한글 글자 단위 인덱스를 위한 검색 유형 정의 및 한글 부호계와의 연관성에 관한 연구

이중화*·이종민*·김성우*
*동의대학교 컴퓨터소프트웨어공학과

A Study of the framework of search patterns for Hangul characters and its relationship with Hangeul code for Hangeul Character based Index

Jung-hwa Lee* · Jong-min Lee* · Seong-woo Kim*
*Dong-eui Univ., Dept. of Computer Software Engineering,
E-mail : junghwa@deu.ac.kr

요 약

본 논문에서는 한글 인덱스를 구현할 때 글자 단위를 기본으로 하는 경우 적용될 수 있는 검색 유형 (search pattern) 들은 어떠한 것들이 존재할 수 있는지에 대해 살펴보고, 검색 알고리즘에 적용시켜 본다. 이 때 부호계와의 연관성과 효율성을 따져보기 위해서 KS X 1001의 두 바이트 조합형과 두 바이트 완성형, 그리고 유니코드 3.0의 조합형 부호계와 완성형 부호계 등 여러가지 부호계를 사용할 때에 대해 기본 검색 알고리즘을 적용해 본다.

키워드
한글인덱스, 한글부호계, 검색

I. 서 론

검색 기능은 데이터베이스에서 원하는 데이터를 찾거나 기존의 데이터를 다른 데이터로 변경하거나, 문서 편집기에서의 검색, 대치 등에 사용되는 데이터 처리의 기본 기능이다.

빠른 단어 검색을 위해서는 효율적인 인덱스를 만드는 것이 가장 일반적인 방법이며 이에 대해서는 많은 연구가 이루어 졌다[1,2,3,4].

그러나 모아쓰기를 하는 한글의 문자적 특성을 고려해 볼 때 검색 기능에서도 글자를 기본 단위로 처리하는 것이 필요한데, 지금 까지 연구된 한글 인덱스는 '가', '각'과 같은 한글의 글자마디를 기본적인 처리대상으로 하기 때문에 한글 글자단위에 대한 처리가 필요할 경우 인덱스의 도움을 받을 수 없는 경우가 대부분이다.

한글 검색을 위한 인덱스 구현에서 글자를 기본 단위로 처리해 주기 위해서는 지금까지의 처리에서와는 다른 여러가지 검색 유형들이 발생할 수 있으며, 특히 불완전한 글자마디를 위한 처리를 고려할 경우는 더 많은 검색 유형을 고려해 주어야 한다. 따라서 본 연구에서는 한글 글자단위 인덱스를 통해 검색을 할 때, 어떠한 검색 유형이 나타날 수 있는지에 대해 살펴보고 한글을 표현하는데 가장 기본이 되는 한글 부호계와의 연관성과 효율성에 대해 살펴본다.

II. 한글 글자 단위 인덱스에서 고려해야 할 검색 유형

서론에서 언급한 것과 같이 한글은 소리글자이며 모아쓰는 특성을 가지고 있다. 즉 'ㄱ','ㄴ','ㄷ' 등의 글자들이 모여서 하나의 글자마디를 구성한다. 지금까지의 한글 검색 기능들을 모두 글자마디를 기본으로 처리하고 있는데 이는 한글의 글자 특성을 볼 때 바람직하다고는 볼 수 없다. 따라서 한글검색에서도 한글 구성의 기본 단위인 글자를 그 기능 구현의 기본 단위로 해야 한다. 이렇게 할 경우 기존의 글자마디를 대상으로 하는 처리는 글자마디가 글자들의 모임으로 간주되

기 때문에 별다른 노력없이 처리할 수 있다.

2.1 한글 글자단위 검색에 필요한 검색 유형

한글에서의 글자마디는 글자의 조합으로 구성된다. 특히 첫소리글자-가운뎃소리글자 또는 첫소리글자-가운뎃소리글자-끝소리글자로 구성된 글자마디를 완전한 글자마디라고 한다.

