韓國 컴퓨터情報學會 論文誌
第10卷 第5號, 2005. 11.

2005-10-5-1-5

# CBDS 트리를 이용한 모바일 기기용 저용량 사전 구현에 관한 연구

정 규 철*, 이 진 관**, 장 혜 숙***, 박 기 홍****

# A Study on the Implementation of Small Capacity Dictionary for Mobile Equipments Using a CBDS tree

Kyu-Cheol Jung*, Jin-Kwan Lee**, Hye-Suk Jang***, Ki-hong Park****

## 요 약

최근 저가의 모바일 기기들이 출시되면서 단순한 휴대용 전자수첩에서 벗어나 학습과 업무용을 많은 이용되고 있으며 일반적인 응용프로그램 또한 많이 생산되고 있다. 그러나 휴대용 모바일 기기들의 단점중 하나가 저속의 소용량 시스템이라는 것이다. 이로 인해 일반 시스템에서 가동 중인 데이터베이스나 검색 알고리즘을 이식 시킬 경우 시스템의 성능을 저하되고 만다. CBDS trie알고리즘을 적용하여 모바일 환경에 맞는 저 용량 색인과 빠른 검색을 실현시킬 수 있게 되었다. 성능을 파악하기 위해 B-tree로 구현된 자바 Treeset API와 비교해본 결과 속도에서는 약간 느렸으나 저장 공간에서는 약 29%의 공간 절약을 할 수 있어 실용 가능함을 보여주고 있다.

## Abstract

Recently So far Many low-cost mobile machinery have been produced. Those are being used for study and business. But those are some weak points which are small-capacity storage and quite low-speed system. If we use general database programs or key-searching algorithm, It could decrease in performance of system. To solve those problems, we applied CBDS(Compact Binary Digital Search) trie to mobile environment. As a result we could accomplish our goal which are quick searching and low-capacity indexing. We compared with some Java classes such as TreeSet to evaluation. As a result, the velocity of searching was a little slow than B-tree based TreeSet. But the storage space have been decreased by 29 percent. So I think that it would be practical use.

▸ Keyword : CDBS, Compact binary digital search trie, 모바일 사전, 사전검색

• 제1저자 : 정규철
• 접수일 : 2005.09.12, 심사완료일 : 2005.11.09
* 군산대학교 컴퓨터정보과학과 박사과정 ** 군산대학교 컴퓨터정보과학과 박사과정
*** 군산대학교 컴퓨터정보과학과 교수, **** 군산대학교 컴퓨터과학과 교수

# Ⅰ. 서 론

최근 모바일 기기의 급속한 발전으로 단순한 게임이나 전자수첩의 기능을 뛰어넘어 일반 컴퓨터에서 수행하는 모든 일을 모바일 기기에서 수행이 가능해지고 있다. 예로 일반 학생들의 경우 PDA나 휴대폰을 이용하여 학습응용프로그램을 구동시키거나 일반 업무용프로그램으로 외부에서 업무를 보고 있다. 또한 GIS를 이용한 위치 추적 및 인터넷을 이용하여 파일의 송수신 및 웹 서핑까지 가능하게 되었다[1][2].

그러나 오랜 시간동안 보조전원만으로 시스템을 가동해야하는 휴대용 모바일 기기들의 특징으로 인해 자연히 저속의 소형 기기들이 개발되고 있다[3]. 이러한 특수성을 고려하지 않고 현재 상용되고 있는 데이터베이스를 적용할 경우 많은 처리속도나 저장 공간에 문제점을 가질 수밖에 없다. 일반적으로 PDA와 같은 모바일 프로그램에 적합한 파일 구조는 아직 특별한 연구가 진행되지 않고 있으며 프로그램개발 업체에서 자체적으로 일반 방법을 사용하여 구현하고 있을 뿐이고 데이터베이스로는 포켓피씨(PDA의 운영체제)용 데이터베이스인 cdb 파일이 있다[4]. 이 파일은 일반PC의 mdb파일을 포켓용으로 변환한 것으로 한 것이나 뒤에서 논할 실험에서 보이는 것처럼 오히려 파일의 크기가 mdb파일보다 약37%정도(2만 단어 삽입의 경우) 늘어나는 것으로 나타났다.

