절단검색을 지원하는 전자사전 구조
(An Electronic Dictionary Structure supporting Truncation Search)

김 철 수 †
(Cheolsu kim)

요 약 역화일을 파일구조로 이용하는 정보 검색 시스템에서는 검색자가 검색할 분야의 완전 단어를 알고 있어야 검색이 가능하다. 그러나 검색자가 완전 단어가 아닌 단어의 부분 문자열을 알고 있는 경우가 많다. 이럴 경우 부분 문자열을 포함하는 색인어들을 검색할 수 있다면 관련 문서들을 검색할 수 있다. 또한 검색된 문헌 수가 너무 적을 경우 부분 문자열을 포함하는 단어를 색인어로 가지는 모든 문서들을 검색하기 위한 방법이 필요하다. 이런 요건들을 충족시키기 위해서는 사용자는 용어 절단 방법을 이용하여 질의어를 구성할 수 있어야 하고, 검색 시스템은 절단 검색을 지원할 수 있는 전자 사전이 필요하다.

본 논문에서는 절단검색을 효율적으로 지원할 수 있는 전자 사전 구조를 설계하고 구현한다. 이 전자사전은 저장된 단어 수에 관계없이 주어진 한 개의 단어 검색 시간 및 역 문자열로 구성된 단어 검색 시간이 빠르고 일정하다. 절단검색을 효율적으로 지원하기 위하여 트라이 구조를 이용하였으며, 빠른 검색 시간을 지원하기 위해 배열을 이용한 방법을 사용하였다. 절단된 용어의 검색 과정에서 확장할 문자열의 길이를 최소화하여 검색 시간을 줄였다.

키워드 : 정보검색시스템, 절단검색, 트라이, 전자사전

Abstract In an Information Retrieval System(IRS) based on an inverted file as a file structure it is possible to retrieve related documents when the searcher know the complete words of searching fields. However, there are many cases in which the searcher may not know the complete words but a partial string of words with which to search. In this case, if the searcher can search indexes that include the known partial string, it is possible to retrieve related documents. Futhermore, when the retrieved documents are few, we need a method to find all documents having indexes which include known the partial string. To satisfy these requests, the searcher should be able to construct a query formulation that uses the term truncation method. Also the IRS should have an electronic dictionary that can support a truncated search term.

This paper designs and implements an electronic dictionary(ED) structure to support a truncation search efficiently. The ED guarantees very fast and constant searching time for searching a term entry and the inversely alphabetized entry of it, regardless of the number of inserted words. In order to support a truncation search efficiently, we use the Trie structure and in order to accommodate fast searching time we use a method using array. In the searching process of a truncated term, we can reduce the searching time by minimizing the length of string to be expanded.

Key words : Information Retrieval System, truncation search, Trie, Electronic Dictionary

1. 서 론

전 세계에 분포되어 있는 엄청난 양의 문서들 가운데 사용자가 필요로 하는 문서들을 신속하고 정확하게 찾을 수 있도록 하기 위해서는 방대한 양의 문서들을 양질의 소량 문서들로 줄여주는, 필터링 역할을 하는 도구가 필요하다. 이러한 필터링 역할을 수행하는 일종의 도구가 정보 검색 시스템(IRS: Information Retrieval System[1][2])이다.

IRS는 검색 대상이 될 수 있는 문서들을 대표할 수 있는 의미 있는 단어(색인어)[3][4]들을 추출하여 검색하

† 정 회 원 : 서남대학교 전산정보학과 교수
chskim@tiger.seonam.ac.kr
논문접수 : 2001년 6월 11일
심사완료 : 2002년 10월 18일

기 용이한 형태로 저장해 두었다가, 검색 요구가 발생하면 질의어에 포함된 검색어들을 색인어들이 저장된 데이타베이스나 파일을 참조하여 정해진 검색 모델[5]의 처리 절차를 거쳐 선별된 문헌들을 보여준다[6][7][8][9].

IRS를 구축하기 위해서는 색인어들을 저장·검색하기 위한 전자 사전이 필수적이며, 빠른 검색을 위해서 역파일(inverted file) 구조를 많이 이용한다. 역파일 구조는 검색 시간은 빠르지만 검색하고자 하는 단어를 정확하게 알고 있어야 검색이 가능하므로 검색할 단어의 문자열을 부분적으로 알고 있는 경우에는 관련 문헌을 검색할 수 없는 문제점이 있다. 이 문제를 해결할 수 있는 방법으로 검색자가 부분적으로 알고 있는 문자열을 이용하여 질의어를 구성하고, IRS는 질의어에 나타난 부분 문자열을 포함하는 색인어들을 가지는 문헌을 검색해 주는 것이다. 즉, 검색자가 질의어에 사용할 완전 단어를 알지 못할 경우 용어 절단 방법을 이용하여 질의어를 구성하고, 시스템은 질의어에서 제공하는 절단된 용어를 확장하여 확장된 검색어를 포함하는 문헌들을 검색할 수 있는 절단 검색(truncation search) 기능을 지원해 주어야 한다. 절단 검색은 검색자가 초보자이거나, 경험이 많고 정보 검색에 익숙한 사람이더라도 새로운 분야의 관련 문헌을 검색할 경우 검색할 완전 단어를 알지 못하는 경우가 많고, 검색된 문헌수가 너무 적어 관련 문헌을 보다 광범위하게 검색하고자 하는 경우에 매우 유용하다. IRS가 절단 검색을 지원함으로 써 IRS를 보다 쉽고 편리하게 이용할 수 있고 성능도 향상시킬 수 있다. 일부 검색 엔진이 절단 검색을 지원하지만 절단 검색을 제대로 지원하지 못하고 있다. 예를 들어 "정보*시스템" 형태와 같이 질의어를 주었을 때 "정보" 문자열과 "시스템" 문자열을 동시에 포함하는 색인어가 나타난 문서들만 검색해 주어야 하지만 "정보" 문자열만 포함하는 색인어가 나타난 문서나 "시스템" 문자열만 포함하는 색인어가 나타난 문서까지 찾아주므로 써 원하지 않는 문서까지 검색해 주는 단점이 있다.

질의어에 포함된 단어의 검색 과정은 사전 구조와 밀접하게 관련되어 있으므로 사전 구조는 검색 기능 및 성능을 좌우하는 중요한 요소이다. IRS를 위한 전자 사전은 빠른 검색 시간을 가질 뿐만 아니라 응용 환경을 잘 지원해 주어야 한다. 즉, 질의어에 완전 단어가 검색어로 주어지면 검색어들을 빠르게 검색할 수 있어야하고, 문서들이 수시로 발생하고 사라지므로 단어들의 추가·삭제가 자유로워야 한다. 또한 완전 단어가 아닌 절단된 용어가 질의어로 주어지더라도 주어진 절단된 용어를 완전한 단어 형태로 확장하여 검색할 수 있어야 한다. 예를 들어 "정보*"를 질의어로 주었을 때 사전에 저장된 색인어 중 "정보"로 시작하는 모든 색인어를 검색해 주어야 한다.