글자를 기본 단위로 하는 한글 검색에서 나타날 수 있는 검색 유형은 다양하다. 이 다양한 유형은 완전한 글자마디들로 구성된 문자열을 찾고자하는 경우 이외에 그렇지 못한 글자마디 즉 첫-가-끝 글자 중 어느 특정 부분만을 포함하는 검색에 대한 고려도 있어야하기 때문이다. 또한 찾고자 하는 문자열이 검색에 성공할 때에도 다시 두 가지로 나눌 수 있다. 입력되는 문자열이 완전한 글자마디로 구성될 경우에는 이러한 경우가 발생하지 않지만 불완전한 글자마디일 경우에는 이와 꼭 같을 경우만 검색을 성공시킬 것인지 또는 불완전한 부분에 어떠한 글자가 올 경우도 검색을 성공시킬 것인지에 대해서 생각할 수 있어야 한다.

우선 위의 상황을 고려하여 한글에서 나올 수 있는 검색 유형들을 정리해 보면 다음과 같다.

① 첫소리글자가 없는 경우
/*-날수록/ -- 할수록, 갈수록, 잘수록, 등등
② 가운뎃소리글자가 없는 경우
/ㄱ-*-ㄴ/ -- 간, 갠, 건 등
③ 끝소리 글자가 없는 경우
이 경우는 두 가지의 의미를 가질 수 있다.
/가/ 일 경우는 끝소리 글자가 없는 경우 즉 /가/를 찾을 수도 있고 또 /간/,/갈/등의 어떠한 끝소리 글자와도 matching되게 하고자 하는 경우도 있다. 따라서 이 경우는 두 가지로 입력을 나누어 생각하여야 한다.
/가/ -- '가'만 matching /가-*/-- 간, 갈 등
④ 첫소리글자만 있는 경우
/ㄱ-*-*/ -- 가, 간, 갈 등
⑤ 가운뎃소리글자만 있는 경우
/*-ㅏ-*/ -- 가, 나, 간, 날 등
⑥ 끝소리글자만 있는 경우
/*-*-ㄴ다면/ -- 한다면, 간다면 등

2.2 글자 단위 검색에서 사용하고자 하는 알고리즘 - SF 알고리즘

이상에서 한글 글자 단위 검색 기능을 위한 검색 유형들을 정리해 보았다.

위에서 생각해 본 검색유형들을 사용해 실제로 검색을 할 때 각 부호체계에서 어떤 동작이 필요한지를 살펴보자.

위와 같은 검색이 일어날 때 부호체계의 특징을 살펴보기 위해서는 본 작업에서는 이중 가장 이해하기 쉽고 간단한 SF (straight forward) 방식을 사용하여 부호체계간의 특성을 살펴보고자 한다.

일반적인 영문 처리에서의 SF 방식의 알고리즘을 살펴보면 다음과 그림 1과 같다.

```
function search_pattern
begin
/* i : 전체 문자열에서 Match가 일어난 위치
   j : 전체 문자열 에서의 현재 위치
   k : 패턴에서의 현재 위치  */
    while( j < String_length &&
                          k < Pattern_length ){
        if( text[ j ] == pattern[ k ] )
            index j, k를 증가시킴
        else {
            j <- i
            i <- 0
            i <- i + 1
        }
    }
    if ( k 가 패턴의 길이와 같을 때 )
          match success !!
    else
          match fail !!
end
```

그림 1. SF 알고리즘

2.3 한글 부호계의 표준 규격

현재 컴퓨터에서 사용되고 있는 한글 부호계의 표준 규격 번호는 KS X 1001이다. 이것은 기존에 사용되던 표준 규격 번호인 KS C 5601이 KS 규격번호 변경에 따라 번호를 바꾼 것으로 기존의 한글 완성형 부호계로 알려져 있던 KS C 5601 완성형 부호계와 한글 조합형 부호계 중에서 가장 널리 사용되었던 두바이트 상용 조합형 부호계가 포함되어 있다. 또한 ISO10646-1/Unicode 등 국제표준 부호계에도 한글이 포함되어 있다 [6,7].