이를 해결하기 위해 저 용량의 모바일 기기인 휴대폰과 PDA의 검색에 적합한 검색시스템 집적한 이진트라이를 통해 구현하고자 한다.

2장에서는 이진트라이(binary trie)와 이진트리의 집적된 표현인 집적한 이진 트라이(compact binary digital search trie)에 대해 설명하고 3장에서는 집적된 이진트라이의 검색, 삽입, 삭제 알고리즘에 대해 설명하고, 4장에서는 이를 이용하여 구축한 모바일사전의 성능을 평가하고 마지막 5장에서 결론 및 향후 과제를 논한다.

# Ⅱ. Compact Binary Digital Search trie

본 장에서 논의하는 트라이(trie)는 용어 retrieval의 중간 글자를 따온 것인데 트리와 구별하기 위해 일반적으로 트라이라 부른다[5]. 트라이는 기본적으로 m개의 포인터로만 구성된 1차원 배열이며 배열 각 원소의 주소는 그 주소에 대응하는 숫자로 나타낸다. 여기서는 키를 버켓주소로 맵핑(mapping)하기위해 이진 트라이를 사용한다. 이것을 이진 디지털 탐색트리라고 불리 우는데 이는 한 키가 다른 키들을 대해 그들 전체적으로 비교하는 것이 아닌 비트단위로 탐색하며 비교를 하게 되기 때문이다.

이진 트리의 일반적인 규약에서처럼 반드시 두 자식노드을 갖는 내부 노드와 자손이 없는 외부노드로 구별하는 속성을 가진다. 내부노드는 디스크 상에서 단지 외부노드들이 버켓에 일치하는지에 대한 탐색의 식별을 위해 사용된다[6][7].

## 2.1 이진 트라이의 개요

이진 트라이에서 이진코드의 순서 열은 문자들의 변환값으로 되어있어 키 값처럼 사용되며 있다. 즉 이진트라이의 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.

〈표 1〉 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.

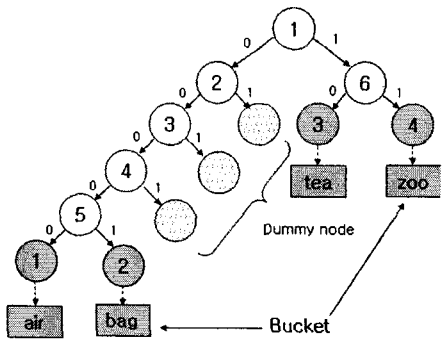


그림 1.이진트라이
Fig 1. Binary Trie

(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.

이 외부노드들을 더미노드라고 한다. 더미노드를 사용하므로 부가적인 이익들이 파생된다. 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 CBDS 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[8]. 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.

이런 경우 이진트라이는 (그림 1)처럼 모두 한 노드에서 두 포인터를 갖는 일반적인 트리주소처럼 구현된다. 저장 공간은 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.

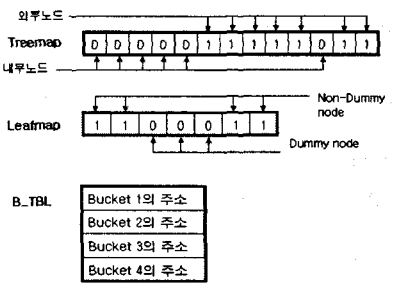


그림 2 그림1에 대해 집적한 이진트라이
Fig 2. Compact binary trie of Fig 1

## III. 알고리즘

집적한 이진트라이를 이용한 검색알고리즘은 오른쪽으로 진행하는 treemap의 첫 번째 비트로부터 비트단위로 처리된다. 즉 검색은 전위순회(pre-order)한다.

검색은 treemap의 비트위치가 '0'을 경우 비트를 전진시키면서 왼쪽 서브트리를 처리한다. 만일 오른쪽으로 서브트리로 나아가고자 할 경우 왼쪽 서브트리를 반드시 건너뛰어야(skip) 한다. 이럴 경우 왼쪽 서브트리에서 외부노드의 수가 내부노드의 수보다 하나 더 많다는 이진트리 속성에 이용하게 된다. 이 처리는 1비트의 수가 0비트의 수보다 하나 더 많은 때까지 treemap의 비트위치를 전진시키면서 왼쪽서브트리의 마지막 위치를 발견한다.

