한국산학기술학회논문지
Vol. 10, No. 8, pp. 1986-1992, 2009

트라이 인덱스를 이용한 이형태 검색

박인철[1*]
[1]호원대학교 컴퓨터게임학부

Searching for Variants Using Trie-Index

In-Cheol Park[1*]
[1]**Division of Computer-game, Howon University**

요 약 사용자는 정보검색에서 단어의 약어나 부분문자열, 혹은 오타가 포함된 단어와 같은 이형태로 자료를 검색하고자 한다. 이형태 검색을 위한 단순한 방법은 사전에 모든 이형태를 등록하는 것이다. 그러나 이 방법은 이형태 사전 구축에 막대한 시간과 비용이 필요할 뿐만 아니라 오타로 인해 생기는 이형태를 처리할 수 없는 문제점이 있다. 이에 대한 대안으로 근사 문자열 매칭 기법을 이용한 방법이 개발되었으나 이 방법 또한 약어 형태의 이형태를 처리하기 어렵다는 단점이 있다. 본 논문에서는 트라이 인덱스를 이용해 약어나 오타를 포함한 대부분의 이형태를 검색할 수 있는 방법을 제안한다. 먼저, 패스 가중치의 계산을 통한 이형태 매칭 방법을 보이고, 검색 속도 향상을 위한 이형태 검색 알고리즘을 제시한다.

Abstract A user often searches a data by inputting a variant such as the abbreviation or substring of a word, or a misspelled word. The simple approach to the searching for variants is to build a variants dictionary. However, it entails enormous cost and time and can not handle variants by misspelling. Approximate searching, searching by approximate string matching, is a good approach to the searching. A problem in the approach is that it cannot handle variants by abbreviations. This paper propose a method for searching various variants including abbreviations and misspelled words, by using the trie indexing. First, this paper shows a variant matching method with the calculation of path weighted-metric. In addition, it provides variant searching algorithm to reduce the search time.

Key Words : Approximate String Matching, Variant Searching, Trie Index

1. 서론

인터넷 사용의 보편화, 내비게이션 사용의 증가 및 IPTV 보급의 활성화에 따라 검색 기술은 더 많은 사용자의 요구 사항을 수용할 수 있어야 한다. 이러한 요구 사항 중의 하나는 이형태를 키워드로 할 경우 원래 사용자가 의도한 키워드를 검색할 수 있어야 한다는 것이다. 예를 들어, “○○대학교 애니메이션학부”가 검색을 위한 정확한 키워드라고 할 때 사용자는 “○○대 애니메이션학부”, “○○대학교 애니학부”, "○○대학교 에니메이션학부", “○○대학교 애니메이션과” 등 다양한 이형태 키워드로 검색하고자 할 것이다.

현재 이형태 검색을 위해 널리 쓰이는 방법은 단순히 사전에 이형태를 저장하는 것이다. 이러한 방법의 문제점은 사전에 방대한 이형태 사전을 구축해야 한다는 것이다. 수동으로 일일이 이형태를 구축하는 것은 많은 비용과 시간을 필요로 할 뿐만 아니라 오류를 포함할 가능성이 있다. 이에 따라 외래어 이형태를 자동으로 생성하려는 연구[1]가 있었으나 예측 가능한 모든 이형태를 사전에 저장하는 것은 비효율적일 뿐만 아니라 오타 등으로 인한 다른 종류의 이형태를 처리하지 못하는 문제점이 있다.

본 논문은 2009년 호원대학교 교내학술연구비 조성비 지원에 의해연구되었음.
*교신저자 : 박인철(icpark@howon.ac.kr)
접수일 09년 06월 03일 수정일 09년 07월 13일 게재확정일 09년 08월 19일

또 다른 접근 방법은 이형태 문자열에 대한 패턴 매칭 연산을 통해 해당 검색어를 찾는 것이다. 편집 거리(edit distance)[2] 계산을 통한 이형태 문자열 검색은 정형적인 접근 방법이나 많은 계산 시간과 메모리 공간을 요구한다. 이를 극복하기 위해 Dynamic Programming을 이용한 문자열 매칭[3], 오토마타를 이용한 문자열 매칭[4], Bit-Parallelism 방법을 이용한 문자열 매칭[5], 필터링 알고리즘을 이용한 문자열 매칭[6] 방법 등이 제안되었고, 각각 장단점을 가지고 있어 응용 분야에 따라 선택적으로 사용되고 있다[7]. 그러나 이러한 방법들은 특정 음절이 변경된 경우에는 잘 작동하나 약어와 같은 이형태는 처리하지 못하는 약점이 있다.