본 논문에서는 역파일 구조를 이용하는 정보검색 시스템에서 질의어에 포함된 검색어가 완전 단어일 때는 빠른 검색 시간을 지원하고, 절단된 용어는 절단 검색을 위해 주어진 용어의 부분 문자열을 정확하고 신속하게 확장하여 검색해 줄 수 있는 사전 구조를 설계하고 구현한다. 본 사전은 트라이[10] 구조를 이용하여 삽입된 단어 수에 관계없이 검색할 단어의 문자열 길이에만 의존하여 검색 시간이 결정되고, 절단된 용어를 검색할 경우에는 확장할 문자열의 길이를 최소화하여 확장 검색 시간을 줄일 수 있으며 확장 검색 시간은 확장되는 문자열의 길이에 따라 검색 시간이 결정된다.

2. 관련연구

단어 검색은 순차 탐색, 이진 탐색, 트리 구조, 해싱 구조[11][12], 트라이 구조[13][14] 등을 이용할 수 있다. 순차 탐색은 절단 기호가 포함된 검색어를 확장하여 관련된 색인어들을 검색할 수 있지만 저장된 색인어 숫자가 증가함에 따라 검색 시간이 비례하여 길어지는 단점이 있다. 트리 구조는 순차 탐색에 비해 검색 시간은 줄일 수 있지만 완전 단어에 대해서만 검색이 가능하다. 이 구조는 절단된 용어의 문자열을 확장할 수 없으므로 절단 검색이 불가능하다. B-트리는 인덱스 세트와 순차 세트의 두 부분으로 구성되어 트리 구조의 특징과 순차 구조의 특징을 모두 가지고 있다. 절단 기호가 포함되지 않은 완전 단어는 인덱스 세트를 통하여 검색하고, 절단된 기호가 포함된 절단된 용어는 순차 세트를 통해 문자열 확장은 가능하지만 삽입 단어 수에 비례하여 문자열 확장을 위한 검색 시간이 증가한다. 해싱 구조는 검색어가 주어지면 해싱 함수를 통하여 관련 문서를 빠르게 찾을 수 있지만 절단 기호가 포함된 검색어의 문자열 확장은 불가능하다. 트라이 구조는 트리 구조의 특수한 형태로 트리 구조는 단어 전체에 의해 분기할 다음 위치가 결정되지만 트라이 구조는 단어를 구성하는 문자열 일부분에 의하여 분기할 위치가 결정되는 구조로써 단어들의 앞부분 문자열이 동일한 문자열(prefix)을 공유하는 특징이 있으므로 절단된 용어의 문자열 확장이 가능할 뿐만 아니라 빠른 검색이 가능하다[14][15].

2.1 트라이

트라이[10]는 트리 구조의 특수한 형태로, 트리 구조는 다음 레벨로의 분할 및 이동이 단어 전체에 의해 결정되지만 트라이는 단어 전체가 아닌 단어를 구성하는

문자에 의해 이동할 위치가 달라진다. 즉, 현재 레벨 i에서 다음 레벨 $j(j=i+1)$로 분기할 때 한 개 단어 전체를 비교하여 분기하는 것이 아니라 한 개의 단어를 구성하는 문자열 중 $i-1$ 번째 문자에 의해 전이된 i번째 레벨의 노드 N_i에서 단어의 i번째 문자에 의해 j번째 레벨의 노드 N_j 위치가 결정된다. 한 개의 단어는 루트 노드에서 시작하여 단말 노드에 도달하는 과정에서 분기할 때 이용된 문자들의 결합으로 구성된다. 따라서 사전에 저장되는 단어들 중 동일한 prefix를 가지는 단어들은 같은 경로에 저장되고 검색된다. 검색을 위한 트라이 깊이는 삽입된 단어 숫자에 관계없이 단어를 구성하는 문자열의 길이에만 의존하므로 검색 시간이 빠르고 단어의 삽입 삭제가 자유로운 특징이 있다. 단어 집합 S1 = {보정, 정신, 정보, 정보검색, 정보검색시스템, 정보통신, 최신}의 트라이 구조를 상태 전이도로 표현하면 그림 1과 같다.

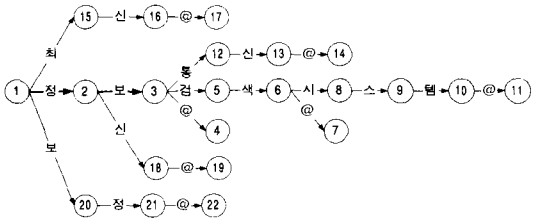

그림 1 상태 전이도로 표현한 트라이 구조

그림 1에서 절단된 용어 "정보*"에 대한 문자열 확장은 루트 노드 ①에서 문자 '정'에 의해 두 번째 레벨의 노드 ②로 분기하고, 노드 ②에서 문자 '보'에 의해 세 번째 레벨의 노드 ③으로 분기한다. 노드 ③에서 다음 문자가 절단 기호 '*'이므로 "정보"로 시작하는 모든 단어는 이 노드 ③에서 분기되어 단말 노드에 도달 가능한 단어들의 집합이다. 즉, "정보"로 시작되는 단어들의 문자열 확장은 노드 ①, ②, ③을 거쳐 단말 노드에 도달 가능한 단어들의 집합이다. 절단된 용어의 확장 과정에서 관련 없는 단어 참조 없이 문자열 확장이 이루어지는 과정을 살펴보자. 노드 ①에서 문자 '정'에 의해 노드 ②로 전이가 이루어질 때 노드 ⑳을 거쳐야하는 단어 "보정"과 노드 ⑫를 거쳐야하는 단어 "최신"이 검색 대상에서 제외되고, 노드 ②에서 문자 '보'에 의해 노드 ③으로 전이가 이루어 질 때 노드 ⑱을 거쳐야하는 단어 "정신"이 검색 대상에서 제외된다. 그리고 다음 문자가 절단기호 '*'이므로 노드 ③에서 단말 노드에 도달 가능한 "정보", "정보검색", "정보통신", "정보검색시스템" 4개의 단어들을 검색하게 된다. 이는 트라이 구조가 단어들의 prefix를 공유하는 특징이 있어 검색에 필요한 최소의 노드만을 참조하여 문자열 확장을 효율적으로 지원할 수 있음을 잘 보여준다.

트라이 구조를 구현하기 위한 일반적인 방법은 리스트 구조를 이용하는 방법이다. 그러나 이 방법은 삽입된 단어 수가 증가함에 따라 연결되는 형제 노드들이 증가하여 검색 시간이 길어지는 단점이 있다. 이런 단점을 해결하기 위한 방법 중 한가지는 두 개의 1차원 배열을 이용하는 방법이다[14][15]. 이 방법은 삽입된 단어 숫자에 관계없이 단어의 길이에만 의존하여 검색 시간이 결정되므로 검색 속도가 빠르다.

2.2 배열을 이용한 트라이 구조

배열을 이용한 트라이 구현 방법[14]은 그림 1과 같이 상태 전이도로 표현된 트라이 구조를 저장하기 위해 두 개의 정수를 이용하여 한 개의 노드를 표현한다. 이 방법은 삽입된 단어 수에 관계없이 한 개의 단어를 검색하기 위해 소요되는 시간이 일정하며, 단어의 삽입과 삭제가 자유로운 특징을 가진다. 키(key)들의 집합 K = $\{k_1, k_2, k_3, \ldots, k_n\}$에 대한 트라이 구조를 다음과 같은 5개의 튜플 <S, I, g, s1, F>를 가지는 디지털 탐색 트리로 정의한다.

