Finite State Transducer를 이용한 한국어 전자 사전의 구조

백 대 호° , 이 호, 임 해 창
고려대학교 전산과학과

A Structure of Korean Electronic Dictionary using the Finite State Transducer

Dae-Ho Baek° , Ho Lee, Hae-Chang Rim
Department of Computer Science, Korea University

요 약

한국어 형태소 해석기와 같은 한국어 정보 처리 시스템은 많은 전자 사전 검색 작업을 요구하기 때문에 전자 사전의 성능은 전체 시스템의 성능에 많은 영향을 미친다. 이에 본 논문은 적은 기억 장소를 차지하면서 탐색 속도가 빠른 Finite State Transducer(FST)를 이용한 전자 사전 구조를 제안한다.

제안된 전자 사전은 Deterministic Finite State Automata(DFA)로 표제어를 표현하고 DFA 상태수 최소화 알고리즘으로 모든 위치에 존재하는 중복된 상태를 제거하여 필요한 기억 장소가 적으며, FST를 일차원 배열에 매핑하고 탐색시 이 배열내에서의 상태 전이만으로 탐색을 하기 때문에 탐색 속도가 매우 빠르다. 또한 TRIE 구조에서와 같이 한번의 탐색으로 입력된 단어로 가능한 모든 표제어들을 찾아 줄 수 있다.

실험 결과 표제어 수가 증가하여도 FST를 이용한 전자 사전의 크기는 표제어 수에 비례하여 커지지 않고, 전자 사전 탐색 시간은 표제어 수에 영향을 받지 않으며, 약 237만 단어를 검색하는 실험에서 TRIE나 B⁺-Tree 구조를 사용한 전자 사전보다 빠름을 알 수 있었다.

1. 서 론

컴퓨터의 대중화와 정보량의 급증에 따라 컴퓨터를 이용한 정보 처리 분야가 점차 확산되고 있다. 이러한 정보를 처리하는 대부분의 시스템은 전자 사전을 필요로 하고, 특히 자연어 처리 분야의 형태소 해석기와 같은 시스템은 전자 사전 참조가 매우 빈번하기 때문에 사전의 성능이 정보 처리 시스템의 성능에 많은 영향을 미친다. 정보 처리 시스템에서 필요로 하는 전자 사전은 사전의 참조가 빈번하므로 검색 속도가 빨라야 하고, 사전 표제어 수가 매우 많기 때문에 사전의 표제어 수가 증가해도 사전의 크기는 많이 커지지 않는 구조를 가져야 한다[1,2].

현재 한국어 정보 처리 분야에서 사용되고 있는 전자 사전 구조는 크게 TRIE 구조, B-Tree/B⁺-Tree 구조, Hashing을 이용한 구조 등으로 분류할 수가 있다. TRIE 구조는 삽입 단어의 개수와는 무관하게 매우 신속한 검색이 가능하며 길이가 일정하지 않은 단어들을 효율적으로 다룰 수 있다. 또한 한국어 문서의 형태소 해석시 어절당 한번의 검색만으로 가능한 모든 표제어를 찾을 수 있어 사전 검색 횟수를 줄일 수 있다[2]. B-Tree/B⁺-Tree구조는 삽입 및 삭제가 용이하고 최하위 레벨에서는 한 노드내에 연속된 표제어들이 존재하기 때문에 유사한 표제어를 연속해서 찾을 경우 캐쉬 기법을 이용하여 탐색 속도를 향상시킬 수 있다. 그러나 탐색 속도가 TRIE보다 떨어진다[2,3]. Hashing을 이용한 구조는 탐색 속도가 매우 빠르다는 장점이 있다. 그러나 해싱값 충돌문제의 처리가 필요하고 가변적인 표제어를 효율적으로 처리하지 못하는 문제점이 있다.

본 논문은 한국어 형태소 해석기와 같은 한국어 정보 처리 시스템에 적합한 전자 사전 구조로써 Finite State Transducer(FST)를 이용한 전자 사전 구조를 제안한다.