본 논문에서는 트라이[8] 인덱스를 이용한 이형태 검색 방법을 제안한다. 이형태 검색의 기본 아이디어는 이형태의 음절을 최대한 포함하는 트라이 사전 내의 패스를 검색하는 것이다. 검색된 단어 중 가장 적합한 후보를 선택하기 위해 우리는 패스 가중치를 계산하여 이용할 것이다. 본 방법은 “무역센타(한국종합무역센터)”와 같이 외래어의 검색 시 흔히 발생하는 음절 일부에 오류가 있거나 “국과수(국립과학수사연구소)”와 같이 약어로 검색하는 경우 유용하다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 다루는 이형태에 대한 정의를 내리고, 3장에서는 이형태로 본래 단어를 검색하기 위해 필요한 기본 연산인 이형태 매칭 방법에 대해 설명한다. 4장에서는 트라이 기반의 이형태 검색 알고리즘을 제안하고, 5장에서는 제안된 방법에 대한 실험 및 평가 결과를 기술한다. 마지막으로 6장에서 결론 및 향후연구방향에 대해 기술한다.

2. 이형태의 정의

이형태는 사용자가 검색하고자 하는 본래의 단어와 다른 형태로 표현된 문자열을 의미한다. 이러한 이형태는 다음과 같이 분류할 수 있다.

① 오타에 의한 이형태
② 외래어 표기에 의한 이형태
③ 약어에 대한 이형태
④ 단어의 일부 문자열로 표현된 이형태
⑤ 별칭(alias)에 의한 이형태

오타에 의한 이형태는 말 그대로 검색어를 잘못 입력한 것이다. “국립대전현충원”을 검색하기 위해 “대전*현*충원”이라고 잘못 입력한 경우이다.

외래어 표기에 의한 이형태는 외래어를 한국어로 표기할 때 소리나는 대로 표기하므로 사람에 따라 다르게 표현할 수 있다. 예를 들어, 영어 단어 center에 대한 올바른 표기법은 “센터”이지만 “센타”라는 표현도 많이 사용한다. 구글을 이용하여 검색해 보면 “센터”는 약 142,000,000개의 문서를 검색하고, “센타”는 약 9,970,000개의 문서를 검색한다. 이러한 오류는 흔히 발생하므로 대부분의 검색 엔진에서 “센터”와 “센타”를 동의어로 처리하지만 모든 외래어 표기에 대해 동의어 사전을 구축하는 것은 쉽지 않은 일이다.

단어의 일부 문자열로 표현된 이형태는 긴 명칭을 사용할 때 흔히 약어를 사용하거나 해당 명칭의 일부만을 사용하는 경우로, “서울문화예술회관”을 전부 부르지 않고 “예술회관”이라고 부르는 경우가 이에 해당한다.

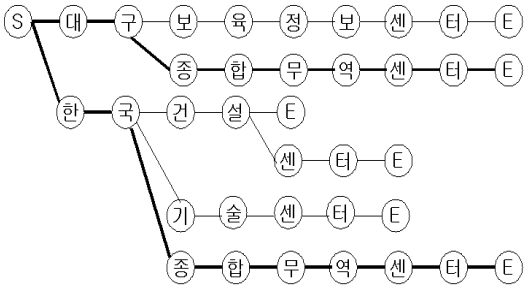
별칭(alias)에 의한 이형태는 원래 지명에 대한 다른 이름을 사용하는 경우로, 예를 들어 금강산을 “봉래산” 등으로 부르는 경우가 이에 해당한다.

