k)\ \cdots\ Hj(k)\ \cdots\ Hn(k)$$

분리 깊이를 L로 정의한다면 H1(k)에서 Hn-1(k)의 길이는 L bit이고 Hn(k)의 길이는 L bit들 보다 작다. 예를 들면 깊이가 2인 HCB트라이에서 "bag"을 탐색하는 과정을 보이고 있다. treemap1에서 position이 2일때 leafmap1의 position 0과 B_TBL1의 1에 의해 treemap2의 position을 0으로 보내고 treemap의 position이 2일때 treemap3의 0으로 보낸다. 그 후 "bag"의 key sequence가 4일때 B_TBL3의 버킷 2를 참조하여 키를 찾게 된다.

이처럼 하위 트리에 존재할 경우 복잡하게 보이지만 〈그림 6〉에서 보여지는 것처럼 HCB트라이에서 키 "zoo"를 탐색할 경우에는 오직 분리된 트리 1의 HCB트라이 사용에 의해 탐색되어 질 수 있다. 그러므로 탐색의 시간비용은 CB 트라이보다 더 좋다.

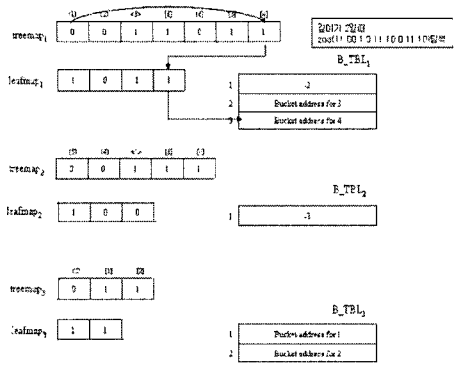

그림 6 깊이가 2인 HCB트라이에서 "zoo"탐색
Fig. 6 Searching the word of "zoo" from the HCB Trie

이렇듯 HCB트라이는 CB트라이의 삽입 삭제 시 비트열의 이동을 최소화하고 트리의 오른쪽에 위치한 키들을 검색하는 시간을 단축시키기 위해 고안된 알고리즘이다. 이러한 아이디어는 새로운 트리를 만들어 트리의 오른쪽을 탐색할 경우 분리한 트리는 탐색에서 제외시키는 방법을 취하였다. 또한 깊이가 정해진 크기를 넘지 않는 한 CB트라이와 같은 방법으로 삽입되기 때문에 CB트라이에서 변환도 가능하게 된다. 따라서 CB트라이에 비해 평균탐색 시간 또한 단축을 시키고 있다.

그러나 버킷테이블에 분리된 트리의 주소를 가져야하기 때문에 주 기억장치에 저장된다면 빠른 속도의 삽입·탐색을 할 수 있지만 보조기억장치에 버킷테이블을 저장한다면 분리된 트리를 참조할 때 마다 디스크를 접근해야한다. 버킷테이블을 주 기억장치에 저장할 경우 실제 데이터의 주소 이외에 링크 주소까지 모두 저장되므로 기억장치에 오버헤드가 발생하게 된다. 그러므로 대량의 사전인 경우 이 HCB트라이는 적합하지 않다.

III. RCB 트라이의 구성

3.1 RCB트라이의 개요

CB트라이는 더미노드를 효율적으로 이용한 집약적인 이진 탐색 트리의 표현법이다. 이 더미노드의 역할은 트리의 균형을 맞추고 트리 순회를 자연스럽게 하기 위해 제안된 방법이다. 더미노드의 장점은 전위탐색으로 트리순회를 가능하게 해주며 실제 외부기억장치에 기억공간을 가지고 있지 않아 외부 메모리의 부담이 적고 탐색 시 더미노드에 도달할 경우 외부 메모리를 참조하기 않고도 탐색 실패를 유도 해낼 수 있다. 삽입 시 더미노드에 도달할 경우 버킷이 비어있는 것으로 간주하여 트리의 구조에는 영향을 주기 않고 외부메모리에만 공간을 확보하여 삽입이 쉽게 하도록 만들어 준다.