디지털 탐색 트리 M = <S, I, g, s1, F>

S : 트라이에 존재하는 상태(또는 노드)들의 유한 집합

I : 단어를 구성하는 기호 또는 문자들의 유한 집합

g : S × I에서 S ∪ { fail }로 사상시키는 전이 함수

s1 : 초기 노드 또는 S 내의 루트 노드

F : 단말 노드의 유한 집합 (F ⊆ S)

또한 디지털 탐색 트리를 이용한 트라이 구성을 위해 다음과 같이 정의한다.

[정의 1] x ∈ I이고, A와 Ω를 입력 기호들로 구성된 임의의 문자열이라 할 때, S 내의 노드들을 다음과 같이 정의할 수 있다.

- 분할 노드(SP) : 임의의 키 $k_i(1 \le i \le N)$ $Ax\Omega$에 대해, 전이 함수g(s1, Ax) = sr을 만족하는 노드 sr이 존재하고, N개의 서로 다른 단어로 구성된 집합 K의 다른 키 kj($1 \le j \le N$, $i \neq j$)와 $Ax\Omega$를 구별하기 충분하다면 sr을 분할 노드라 부른다.
- 다중 노드(SM) : 루트 노드에서 각 분할 노드까지의 경로 상에 존재하는 모든 노드들을 다중 노드라 부른다.
- 단일 노드(SI) : 각각의 분할 노드에서 단말 노드까지의 경로 상에 존재하는 모든 노드들을 단일 노드라 부른다.

2.3 두 개의 트라이를 이용한 방법

트라이 구현 방법 [14]는 검색 시간은 빠르지만, 단일 노드에서 나타날 수 있는 공통된 뒷부분 문자열(suffix)들을 중복하여 저장하는 단점이 있다. 그림1에서 단어 "최신"의 '신', "정보통신"의 '신'과 단어 종료기호 '@'는 중복하여 저장된다. 이런 단점을 개선하여 공통 prefix뿐만 아니라 공통된 suffix도 한 번만 표현하는 방법이다. [14]에서와 같이 루트 노드에서 분할 노드까지의 경로에 존재하는 모든 노드들에 대해 트라이(왼쪽-트라이)를 구성한 후 단일 노드들에 대한 문자열을 역 방향 문자열(역 문자열)로 변환한 후, 변환된 역 문자열들을 동일한 방법으로 또 하나의 트라이(오른쪽 트라이)를 구성하여 두 개의 트라이를 연결하여 구성하는 방법이다[15][16]. 그림 1을 두 개의 트라이를 이용하여 표현한 구조는 그림 2와 같다.

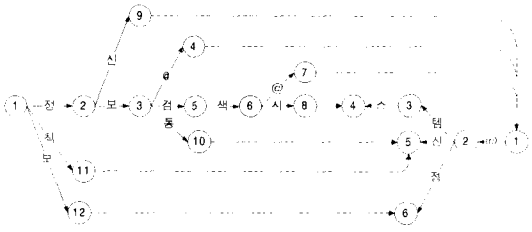

그림 2 두 개의 트라이를 이용한 구조

그림 2는 공통된 suffix를 공유할 수 있도록 한 방법으로 기억 장소는 절약할 수 있지만 역 문자열 단어 검색은 불가능하다. 역 문자열 단어 검색이 가능하기 위해서는 오른쪽 트라이의 루트 노드 ①에서 시작하여 왼쪽 트라이의 루트 노드 ①에 도달할 수 있어야 한다. 그림 2에서 보는 것처럼 왼쪽 트라이의 단말 노드는 오른쪽 트라이의 특정 노드로 전이한다. 이 때 오른쪽 트라이를 구성하는 임의의 노드로 전이하는 왼쪽 트라이의 단말 노드 수가 2 이상일 때 즉, 오른쪽 트라이 내의 임의의 노드로의 진입 차수가 2 이상 일 때 진입 차수가 2 이상인 그 노드에서 왼쪽 트라이로 전이할 노드를 결정할 수 없으므로 역 문자열 단어는 검색할 수 없다. 예를 들어 왼쪽 트라이의 노드 ④, ⑦, ⑨는 오른쪽 트라이의 노드 ①로 전이하고, 왼쪽 트라이의 노드 ⑩과 ⑪은 오른쪽 트라이의 노드 ⑤로 전이한다. 즉, 오른쪽 트라이 노드 ①과 ⑤는 왼쪽 트라이에서 전이되는 진입차수가 3과 2인 노드이다. 역 문자열 단어 검색을 위해서는 오른쪽 트라이의 루트 노드 ①에서 왼쪽 트라이의 노드 ④, ⑦, ⑨로, 오른쪽 트라이의 노드 ⑤에서 왼쪽 트라이의 노드 ⑩과 ⑪로 각각 전이가 이루질 수 있어야 한다. 그러나 전이될 노드를 결정할 수 없는 문제가 발생한다. 이처럼 오른쪽 트라이의 노드 중 진입 차수가 2 이상이 노드가 존재할 때 역 문자열 검색이 불가능하다.

3. 새로운 전자 사전 구조의 설계 및 구현

3.1 설 계

검색어의 왼쪽이 절단(전절단)된 경우 단어의 앞부분 문자열 확장, 오른쪽이 절단(후절단)된 경우 단어의 뒷부분 문자열 확장, 중간 부분이 절단(중앙절단)된 경우 단어의 중간 문자열 확장을 지원하기 위해서는 새로운 사전 구조가 필요하다. 본 절에서는 다양한 용어 절단 검색을 지원하기 위한 새로운 사전 구조를 설계한다. 그림 1과 같은 트라이 구조는 후절단(right truncation)된 용어("정보*")의 뒷부분 확장은 가능하지만 전절단(left truncation)된 용어("*검색")의 단어의 앞부분 확장은 불가능하므로 단어의 앞부분 확장을 위해서는 새로운 방법이 요구된다. 단어의 앞부분 확장을 위해서 단어의 역 문자열로 구성된 트라이 구조를 구성할 수 있다. 단어 집합 S1의 역 문자열 단어들을 트라이 구조로 표현하면 그림 3과 같다.

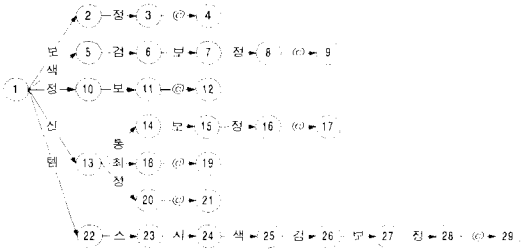

그림 3 역 문자열 트라이 구조

그림 3의 구조는 단어의 앞부분 확장("*검색")은 가능하지만 단어의 뒷부분 확장("정보*")은 불가능하다. 따라서 다양한 형태의 문자열 확장을 동시에 지원하기 위한 방법이 필요하다. 단어의 앞부분 문자열 확장과 뒷부분 문자열 확장을 동시에 지원하기 위하여 그림 1, 3을 단순히 통합한 구조는 그림 4와 같다.