본 연구는 위에서 언급한 KS X 1001의 완성형, 조합형 부호계, 유니코드 3.0의 완성형, 조합형 부호계(이하, 유니코드 조합형, 완성형)를 대상으로 한다.

2.4 한글 글자 단위 검색알고리즘

기존의 알고리즘들을 사용하여 한글을 처리할 때는 한글을 특성과 사용하는 부호계에 맞게 몇 부분을 수정하여야 하는 것이 일반적이다. 위와 같은 검색을 할 때에도 알고리즘에 글자를 분리하거나 또는 '*' 등과 같은 특수 기능을 하는 문자를 처리하기 위해서 몇 부분이 추가 되어야 한다. 다음에서는 각 부호계를 사용할 때 SF 알고리즘에 어떻게 추가되는지 살펴보겠다.

1) KS X 1001 조합형을 사용하는 경우

KS X 1001 조합형의 경우는 첫소리, 가운뎃소리, 끝소리 글자에 대한 부호같이 두 바이트에 걸쳐 들어 있기 때문에 다음과 같은 작업을 거친 후에 글자를 글자마디에서 분리해 낼 수 있다.

- 첫소리 글자 분리 : ((1'st byte) << 1) >> 3
- 가운뎃소리 글자 분리:
  ((1'st byte) << 6) >>2)+((2'nd byte) >> 5)
- 끝소리 글자 분리 : ((2'nd byte) << 3) >> 3

글자단위의 한글 검색을 할 경우 위와 같은 분리 루틴이 검색 알고리즘에 그림 2와 같이 포함되어야 한다.

```
function search pattern
begin
   .....
  while( j < String_length &&
         k < Pattern_length ){
1. 입력 패턴에서 첫-가-끝소리글자를 분리
2. text에서 첫-가-끝소리글자 분리
  if( 분리해 낸 각 글자 = 패턴의 각 글자 or
              패턴이 '*'일 경우 )
              match 가 진행중....
        }
   else
        .....
    }
   .....
end
```

그림 2. KS X 1001-조합형의 경우 글자 검색

2) KS X 1001 완성형을 사용하는 경우

KS X 1001 완성형을 사용하는 경우는 KS X 1001 조합형의 경우 보다 더욱 복잡하다. 그 이유는 이미 알려진 바와 같이 부호체계 자체가 글자마디를 기본으로 하고 있기 때문에 글자마디에서 글자를 바로 분리해 낼 수 없기 때문이다.

일반적으로 KS X 1001 완성형에서 글자마디를 분리해 내는 방법은 각 부호계에 해당하는 글자마디 변환 테이블을 사용하여 글자마디를 분리해 내게 되는데 이는 완성형 부호계를 조합형 부호계로 변환시켜 글자를 알아내는 것과 같다. 이렇게 할 경우 코드 변환에 필요한 별도의 메모리를 차지 할 뿐만 아니라 코드 변환에 필요한 시간이 더 필요하게 된다.

이때는 KS X 1001 조합형을 사용하는 경우의 글자분리 부분에 추가적으로 완성형 부호계를 조합형 부호계로 변환 시키는 다음의 4단계가 추가된다.

```
function search pattern
begin
   .....
 while( j < String_length &&
        k < Pattern_length ){
    /*- 코드 변환을 위해 추가되는 부분 -*/
 1. 세그먼트 = 상위 1바이트 - B0H
 2. 옵셋 = 하위 1바이트 - A1H
 3. 절대 위치 = 세그먼트 * 94 + 옵셋
 4. 조합형 한글 코드 = 참조 배열[절대 위치]

                  .....
end
```

그림 ?. KS X1001Unicode-조합형의 경우 글자 검색

4) 유니코드 조합형을 사용하는 경우

유니코드 조합형은 앞에서도 설명한 바와 같이 부호계 자체가 글자를 기본 단위로 하고 이를 조합하여 글자마디를 구성하는 조합방식 부호계이다. 또한 글자들이 3-바이트 부호계처럼 바이트 단위로 나누어져 있기 때문에 두 바이트 조합형 부호계에서 글자를 분리 해 내는데 필요한 비트 마스킹 (bit masking) 이나 좌,우 쉬프트 (shift) 작업이 필요 없다는 장점을 가진다.