본 논문에서 이형태 검색은 별칭에 의한 이형태를 제외한 나머지 이형태를 입력 대상으로 다룬다.

3. 트라이 기반 이형태 매칭

트라이 인덱스를 이용한 이형태 검색의 기본 아이디어는 트라이 노드를 역추적하여 주어진 이형태 문자열을 최대한으로 포함된 모든 패스를 찾아내는 것이다. [그림 1]은 “무역센터”에 대해 매칭된 패스에 대한 예를 보여주고 있다. 여기서 Ⓢ는 시작 노드를 Ⓔ는 종단 노드를 의미한다.

3.1절에서는 이형태 검색을 위한 트라이 사전의 구축 방법에 대해, 3.2절에서는 음절과 트라이 노드의 매핑과 조상 노드에 대한 역추적 방법에 대해 살펴본 후, 3.3과 3.4절에서는 이형태 매칭 방법에 대해 살펴본다.

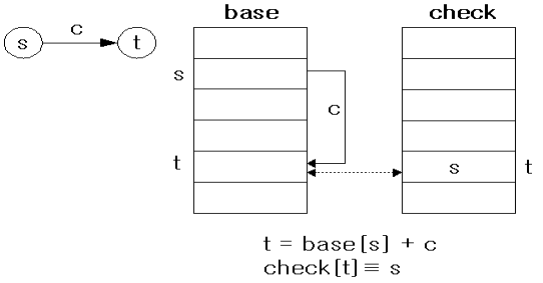


[그림 1] 매칭된 패스의 예

3.1 트라이 사전의 구축

트라이의 특성상 두 노드가 같은 패스에 속하는지를 판단할 때 역추적으로 판단하는 것이 효율적이다. [그림 1]에서 노드(한)에서 노드(센)이 같은 패스에 있는 지를 판단할 때, 부모 노드인 (한)에서 시작한다면 노드 (한)에서 시작하는 모든 패스를 검사해야 한다. 설령 중간에 노드(센)를 찾았다 하더라도 [그림 1]에서 볼 수 있는 바와 같이 원하지 않는 패스의 노드(센)일 수 있으므로 언제나 모든 패스를 검색해야 하는 문제점이 있다. 그러나 자손 노드인 노드(센)에서 조상 노드인 노드(무)가 같은 패스에 존재하는가를 검사할 때에는 언제나 단일 패스만 한 번 검사하면 된다.

Jun-Ichi Aoe가 제안한 이중 배열을 이용한 트라이[9]는 [그림 2]와 같이 base와 check라는 두 배열을 사용하여 트라이 구조를 표현한다. 이중 배열 트라이는 생성 속도가 매우 느리다는 단점이 있지만 매우 빠른 검색을 수행할 뿐만 아니라 check 배열을 이용하여 노드의 역추적도 가능하다. 즉, 별도의 부모 노드에 대한 링크없이 자손 노드에서 조상 노드에 대한 패스를 검사할 수 있다. 따라서 본 논문에서는 트라이 인덱스 구축을 위해 이중 배열 트라이를 사용한다.



[그림 2] base와 check 배열

한국어 트라이는 음절 단위나 음소 단위로 구축이 가능하다. 이론적으로 음소 단위의 한국어 트라이는 음절 단위 트라이보다 더욱 정확한 이형태 검색을 가능하게 한다. 예를 들어, “센터”에 대한 이형태 “센타”를 검색할 때, 음절 단위는 “센*” 문자열을 포함한 모든 문자열을 검색하지만 음소 단위는 “ㅅ-ㅔ-ㄴ-ㅌ-*-ㅏ” 문자열을 포함한 모든 문자열을 검색하기 때문이다(‘ㅏ’는 무종성을 의미한다). 그러나 음소 단위 트라이는 음절 단위에 비해 3배의 노드를 생성하기 때문에 패턴 매칭 연산 시간을 크게 증가시킨다.

따라서 본 논문에서는 트라이를 음절 단위로 구축하였으며, 한국어의 음절 단위로 검색과 역추적이 가능하도록 이중 배열 트라이를 구현하였다.