그림 4의 구조는 순방향 문자열로 구성된 단어와 역 문자열로 구성된 단어들을 통합 저장하여 앞부분 확장과 뒷부분 확장이 가능하지만 한가지 큰 결점이 있다.

순방향 문자열 단어와 역 문자열 단어 삽입 과정에서 임의의 두 개의 서로 다른 단어 K_i, K_j ($i \neq j$)에 대해 K_i의 순방향 문자열 F_i와 K_j의 역 문자열 R_j의 문자열 순

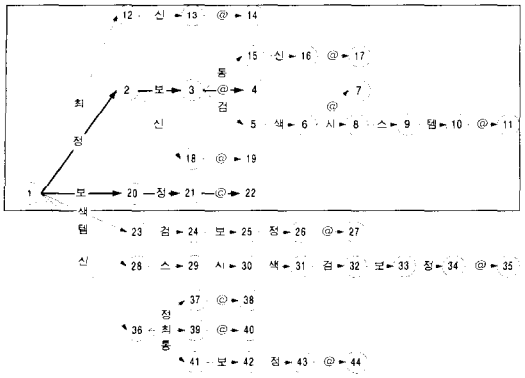

그림 4 그림 1과 3의 단순 통합

서가 일치하는 경우(F_i=R_j)가 발생한다. 이럴 경우 역시 Ki의 역 문자열 R_i와 K_j의 순방향 문자열 F_j의 문자열 순서도 일치(R_i=F_j)한다. 이 때 F_i와 R_j 그리고 R_i와 F_j는 각각 서로 다른 단어이지만 한 개의 동일 단어로 인식하여 동일 위치에 저장되는 것이다. 예를 들어 "정보"와 "보정"은 완전히 서로 다른 단어이므로 각각 다른 위치에 저장되어야 하지만 그림 4와 같이 통합하여 유지할 경우 "정보"와 "보정"의 역 문자열 "정보"가 동일 위치에 "보정"과 "정보"의 역 문자열 "보정"이 동일 위치에 저장된다. 주어진 단어들의 집합 S1을 먼저 순방향으로 삽입하면 그림 4의 사각형 내의 구조가 된다. 역 문자열 단어를 삽입하기 위하여 "정보"의 역 문자열인 "보정"의 삽입 과정을 보자. 노드 ①에서 문자 '보'에 의해 노드 ⑳으로, 노드 ⑳에서 문자 '정'에 의해 노드 ㉑로, 노드 ㉑에서 '@'에 의해 ㉒로 진행하여 이미 순방향 단어 "보정"이 저장되어 있음을 알 수 있다. 이처럼 "정보"의 역 문자열과 순방향 단어 "보정"의 관련 정보가 동일한 위치에 저장되는 단점이 있다. 이런 현상은 임의의 단어 K_i의 순방향 문자열과 다른 단어 K_j(i≠j)의 역 문자열 순서와 일치하는 단어가 존재할 때 발생함을 알 수 있다. 따라서 그림 4와 같이 단순 통합한 사전 구조는 순방향 문자열 단어와 역 문자열 단어의 중복 여부 확인을 위한 방법이 필요하다.

지금까지 지적한 단점을 해결하기 위해서는 새로운 방법이 필요하다. 즉, 전 절단된 용어, 후 절단된 용어는 물론 중앙 절단된 용어의 확장도 효율적으로 지원할 수 있는 방법이 필요하다. 그림 1의 트라이 구조는 단어의 prefix를 공유하고, 그림 3의 구조는 단어의 suffix를 공유하는 특징을 가지므로 prefix와 suffix를 동시에 공유할 수 있도록 하는 것이 바람직하다. 그림 2의 구조[15][16]는 suffix를 부분적으로 공유하고 있지만 역 문자열 단어 검색이 불가능하고 전절단(*정보)된 용어의 확장도 불가능하다.

순방향 문자열 단어는 물론 역 문자열 단어, 후 절단된 용어, 전 절단된 용어, 중앙 절단된 용어 검색을 지원하기 위해 그림 1과 3의 특징을 동시에 최대로 활용하여 새로 설계한 사전 구조는 그림 5와 같다.

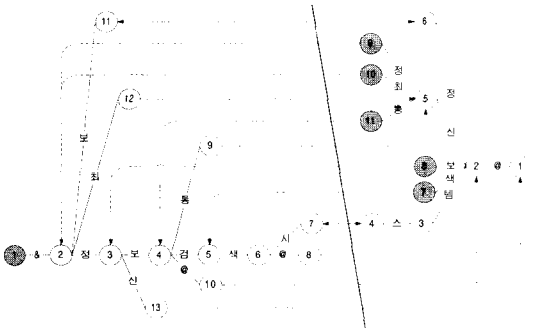

그림 5 새로 설계한 사전 구조

그림 5의 구조는 순방향 문자열 단어, 역 문자열 단어 검색은 물론 다양한 절단 검색이 가능하다. 그림 5에서 기호 '@'는 임의의 단어에 대한 순방향 문자열이 다른 순방향 문자열 단어의 부분 문자열일 때 두 개의 단어(정보, 정보처리)를 구분하기 위한 단어의 종료 기호이고, 기호 '&'는 임의의 단어에 대한 역 문자열이 다른 역 문자열 단어의 부분 문자열일 때 두 개의 서로 다른 단어(신통: 통신 역 문자열, 신통보정: 정보통신 역 문자열)를 구분하기 위한 단어의 시작 기호이다. 왼쪽 트라이와 오른쪽 트라이 내에서 방향이 없는 실선으로 연결된 노드끼리는 선을 따라 앞뒤로 이동이 가능하다. 그러나 좌·우 트라이 두 개를 점선으로 연결한 노드끼리는 방향성이 있다. 즉, 두 개의 트라이를 구분하는 사선의 통과는 반드시 화살표가 표시된 방향으로만 이동이 가능하다.

순방향 문자열 단어는 왼쪽 트라이의 루트 노드 ①에서 출발하여 오른쪽 트라이의 루트 노드 ①에 도달하면 검색에 성공하고, 역 문자열 단어는 오른쪽 트라이 루트 노드 ①에서 출발하여 역 문자열에 의해 왼쪽 트라이 루트 노드 ①에 도달하면 검색에 성공한다. 그림 5에서 음영으로 표시된 노드들(왼쪽 트라이의 노드 ①, 오른쪽 트라이의 노드 ⑦~⑪)은 역 문자열 단어 검색 및 전 절단 검색을 지원하기 위해 새로이 추가된 노드들이다.

[정의 2] 역 문자열 단어 검색을 위한 트라이 구조를 생성하기 위해 다음과 같이 정의한다.