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

아래의 변수들을 사용하여 검색 알고리즘을 표현하고 있다.

### 3.1 검색알고리즘

**[검색 알고리즘]**

- keypos : 검색비트 문자열에서 현재위치를 나타내는 포인터
- treepos : treemap에서 현재위치를 나타내는 포인터
- leafpos : leafmap에서 현재위치를 나타내는 포인터
- Input : 검색하고자 하는 키;
- Output : 키가 발견되면 True 아니면 False;

**[Step-1]** : {초기화}

- keypos =1 , treepos =1 , leafpos = 1;

**[Step-2]**: {키의 bit값 검증}

- 만일 keypos에 의해 가리키는 포인터의 비트 =1 이면 Step-3 아니면 Step-4;

**[Step-3]**: {leaf sub-tree 건너뜀}

- '1'비트의 수가 '0'비트의 수보다 1개더 많을 때 까지 전진 그리고 Step-4 처리;

**[Step-4]**: {간선 순회}

- treepos를 하나 전진하고 treepos가 지시하는 treemap의 비트가 "0"(내부노드이면) 하나에 의해 keypos를 '1' 비트 전진한 후에 Step-2로 돌아가고 아니면 Step-5를 처리;

**[Step-5]**: {일치하는 외부노드의 더미인지 아닌지 검증}

- 첫 번째 비트부터 treepos까지 treemap에 1비트의 수에서 leafpos를 고정시키고 leafpos에 의해 가리켜지는 leafpos에 비트가 '1'이면 Stop1-6으로 아니면 실패;

**[Step-6]**: {B_TBL로부터 일치하는 버켓 주소의 획득}

- 첫 번째 비트부터 leafpos까지 leafmap에 1비트의 수를 세고 센 수가 가리켜지는 B_TBL의 슬롯으로부터 버켓 주소를 얻는다.

**[Step-7]**: {일치하는 버켓내의 키의 검색}

- 만일 얻어진 버켓주소에 의해 가리켜진 버켓이 키를 포함하면 True 아니면 FALSE;

**[알고리즘 끝]**

예를 들면 (그림 2)에서 'tea'를 검색한다고 할때, tea의 첫 번째 bit가 '1'이므로 루트는 노드 2를 가진 왼쪽 서브트리를 Skip하기 위해 treepos는 10번째 bit를 전진 되어지고 노드 6에서 처리에 의해 11번째로 이동한다.

두 번째 bit가 '0'이므로 비트위치를 12번째로 옮겨간다.

그런 후 treemap의 12번째 값이 '1'이므로 leafmap을 검증한다. 그런데 첫 비트에서 12번 비트까지 treemap의 1비트의 수는 6이고 leafmap의 6번째 비트값은 1이이므로 더미가 아닌 외부노드를 발견할 수 있다. 첫 번째부터 6번째까지 leafmap의 1비트의 수는 3이다.

따라서 버켓 3의 주소는 B_TBL의 3번째 슬롯으로부터 포함되어 있다는 것을 알 수 있고 바로 키가 발견된다.

### 3.2 삽입 알고리즘

새로운 키를 삽입하고 할 경우 leafmap을 확인하여 leafpos가 가리키는 값이 0이면 더미노드 이므로 바로 B_TBL에 키를 할당하지만 '1'일 경우 해당 버켓에 다른값이 들어있기 때문에 외부노드에 새로운 서브트리를 추가시켜야한다.

예를 들면 (그림 1)에서 'eat(00100 00000 10011)'를 삽입하고자 할 때, eat의 첫 번째 비트가 '0'이고 treemap의 첫 비트도 '0'이므로 treepos를 2로 바꾸고 keypos도 2

로 변경한다. keypos 2의 값도 '0'이므로 treepos도 3로 바꾸고 keypos를 3으로 바꾼다. 그런데 keypos가 가리키는 값이 1이므로 treepos를 9번 비트로 전진시킨다. 왜냐하면 treepos의 3번 비트에서 시작하여 '1'의 개수가 하나 더 큰 비트 위치가 9번 비트이기 때문이다.