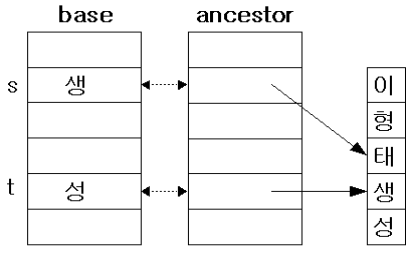
3.2 트라이 노드의 역추적

트라이 노드를 역추적하려면 먼저 특정 음절이 포함된 노드가 어느 곳에 위치해 있는지를 알아야 한다. [그림 2]는 이중 배열 트라이에서 문자 c에 의해 상태 s에서 상태 t로 전이할 때 base 배열과 check 배열의 역할을 보여준다. 문자 c에 의해 s에서 t로 전이되므로 문자 t에 대한 노드는 base[t]로 정의할 수 있다. 따라서 문자 c에 대한 노드의 base 배열의 인덱스로 정의할 수 있다.

문자 c에 대한 노드는 트라이의 여러 노드에서 나타날 수 있으므로 아스키 문자 c와 트라이 노드에 대한 매핑 Map(c)는 다음 식 (1)로 표현할 수 있다.

(1) Map(c) → {t_1, t_2, ⋯, t_n}, (단, t_i는 배열의 인덱스)

한글의 음절은 2바이트로 구성되어 있기 때문에 하나의 음절은 두 개의 트라이 노드로 표현해야 한다. 따라서 한글 음절에 대한 노드는 <s_i, t_i>와 같이 두 인덱스의 쌍으로 표현될 수 있다. 그러나 한글 음절과 트라이 노드의 매핑 목적은 조상 노드에 대한 역추적이 목적이므로 단순히 s_i만을 사용하여 표현해도 문제가 발생하지 않는다.



[그림 3] ancestor 배열의 사용

현재 노드에서 특정 문자 c를 갖는 조상 노드가 존재하는지 판별하려면 현재 트라이 노드가 어떤 문자를 표현하는지 알 수 있어야 한다. [그림 3]에서 트라이 노드 base[t]가 표현하는 문자 c는 다음 식 (2)에 의해 구할 수 있다.

(2) c = t - base[check[t]]

또한 한글의 음절은 2바이트 문자이므로 한글 음절 w = <c, d>는 다음 식 (3)에 의해 구할 수 있다.

(3) s = check[t];
d = t - base[s];

```
c = s - base[check[s]];
w = (c << 8) | d;
```

만일 검색 포털 서버와 같이 처리 속도가 중요한 시스템인 경우에는 조상 노드의 음절을 저장하는 ancestor 배열을 사용하여 조상 노드에 대한 역추적을 처리할 수 있다. 예를 들어, [그림 3]과 같이 "이형태생성"에 대한 ancestor 배열을 구성할 수 있다. 그러면 단순히 문자열 검색 함수에 의해 해당 음절이 조상 노드에 존재하는지를 판단할 수가 있다.

3.3 이형태 매칭

두 단어 사이의 유사도를 계산하기 위해서 널리 쓰이는 방법은 편집 거리를 계산하는 것이다. 검색 엔진은 이형태와 가장 유사한 단어를 검색하기 위해 편집 거리가 가장 작은 값을 갖는 문자열을 선택한다. 그러나 비교할 패턴의 개수가 늘어날수록 편집 거리의 계산을 위해 필요한 시간과 메모리 공간은 크게 늘어나는 단점이 있다.

서론에서 언급한 바와 같이 이를 해결하려는 여러 방법이 제안되었으나 각각 장단점이 있어 응용 분야에 적절한 방법을 선택해서 사용해야 한다.

본 논문은 편집 거리를 계산하는 대신에 음절을 표현하는 노드들이 같은 단어를 이루는 패스에 존재하는지 여부를 계산하는 패스 가중치(path weighted-metric)에 의한 이형태 매칭 방법을 제안한다. 예를 들어, [그림 1]에서 진하게 표현된 단어 패스에 이형태를 이루는 음절들을 모두 포함하므로 패스 가중치의 값은 다른 단어 패스보다 큰 값을 갖게 된다.