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 Shift연산이 발생하게 된다. 이러한 문제를 해결하기 위해 제안된 HCB트라이는 트리를 분리하여 treemap과 leafmap이 길어지는 것을 방지하였다. 그러나 주 기억장치에 로드 되는 데이터를 줄이기 위해 버킷테이블을 주 기억장치에 두지 않고 보조기억장치에 둘 경우 링크된 트리 주소를 탐색하기 위한 잦은 입출력으로 오히려 더 많은 시간을 요구하게 된다.

정의 1. RCB트라이의 데이터 타입

```
class MapSet implements Serializable{
    boolean[] treemap;
    boolean[] innermap;
    boolean[] skipmap;
    int[] btbl;
    MapSet(){
        this.treemap =new boolean[]{false};
        this.innermap = new boolean[]{false};
        this.skipmap = new boolean[]{false};
        this.btbl =  new int[]{0};
       }
}
```

본 장에서는 더미노드를 별도로 관리하기 위한 맵의 사용으로 treemap의 길이를 짧게 하고 보조기억장치를 한번의 접근으로 탐색하여 잦은 입출력으로 인한 시간 증가를 회비하기 위한 방법으로 RCB(Reduced CB) 트라이라고 명명한 새로운 트리구조를 제안하고자 한다.

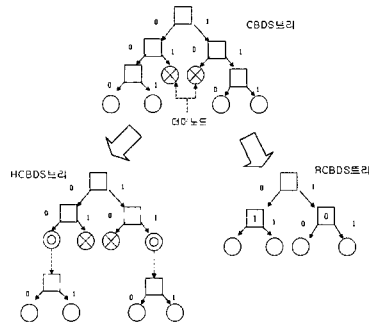

그림 7 트리구조의 비교
Fig. 7 Comparison with tree structures

RCB트라이에 대한 데이터 타입은 〈정의 1〉과 같이 정의한다. CB트라이에서는 초기상태가 treemap이 '011', leafmap이 '00'이었으나 RCB트라이에서는 모두 '0'으로 초기화 하였다. 왜냐하면 CB트라이는 더미노드를 허용하면서 이진트리의 구조를 유지해야 하기 때문인데 RCB트라이에서는 둘이상이 삽입이 되어야 이진트리의 구조를 이룰 수 있기 때문이다.이 구조는 〈그림 7〉에서 볼 수 있듯이 HCB트라이에서 변형한 형태가 아닌 CB트라이에서 변형을 시도하였고 더미노드를 줄여 treemap의 길이를 짧게 만들기 위한 방법을 제안하였다.

3.2 RCB트라이의 탐색

RCB트라이는 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 삽입 키의 물리적 위치를 가지고 있는 B_TBL을 포함한다.

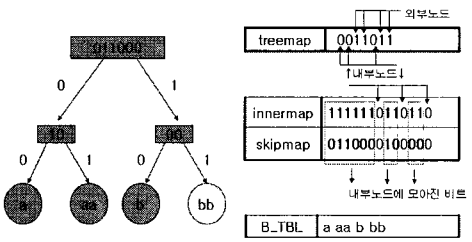

그림 8 트리 구조
Fig. 8 Tree Structure

〈그림 8〉에서 보듯이 treemap과 B_TBL는 일반적인 CB트라이와 같음을 볼 수 있다. 이때 'a' 는 "01100001" 이고 'b'는 "01100010" 이다.

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 추가 시킨다. 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 RCB트라이에서는 외부노드에는 반드시 키값이 들어가기 때문이다.

먼저 RCB트라이의 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 삽입 키의 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 RCB트라이의 삽입 · 삭제

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

1. 모아진 내부노드 안에서 분기가 일어날 경우
2. 기존 버킷에 다른 키값이 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 비트를 생성하는 서브트리의 내부노드에 모아두고 다른 비트들의 크기를 비교하여 '0'이면 왼쪽, '1'이면 오른쪽으로 이동 시킨다.

