한글 상호(商號)를 로마자로 변환하기 위한 고속 부분문자열 분석 알고리즘

황명진*, 조선호, 권혁철
부산대학교 컴퓨터공학과
e-mail:(holgabun*, sean, hckwon)@pusan.ac.kr

High Speed Substring Analysis Algorithm for Converting from the Korean Company Name to Roman Characters

Myeong-jin Hwang, Sun-ho Jo, Hyuk-chul Kwon
Korean Language Processing Lab,
School of Electrical & Computer Engineering,
Pusan National Universty

요 약

한글 상호(商號) 로마자 변환기는 한글로 만들어진 상호를 로마자로 자동 변환하는 시스템이다. 이 변환기는 기사용 로마자 상호명과 업종명, 그리고 표준 한글 로마자 변환 규칙에 의해 생성한 로마자를 조합하여 로마자 상호를 생성한다. 이때, 조합을 위한 알고리즘이 필요한데, 기존에 비슷한 용도에 사용되었던 stack 알고리즘을 적용할 경우 비효율적이다. 본 논문은 이를 대체할 새 알고리즘을 제안한다. 새 알고리즘은 기존 stack 알고리즘을 사용할 때에 비해 복잡도를 $O(b^d)$에서 $O(b*d)$로 줄여 성능을 높인다.

1. 서론

국내에서 법인의 상호는 대법원의 등기소에서 등기되어 관리되고 있다. 2007년 개정된 상업등기법에 따라 전산화 환경에 맞추어 전자신청 제도 등의 도입이 시행되었는데, 등기기록과 신청서 등 등기에 관한 서면에서는 한글과 아라비아 숫자를 사용하도록 하되, 상호와 외국인의 성명은 대법원 예규로 정하는 바에 따라 한글, 아라비아 숫자와 함께 괄호 안에 로마자, 한자, 아라비아 숫자 그리고 부호를 병기할 수 있도록 한다. [2]

상호 로마자 변환이란 등기되었거나 등기하려는 한글 상호(商號)를 로마자로 표기하고자 할 때 적합하게 변환하는 것을 말한다. 여기에서 가장 기준이 되는 방법은 한글의 로마자 변환 방법을 따르는 것으로 이는 '문화관광부고시 제2000-8호(2000. 7. 7.)'로 정해져 있다. [3]

그러나 어떤 로마자 상호는 이전부터 각자 사용해온 로마자 표기가 있으므로 대부분 사람이 이 표기에 대해 그 표기의 유명성을 인지하고 있다면, 권리자에게 이를 허용해 주어야 한다. 예를 들어 '삼성전자'에서 '삼성'을 로마자 변환 표준에 따라 변환한다면 'Samseong'이 되나, 이미 이 회사에서는 오랫동안 'Samsung'을 사용하고 있고 이것이 보편적으로 사회에서 받아들여져 있으므로 이를 허용할 수도 있다. 또한 '쓰리넷'의 경우 개별 기업의 니드에 맞춰 원한다면 'Sseurinet'이 아닌 '3Net'으로도 변환할 수 있도록 숫자나 영어를 음차한 표현은 원래 형태로 복원해 줄 필요도 있다. 심지어 기존에 사용 중인 업체명 중에는 '오륙'을 '5.6 Co., Ltd.'로 사용하는 것처럼 부분 문자열 조합으로는 생성할 수 없는 경우도 많으므로, 이런 경우는 기존 명칭도 함께 제공하도록 한다.

따라서 '상호 로마자 변환기'를 구현하려면 '영어 상호 고유명사 사전', '상호명 사전', '업종명 사전', 알파벳과 숫자 변환을 위한 '기호 사전' 검색 결과와 로마자 변환 규칙에 따라 생성한 결과를 조합하여 로마자 상호를 생성해야 한다.

본 논문에서는 이를 달성하기 위한 최적화된 '고속 부분문자열 분석 알고리즘'(HSSA 알고리즘)을 제시한다.

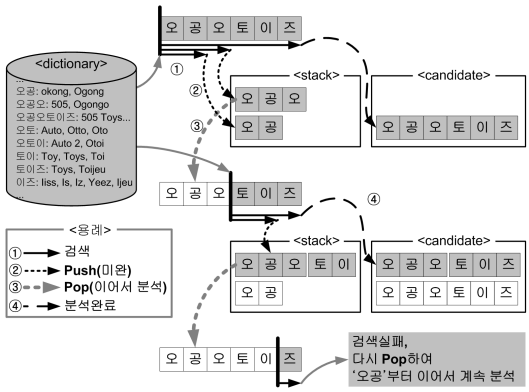

그림 1. 스택 알고리즘을 이용한 부분문자열 분석 과정

2. 기존 방법

로마자 상호 생성 시 부분문자열을 검색/조합하는 과정이 필요하다. 스택 알고리즘을 이용해 이 과정을 처리할 수 있다. 그러나 필요치 않은 작업을 반복 수행하는 문제가 있다.

