제31회 한국정보처리학회 춘계학술발표대회 논문집 제16권 제1호 (2009. 4)

일반화된 접미사 트리의 온라인 동반 생성 알고리즘

나중채*
*세종대학교 컴퓨터공학과
e-mail:jcna@sejong.ac.kr

An Algorithm for Constructing On-line and Concurrently the Generalized Suffix Tree

Joong Chae Na*
*Dept. of Computer Engineering, Sejong University

요 약

접미사 트리는 주어진 하나의 문자열의 모든 접미사를 표현하는 트리로, 문자열 처리, 압축 등 다양한 분야에서 활용된다. 접미사 트리는 문자열 집합에 대한 자료구조로 확장될 수 있는데, 이를 일반화된 접미사 트리라 부른다. 본 논문에서는 일반화된 접미사 트리를 동반적이면서 온라인으로 생성하는 문제를 다룬다. 기존의 생성 알고리즘은 정방향의 문자열이 아닌 역방향의 문자열들에 대한 일반화된 접미사 트리를 생성하여, 부자연스럽다. 본 논문에서는 정방향 문자열들의 일반화된 접미사 트리를 동반적이면서 온라인으로 생성하는 알고리즘을 제시한다.

1. 서론

접미사 트리(*suffix tree*)는 주어진 문자열(string)의 모든 접미사(suffix)를 표현하는 압축(compacted) trie로, 선형 시간에 생성될 수 있다 [1,2,3]. 접미사 트리는 문자열 알고리즘 분야에서 기본적인 자로 구조이면서, 텍스트 처리, 압축, 비전, 계산 분자 생물학 등 다양한 분야에 활용되고 있다.

*Generalized 접미사 트리*는 접미사 트리를 문자열 집합(set)으로 확장한 것으로, 집합 안에 있는 모든 문자열들의 모든 접미사들을 표현하는 압축 trie이다. 일반적으로 generalized 접미사 트리는 일반 접미사 트리 생성 알고리즘으로 생성될 수 있다. 지금부터는 혼동의 여지가 없는 경우 generalized 접미사 트리를 간단히 접미사 트리라 칭한다.

본 논문에서는 문자열 집합에 대한 접미사 트리를 동반적이면서(concurrently) 온라인(on-line)으로 생성하는 문제를 다룬다. 공식적인(formal) 문제 정의는 다음과 같다.

문제: 길이가 n인 m개의 문자열 집합 $A=\{S_1,\ldots,S_m\}$가 주어져 있다. 시각 p때 ($p=1,\ldots,n$), 각 문자열의 앞 p개의 문자들(즉, 길이가 p인 접두사)만 알려져 있다. 매 시각 p때, 길이가 p인 각 접두사들의 모든 접미사를 표현하는 접미사 트리를 생성하라.

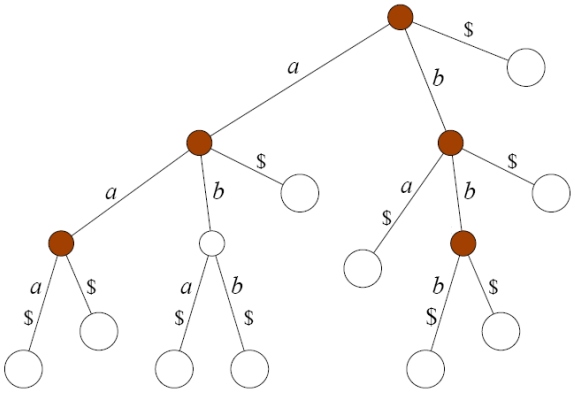
위 문제는 이차원 접미사 트리 생성 과정에서 부분 문제로 제시되었고, 문자열 하나에 대한 접미사 트리를 생성하는 Weiner의 알고리즘 [1]을 확장하여 위 문제를 해결하는 알고리즘이 개발되었다 [4]. 그러나 기존에 제시된 알고리즘은 정방향 문자열들의 접미사들을 표현하는 접미사 트리가 아닌, 역방향 문자열들의 접미사들을 표현하는 접미사 트리를 생성하여, 부자연스러운 해결책이다.