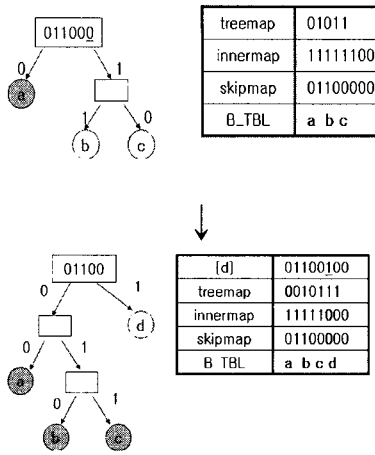

그림 9 "a,b,c"에 "d"를 삽입
Fig. 9 Inserting the character of "d" into "a,b,c"

RCB트라이에서 삭제는 삭제하고자 하는 키를 탐색한 후 삽입의 역순으로 처리하게 된다. 그러나 삭제처리는 삽입과 달리 크게 두 가지의 경우로 나눌 수 있다.

1. 형제 노드에 서브트리가 없는 경우
2. 형제 노드에 서브트리가 존재하는 경우

첫 번째의 경우는 두 노드가 모두 외부노드를 의미한다. 이런 경우 삭제 후 부모노드와 남은 외부노드가 합병하여 하나의 외부노드를 만들게 되는 이 경우 다시 합병하고자 내부 노드에 모아진 비트의 여부에 따라 다시 두가지의 경우로 나타난다.

또한 두 번째의 경우에는 형제 노드에 서브트리가 있다는 것은 형제 노드가 내부노드를 의미하게 된다. 이럴 경우 외부 노드를 삭제하게 되면 내부 노드 하나만 존재하므로 남은 내부노드는 모아진 비트에 삭제 노드의 방향을 나타내는 비트를 포함시켜 형제노드와 통합하여 형제노드를 부모노드로 구성하는 형태를 취하게 된다. 그리고 삽입 시와 마찬가지로 탐색 시 키의 삭제 포인트가 가리키는 비트가 '1'일 경우와 '0'일 경우에 따라 삭제 방법이 달라진다. 우선 '1'일 경우에는 treemap에서 현재 위치가 가리키는 비트 '1'에 해당하는 부모노드('0 xxx 1')를 왼쪽 서브트리를 역으로 건너뛰어 찾아 함께 삭제하여야 하고 '0'일 경우에는 바로 옆에 있는 내부노드('01')를 같이 삭제하면 된다. 또한 innermap에서는 형제노드가 부모노드가 되는 것이므로 단순히 '0'으로 되어있는 포인터를 '1'로만 바꿔주면 된다.

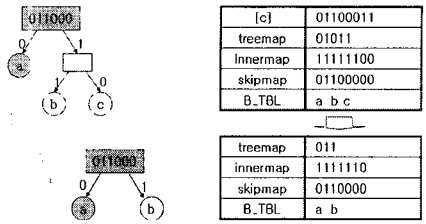

그림 10 "c"의 삭제
Fig. 10 Deleting the character of "c"

IV. 실험 및 평가

탐색 실험을 위해 형태소 분석기용 명사사전[13]에 수록된 1만 단어를 이용하여 기본데이터를 만들어 CB트라이와 HCB트라이, 그리고 RCB트라이를 탐색 시간과 생성되는 맵의 크기를 실험 하였다. 개발 언어는 범용성을 고려하여 Java 1.5를 이용하였다.