- L_prefix : 임의의 단어 검색 및 후 절단된 단어 검색을 위해 [정의 1]의 분할 노드와 다중 노드를 구성하는 문자열(Ax)을 L_prefix로 정의한다. 이 문자열들은 왼쪽 트라이에 저장된다.
- R_suffix : 임의의 단어 검색 및 후 절단된 용어 검색을 위해 [정의 1]의 단일 노드를 구성하는 문자열(Ω)을 R_suffix로 정의한다. 이 문자열들은 역 문자열 순서로 오른쪽 트라이에 저장된다.
- R_prefix : 임의의 단어들에 대한 역 문자열 단어 검색 및 전 절단된 용어의 문자열 확장을 위해서 R_suffix를 확장한다. R_suffix의 단말 노드는 역 문자열 단어들의 집합에 대해서 [정의 1]의 분할 노드 조건을 만족하지 못할 수 있으므로 분할 노드 조건을 만족하기 위한 최소한의 문자들을 추가한다. 임의의 단어를 $Ax\Omega$(Ax : 다중 노드를 구성하는 문자열, Ω : 단일 노드를 구성하는 문자열)라 할 때 [정의 1]을 만족하는 이 단어의 역 문자열을 $\Omega''yA''$ (Ω'' : 다중 노드를 구성하는 역 문자열, A'' : 단일 노드를 구성하는 역 문자열)라 하자. 이 때 $\Omega''y$를 R_prefix로 정의한다. 이 문자열은 오른쪽 트라이에 저장되며, R_prefix를 구성하는 노드들이 구성하는 노드들은 R_suffix 노드들이 구성하는 모든 노드들을 포함한다.
- L_suffix : 임의의 단어에 대한 역 문자열 단어 검색 및 전 절단된 단어의 확장을 위해서 단어의 역 문자열($\Omega''yA''$) 트라이를 구성했을 때 R_prefix에 저장된 역 문자열($\Omega''y$)을 제외한 나머지 문자열(A'')을 왼쪽 트라이에 저장한다. 이 과정에서 삽입된 문자열을 L_suffix로 정의한다. 이 문자열들은 왼쪽 트라이에 저장되며 L_prefix의 일부가 된다.

순방향 문자열 단어 검색 및 후 절단된 단어 확장을 위한 트라이 구성 과정은 다음과 같다.

1. L_prefix 문자열(Ax)들이 구성하는 노드들을 왼쪽 트라이에 삽입한다. L_prefix의 마지막 문자에 의해 전이한 노드는 왼쪽 트라이의 분할 노드(L_SP)가 된다.

2. R_suffix 문자열들(Ω)이 구성하는 노드들은 오른쪽 트라이에 삽입한다. R_suffix 문자열을 역 문자열로 변환한 다음 오른쪽 트라이에 삽입한다. R_suffix를 삽입한 마지막 노드는 Accept 노드(R_A)가 된다.

3. 순방향 단어 검색을 위해 노드 L_SP가 노드 R_A를 가리키도록 한다. 역 문자열 단어 검색 및 전 절단된 단어의 확장을 위해 다음과 같이 트라이를 구성한다.

4. R_prefix 문자열($\Omega''y$)이 구성하는 노드들을 오른쪽 트라이에 삽입한다. 위의 3까지 과정에서 R_suffix의 역 문자열이 단말 노드는 분할 노드 조건을 만족하지 못할 수 있으므로 R_suffix의 단말 노드에서 분할 노드가 생성될 때까지 노드를 삽입한다. R_prefix가 구성하는 노드들은 R_suffix가 구성하는 노드들을 포함하고, R_prefix의 마지막 문자에 의해 전이한 노드는 오른쪽 트라이의 분할노드(R_SP)가 된다.

5. L_suffix 문자열의 역 문자열이 구성하는 노드들은 왼쪽 트라이에 삽입한다. L_suffix를 삽입한 마지막 노드는 Accept 노드(L_A)가 된다. L_Suffix에 의해 구성된 모든 노드들은 L_prefix 삽입 과정에서 이미 삽입되어 있다.

6. 역 문자열 단어 검색을 위해 노드 R_SP가 노드 L_A를 가리키도록 한다.

과정 1~6을 반복하여 구성한 트라이 구조는 그림 5와 같다. 그림 5와 같은 트라이 구조는 다음과 같은 특징이 있다.

1. 검색 시간이 빠르다. 검색과정에서 관련 없는 노드들은 참조하지 않고 단어를 구성하는 문자열 길이+2개의 노드를 방문하므로 삽입된 단어 수에 관계없이 주어진 한 개의 단어에 대한 검색 시간은 단어의 길이에만 의존한다.

2. 공유되는 prefix, suffix 노드 수를 최대화하여 사전 구성에 필요한 노드 수를 최소화한다(L_suffix 구성 노드 ⊂ L_prefix 구성 노드, R_suffix 구성 노드 ⊂ R_prefix 구성 노드). 즉, 양방향 단어 검색을 위해 필요한 최소 노드는 L_prefix와 R_prefix를 표현하는데 필요한 노드들이다. 그림 1과 3처럼 각각 트라이를 구성한다면 22개의 노드와 29개의 노드를 각각 필요로 하여 전체 51개의 노드가 필요하다. 그러나 그림 5에서는 왼쪽 트라이에 13개의 노드 오른쪽 트라이에 11개의 노드 전체 24개의 노드를 요구하여 필요로 하는 노드 수를 최소화할 수 있다.

3. 절단된 용어의 문자열 확장 과정에서 노드 참조 수를 최소화한다. 그림 1과 같은 트라이 구조는 단말 노드에 도달해야만 단어의 관련 정보를 가져올 수 있으며, 단어의 모든 문자열을 검색한 이후에 드디어 단말 노드(④, ⑦, ⑪, ⑭, ⑰, ⑲, ⑳)에 도달할 수 있다. 그러나 그림 5와 같은 사전 구조는 왼쪽 트라이의 단말노드 및 오른쪽 트라이의 단말 노드를 통해서 관련 정보를 접근할 수 있으므로 왼쪽 트라이의 단말 노드 ⑦~⑬을 통해서 순방향 문자열의 관련 정보를 접근할 수 있고, 오른쪽 트라이의 단말노드 ④~⑪을 통해서 역 문자열 단어의 관련 정보를 접근할 수 있다. 이런 성질을 이용하여 참조하는 노드 수를 줄일 수 있다. 노드 참조 수 감

소는 두 가지로 나누어 생각할 수 있다.