패스 가중치는 단어를 이루는 패스에 존재하는 음절의 개수에 의해 영향을 받으므로 단어 w와 이형태 t의 패스 가중치는 기본적으로 다음 식 (4)와 같이 계산될 수 있다.

$$(4)\ \sum_{k=2}^{m}\sum_{j=1}^{k-1} EP(\beta_k, \beta_j)$$

$$EP(\beta_k, \beta_j) = \begin{cases} 1\ \beta_k = \alpha_h, \beta_j = \alpha_i, h > i \\ 0\ otherwise \end{cases}$$

$$w : \alpha_1\alpha_2\alpha_3 \cdots \alpha_n$$

$$t\ : \beta_1\beta_2 \cdots \beta_m$$

EP는 패스 평가 함수로 이형태에 나타나는 음절이 먼저 나타나는 순서대로 β_k, β_j일 때 단어(원래 검색하고자 하는 단어) w에 속한 α_h, α_i가 각각 이형태 음절 β_k, β_j와 같고 α_h가 α_i보다 먼저 나타나면 1을 갖고 그렇지 않으면 0값을 갖는다. 예를 들어, EP("대", "건")은 "건←국←대"로 이어지는 역패스 관계가 존재하므로 1값을 갖는다. 반면에, EP("대", "건")의 값은 0이 된다. 이 패스 평가 함수를 이용하여 이형태 "건대주차장"과 단어 "건국대학교주차장" 사이의 패스 가중치를 계산하면 10(=4+3+2+1)이 되며, 단어 "이화여자대학교주차장"과의 패스 가중치는 6(=3+2+1)을 갖게 된다. 설령 비교하고자 하는 단어가 이형태에 속한 모든 음절을 포함하고 있어도 식 (4)는 거의 대부분 만족할만한 결과를 보여준다. 예를 들어, 단어 "대한건설주차장"과의 패스 가중치는 9(4+3+2+0)를 갖게 되므로 올바른 검색 결과를 산출할 수 있을 것이다.

식 (4)는 영어의 한글 표기나 오타로 인해 특정 음절이 변형된 경우에는 대부분 1순위로 원하는 검색어를 찾을 수 있으나 약어 형태의 "이형태"를 찾을 때에는 적지 않은 오류를 보여준다. 이러한 오류의 발생은 식 (4)에서 다음과 같은 경우를 고려하지 않았기 때문이다.

㉮ 서로 인접한 음절에 더 많은 가중치를 줄 필요가 있다.
㉯ 이형태의 한 음절이 어떤 단어의 첫 문자일 때 더 많은 가중치를 줄 필요가 있다.

㉮의 경우는 "대한식당"을 검색 문자열로 입력하였을 때 "대한식당"과 "대중한식당"이 같은 패스 가중치를 갖는 문제를 해결하기 위해 필요하다. 이에 대한 간단한 해결은 두 음절이 인접한 경우 패스 평가 함수의 값을 1 대신에 2로 설정하는 것이다. 다음 식 (5)는 이를 반영한 새로운 패스 평가 함수의 정의를 보여준다.

$$(5)\ EP(\beta_k, \beta_j) = \begin{cases} 2\ \beta_k = \alpha_h, \beta_j = \alpha_i, h - i = 1 \\ 1\ \beta_k = \alpha_h, \beta_j = \alpha_i, h - i > 1 \\ 0\ otherwise \end{cases}$$

식 (4)에서 정의한 것처럼 β_k, β_j는 이형태에 속한 음절을 의미하며, α_h, α_i는 실제 검색하고자 하는 단어에 속한 음절을 의미한다. 식 (5)에 의해 검색 문자열 "대한식당"과 단어 "대한식당"의 패스 가중치는 9(4+3+2)인 반면에, "대중한식당"과의 가중치는 8(4+3+1)을 갖게 되어 좀 더 정확한 검색 결과를 보여줄 수 있다.