본 논문에서는 위 문제에 대해서 정방향 문자열들의 접미사 트리를 생성하는 알고리즘을 제시한다. 본 알고리즘은 하나의 문자열에 대한 접미사 트리 생성 알고리즘 중 하나인 [3]의 알고리즘을 확장한 것이다. 본 알고리즘은 기존 알고리즘과 동일하게 문자열 집합 A의 모든 문자열 길이의 합에 비례하는 시간에 수행되지만, 정방향 문자열의 접미사 트리를 생성한다는 장점을 가지고 있다. 접미사 트리가 활용되는 대부분의 응용(application)에서 정방향 접미사 트리가 사용된다는 점을 감안할 때, 본 논문에서 제시하는 알고리즘이 기존의 알고리즘보다 더 효용성이 좋을 것으로 기대된다.

본 논문의 구성은 다음과 같다. 2절에서 접미사 트리에 대한 배경 지식 및 기존 연구를 소개하고, 3절에서 새로운 알고리즘을 제시한 후, 4절에서 결론을 맺는다.

2. 접미사 트리

S를 길이가 n이고, 고정 알파벳 Σ에 대한 문자열이라 하자. S의 i번째 문자를 $S[i]$로, 부분 문자열 $S[i]\cdots S[j]$를 $S[i..j]$로 표기한다. 마지막 문자 $S[n]$은 Σ의 어떤 문자보다 사전 순으로 작은 특수 문자 \$로 가정한다.



(그림 1) 문자열 집합 $\{aaa\$, aba\$, abb\$, bbb\$\}$의 일반화된 접미사 트리

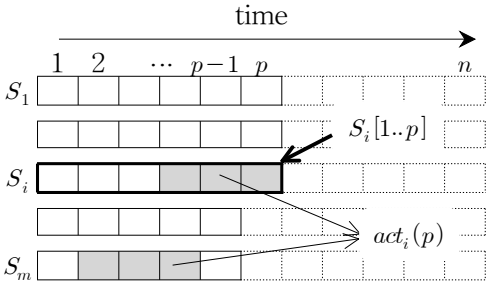
문자열 S의 접미사 트리 T는 S의 모든 접미사를 표현하는 trie이다. 루트 이외의 각 내부(internal) 노드는 두 개 이상의 자식을 가진다. 각 간선은 S의 부분 문자열을 나타내고, 이를 간선의 *레이블*(*label*)이라 칭한다. 한 노드에서 나가는(outgoing) 임의의 두 간선의 label은 같은 문자로 시작하지 않는다. 각 내부 노드는 접미사 링크(suffix link)를 가진다. 노드 u의 레이블을 $c\alpha$라 할 때, u의 접미사 링크는 레이블이 α인 노드 v를 가리킨다. 보다 자세한 접미사 트리에 대한 정의는 [2,3]을 참조하기 바란다.

일반화된 접미사 트리(generalized suffix tree) [5]는 문자열 집합에 속한 모든 문자열들의 모든 접미사들을 표현하는 trie로, 위의 문자열 하나에 대한 접미사 트리의 정의를 문자열들의 집합으로 확장한 것이다. 트리가 가지는 구조 및 성질은 동일하다. 그림 1은 일반화된 접미사 트리의 예를 보여준다.

접미사를 선형시간에 생성하는 알고리즘에는 [1,2,3]이 있다. (분할 정복 기법을 사용하는 알고리즘도 있지만, 본 논문과 무관하기 때문에 제외시킨다.) [1]의 알고리즘은 접미사를 짧은 것부터 긴 것 순으로 트리에 차례차례 삽입하는 방식이다. 따라서 맨 뒤의 문자부터 하나씩 앞으로 읽어나가면서 처리한다. [2]의 알고리즘은 긴 접미사부터 짧은 것 순으로 삽입한다. 하나의 접미사를 삽입할 때, 최악의 경우, 전체 문자열을 다 알아야 하므로, 진정한 의미의 온라인 알고리즘이라 볼 수는 없다. [3]의 알고리즘은 맨 앞 문자부터 차례로 입력될 때, 지금까지 입력된 문자열, 즉 접두사에 대한 접미사 트리를 생성해나간다. 본 논문에서 다루는 문제를 위해 [4]에서는 [1]의 알고리즘을 확장하였다.

3. 온라인 동반 생성 알고리즘

본 논문에서는 집합 A의 각 문자열 $S_q (1 \leq q \leq m)$가 처음 문자부터 한 문자씩 온라인으로 입력되는 상황을 고려한다. 즉, 시각이 1부터 n까지 변하고, 시각이 p일 때, 각 문자열 S_q의 p번째 문자 $S_q[p]$가 입력되고, 결과적으로 $S_q[1..p]$를 알고 있다. 하지만, p번째 문자 뒤의 문자들은 모르는 상태이다 (그림 2 참조).