유니코드 조합형은 끝소리글자가 있는지 없는지에 따라 4 또는 6 바이트로 한 글자마디를 나타내게 되는데 이러한 특징을 검색에서 고려해야 한다.

```
function search pattern
begin
 .....
 while( j < String_length &&
        k < Pattern_length ){
    if( text[j] == pattern[k] ){
        j++; k++;     }
    else if ( pattern [k] == '*' ){
        k++ ;
        if( text[j] != 끝소리 글자 )
            j++ ;
    }
    else
        .....
}
        :
```

그림 3. 유니코드 조합형 경우 글자 검색

4) 유니코드 완성형을 사용하는 경우

유니코드 완성형은 기존의 KS X 1001 완성형 방식에서의 2,350 글자마디를 다루는 방법과는 다르게 요즘 사용되고 있는 한글 11,172 글자마디를 모두 포함하고 있다. 따라서 유니코드 완성형의 경우 조합형 방식으로 부호계를 변환해야 하는 번거로움 없이 다음과 같은 계산식에 의해 첫, 가 끝소리 글자의 가나다순 인덱스를 찾을 수 있다.

```
Index = 부호값 - 0xAC00
첫소리글자인덱스=Index/
            (가운뎃소리글자수 * 끝소리글자수)
가운데소리글자인덱스 = (Index %
 (가운뎃소리글자수 * 끝소리글자수)) / 끝소리글자수
끝소리글자인덱스 = Index % 끝소리글자수
```

위와 같이 글자를 글자마디에서 분리해 낸 후의 검색 방법은 KS X 1001의 조합형 방식과 동일하다.

2.5 결과 비교와 연구

지금까지 한글 글자 단위의 한글 검색 시스템을 구현할 때 필요한 알고리즘의 변형에 대해 SF 방식의 검색 알고리즘을 사용하여 알아보았다.

이때의 알고리즘 변형은 여러 부호계를 사용할 경우 부호계의 특성에 맞게 이루어 져야 한다.

각 경우의 알고리즘 추가에 대한 부담정도를

살펴보면 다음과 같다.

⊙ KS X 1001 - 조합형 - 조합방식이기 때문에 글자마디에서 각 검색에 필요한 글자를 분리해 낼 수 있다. 그러나 각 글자를 분리해 내는데 필요한 7번의 쉬프트 연산과 1번의 논리곱(AND)연산, 또는 그에 상응하는 논리 연산을 필요로 한다.

⊙ KS X 1001 - 완성형 - 완성형 부호계에서는 그 자체로는 글자마디에서 글자를 분리해 낼 수 없다. 따라서 조합형 부호계로의 변환이 필요하다. 변환을 하지 않고 처리하더라도 그에 해당하는 만큼의 연산 부담을 가지게 된다. 위의 예와 같이 상용조합형으로 바꾸어서 처리할 경우에는 4번의 (더하기, 곱하기 각 1 번, 빼기 2번, 배열 참조 1번) 산술연산이 필요하다. 그리고 두 바이트 조합형의 경우와 마찬가지로 글자를 분리하는데 필요한 연산을 해야 한다.

⊙ 유니코드-조합형 - 유니코드 조합형 부호계는 별다른 연산 부담 없이 검색을 할 수 있다. 그러나 끝소리 글자 유b에 따라 4-6바이트로 나타내는 가변 부호계이기 때문이 이를 처리해 주는 부분이 필요로 하게 된다.

⊙ 유니코드-완성형 - 이 경우는 완성형 부호계이기는 하지만 별도의 코드 변환 없이 계산식에 의해 각 글자를 분리해 낼 수 있다. 각 글자를 분리해 낼 때 드는 비용은 총 9번의 산술연산(빼기 3번, 나누기 2번, 곱하기 2번, 나머지연산 2번)이 필요하다.