2. Finite State Transducer를 이용한 한국어 전자 사전의 구조

본 논문에서 제안된 전자 사전은 표제어의 인덱스 값을 계산

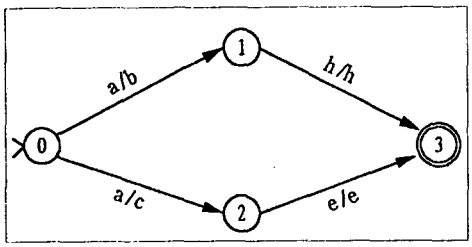
하기 위하여 해싱 기법을 사용한다. 제안된 해싱 기법은 FST를 사용하여 Perfect Hashing Function을 만드는 것이다. 2.1에서 FST에 대해서 정의를 내리고, 2.2에서 FST로 표현된 표제어를 Perfect Hashing Function이 인덱스 값을 계산하는 방법을 설명한다.

2.1 FST의 정의

본 논문에서 제안한 전자 사전의 기본 구조인 FST는 다음과 같이 다섯 개의 요소로 이루어진다.

$$T=(\Sigma, Q, i, F, E)$$

여기서, Σ는 유한 개수의 알파벳, Q는 상태의 유한 집합, $i \in Q$는 초기 상태, $F \subseteq Q$는 최종 상태의 집합, $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \Sigma^* \times Q$는 전이함수의 집합이다[8].



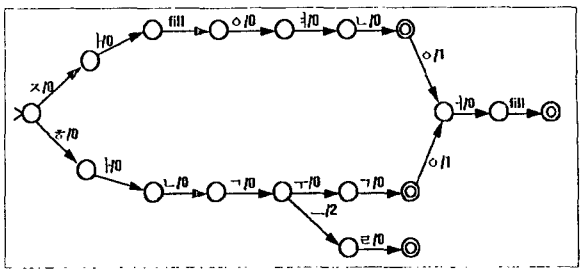
[그림 1] 간단한 Finite State Transducer

예를 들어, [그림 1]은 다음과 같이 나타낼 수 있다.

$$T1 = (\{a, b, c, h, e\}, \{0, 1, 2, 3\}, 0, \{3\}, \{(0, a, b, 1), (0, a, c, 2), (1, h, h, 3), (2, e, e, 3)\})$$

2.2 사전 표제어의 표현 방법

FST를 이용한 Perfect Hashing Function은 단순히 FST의 출력값을 합하는 함수이다. 이렇게 계산된 출력값의 합이 그 표제어의 인덱스 값이 될 수 있도록 하기 위해서는 FST의 출력값을 적절히 설정해야 한다. FST의 출력값을 설정하는 방법은 3장에서 자세히 설명하고, 여기서는 만들어진 FST의 모양과 이 FST를 이용하여 표제어들의 인덱스 값을 계산하는 것을 보인다.



[그림 2] FST를 이용한 Perfect Hashing Function

[그림 2]가 포함하고 있는 표제어는 "자연, 자연어, 한국, 한국어, 한글" 이다. 각 표제어의 인덱스 값은 해싱 함수에 의해서 다음과 같이 계산된다.

H(자연) = 0 + 0 + 0 + 0 + 0 + 0 = 0
H(자연어) = 0 + 0 + 0 + 0 + 0 + 0 + 1 + 0 + 0 = 1
H(한국) = 2 + 0 + 0 + 0 + 0 + 0 = 2
H(한국어) = 2 + 0 + 0 + 0 + 0 + 0 + 1 + 0 + 0 = 3
H(한글) = 2 + 0 + 0 + 0 + 2 + 0 = 4

3. FST를 이용한 전자 사전의 구현 방법

사전 표제어를 FST로 표현하고 각 상태의 출력값들의 합이 그 표제어의 인덱스 값이 되게 하기 위해서는 다음과 같은 과정을 거쳐야한다. 사전 표제어를 DFA로 표현하고, 중복되는 상태들을 제거하기 위해서 DFA 상태수 최소화 알고리즘을 적용하고, DFA의 각 상태에 출력값을 설정하여 FST로 만든다.

3.1 Deterministic Finite State Automata(DFA)를 이용한 표제어의 표현

DFA도 FST와 마찬가지로 다음과 같이 다섯 개의 요소로 표현된다.