**〔삽입 알고리즘〕**

**〔Step-1〕**: 〔초기화〕
- treemap = 011, leafmap = 00,
  B_TBL = null

**〔Step-2〕**: 〔키의 bit값 검증〕
- 만일 keypos에 의해 가리키는 포인터의 비트 가'1' 이면 Step-3 아니면 Step-4;

**〔Step-3〕**: 〔leaf sub-tree 건너뜀〕
- '1'비트의 수가 '0'비트의 수보다 1개더 많을 때 까지 전진 그리고 Step-4;

**〔Step-4〕**: 〔간선 순회〕
- treepos를 하나 전진하고 treepos가 지시하는 treemap의 비트가 "0"(내부노드이면) 하나에 의해 keypos를 '1'비트 전진한 후에 Step-2로 돌아가고 아니면 Step-5;

**〔Step-5〕**: 〔일치하는 외부노드의 더미인지 아닌지 검증〕
- 첫 번째 비트부터 treepos까지 트리맵에 1비트의 수에서 leafpos를 고정시키고 leafpos에 의해 가리켜지는 leafpos에 비트가 "1"이면 S-6로 가고 아니면 Step-7;

**〔Step-6〕**: 〔 treemap에 서브 트리추가〕
- treepos가 가리키는 '1'의 값을 '011'로 치환하고 leafmap에 한 비트를 추가하고 keypos를 한비트 전진한후 step-2로 RETURN;

**〔Step-7〕**: 〔B_TBL에 키 할당〕
- leafpos가 가리키는 값이 '0'(더미노드) 이므로 B_TBL에 새로운 키 할당

**〔알고리즘 끝〕**

이후 treepos 9가 가리키는 값이 '1'이므로 treemap의 1에서 9 비트까지의 1의 개수가 4이므로 leafpos를 4로 설정하고 가리키는 값이 '0(더미노드)'이므로 B_TBL〔leafpos〕에 킷값을 할당한다.

## 3.3 삭제 알고리즘

집적한 이진트라이에서 키의 삭제는 검색키의 삭제 이후에 실행되어진다.

맨 먼저 검색된 삭제키에 의해 키가 이미 저장된 버켓을 확인하는데 이 버켓을 삭제 버켓이라고 부른다. 만일 삭제 버켓이 발견되면 키는 버켓으로부터 제거 된 후 삭제처리는 버켓내용에 따라 아래 3가지 경우 중 한 가지가 된다.

(1) 삭제버켓이 부분적으로 차있다.
(2) 삭제버켓이 비어있다. 그리고 삭제버켓과 연결된 외부노드의 형제노드는 더미노드가 아니다.
(3) 삭제 버켓이 비어있다. 그리고 삭제버켓과 연결된 외부노드의 형제노드는 더미노드이다.

먼저 첫 번째 경우는 바뀌지 않고 삭제 처리를 마칠 수 있다.

두 번째 경우 형제 노드가 더미노드가 아니거나 형제 노드가 내부노드이면 삭제 버켓과 연결된 외부 노드를 더미노드로 수정되고 삭제버켓은 제거된다.

수정은 '1'로부터 '0'까지 leafmap에서 일치하는 비트의 변경만으로 마치게 된다. 세 번째 경우 합병처리가 반드시 필요한데 이것은 새로운 더미 외부노드로 두 외부노드는 삭제된 버켓과 연결된 외부노드와 형제 노드이다.(더미 외부노드)

합병은 (그림 3)처럼 다음의 처리에 의해 실행된다.

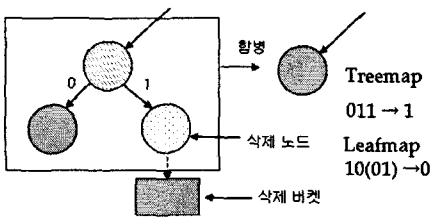


그림 3. 삭제 처리
Fig 3. Deletion process

제일 먼저 treemap에서 삭제 노드를 포함하는 가장 작은 서브트리를 표현한 비트 '011'은 단일 신규더미 노드를 표현한 '1'로 바뀐다.

다음으로 leafmap상에서 삭제노드로 표현된 '10' 또는 '01'과 형제 더미노드는 단일 신규더미가 가리키는 '0'비트로 바뀐다.