첫 번째 경우는 후 절단된 용어의 문자열이 왼쪽 트라이의 분할 노드에 도달하기 이전에 모두 사용되거나, 전 절단된 용어의 역 문자열이 오른쪽 트라이의 분할 노드에 도달하기 이전에 모두 사용된 경우이다. 즉, 그림 5에서 두 개의 트라이 경계를 통과하지 않은 상태에서 절단된 용어의 문자열이 소진된 경우이다. 그림 1과 같은 일반적인 트라이 구조라면 확장 검색을 위해 절단된 용어의 문자열이 모두 소진되더라도 관련 정보를 가져오기 위해서는 트라이의 단말 노드에 도달할 때까지 모든 노드를 검색해야 한다. 그러나 그림 5의 구조에서는 왼쪽 트라이 및 오른쪽 트라이의 분할 노드까지만 검색한다. 즉, 후 절단된 용어는 왼쪽 트라이, 전 절단된 용어는 오른쪽 트라이만 검색하면 된다. 예를 들어 그림 5에서 절단된 용어가 "정보*"라면, 루트 노드 ①에서 단어 시작 기호 '&'에 의해 노드 ②로, 글자 '정'에 의해 노드 ③으로, 노드 ③에서 글자 '보'에 의해 노드 ④로 진행한다. 검색할 다음 글자가 확장기호 '*'이므로 노드 ④에서 왼쪽 트라이의 단말 노드에 도달 가능한 단어를 검색하여야 한다. 그림 1의 경우라면 노드 ①, ②, ③외에 노드 ④~⑭까지 11개의 노드를 추가하여 전체 14개의 노드를 방문하여 "정보"로 시작되는 단어 "정보", "정보검색", "정보통신", "정보검색시스템" 4개의 단어를 검색하게 된다. 그러나 그림 5와 같은 트라이 구조는 왼쪽 트라이의 분할 노드에 도달하면 [정의 1]의 분할노드 성질에 의해 한 개의 단어가 존재함을 알 수 있으므로 노드 ①, ②, ③, ④외에 노드 ⑤~⑩까지 6개의 노드를 추가하여 전체 10개의 노드만을 방문하고도 동일 단어들이 존재함을 알 수 있으므로 확장을 위해 방문할 노드 수를 줄여 검색 시간을 줄일 수 있다.

두 번째 경우는 후 절단된 용어 검색을 위해 왼쪽 트라이의 분할 노드를 거쳐 오른쪽 트라이를 검색하는 과정에서 검색할 문자열이 소진되거나, 반대로 전 절단된 용어 검색을 위해 오른쪽 트라이의 분할 노드를 거쳐 왼쪽 트라이를 검색하는 과정에서 검색할 문자열이 소진되는 경우이다. 이 때 후 절단 검색을 위해 오른쪽 트라이 노드 ①에 도달하기 이전에, 전 절단 검색을 위해 왼쪽 트라이의 노드 ①에 도달하기 이전에 전 절단 및 후 절단된 용어의 문자열이 소진되는 시점에서 검색이 종료된다. 이는 [정의 1]의 분할노드 성질에 의해 분할 노드가 검색되면 분할 노드에서 검색된 단어의 정보를 얻을 수 있기 때문이다. 예를 들어 절단된 용어가 "*시스템"일 때, 전 절단된 용어이므로 오른쪽 트라이의 루트 노드 ①에서 기호 '@'에 의해 노드 ②로, 노드 ②에서 문자 '템'에 의해 노드 ③으로, 노드 ③에서 문자 '스'에 의해 노드 ④로 진행한다. 노드 ④는 오른쪽 트라이의 분할 노드이므로 오른쪽 트라이 노드 ④에서 점선을 따라 두 트라이의 경계를 넘어 왼쪽 트라이의 노드 ⑦로 이동한다. 왼쪽 트라이의 노드 ⑦에서 문자 '시'에 의해 노드 ⑥으로 진행한다. 검색할 다음 문자가 확장 기호 '*'이므로 문자열 확장을 수행하여야 한다. 그러나 오른쪽 트라이의 단말 노드 ④를 통과하여 왼쪽 트라이로 트라이 경계를 넘는 시점에서 "스템" 문자열로 종료되는 단어가 유일함을 확신(∵[정의 1]의 분할노드)할 수 있으므로 오른쪽 트라이의 루트 노드 ①에서 출발하여 오른쪽 트라이 분할 노드를 거쳐 왼쪽 트라이의 노드 ⑥까지 도달할 때까지 사용된 문자열 "시스템"으로 끝나는 단어 역시 당연히 한 개뿐이다. 따라서 왼쪽 트라이 검색 과정(⑥—색→⑤—검→④—보→③—정→②—@→①)을 생략할 수 있다. 그러나 그림 3과 같이 역 문자열에 의해 구성된 트라이 구조라면 ①—템→㉒—스→㉓—시→㉔—색→㉕—검→㉖—보→㉗—정→㉘—@→㉙를 검색한 다음 관련 정보를 얻을 수 있으므로 노드 ㉕, ㉖, ㉗, ㉘, ㉙를 반드시 검색하여야 한다. 이처럼 그림 5의 구조는 절단된 용어의 문자열 확장 과정에서 불필요한 노드 참조를 제거하여 검색 노드 수를 줄일 수 있다.

4. 중앙 절단을 지원한다. 중앙 절단("정보*시스템")은 후 절단("정보*")과 전 절단("*시스템")의 두 가지 조건을 동시에 만족해야 하므로 먼저 후 절단에 의해 얻어진 단어들의 집합 Fs와 전 절단에 의해 얻어진 단어들의 집합 Rs를 구한 후 두 집합에 공통으로 들어있는 단어들의 집합 Ls=(Fs∩Rs)가 중간 절단에 의해 얻어진 단어들의 집합이다. 예를 들어 그림 5에서 "정보*"에 의해 얻어진 단어들의 집합 Fs = {정보, 정보검색, 정보통신, 정보검색 시스템}이고, "*시스템"에 의해 얻어진 단어들의 집합 Rs = {정보검색 시스템}이다. 따라서 Ls = Fs∩Rs = {정보검색시스템}을 최종적으로 얻게 된다.

3.2 구 현

트라이 구조를 구현할 수 있는 방법으로 리스트 구조를 이용한 방법, 이진 트라이 구조를 이용하는 방법, 상태 전이도를 이용한 방법, 배열을 이용한 방법들이 있다. 이진 트라이 구조는 추가적인 장치를 요구하고, 상태 전이도를 이용한 구현 방법은 검색 시간은 리스트 구조보다 빠르지만 절단 검색을 지원하기 위한 단어 확장이 불가능하다. 따라서 절단 검색을 지원하기 위해서는 리스트 구조나 배열을 이용하는 방법이 가능하다. 리스트 구조를 이용하여 트라이 구조를 구현하기 위해서는 기본적으로 4개의 필드(글자저장필드, 단어 종료필드,

자식 링크 필드, 형제 링크 필드)가 필요하다. 그러나 이 리스트 구조는 역 문자열 단어 검색이 불가능하다. 역 문자열 단어 검색이 가능하기 위해서는 추가적으로 3개의 필드(부모링크필드, 왼쪽 형제링크필드, 글자 시작기호)가 필요하다. 이는 기억장소 요구 량을 증가시키는 요인이 된다. 링크 필드를 4바이트, 한글 1글자 2바이트, 글자 종료기호 1바이트로 가정할 때 일반적인 리스트 구조에서는 노드 당 11바이트(두개 링크*4바이트+2바이트+1)이지만 역 문자열 단어 검색을 지원하기 위한 리스트 구조는 노드 당 20바이트(4바이트*4개의 링크+한글 1글자 2바이트+시작기호 1바이트 + 종료기호 1바이트)로 노드 당 9바이트의 기억공간이 추가적으로 필요하다. 결과적으로 필요한 기억 공간은 거의 두 배로 증가시키게 된다. 또한 삽입되는 단어 수 증가에 따라 연결되는 형제 노드 수가 증가하여 검색 시간이 증가하는 단점을 가진다. 두 개의 1차원 배열을 이용한 구현 방법에서는 트라이를 구성하는 한 개 노드는 두 개의 정수를 이용하여 표현하며 역 문자열 단어 검색을 위해서 한 개의 노드 표현을 위한 추가적인 기억장소를 요구하지 않고, 검색 시간이 일정한 특징을 가진다[14][15]. 삽입된 단어 숫자가 증가함에 따라 리스트 구조를 이용한 방법보다 두 개의 일차원 배열을 이용한 방법이 평균적으로 1.2배에서 3.1배 빠른 검색 시간을 가지는 것으로 알려졌다[17].