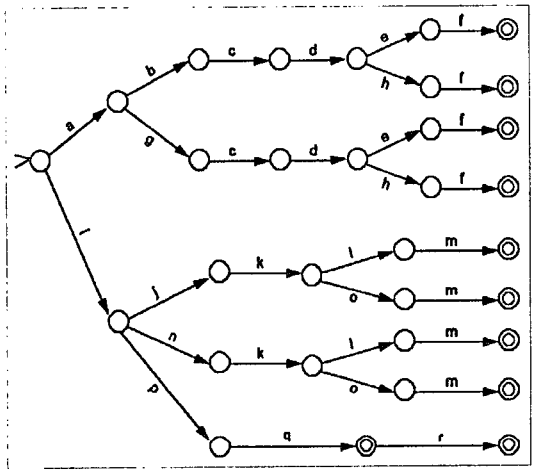
$$M=(\Sigma, Q, i, F, E)$$

여기서, Σ는 유한 개수의 알파벳, Q는 상태의 유한 집합, $i \in Q$는 초기 상태, $F \subseteq Q$는 최종 상태의 집합, E는 $Q \times \Sigma$에서 Q로의 전이함수이다[9].

한국어를 표현하기 위해서 Σ는 한글의 자소 즉, 'ㄱ', 'ㄴ', ...,'ㅏ','ㅐ',... 등이 된다. 사전 표제어를 DFA로 만들기 위해서

표제어가 들어오면, 표제어를 DFA의 입력 alphabet인 한글의 자소로 분리한다. 분리된 자소의 열을 가지고 DFA를 탐색하면서 존재하지 않는 상태전이가 생기면 새로운 상태를 만들어 나간다. 이러한 탐색과 새로운 상태를 생성하는 과정을 모든 표제어의 입력이 끝날 때까지 계속한다.

이렇게 해서 만들어진 DFA는 다음 그림과 같이 표제어의 중복된 앞 부분만이 압축된 일반적인 TRIE 형태이다. [그림 3]은 표제어 abcdef, abcdhf, agcdef, agcdhf, ijklm, ijkom, inklm, inkom, ipq, ipqr 이 입력된 DFA이다.



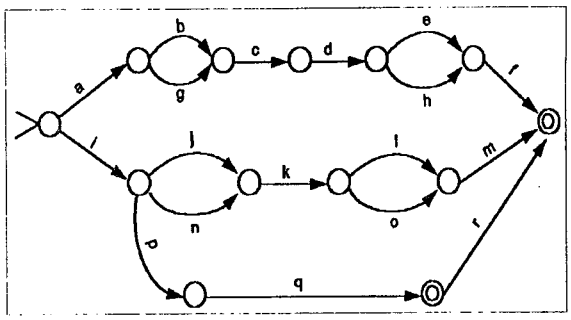
[그림 3] 표제어가 입력된 DFA

3.2 DFA의 상태수 최소화

3.1 에서 만들어진 DFA에서는 앞부분에서 중복되는 상태들은 제거되었지만, 그 이후에 중복되는 많은 상태들은 전혀 제거되지 않고 존재한다. 이러한 중복된 상태가 많이 존재하면 DFA를 표현하기 위해서 많은 기억공간이 필요하게된다. 그러므로 중복된 상태들을 줄이면서 원래의 DFA가 포함하고 있는 표제어를 변화시키지 않는 방법이 요구된다. 즉, DFA의 상태수를 최소화하면서 그 특성을 변화시키지 않는 방법이 필요하다. 이 방법을 DFA 상태수 최소화 알고리즘이라 한다.

상태수 최소화 알고리즘은 입력 일에 의해서 구분할 수 있는 모든 상태들의 집합을 구해서, 구분할 수 없는 상태들의 집합을 하나의 상태로 합침으로써 DFA의 상태수를 줄인다[5,6].

[그림 4]는 [그림 3]에 있는 DFA의 상태수를 최소화한 DFA를 나타낸 것이다. [그림 3]에서는 상태수가 32개 였는데, 최소화 과정을 거친 후에는 상태수가 13개로 줄어들었다. 그러나 [그림 3]과 [그림 4]가 나타내는 DFA는 완전히 같은 표제어를 포함하고 있는 DFA이다.