우선 실2험에 앞서 전제가 되어야하는 것은 CB트라이와 HCB트라이 모두 참고 자료를 이용하여 직접 프로그램을 작성하였고 실험 데이터는 참고 저자들이 제시한 데이터가 아닌 RCB트라이를 구성하면서 만든 데이터를 사용하였기 때문에 실제 참고 저자들의 실험과 결과가 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 선정한 후 탐색 시간을 측정하였다. 그리고 측정단위가 미세한 밀리초가 되어 시스템의 상태에 따라 다른 결과가 나올 수 있으므로 정확한 측정을 위해 같은 조건에서 3번의 탐색을 더하여 평균탐색 시간을 계산하였다. 〈그림 11〉에서 보듯이 HCB트라이는 보조기억장치에 버킷테이블을 둔 것으로 버킷테이블 접근 시간으로 인해 탐색시간이 많이 걸리는 것으로 판단된다. CB트라이에 비해서는 RCB트라이 속도가 평균 60%정도 향상되었다. 이 결과는 더미노드가 없어지면서 트리의 크기가 전체적으로 줄어든 것에 따른 효과라고 볼 수 있다.

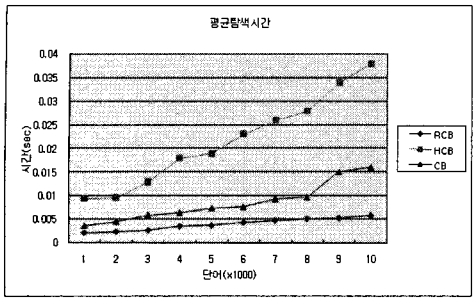

그림 11 평균탐색시간 비교
Fig. 11 Comparison with the AVTs

4.2 색인 용량의 비교 실험

〈그림 12〉는 CB트라이에서 실험한 데이터를 가지고 맵의 크기를 살펴본 결과이다. treemap이 leafmap의 2배 정도 많은 공간을 차지하는 것을 볼 수 있다. 그리고 CB트라이의 leafmap에 존재하는 더미노드의 비중을 살펴보면 〈그림 13〉와 같다.

더미노드의 비율은 leafmap의 64%에 해당할 정도로 많은 부분을 차지하고 있다.

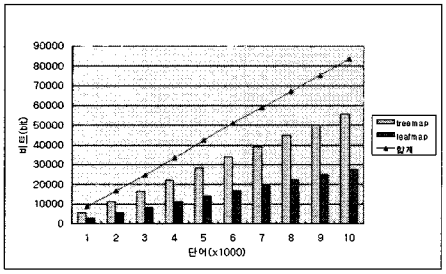

그림 12 CB 트라이의 map 크기 변화
Fig. 12 Variation of the map size in the CB Trie

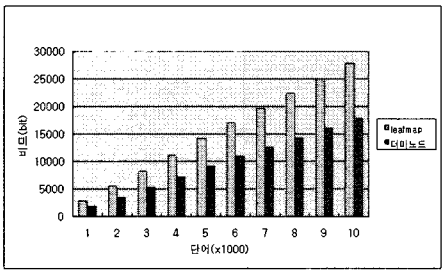

그림 13 CB트라이의 leafmap과 더미노드의 크기
Fig. 13 The Size of leafmap and dummy nodes in CB Trie

이 때문에 탐색과 삽입 삭제 시 많은 시스템의 부담을 초래하고 있다. RCB트라이의 경우 탐색을 위해서는 treemap과 innermap만 필요하기 때문에 주 기억장치에 로드되는 두 트리의 합만 계산하였다. 만약 여기에 skipmap의 크기를 같이 포함하더라도 innermap만큼의 용량이 더 필요한 것이기 때문에 실험에서는 제외를 시켰다. 위의 CB트라이와 RCB트라이의 생성 크기를 비교해보면 평균 57%정도 적게 생성된다. 그리고 깊이가 11인 HCB트라이의 생성 크기를 보면 CB트라이와 많은 차이는 없는 것을 볼 수 있다. 이유는 실제로 CB트라이에서 추가되는 것은 다음 트리의 포인터 정보를 가지고 있는 링크가 더 추가되기 때문에 실제로 많은 비트가 추가되지는 않는다. 하지만 기억장치에는 treemap과 leafmap이외에 버킷테이블이 함께 올라간다. 왜냐하면 다음 트리를 탐색해나가기 위해 버킷테이블을 찾아야하기 때문이다.