### 3.4 구현 및 고찰

사전을 구축하기 위해 형태소 분석을 이용하였다. 구축환경은 Pentium4-2.6Ghz 프로세서와 Main memory 1Gbyte환경에서 J2SE 5.0을 이용하여 CBDS tree를 구축하였으며, 검색환경은 Intel PXA 255 프로세서와 32MB의 Main memory, 256MByte의 SD memory를 가진 PocketPC 2002 iPaq PDA에서 구동될 수 있도록 Personal Java로 구현하였는데 이유는 J2ME의 CDC와 CLDC에는 필요한 API(파일 액세스관련 API)가 없기 때문이다[9].

사전은 SD메모리에 두고 인덱스만 메인 메모리에 올려놓고 검색을 실시하였다. 성능의 우수성을 실험하기 위해 같은 환경에서 B-Tree로 구현된 Treeset API[10]에 같은 데이터를 등록시키고 검색시간에 대해 평가를 실시하였다.

먼저 5만개단어 중 100개를 무작위로 추출하여 검색하면서 검색 시간과 필요 공간을 계산하였다.

(그림 4)는 100개에 대한 검색 시간을 그래프로 표현한 것이다. 최고 113ms에서 최저 104ms까지의 검색 시간을 나타내고 있다. 아래는 자바에서 기본적으로 제공하는 Treeset API를 이용하여 검색을 실시한 것인데 검색 속도 면에서 최적화된 API와 거의 비슷한 속도를 내고 있다.

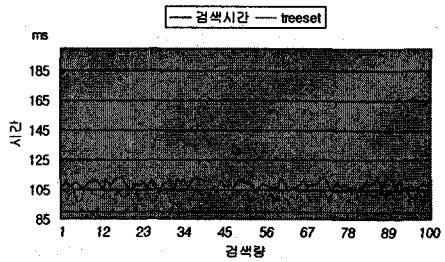


그림 4. 검색소요 시간
Fig 4. Search necessary time

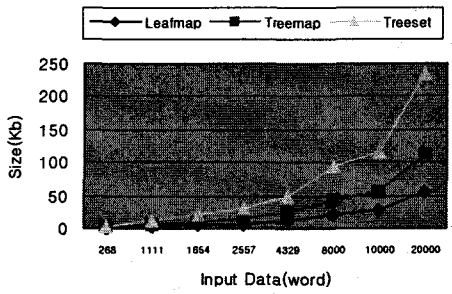


그림 5. TreesetAPI와 크기 비교
Fig 5. Compared Size with Treeset API

(그림 5) 는 기억장치의 필요량을 그래프로 표현한 것이다. Treeset API는 자료의 양이 많아지면서 급격하게 용량이 커지는 것을 볼 수 있다. 반면 접적한 이진트라이에서는 공간을 두개로 분리하는 대신에 Treeset 보다 적은 공간을 차지하는 것을 볼 수 있다.

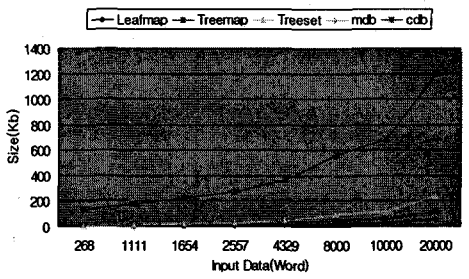


그림 6. 데이터베이스와의 비교
Fig 6. Compare Size with Database

데이터 20,000개를 기준으로 볼 경우 Treeset에서 요구하는 공간은 234KB인 반면 leafmap은 56KB, Treemap은 112KB로 약 29%정도 공간 절약의 효과를 거두는 것으로 나타났다. 또한 (그림 6)을보면 일반 PC에서 사용되는 데이터베이스인 MS-ACCESS의 MDB파일과 PocketPC용으로 변환한 CDB파일과 크기를 비교하였다. 도표의 값은 최대한 크기를 줄이기 위해 인덱스 기능을 없앤 크기이다. 그런데 특이하게도 PocketPC용으로 변환하였을 때 일정크기(2만 단어 삽입의 경우) 이상이 되면 MDB보다 오히려 37% 더 커지는 현상이 발생하여 많은 용량의 데이터 처리에는 부적합한 것으로 나타나고 있다. 제안한 방법의 데이

터베이스와 비교할 때 19%의 공간만 필요로 하므로 매우 효율적임을 보여주고 있다.