[그림 4] 상태수가 최소화된 DFA

3.3 FST의 출력값 계산 방법

이 장에서는 표제어를 찾기 위해서 FST의 상태를 전이하면서 발생하는 출력값들의 합이, 그 표제어의 인덱스 값이 되도록 FST의 출력값을 설정하는 것을 설명한다. 각 상태에서의 FST의 출력값은 다음과 같이 정의된다.

S_i : i 번째 상태
G_i : S_i 에서 생성될 수 있는 부분 표제어의 수 (*Generation Number*)
C_i : S_i 에서 입력받을 수 있는 입력 심볼의 집합 $= \{C_{i,1}, C_{i,2}, \ldots, C_{i,n}\} \subseteq \Sigma$ (단, $C_{i,j} < C_{i,j+1}$)
$N_{i,j}$: S_i 에서 $C_{i,j}$ 가 입력되었을 때 전이되는 상태
$O_{i,j}$: S_i 에서 $C_{i,j}$ 가 입력되었을 때의 출력값

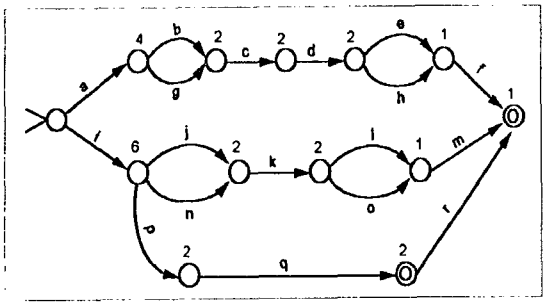
$$= \begin{cases} 0 & \text{if } j=1,\ S_i \notin F \\ 1 & \text{if } j=1,\ S_i \in F \\ O_{i,j-1} + G_{N_{i,j-1}} & \textit{otherwise} \end{cases}$$

FST의 각 상태에서의 출력값을 계산하기 위해서는 DFA 각 상태에서의 Generation Number를 알아야만 한다. DFA의 Generation Number를 계산하는 알고리즘은 [그림 5]와 같다.

```
int SetGenNum(int StateNum)
{
    CurrentValue = 0;

    for (ch = 0; ch < 마지막_알파벳; ch++) {
        if (상태전이가 없으면) continue;
        NextStateGenNum = DFA[NextState][GenNum];
        if (NextStateGenNum != 0) CurrentGenNum += NextStateGunNum;
        else CurrentGenNum += SetGenNum(NextState);
    }

    if (종료 상태이면) CurrentGenNum++;
    DFA[StateNum][GenNum] = CurrentGenNum;

    return CurrentGenNum;
}
```

[그림 5] DFA 각 상태의 Generation Number를 계산하는 알고리즘

[그림 5]의 알고리즘을 사용하여 [그림 4]의 각 상태에서의 Generation Number를 계산하면 [그림 6]과 같다.

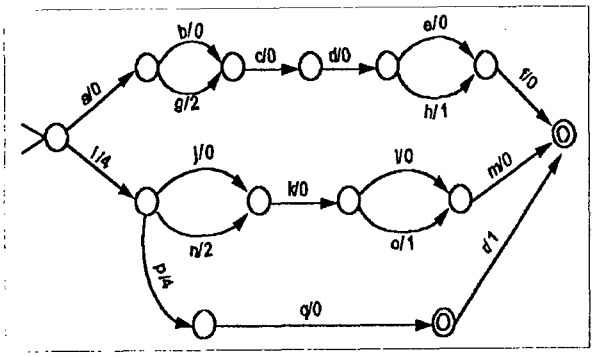


[그림 6] DFA 각 상태에서의 Generation Number

이제 계산된 Generation Number를 가지고 FST 각 상태의 출력값을 설정해야 한다. FST의 각 상태에서 그 상태의 Generation Number를 가지고 FST의 출력값을 계산하는 수식은 다음과 같다.

$$O_{i,j} : S_i \text{ 에서 } C_{i,j} \text{ 가 입력되었을 때의 출력값}$$
$$= \begin{cases} 0 & \text{if } j=1,\ S_i \notin F \\ 1 & \text{if } j=1,\ S_i \in F \\ O_{i,j-1} + G_{N_{i,j-1}} & \textit{otherwise} \end{cases}$$

위의 수식에 의해서 [그림 6]의 DFA에 출력값을 계산하면 [그림 7]과 같이된다.