㉯의 경우는 "서울터미널"을 검색 문자열로 입력하였을 때 "서울남부버스터미널", "동서울종합터미널"이 같은 패스 가중치를 갖는 문제를 해결하기 위해 필요하다. 이를 해결하기 위해서는 독립된 단어를 이루는 문자열에 가중치를 줄 필요가 있다. 즉, "동서울종합터미널"에서

"서울"은 독립된 단어가 아니지만, "서울남부터미널"에서 "서울"은 독립된 단어로 사용하였으므로 이에 대한 가중치를 부여하는 것이다. 4장에서 독립 단어 가중치 부여에 대해 자세히 살펴 볼 것이다.

3.4. 단어 가중치 부여

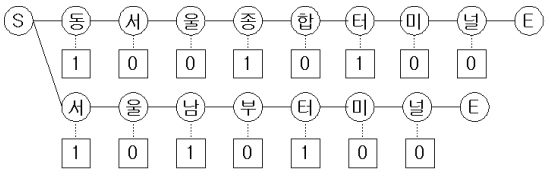
단어 가중치(word weight)는 단어를 이루는 각 음절에 붙는 값으로 0, 1, 2 중에 하나의 값을 갖는다. 단어 가중치의 각 값은 다음과 같이 결정된다.

㉮ 0 : 해당 음절이 단어의 시작이 아닌 경우, "남서울대학교"에서 음절 "서", "울", "학", "교" 등이 해당된다.
㉯ 1 : 해당 음절이 단어의 시작이나 약어로 자주 쓰이지 않는 경우, "서울식당"에서 "서", "식"이 해당된다.
㉰ 2 : 해당 음절이 단어의 시작이고 약어로 자주 쓰이는 경우, "경기도발전연구소"에서 "경", "연" 등이 해당된다.

다음 식 (6)은 단어 가중치를 고려한 패스 평가 함수의 최종식을 보여준다.

(6) $$EP(\beta_k, \beta_j) = \begin{cases} 2+v & \beta_k = \alpha_h, \beta_j = \alpha_i, h-i = 1 \\ 1+v & \beta_k = \alpha_h, \beta_j = \alpha_i, h-i > 1 \\ 0 & otherwise \end{cases}$$

식 (6)에서 v는 β의 단어 가중치를 의미한다.



[그림 4] 단어 가중치의 예

[그림 4]는 각 "서울남부터미널"과 "동서울종합터미널"에 대해 각 음절에 단어 가중치를 부여한 모습을 보여준다. 식 (6)에 의해 "서울터미널"과 "동서울종합터미널" 사이의 패스 가중치는 15(6+5+2+2)이며, "서울남부터미널"과의 패스 가중치는 19(7+6+3+3)가 됨을 알 수 있다.

트라이 구현 시 가중치는 [그림 2]에서 표현된 base와 check 배열 이외에 weight 배열을 사용하여 저장된다. 이중 배열 트라이에서 각 노드는 바이트 단위이므로 식 (3)에서처럼 음절이 {c, d}로 구성되어 있고 바이트 c를 표현하는 base 배열의 인덱스가 s라면 단어 가중치 w를 구하는 식은 (7)과 같다.

(7) ω = weight[base[s]+d]

4. 이형태 검색 알고리즘

이형태 검색을 위해 먼저 필요한 것은 이형태를 이루는 음절이 트라이의 어느 노드에 해당되는 것인지 아는 것이다. 이는 검색 기법을 이용하여 간단히 해결할 수 있으나 문제는 너무나 많은 음절에 대해 이형태 매칭을 수행해야 하기 때문에 사전에 저장된 단어의 크기에 따라 검색의 응답 시간이 크게 증가한다는 것이다.

하나의 음절에 연결된 트라이 노드의 평균 개수를 n이고, 이형태의 평균 음절 개수를 m이라 가정하자. 모든 후보에 대해 패스 가중치를 계산하면 모두 n*(m-1)*(m-2)번의 패스 평가 함수를 실행해야 한다. 만일 트라이 사전에 구축된 엔트리 개수가 100만개이고 엔트리의 평균 음절 개수 m의 길이가 6.3이라 가정해 보자. 2,350개의 완성형 음절을 사용한다면 n의 값은 2,680.9가 되며, 한 번의 검색을 위해서 약 60,000번 정도의 패스 평가 함수를 수행해야 한다는 계산이 나온다. 이는 사용 빈도를 고려하지 않은 수치이고 실제에 있어서는 패스 평가 함수의 실행 횟수는 훨씬 더 커지게 된다.