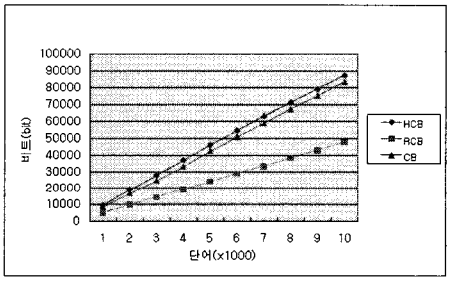

그림 14 세 트라이의 map크기 비교
Fig. 14 Comparison with the three Tries

이 때문에 treemap과 leafmap의 크기만으로는 의미가 없다. 그리고 CB트라이는 아래 그림들에서 보듯이 83,000비트 즉, 10Kbyte면 트리를 다 표현할 수 있고, RCB트라이는 47,000비트 즉, 5Kbyte면 트리를 전부 표현이 가능하다.

그러나 HCB트라이는 CB트라이와 비슷한 10Kbyte이외에 버킷테이블을 한꺼번에 로드 시켜야한다. CB트라이와 HCB트라이, RCB트라이의 map들의 전체 크기만을 표현한 그림이 〈그림 14〉이고 그림에서 보듯이 RCB트라이가 월등히 적은 공간을 필요로 하는 것을 볼 수 있다. 약 50% 정도의 공간을 절약된 것으로 나타났다.

V. 결 론 및 향후 과제

본 논문에서는 이진트라이를 축약된 배열 구조로 표현한 CB트라이와 CB트라이를 계층적으로 표현한 HCB트라이의 단점을 보완하여 한 번의 보조기억장치의 접근으로 빠르게 탐색할 수 있고 트리 길이에 부담을 주는 더미노드를 없애고자하여 본 논문에서는 RCB트라이를 제안하였다. 이 RCB트라이는 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 CB트라이보다 50%감소됨을 실험을 통해 증명하였고 탐색 시간도 CB트라이에 비해 60%향상되었음을 알 수 있었다. 향후 저속의 모바일 환경에서 빠른 탐색 뿐남 아니라 빠른 map의 갱신이 가능하도록 연구하고자 한다.

참고문헌

[1] 강성윤외 2, "자바 모바일 프로그래밍", 대림, 2002

[2] 문봉재, "모바일 디바이스에서 닷넷 애플리케이션 구축하기", Microsoft Press, 2003

[3] 정규철, 이진관, 장혜숙, 박기홍, "CBDS 트리를 이용한 모바일 기기용 저용량 사전 구축에 관한 연구", 한국컴퓨터정보학회, 제10권 제5호, 2005

[4] 안원국, "C#.NET Mobile Programming", 영진닷컴, 2005

[5] 리처드 헌터, "유비쿼터스 공유와 감시의 두얼굴", 21세기 북스, 2003

[6] 김철수,"절단검색을 지원하는 전자사전 구조 ", 한국정보과학회, 제9-C권, 제1호, 2003

[7] Aho A. V., Hopcroft J.E., & Ullman J. A. (1983). "Data Structures and Algorithms." Reading, MA: *Addison - Wesley*

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

[9] Masami, S. Aoe,J. "An Order Searching Algorithm of Extensible Hashing" Inter. j. Computer Math., Vol. 63, pp.179-201, 1997.

[10] Aoe,J, Park K(1996), A trie compaction algorithm for a large set of keys, IEEE Trans. on Knowledge and Data Engineering, 8(3)

[11] Jonge, W. D(1987), Two access methods using compact binary tree, IEEE Trans. on Software Engineering, 13(7), 7999-809

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

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

저 자 소 개

정 규 철

1995 군산대학교 컴퓨터과학과(학사)
1999 군산대학교 컴퓨터과학과(석사)
2006 군산대학교 컴퓨터과학과(박사)
관심분야 : 자연어처리, 정보검색, 유비쿼터스, 텔레매틱스