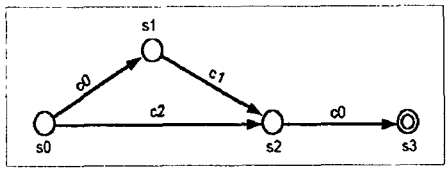


[그림 7] 출력값이 설정된 FST

3.4 FST의 저장 구조

지금까지 설명한 FST를 구현하는 데에는 링크드 리스트, 배열, 다른 복합적인 구조 등이 사용될 수 있다. 어느 것이 더 효율적인 가하는 것은 FST의 각 상태에서 사용되는 알파벳의 수와 전이되는 알파벳 수의 평균, 구조가 자주 변해야 하는가 그렇지 않은가, 여기에 적용되는 연산의 종류 등에 따라 달라진다. 링크드 리스트 방법은 삽입과 삭제가 많아서 구조가 자주 변해야 하는 응용분야에 적합하고, 배열을 사용한 방법은 구조가 변하지 않고 임의의 접근이 많은 응용분야에 더 적합하다. 본 논문에서 제안한 전자 사전은 형태소 분석기와 같은 빈번한 사전 탐색을 요구하고, 표제어가 잘 바뀌지 않는 시스템을 목표로 하여 만들어 졌기 때문에 FST를 배열로 표현한다.

[그림 8]은 DFA를 2차원 배열로 나타내는 것을 보여준다. [그림 8 (B)]의 배열은 열이 알파벳을 나타내고, 행이 상태를 나타낸다. 그리고 배열의 내용은 다음 상태를 나타내고, 내용이 -1 이면 종료 상태이다.



[그림 8] (a) 간단한 DFA

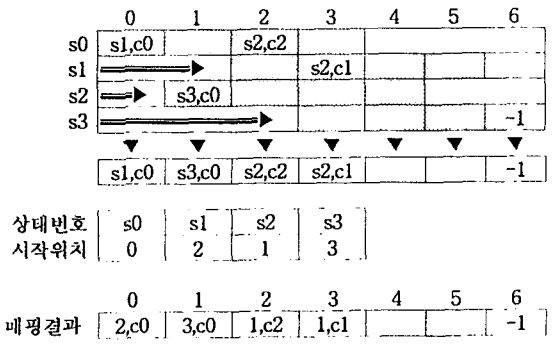
	c0	c1	c2	Final
s0	s1		s2	
s1		s2		
s2	s3			
s3				-1

[그림 8] (b) 배열을 이용한 DFA의 표현

한국어 사전을 FST로 구성할 때 알파벳은 자소, 음절, 1 바이트 문자 등이 될 수 있다. 그러나 자소, 음절이나 1 바이트 문자를 FST의 알파벳으로 사용할 경우에는 알파벳의 종류가 너무 많기 때문에 배열로 FST를 표현할 경우에 하나의 행이 너무 많은 열을 가지므로 기억 장소의 낭비가 심하다. 이를 해결하기 위해서 "초성 다음에는 중성이 중성 다음에는 종성이 나타난다"는 한국어 코드의 특성을 고려하여 새로운 알파벳을 정의한다. FST를 정의할 때 알파벳은 한글 상용 조합형 코드에서 초성, 중성, 종성을 분리하였을 때 분리된 값으로 한다. 그리고 초성, 중성, 종성의 구분은 그 상태가 종료 상태인지 아닌지 구분하는 것을 변형하여 초성, 중성, 종성, 종료 종성 등으로 분류하였다. 이렇게 함으로써 알파벳의 수를 줄였다.