패스 평가 함수의 실행 횟수를 줄이기 위해 우리는 병렬 처리 기법을 사용한다. 즉, 음절을 포함한 모든 단어에 대해 동시에 패스 가중치를 구하되, 패스 가중치의 최대값과의 차이(이 차이를 "가중치 편차"라 부르자)가 d(이를 "최대 허용 편차"라 부르자)이하이면 후보에서 제외한다. 만일 남은 후보가 p(이를 "최소 후보 개수"라 부르자) 이하이면 더 이상 후보 검색을 진행하지 않는다.

예를 들어, 트라이 사전이 [그림 1]과 같이 구축되어 있다고 가정할 때, 이형태 "무역센터"에 대한 검색을 고려해 보자. 음절 "터"와 "센"에 대한 패스 평가 함수를 수행한 결과는 다음과 같다. (단순함을 위해 d의 값은 1, p의 값은 2라고 가정하자)

[대구보육정보센터] → 3
[대구종합무역센터] → 3
[한국건설센터] → 3
[한국기술센터] → 3
[한국종합무역센터] → 3

탈락할 후보가 없으므로 모든 후보에 대해 다시 “터”와 “역”에 대한 패스 평가 함수를 실행한다.

[대구보육정보센터] → 3
[대구종합무역센터] → 4
[한국건설센터] → 3
[한국기술센터] → 3
[한국종합무역센터] → 4

최대 허용 편차 d에 의해 세 개의 후보는 제외되고 남은 후보는 p 이하이므로 패스 가중치 계산은 더 이상 진행하지 않는다.

만일 이형태가 “무역센타”인 경우 위 방식에 문제가 발생한다. 위에서 언급한 후보 대신에 “타”가 들어간 다른 단어가 검색될 것이기 때문이다. 이를 방지하려면, 음절 “타”와 마찬가지로 “센”, “역”에 대해서도 같은 방식으로 계산을 한 후 가장 높은 갖는 값을 갖는 후보들을 검색 결과로 선택하면 된다. [그림 5]는 이형태 검색 수행 과정을 보여준다.

```
var
  Sk : 이형태의 k번째 음절;
  n : 이형태의 길이;
  EP(Sk, Sj) : 음절 wk와 wj의 패스 평가 함수 결과값;
  No : 단어 Wo에 연결된 트라이 노드;
  PWo : 단어 Wo의 패스 가중치;
  WSk : k번째 수행과정에서 구해진 모든 단어 후보;
  d : 최대 허용 편차;
  p : 최소 후보 개수;
begin
  for k ::= n to 2
  begin
    Sk에 연결 노드 집합을 구하여 WSk에 저장
    for j ::= k - 1 to 1
    begin
      foreach Wo in WSk
      begin
        PWo += EP(Sk, Sj);
      end
      d보다 큰 가중치 편차를 갖는 후보를 WSk에서 제거;
      WSk의 원소 개수가 p이하이면 반복에서 빠져나옴;
    end
    if (k < n) then
      WSk := WSk ∪ WSk+1;
    endif
  end
  d보다 큰 편차를 갖는 후보를 WS2에서 제거;
  WS2을 패스 가중치에 따라 내림차순으로 정렬;
end.
```

[그림 5] 이형태 검색 알고리즘

5. 실험 및 평가

실험은 작동 속도 2.2GHz인 AMD 듀얼 코어 CPU를 장착한 리눅스 서버에서 수행하였다. 우리는 이미 구축되어 있는 3만여 개의 지명 코퍼스 중 5음절 이상의 지명 데이터 8,361개 단어를 대상으로 트라이 사전을 구축하였다. 실험 및 평가를 위해 [표 1]과 같이 약어로 쓰인 이형태 50개, 한 음절을 잘못 표기한 이형태 50개, 두 음절을 잘못 표기한 이형태 50개, 일부 문자열을 사용한 이형태 50개로 총 200개의 이형태를 구축하였다. 이형태의 평균 음절수는 4.6이며, 5음절 이상의 지명 데이터의 평균 음절수는 6.3이었다.