따라서 본 논문에서는 절단 검색을 지원하기 위한 사전 구조를 구현하기 위한 방법으로 두 개의 1차원 배열을 이용한 방법[14]을 이용하였다. 이는 검색 과정에서 완전 단어에 대한 검색 요구가 많이 발생하므로 기본적으로 빠른 검색 시간을 지원하면서 절단 검색도 잘 지원하기 위해서이다. 배열을 이용한 트라이를 구현한 방법으로 3.1에서 언급한 단어의 순방향 문자열 트라이와 역 문자열 트라이를 따로따로 구현하여 저장하는 방법과(그림 1+그림 3), 한 개로 통합하여 저장하는 방법(그림 4), 본 논문에서 제안한 전 절단, 후 절단, 중간절단을 지원하는 트라이 구조(그림 5)를 각각 구현하였다. 기억 장소를 절약하기 위하여 3가지 방법 모두 그림 2와 같이 두 개의 트라이를 이용한 방법을 사용하였다.

4. 실험 평가

실험을 위한 시스템 환경으로 CPU: 펜티엄III 500Mhz, RAM: 256MB를 이용하였으며 배열을 위한 정수는 4바이트 정수를 이용하였다. 실험을 위한 단어는 동아 새국어사전[18]에 수록된 단어를 대상으로 하였으며 8만개의 단어를 저장하기 위해 242,000개의 음절이 사용되어 평균 음절수는 3.03이다. 따라서 완전 단어의 검색 과정에서는 평균적으로 5개의 노드를 참조한다.

절단 검색을 지원하기 위해 역 문자열 트라이와 순방향 트라이를 단순 통합하여 저장한 방법(그림 4)에서 순방향 문자열과 역 문자열 단어가 동일한 위치에 저장될 수 있으므로 단어를 검색했을 때 획득한 정보가 역 문자열 단어의 정보인지 순방향 문자열 단어의 정보인지 애매하다. 단어를 삽입함에 따라 순방향 문자열과 역 문자열 순서가 동일한 단어의 발생 비율을 알아보기 위하여 순방향 문자열 단어와 역 문자열 단어의 중복 비율을 실험하였다. 삽입단어가 증가함에 따라 발생하는 중복 비율은 그림 6과 같다.

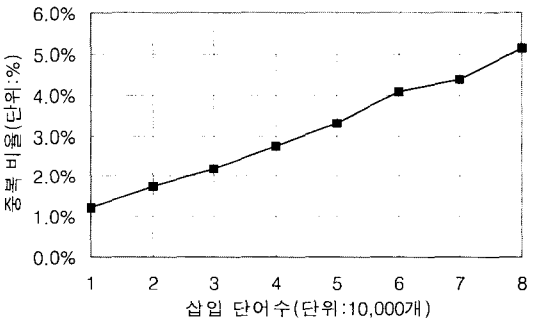

그림 6 순방향 문자열 단어와 역 문자열 단어의 중복 비율

그림 6에서 보는 것처럼 삽입 단어수가 증가함에 따라 중복되는 단어수가 점점 증가함을 알 수 있다. 단어 8만개를 삽입했을 때 5% 이상을 차지하였으며 이는 중복되는 단어 수의 증가에 따른 단어의 모호성을 증가시키는 요인이 된다. 따라서 1개의 트라이에 통합하여 저장하는 것은 바람직하지 않음을 알 수 있다.

기억장소 실험을 위하여 두 개의 트라이 구조에 순방향 문자열 단어와 역 문자열 단어를 별개로 저장하는 방법(방법 1)과 통합된 트라이에 저장하는 방법(방법 2), 본 논문에서 설계한 트라이 구조에 저장한 방법(방법 3)을 각각 이용했을 때 필요한 기억 장소를 보면 그림 7과 같다. 그림 7에서 보는 것처럼 단어 1만개를 삽입했을 때 방법 1과 방법 2가 필요한 기억장소는 200KB이고 방법 3은 170KB를 요구하여 비슷하지만 8만개의 단어를 삽입했을 때 방법 1은 1,380KB, 방법 2는 1,300KB, 방법 3은 1,180KB가 필요하였다. 이는 논 논문에서 설계 구현한 방법이 기억장소를 적게 차지함을 알 수 있다.

검색 시간은 삽입 단어 수에 관계없이 단어 길이에만 의존하여 검색 시간이 결정되므로 1개의 단어 검색 시

간 및 절단된 용어의 검색 시간은 방법 1, 2, 3 모두 같으므로 비교하지 않았다.

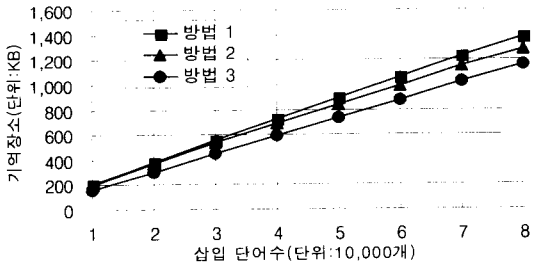

그림 7 기억 장소 요구량

B+트리를 이용하여 사전을 구성한다면 후 절단된 단어를 위해 순차 세트를 순차적으로 검색하므로 삽입된 단어의 절반 정도를 비교 검색하여야 하고, 후 절단된 단어와 중앙 절단된 단어 검색 검색을 위해서는 순차 세트에 연결된 모든 단어를 반드시 검색하여야 한다. 이는 절단 검색은 지원하지만 매우 비효율적임을 알 수 있다. 리스트 구조를 이용한 방법은 기억 장소를 많이 요구하는 단점이 있으며 본 논문에서 설계·구현한 방법은 한 개의 단어를 검색하기 위해 소요되는 시간이 삽입된 단어 수에 무관하게 일정할 뿐만 아니라 문자열 확장을 위한 시간은 확장할 길이에만 영향을 받는다. 방법 1을 이용할 경우 전 절단 및 후 절단 단어 검색은 문제가 없지만 중앙 절단의 경우 전 절단과 후 절단을 동시에 만족해야 하므로 동시에 조건을 만족하는 단어를 조사하기 위한 추가적인 과정이 필요하다. 방법 2는 임의의 단어에 대한 역 문자열이 다른 단어의 문자열과 동일한 경우에 두 단어를 식별하기 위한 추가적인 처리가 필요하다. 방법 3은 방법 1과 방법 2의 단점을 해결한 방법이다.

본 사전의 단점으로는 양쪽 절단을 지원하지 못한다. 단어의 검색 과정은 항상 루트 노드(왼쪽 트라이 및 오른쪽 트라이 노드 ①)에서 시작하여 오른쪽 및 왼쪽 트라이 시작 노드 ①에 도달하므로 써 한 개의 단어 검색이 종료된다. 그러나 트라이의 특성에 의해 검색을 시작할 중간 레벨의 특정 노드의 위치를 결정할 수 없으므로 양쪽 절단("*검색*")은 지원하지 못한다.