위의 결과에 따른 검색 효율의 순서를 따져 보면 다음과 같다.

KS X 1001 완성형 << 유니코드 완성형
<< KS X 1001 조합형 << 유니코드 조합형

위에서 알 수 있는 가장 중요한 결과로는 한글의 구성단위를 글자(첫소리, 가운뎃소리, 끝소리)로 할 때 조합형 부호계와 같이 그 부호계가 이를 잘 반영하고 있어야만 프로그램 작성뿐만 아니라 속도, 이해도 면에서 더 효율적이다.

또한 같은 조합방식으로 글자마디를 표현하는 부호계라 할지라도 글자가 바이트 단위로 나누어져 있는 경우(유니코드 조합형)와 그렇지 않은 경우(KS X 1001 조합형)가 다르다. 즉 바이트 단위로 나누어져 있는 경우는 글자를 분리해 내기 위해서 쉬프트 (shift) 연산을 하거나 비트 마스킹(bit masking)을 할 필요가 없지만, 그렇지 않은 경우는 비트 단위의 연산을 통해 글자마디에서 글자를 분리해 내어야 한다.

따라서 글자단위의 한글 검색을 위한 인덱스 구성 등 글자단위의 정보를 필요로 하는 작업들에서는 이러한 정보를 부호계에서 반영하고 있어야 하며 또한 부호계 자체가 바이트 단위로 나누어져 있는 경우에서는 더욱 유리하다는 것을 잘 알 수 있다.

III. 결론

이상에서 한글 글자단위 인덱스 구현을 위한 글자단위 검색 기능에서 기존의 글자마디 방식의 검색 이외에 글자단위의 검색 기능을 구현하는데 있어서 가능한 입력 패턴들을 정리하고, 이를 현재 사용되고 있는 부호계들에서 구현할 때 필요한 알고리즘의 추가에 대해 살펴보았다.

글자단위 검색에서 꼭 필요한 글자 분리 부분은 부호계 자체가 이를 반영하고 있을 경우, 즉 조합 방식의 부호계일 경우가 효율적이다. 이는 단순히 글자 검색 기능에서 유리하다고 말할 수 있기보다는 한글의 특성을 더 잘 반영하고 있는 부호체계라는 말이 될 것이다. 또한 조합방식의 부호체계에서도 별도의 연산을 하지 않고 글자마디에서 글자를 분리해 낼 수 있는 경우는 글자를 기본 단위로 처리하고자 하는 분야에서는 더욱 편리하게 사용될 수 있을 것이다.

참고문헌

[1] 박미란, 나연묵멀티미디어학회 논문제 제1권 제2호(1998.12) 162-172

[2] 김철수(Cheol-Su Kim)외 2인, 이중 배열 트라이 구조를 이용한 학국어 전자 사전의 구축, 정보과학회논문지(B) 제23권 제1호, 1996. 1, pp. 85 ~ 94 (10pages)

[3] 이근용 외 2인, 사전 검색 알고리즘을 이용한 자소 단위 한국어 형태소 분석, 한국정보과학회 1995년도 가을 학술발표논문집 제22권 제2호(A), 1995. 10, pp. 619 ~ 622

[4] 김희철 외2인, 다차원 이진트리를 한글색인사전의 구현, ,한국정보과학회 1998년도 봄 학술발표논문집 제25권 제1호(B), 1998. 4, pp. 452 ~ 454

[5] 김경석, 컴퓨터속의 한글이야기 둘째보따리-유니코드 3.0 및 ISO/IEC 10646 소개, 부산대학교 출판부, 1999

[6I ISO/IEC 10646-1:1993(E). International Standard. 1st edition. Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane, May 1, 1993. ISO

[7] The Unicode Consortium, The Unicode Standard, Version 5.0, Addison-Wesley Professional, 2006