배열을 사용하여 FST를 표현하면 각 상태에 대한 접근이 빠르다는 장점이 있지만, FST를 표현한 2차원 배열은 사용되지 않는 부분이 너무 많아서 기억장소의 낭비가 너무 심하다. 이 기억장소의 낭비를 줄이기 위해서 2차원 희소 배열을 1차원 배열상에 매핑하는 기법을 사용한다[10]. DFA를 2차원 배열로 나타내는 것은 [그림 8]과 같지만, FST를 2차원 배열로 나타내기 위

해서는 배열의 요소가 전이될 상태뿐만 아니라 출력값을 포함하고 있어야 한다. 그리고 1차원 배열 상에 매핑하기 위해서 입력 알파벳을 포함해야 한다. [그림 9]는 [그림 8]의 DFA를 1차원 배열에 매핑하는 과정을 보이고 있다. FST를 1차원 배열에 매핑하는 과정은 DFA에서와 같지만 출력값이 첨가되어야 한다.



[그림 9] DFA를 1차원 배열에 매핑하는 방법

2차원 배열의 행을 하나의 슬롯으로보고 슬롯의 시작 위치가 겹치지 않고, 배열의 내용이 있는 부분이 겹치지 않게 슬롯을 우측으로 움직이면서 1차원 배열에 매핑시킨다. 이 때, 각 슬롯의 시작 위치를 기록해 둔다. 그리고 나서 매핑된 1차원 배열에서 내용의 각 상태 번호를 그 상태의 시작 위치로 모두 바꾼다. 이렇게 만들어진 1차원 배열에서 각 상태를 전이하는 과정은 4장에서 설명한다. FST를 배열로 표현했을 때의 또 다른 장점은 디스크에 있는 사전을 메모리로 로드하는데 다른 부가적인 연산이 필요하지 않으므로 처음에 사전을 디스크로부터 읽어들이는데 매우 빠르다는 것이다.

4. Finite State Transducer를 이용한 전자 사전 탐색 방법

[그림 10]은 [그림 9]에 나타난 1차원 배열을 이용하여 입력이 "c0, c1, c0", "c0, c2" 일 때의 전이하는 과정을 나타낸 것이다.

먼저 입력이 "c0, c1, c0"인 경우에 c0가 들어오면 초기 상태는 s0이고 s0의 시작 위치에서 c0의 오프셋을 더한 곳에 있는 ch가 c0인지 본다. 만약 c0가 아니면 다음 상태로 가는 전이가 없다는 것이고, c0이면 다음 상태로 c0를 가지고 전이를 할 수가 있다는 것이다. 전이할 수 있는 다음 상태의 시작위치는 Next란 곳에 저장되어 있다. 다음 상태가 종료 상태인지를 알기 위해서 다음 상태의 시작 위치에 전체 알파벳의 수인 3을 더한 위치에 있는 값을 본다. 만약 이 값이 -1 이면 종료 상태이고 그렇지 않으면 종료 상태가 아니다. 종료 상태가 아니면 다음 입력을 받아들인다. 이러한 과정을 계속 반복하다가 마지막으로 c0가 들어왔을 때에 다음 상태가 종료 상태이기 때문에 "c0, c1, c0"는 accept된다.

입력이 "c0, c2"인 경우에는 처음 c0는 첫 번째 경우와 같고 c2가 들어오면 상태의 시작 위치에 입력의 오프셋을 더한 곳에 있는 ch가 c2가 아니므로 이 상태에서는 c2를 입력으로 하는 전이가 없음을 나타낸다. 따라서 "c0, c2"의 입력은 accept되지 않는다.

형태소 분석기와 같은 시스템에서 필요로 하는 한번의 탐색으로 가능한 모든 단어를 찾을 수 있는 기능은 다음 상태가 종료 상태이면 그 단어를 표제어가 될 수 있다고 표시하면서 계속해서 입력을 받아들이면서 상태를 전이해 가면 된다.