스택 알고리즘을 이용하여 부분문자열을 조합하는 과정은 다음과 같다. [1]

```
스택-알고리즘 ( inputString )
    Clear ( result )
    Clear ( foundSubStr )
    Clear ( 스택 )
    startIdx := 0
    WHILE startIdx <= end of inputString DO
        FOR i := startIdx to end of inputString DO
            substr := GetSubstr ( from startIdx to i )
            SearchDic ( substr )
            IF found substr in the dictionary THEN
                newFoundSubStr := foundSubStr + substr
                IF newFoundSubStr is complete THEN
                    Add ( result to newFoundSubStr )
                ELSE
                    Push ( newFoundSubStrList to 스택 )
                ENDIF
            ENDIF
        ENDFOR
        IF 스택 is not empry THEN
            foundSubStr := Pop ( from 스택 )
            startIdx := lenOfStr ( foundSubStr )
        ELSE
            startIdx := end of inputString + 1
        ENDIF
    ENDWHILE
    Display result
STOP
```

스택 알고리즘을 이용하여 '아이텍스포츠'를 분석하면 아래와 같다.

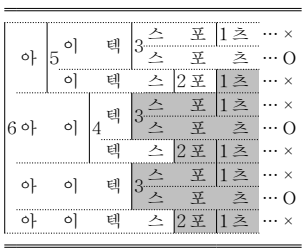

그림 2. 스택 알고리즘을 이용한 부분문자열 분석

전체 9가지 경우를 끝까지 분석해 보았고, 그 중 3개만 분석이 완료되었음을 확인할 수 있다. 위 그림의 숫자는 각 검색 시작점에서 사전 검색을 한 횟수이다. 예를 들어, '아이'까지는 분석을 마쳤을 경우, 남은 문자열은 '텍스포츠'이며, 이에 대해, '텍', '텍스', '텍스포', '텍스포츠'을 각각 1회, 총 4회 검색한 후에야 '텍'과 '텍스'만 사전에 있음을 알 수 있다. 따라서 '아이텍스포츠'를 분석하기 위해서는 총 36회의 사전 검색을 하여야 한다.

위 그림에서 음영 부분은 스택 알고리즘이, '분석 결과는 다르지만, 경계가 같은 여러 후보'들의 뒷부분을 각각 따로 중복으로 분석했음을 보여준다. 즉, '아+이텍', '아이+텍', '아이텍'은 경계가 모두 동일하여 뒷부분 '스포츠'에 대해서 한 번만 분석(3회+1회 검색)하면 되지만, 스택 알고리즘은 스택을 사용하므로 같은 분석을 2번 더 반복한다. 위의 예에서, 이러한 중복 분석을 방지하면 사전 검색 횟수를 36회에서 21회로 줄일 수 있다.

최악의 경우 하나의 한글 상호명을 변환하기 위해 수만 번의 검색을 해야 할 수도 있다. '뉴이십일세기영화사'의 경우, 분석 가능한 토큰이 '뉴', '이십일', '이십일세기', '일', '세기', '기', '기영', '영', '영화', '영화사'이고, 영어 대역의 수가 각각 '이십일세기'는 8가지, '세기'는 20가지, '영화'는 12가지 등으로 많다. 이 예는, 아무 생각 없이 스택 알고리즘을 사용했을 때 '뉴|이십일|세기|영화|사'로 분석되는 경우만 176만 번 이상의 검색이 필요하지만, 중복 검색을 피할 경우 1천여 번으로 검색 횟수를 줄일 수 있다.

따라서 중복 검색을 피하는 고속 부분문자열 분석 방법이 필요하다.

3. 개선된 방법

제안하는 고속 부분문자열 분석 알고리즘(이하 HSSA 알고리즘)은 중복 분석을 방지하여 분석 속도를 향상시키는 알고리즘이다. HSSA 알고리즘의 기본 처리 과정은 아래와 같다.

```
HSSA-알고리즘 ( inpStr )
    ;each substrStart in inpStr has saved there end position
    Clear ( 스택 )
    startIdx := 0
    WHILE startIdx <= end of inpStr DO
        FOR i := startIdx to end of inpStr DO
            substr := GetSubstr ( from startIdx to i )
            SearchDic ( substr )
            IF found substr in the dictionary THEN
                SaveEndIdx ( i at startIdx in inpStr )
                Push ( i to 스택 )
            ENDIF
        ENDFOR
        IF 스택 is not empry THEN
            startIdx := Pop ( from 스택 )
        ELSE
            startIdx := end of inpStr + 1
        ENDIF
    ENDWHILE
    Clear ( result )
    Compile ( inpStr, "", result )
END

Compile ( inpStr, compiled, result )
    ;compiled: partially compiled substring
    ;each startIdx in inpStr has saved there end position
    startIdx := end of compiled
    FOREACH endIdx DO
        endIdx = GetEnd ( from startIdx in inpStr )
        substr := GetSubstr ( startIdx, endIdx )
        IF endIdx is end of inpStr THEN
            Add ( compiled + substr at result )
        ELSE
            Compile ( inpStr, compiled + substr, result )
        ENDIF
    ENDFOR
STOP
```

HSSA 알고리즘은 각 부분문자열의 시작 위치에 해당 부분문자열의 끝 위치 정보를 기억해 두는 방식으로 부분문자열을 분석해 낸 후, 전방에서 후방으로, 후방에서 전방으로 탐색하며 부분문자열을 조립하는 과정을 거쳐 전체 문자열을 만들어 낸다.