5. 결 론

본 논문에서는 완전 단어의 검색 시간이 빠를 뿐만 아니라 다양한 절단 검색을 지원하기 위한 새로운 전자 사전 구조를 설계하고 구현하였다. 구현된 전자 사전은 문자열 확장을 필요로 하지 않은 환경에서 빠른 검색 시간을 지원할 뿐만 아니라 확장을 필요로 하는 경우에도 확장을 위해 방문해야 하는 노드 수를 최소화하여 검색 시간을 줄였다. 지원하는 문자열 확장은 전 절단된 용어의 앞부분 문자열 확장, 후 절단된 용어의 뒷부분 문자열 확장, 중간 절단된 경우의 중간 문자열 확장을 지원한다. 이 사전은 정보 검색 환경뿐만 아니라 문자열 확장이 필요한 분야에서 유용하게 사용할 수 있다. 빠른 검색 시간을 지원할 뿐만 아니라 문자열 확장이 가능하므로 다음과 같은 분야에 적용하여 유용하게 이용할 수 있다.

1. IRS에서 매우 효율적으로 이용할 수 있다. 본 사전 구조는 삽입 단어 수에 관계없이 단어 길이에만 의존하여 검색 시간이 결정되므로 검색 시간이 빠르고, 문자열 확장이 가능하므로 다양한 형태의 절단 검색을 지원한다. 따라서 이 사전 구조 및 검색 방법을 정보 검색 엔진에 적용할 경우 첫째, 검색할 분야에 익숙하지 않은 사람도 용어 절단 방법을 이용한 검색이 가능하고, 둘째, 검색 엔진 환경에 익숙한 사람도 검색된 문헌 숫자가 적을 경우 절단 검색을 통해 관련 있는 문서를 보다 광범위하게 찾을 수 있다. 셋째, 완전 단어에 대한 검색뿐만 아니라 절단 검색도 빠른 검색이 가능하다.

2. 형태소 분석 과정에서는 형태소 분석을 위해 전자 사전 참조가 매우 많이 발생한다. 따라서 전자 사전의 검색 성능은 분석기의 성능에도 큰 영향을 미친다. 형태소 분석 과정에서 분석의 대상이 되는 어간과 어미, 체언과 조사 분리 과정에서 필요할 경우 어간 및 체언은 순방향 단어 검색을 통하여 어미 및 조사는 역 문자열 단어 검색을 통하여 분리 가능하다.

3. 음성 인식 후처리 과정에서 이용할 수 있다. 음성을 입력으로 하여 처리된 결과의 후보가 정확하게 발생하는 경우뿐만 아니라 부분적인 잡음으로 인하여 일부 문자가 애매 모호할 경우 애매 모호한 단어를 사전에서 확장 가능한 문자들이 존재하는지 확장하여 조사하므로써 보다 정확한 결과를 얻을 수 있다.

4. 문자 인식 후처리 과정에 이용할 수 있다. 전자 사전은 문자 인식 과정에서 성능 향상을 위한 수단으로 사용된다. 오프라인으로 인식하는 경우에도 검색 시간이 매우 빨라 잘 이용할 수 있으며, 특히 실시간으로 인식하는 경우 트라이 성질을 이용하여 단어를 구성하는 문자 각각을 처리할 때마다 현재까지 처리된 상태에서 발생 가능한 다음 글자들을 예측할 수 있으므로 실시간으로 처리하는 경우에도 이용할 수 있다.

본 사전 구조의 단점으로는 양쪽 절단을 지원하지 못한다. 따라서 양쪽 절단이 필요한 분야에 활용하기 위해서는 양쪽 절단을 지원하기 위한 연구가 있어야 한다

참 고 문 헌

[1] 정영미, 정보검색론, 구미무역출판부, 1993.

[2] Salton, G. and M. J. McGill, Introduction to modern Information Retrieval, New Yourk: McGraw-Hill, 1983.

[3] 김판구, 조유근, "상호 정보에 기반한 한국어 텍스트의 복합어 자동색인", 한국정보과학회 논문지, 21권 7호, pp. 1333-1340, 1994.

[4] 최재혁, "형태소 분석을 통한 한·영 자동 색인어 추출 시스템", 한국정보과학회 논문지, 23권 12호, pp. 1279-1288, 1996.

[5] Salton, G., E. A. Fox and Hwu "Extended Boolean Information Retrieval," CACM Vol. 26, No. 11, pp. 1022-1036, 1983.

[6] 강현규, 박세영, 최기선, "자연어 정보 검색에서 상호 정보를 이용한 2단계 문서 순위 결정 방법", 한국정보과학회 논문지, 23권 8호, pp. 852-861, 1996.

[7] 고미영, P-NORM 검색의 문헌 순위화 기법에 관한 실험적 연구, 연세대학교 박사학위 논문, 1999.2.

[8] Harman D. "An Experimental study of factors important in document ranking," Paper presented at ACM Conference on Research and Development in information Retrieval, Pisa, Italy, 1986, pp. 186-193.

[9] Belkin, N.J. and W.P. Cropt. "Retrieval technique," Annual Review of Information Science and Technology, 22, pp. 109-145.

[10] E. Fredkin, B. Beranek and Newman, "Trie memory," CACM, Vol 3, pp. 490-499, 1960.

[11] T. G. Lewis and C. R. Cook, "Hashing for dynamic and static internal tables," IEEE Computer, pp. 45-56, Oct. 1988.

[12] Margo Selter, A New Hashing Package for UNIX, USENIX-Winter '91 - Dallas, TX, 1991.

[13] Masami Shishibori, Kazuhiri M. and J. I. Aoe, "The Design of a Compact Data Structure for Binary Tries", ___, pp. 573-479, 1996.

[14] J. I. Aoe, "An Efficient Digital Search Algorithm by Using Double-array Structure," IEEE Transaction on S/W Eng., Vol. 15, No. 9, pp. 1066-1077, 1989.

[15] K. Moromoto, H. Iroguchi and J. I. Aoe, "A Retrieval Algorithm of Dictionary by using Two trie Structures," 일본 전자 공학회 논문집 D-II Vol. J76-D-II No. 11, pp. 2374-2383, 1994.

[16] 김철수, 배우정, 이용석, J. I. Aoe, "이중배열 트라이 구조를 이용한 한국어 전자 사전 구축", 한국정보과학회 논문지 23권 1호, pp. 85-94, 1996.

[17] J. I. Aoe and K. Morimoto, "An Efficient Implementation of Trie Structure," S/W Practice and experience, Vol. 29(9), pp. 695-721. 1992.

[18] 동아 새 국어사전, 동아출판사, 1994.

김 철 수

1987 전북대학교 전산통계학과 학사
1989 전북대학교 전산통계학과 석사
1998 전북대학교 전산통계학과 박사
1995.3~현재 서남대학교 전산정보학과 조교수. 관심분야는 자연어처리, 정보검색 지식표현