5. 실험 및 평가

FST를 이용한 전자 사전 구조의 성능 평가를 위해서 표제어 수에 따른 사전의 크기와 탐색 속도를 측정하였다. 전자 사전 크기 측면에서는 표제어 수의 증가에 따른 사전 크기의 변화를 관측하였고, 전자 사전 탐색 속도 측면에서는 표제어 수의 변화에 따른 전자 사전 탐색 시간을 측정하였다. 실험에 사용된 표제어는 약 105,000개의 형태소 사전에서 고르게 추출한 것이고, 탐색에 사용된 단어는 코퍼스에서 추출된 약 237만 단어이다. 실험은 SUN Sparc 10 Workstation에서 이루어졌다. 그림에 나타난 탐색 시간은 여러 번 측정한 다음 중간값을 취하였다. 실험 결과는 [그림 11]과 같다.

표제어가 2만 개일 때 DFA의 상태수는 93,766개 였는데, 상태수 최소화 알고리즘을 수행한 후의 상태수는 19,196으로 원래 상태수의 20% 수준으로 줄어들었다. 또한 표제어가 2만 개씩 늘어감에 따라 DFA 상태수의 증가 비율은 줄어들었다. 이에 따라 표제어 수의 증가에 따른 전자 사전 크기의 증가 비율 또한 조금씩 적어졌다. Compact TRIE의 경우 표제어 수가 약 11만개일

입력	현재 상태의 시작위치	입력의 offset	전이가 가능한가?	다음 상태는?	다음 상태가 종료 상태인가?
c0	0	0	FST[0].ch = c0	FST[0].Next = 2	FST[5] != -1
c1	2	1	FST[3].ch = c1	FST[3].Next = 1	FST[4] != -1
c0	1	0	FST[1].ch = c0	FST[1].Next = 3	FST[6] = -1
c0	0	0	FST[0].ch = c0	FST[0].Next = 2	FST[5] != -1
c2	2	2	FST[4].ch != c2		

[그림 10] 1차원 배열에서 FST의 전이

표제어 수	DFA의 상태수	상태수가 최소화된 DFA의 상태수 (증가수)	사전의 크기 (Bytes)	탐색 시간 (초)
2만	93,766	19,196 (19,196)	478,716	19.21
4만	152,686	29,206 (10,010)	792,948	19.28
6만	209,251	39,983 (10,777)	1,111,644	19.25
8만	267,592	49,340 (9,357)	1,419,672	19.94
10만	318,916	55,738 (6,398)	1,683,252	20.06

[그림 11] 표제어 수에 따른 FST 사전의 크기와 탐색 시간

때 노드 수가 148,000여개인 것에 비하면 FST의 상태수가 매우 적음을 알 수 있다[2].

전자 사전 탐색 시간은 표제어 수가 증가해도 일정했다. 그 이유는 전자 사전의 탐색이 FST 상태의 전이만으로 이루어져, 전자 사전 탐색 속도는 사전 표제어 수에 관계없이 단지 탐색키의 길이에만 영향을 받기 때문이다.

기존의 다른 사전 구조들과 비교하기 위해서 일반적으로 많이 사용되는 자소별 양방향 TRIE[4]와 B$^+$-Tree를 사용한 사전[3]을 사용하였다. 전자 사전에 입력된 표제어 수는 약 105,000개였고, 검색된 단어는 코퍼스에서 추출된 약 237만 단어였다. 실험 환경과 탐색 시간 측정 방법은 위와 동일하다.

사전 구조	사전의 크기 (Bytes)	탐색 시간 (초)	삽입 삭제
FST	1,759,044	20.69	불가능
양방향 TRIE	1,506,638	594.90	불가능
Compact TRIE	?	104.36	가 능 (비효율적)
B$^+$-tree	1,644,544	343.16	가 능

[그림 12] 각 사전 구조를 이용한 사전의 탐색 시간

실험 결과, FST를 이용한 전자 사전이 탐색 속도의 측면에서 다른 구조를 사용한 전자 사전 보다 탁월한 성능을 보였다. 양방향 TRIE의 탐색 속도가 B$^+$-Tree보다 더 늦게 나온 것은 주 기억 장치를 적게 차지하기 위하여 첫 글자를 위한 배열만을 주 기억 장치에 로드하고 나머지는 사전에 두기 때문이다. Compact TRIE의 탐색 시간은 [2]에 의하면 [4]의 양방향 TRIE보다 최고 5.7배가 빠르다고 한다. [2]에 의해서 Compact TRIE의 탐색 시간은 약 104.16초가 될 것이라고 가정하였다. 사전의 크기는 자소별 양방향 TRIE가 제안한 전자 사전보다 작았지만, 표제어수가 증가하면 FST가 더 작아질 것이다. 삽입과 삭제는 B$^+$-Tree를 사용한 사전이 가장 효율적이고 제안한 전자 사전은 불가능하였다.