이 알고리즘은 하나의 부분문자열에 대해 한 번만 검색을 함으로써 중복 검색을 피할 수 있고, 결국 부분 문자열 분석 속도를 향상시킬 수 있다.

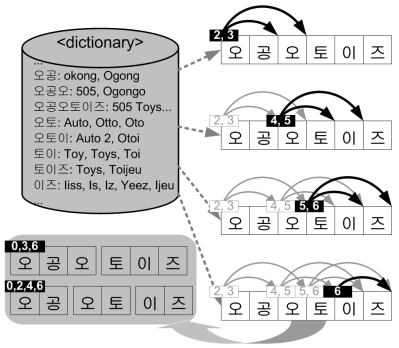

그림 3. 고속 부분문자열 분석(HSSA) 알고리즘을 이용한 분석 과정

4. 계산 복잡도

아래 그림은 스택 알고리즘을 이용한 부분문자열 분석 결과인 그림 2를 트리형태로 표현한 것이다. 그림에서 음영으로 표한 부분은, 첫 음절의 위치와 마지막 음절의 위치가 동일한 부분문자열을 나타낸다. 이것을 앞으로 '부분문자열 노드'라 하겠다. 이 부분문자열 노드는 복잡도 계산을 단순하게 만들어 준다.

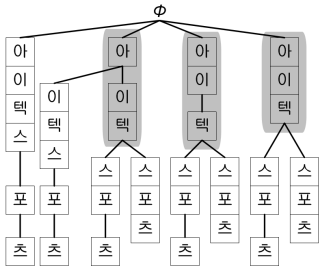

그림 4 . 스택 알고리즘을 이용한 부분문자열 분석 결과를 트리 형태로 표현

그림 5는 부분문자열 분석 결과를 트리 형태로 나타낸 뒤 부분문자열 노드를 이용해 단순화하여 표현한 그림이다. 부분문자열 노드의 평균 자식 수가 b이고, leaf 노드까지의 평균 깊이가 d라고 할 때, 그림 (A)와 (B)의 전체 노드 수는 다음과 같이 계산된다. [4]

(A) 스택 알고리즘: b^d
(B) HSSA 알고리즘: $b \times d$

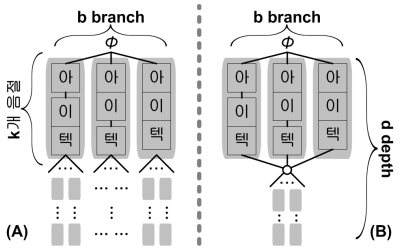

그림 5. 단순화한 부분문자열 분석 결과의 트리 형태(A: 스택 알고리즘을 이용한 경우, B: HSSA 알고리즘을 이용한 경우)

5. 결론

이 논문에서는 상호 등기를 위해 한글 상호(商號)에 대응하는 적합한 로마자 상호 후보를 생성해내는 변환을 위한 고속 부분문자열 분석 알고리즘을 제시했다. 로마자 상호의 생성을 위해서는 로마자 변환으로 생성하는 문자열과 기존에 사용되는 고유 문자열을 구분하는 것이 중요하다. 제시된 한글 상호(商號)의 문자열에서 부분문자열을 분석하는 알고리즘이 핵심적으로 필요한 데, 기존의 비슷한 용도에 사용되었던 stack 알고리즘을 적용할 경우는 복잡도가 $O(b^d)$으로 비효율적이다. 이 문제를 분석하여 제안된 알고리즘은 복잡도를 O(b*d)로 줄일 수 있어 성능을 비약적으로 향상시켰다.

Acknowledgements

이 논문은 2008년도 정부(과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임
(No. R01-2007-000-20517-0)

참고문헌

[1] 김영택 외, "*자연언어처리*", 생능출판사, 2001

[2] (대법원규칙 제2129호)상업등기규칙,http://www.iros.go.kr, 대법원 인터넷등기소, 2007. 12. 24

[3] 로마자표기법, http://www.korean.go.kr/08_new/data/rule04.jsp, 국립국어원, 2000. 7. 7.

[4] R. P. Grimaldi : "Discrete And Combinatorial Mathematics", Addison Wesley Longman, 2000