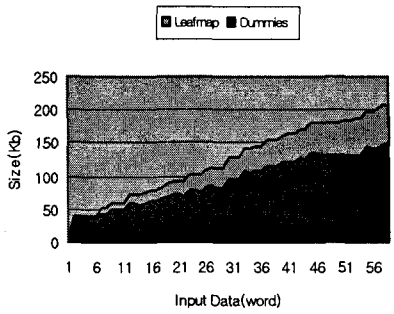


그림 7. 더미노드의 변화
Fig 7. a Change of Dummies

그런데 (그림 7)을 보면 사용하지 않는 더미 노드가 3/4 정도를 차지 할 정도로 많이 나타났다. 이는 이진 트리의 속성을 맞추기 위한 것이고 보조기억장치 공간을 전혀 필요로 하지는 않지만 검색의 효율성을 떨어트릴 가능성이 있으므로 향후 더미노드의 수를 줄이기 위한 노력을 해야 할 것이다.

## V. 결 론

본 논문에서는 집적한 이진 트라이를 이용하여 모바일 환경에서도 적은 용량으로 빠르게 검색할 수 있는 시스템을 제안하였다. 자바에서 일반적으로 사용되는 treeset API에 비해 속도에서는 약간 떨어지는 것으로 나타났으며 저장 공간에서는 약 29%정도 적게 들고 데이터베이스에 비해서는 86%정도의 공간이 절약되는 것으로 나타나 실용 가능성이 뛰어나다고 판단된다. 그러나 더미노드가 상당히 많이 생성되어 향후 더미노드를 줄이기 위한 노력과 키 길이를 맞추기 위해 해싱함수를 이용하는데 걸리는 시간을 갖지 않고 바로 이진 트라이로 구축하는 방법을 모색한다면 보다 뛰어난 시스템이 될 것이다.

## 참고문헌

[1] 이기영,노경택, "Mobile Gis를 위한 클라이언트 인터페이스의 설계 및 구현", 한국 컴퓨터정보학회 논문지, 7권 4호, 2002

[2] 이유리, 박동규, "모바일 환경에 적합한 헬스 케어 정보시스템에서의 역할기반 접근제어", 한국컴퓨터정보학회 논문지, 10권 3호, 2005

[3] 리처드 헌터(2003), 유비쿼터스 공유와 감시의 두얼굴, 21세기 북스

[4] 안원국(2005), C#.NET Mobile Programming, Yongjin.com

[5] Aho, A. V(1984), Data structures and algorithms, Addion-Wesley

[6] Gonnet, G. H(1992), Information retrieval-data structures and algorithm, Prentice Hall

[7] Aoe,J, Park K(1006), A trie compaction algorithm for a large set of keys, IEEE Trans. on Knowledge and Data Engineering, 8(3)

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

[9] 정영오(2002), 모바일자바 PDA 핸드폰프로그래밍, PC BOOK

[10] The Java Tutorial, http://java.sun.com/docs/books/tutorial/index.html

## 저 자 소 개


**정 규 철**

1999 군산대학교 컴퓨터과학과(석사)
2000~현재 군산대학교 컴퓨터과학과 박사수료
〈관심분야〉 정보검색, 유해차단, 텔레매틱스

**이 진 관**

1999 군산대학교 컴퓨터과학과(석사)
2002~현재 군산대학교 컴퓨터과학과 박사과정
〈관심분야〉 정보검색, 음성인식, 텔레매틱스

**장 혜 숙**

2000 군산대학교 컴퓨터과학과(석사)
2004~현재 군산대학교 컴퓨터과학과 박사과정
〈관심분야〉 자연어처리, 음성인식, 텔레매틱스

**박 기 홍**

1995 일본 토쿠시마대학 지능정보과학과(박사)
1987~현재 군산대학교 컴퓨터과학과 교수
2004~현재 NURI사업 텔레메틱스 인력양성 사업단(군산대) 단장
〈관심분야〉 자연어처리, 정보검색, 텔레매틱스