[표 1] 실험에 사용한 이형태

종 류	개 수
약 어	50
오타(1음절)	50
오타(2음절)	50
부분문자열	50

실험 결과 최대 허용 편차를 3으로, 최소 후보 개수를 6으로 한 경우에 검색한 후보 리스트 중에 원하는 결과를 포함하는 100% 재현율을 보였다. 평균 21.7개의 단어를 갖는 10,000개 문장 전체에 대한 검색 시간은 1.32초이었다.

6. 결론

이형태 검색은 사용자가 문서에 나타난 정확한 단어를 입력하지 않아도 해당 문서를 찾을 수 있도록 한다. 이를 위한 간단한 방법은 나타날 수 있는 모든 이형태를 사전에 구축하는 것이나 이는 거의 불가능한 작업이다. 과도한 이형태 사전을 구축하지 않고도 이형태에 대한 검색이 가능하게 위해서 현재 널리 사용되는 방식은 근사 문자열 매칭 기법을 이용하는 것이다. 그러나 근사 문자열 매칭은 오타나 부분 문자열에 대한 매칭에 비해 약어에 대한 매칭은 계산 복잡도가 크게 증가하기 때문에 약어에 의한 이형태 검색을 지원하지 못하는 문제점이 있다.

본 논문에서는 트라이 인덱스를 이용하여 오타나 부분 문자열뿐만 아니라 약어에 의한 이형태도 성공적으로 검색할 수 있는 방법을 제안하였다. 또한 실험을 통하여 제안한 방법을 상용 시스템 구현에 적용할 수 있음을 보

였다. 한국어 문장에서 단어 시작 음절을 중심으로 한 약어(즉, "국가정보원"을 "국정원"으로 표현)가 비교적 많이 나타나므로 이형태 검색을 지원하는 한국어 검색 시스템을 구현하는 데 유용하게 사용할 수 있을 것으로 기대된다.

참고문헌

[1] 이재성, “효과적인 외래어 이형태 생성을 위한 확률 문맥 의존 치환 방법”, 한국콘텐츠학회논문지 제7권 제2호 pp. 73-83, 2007. 2.

[2] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals.” Soviet Physics-
Doklady, pp. 707-710, February 1966.

[3] Eiko Yamamoto et al., "Dynamic Programming Matching for Large Scale Information Retrieval" Proceedings of the Sixth International Workshop on Information, pp. 100-108, July 2003.

[4] Holub, J., “Reduced Nondeterministic Finite Automata for Approximate String Matching“ Proceedings of the Prague Stringologic Club Workshop, pp. 19-27, 1996.

[5] Myers, G., "A fast bit-vector algorithm for approximate string matching based on dynamic programming" J. ACM 46, 3, pp. 395–415, 1999.

[6] Navarro, G., "A Guided Tour to Approximate String Matching", ACM Computing Survey, 33(1), pp. 31-88, 2001.

[7] Chung W. Ng, "Inexact Pattern Matching Algorithms via Automata" http://biochem218.stanford.edu/Projects%20Winter%202007/Ng.pdf, Mar. 2007.

[8] Edward Fredkin, "Trie Memory" Communications of the ACM 3 (9), pp. 490-499, 1960.

[9] Aoe, J., "An Efficient Digital Search Algorithm by Using a Double-Array Structure" IEEE Transactions on Software Engineering. Vol. 15 (9), pp. 1066-1077. Sep. 1989.

박 인 철(In-Cheol Park) **[정회원]**

- 1984년 2월 : 전북대학교 전신통계학과 (이학사)
- 1986년 2월 : 전북대학교 전산통계학과 (이학석사)
- 1998년 8월 : 전북대학교 전산통계학과 (이학박사)
- 1992년 10월 ~ 현재 : 호원대학교 컴퓨터게임학부 교수

<관심분야>
한국어정보처리, 정보검색, 시멘틱웹. 지식표현