6. 결 론

본 논문은 한국어 형태소 해석기와 같은 한국어 정보 처리 시스템에 적합한 전자 사전 구조로써 FST를 이용한 전자 사전 구조를 제안하였다. 제안한 전자 사전 구조는 다음과 같은 장점을 가진다. 첫째, 표제어를 표현하는데 DFA를 사용하고, DFA를 상태수 최소화 알고리즘으로 모든 위치의 중복된 상태들을 제거하여 적은 기억 공간에 나타낼 수 있다. 둘째, FST를 일차원 배열로 나타냄으로써 사전을 메모리로 읽어들일 때 별도의 연산이 필요하지 않으므로 사전 로드 시간이 적다. 셋째, 표제어를 탐색할 때 일어나는 상태 전이가 메모리를 직접 접근함으로써 이루어지기 때문에 전자 사전 탐색 속도가 매우 빠르다. 넷째, 한국어 특성을 고려한 형태소 해석기에서 어절 당 한번의 탐색으로 가능한 모든 표제어를 찾을 수 있어 사전 탐색 횟수를 줄일 수 있다.

실험 결과 표제어 수가 늘어나도 전자 사전의 크기의 변화는 점차 줄어들었으며, 사전 탐색 시간은 사전의 크기에 영향을 받지 않음을 알 수 있었다. 다른 사전 구조들과의 비교에서도 탐색 시간이 매우 효율적임을 알 수 있었다. 그러나 제안된 전자 사전은 한번 사전이 구축되고 나면, 표제어의 삽입과 삭제가 불가능하다는 단점을 가지고 있으므로 삽입 삭제가 요구되지 않고 빠른 탐색을 필요로 하는 응용분야에 적합하다.

참고문헌

[1] 김철수, 배우정, 이용석, "이중 트라이를 이용한 한국어 단어 검색", *제6회 한글 및 한국어정보처리 학술대회*, pp. 113-118, 1994

[2] 이승선, 송주원, 조완섭, 황규영, 최기선, "Compact TRIE Index(CompTI):한국어 전자 사전을 위한 데이타베이스 색인 구조", *한국정보과학회논문지* 제22권 제1호, pp. 3-12, 1995

[3] 이호, 김진동, 임해창, "한국어 용례 추출기를 위한 사전 및 정보 화일의 구조", *고려대학교 이학론집* 제35집, pp. 97-108, 1994

[4] 최기선 외 5인, "한국어 철자 및 띄어쓰기 교정 시스템에 관한 연구(II)", *최종 보고서*, 과학기술처, pp. 23-35, July 1992

[5] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, *Compilers Principles, Techniques, and Tools*, Addison-Wesley, 1988

[6] ALLEN I. HOLUB, *COMPILER DESIGN IN C*, Prentice-Hall International Editions, pp. 132-152, 1990

[7] Emmanuel Roche, "Finite-State Tools For Language Processing", *ACL'95 Tutorial*, 1995

[8] Emmanuel Roche, Yves Schabes, "Deterministic Part-of-Speech Tagging with Finite-State Transducers", *Computational Linguistics*, Volume 21, Number 2, pp. 227-253, June 1995

[9] Harry R. Lewis, Christos H. Papadimitriou, *ELEMENTS OF THE THEORY OF COMPUTATION*, PRENTICE-HALL SOFTWARE SERIES

[10] Robert Endre Tarjan, Andrew Chi-Chin Yao, "Storing a Sparse Table", *Communications of the ACM*, Volume 22 Number 11, pp. 606-611, 1979

[11] Yves Schabes, "Parsing Technologies:The Right Tools for The Right Problems", *Coling'94*, pp. 7-28, August, 1994