(그림 2) n개의 문자열 및 문제 상황

본 알고리즘은 시각의 흐름에 따라 n개의 단계(stage)로 구성된다. 시각 p가 끝난 후에, m개의 문자열 $S_q[1..p]$들의 접미사 트리가 생성된다. 즉, 이 접미사 트리를 T_p로 표기할 때, 초기 시각 $p=1$의 시작 시점에는 오직 루트 노드만을 가지고 있는 T_0가 존재한다. 시각 p의 시작 시점에 T_{p-1}이 주어져있고, 알고리즘은 시각 p일때 트리 T_{p-1}에 있는 기존의 문자열 $S_q[1..p-1]$에서 한 문자씩 확장함으로써 T_p를 생성한다. 이 때 m개의 문자열 $S_1[1..p-1]$, $S_2[1..p-1]$, ..., $S_m[1..p-1]$을 차례로 처리한다. 시각 p에서 q번째 문자열까지 처리했을 때의 접미사 트리를 $T_{p,q}$라 표기하면, 알고리즘의 개략적인 의사 코드는 다음과 같다.

```
Construct_GST(A)
1: 초기에 루트를 생성하여 T_0를 만든다.
2: for p=1 to n do
3:     for q=1 to m do
4:         transform(T_{p,q-1});   // T_{p,q-1}을 T_{p,q}로 변형
```

지금부터는 트리 $T_{p,q-1}$을 $T_{p,q}$로 변형하기 위해, 시각 p에서 문자열 S_q를 처리하는 방법에 대해서 기술한다. 이 부분의 알고리즘은 하나의 문자열을 온라인으로 생성하는 Ukkonen의 알고리즘 [3]과 유사하다. 문자열 $S_q[1..p-1]$에 문자 $S_q[p]$가 더해져 $S_q[1..p]$가 될 때, $S_q[1..p-1]$의 각 접미사의 길이는 하나씩 길어지고, 길이가 1인 새로운 접미사 $S_q[p]$가 트리에 삽입된다. $S_q[1..p-1]$의 접미사를 α라 표기하고, 문자 $S_q[p]$를 c로 표기하자. 이 때 $S_q[1..p]$의 각 접미사는 αc로 표현되고, 정확히 다음 3 경우 중 하나에 해당한다.

(가) α가 $S_1[1..p],...,S_{q-1}[1..p],S_q[1..p-1],...,S_m[1..p-1]$에서 한번만 발생(occur)한다 (즉, 자기 자신). 이는 αc가 트리에 표현되지 않았다는 것을 의미하므로, α를 표현하는 단말노드를 αc를 표현하도록 바꿔줘야 한다.

(나) α가 $S_1[1..p],...,S_{q-1}[1..p],S_q[1..p-1],...,S_m[1..p-1]$에서 두 번 이상 발생하지만, αc는 그렇지 않다. 이는 트

리에 αa $(a \neq c)$가 표현되어 있음을 의미하여 α를 나타내는 내부노드가 없으면 생성하고, 이 내부 노드의 자식으로 αc를 표현하는 단말노드를 생성해야 한다.

(다) αc가 이미 $S_1[1..p],...,S_{q-1}[1..p],S_q[1..p-1],...,S_m[1..p-1]$에 존재한다. 이 경우 αc가 이미 트리에 표현되어 있으므로, 트리의 모양은 변하지 않는다.

접미사가 (가)에 속하는 경우, implicit update [3]라 불리는 방법을 이용하여 아무런 추가 작업 없이 (가)의 경우를 처리할 수 있다. 또한 (다)의 경우도 처리할 필요가 없으므로, (나)에 속하는 접미사들만 처리해주면 된다. (나)에 속하는 가장 긴 접미사를 x, 가장 짧은 접미사를 y라 했을 때, (나)에 속하는 모든 접미사들의 길이는 $|x|$보다 같거나 짧고, $|y|$보다 같거나 길다.

$T_{p,q-1}$을 $T_{p,q}$로 변형하기 위해서는 (나)에 속하는 접미사들만 처리해주면 되는데, 가장 긴 x부터 길이가 짧아지는 순서대로 처리하고 마지막에 y를 처리한다. 트리에서 x를 표현하는 위치를 active-위치, y를 표현하는 위치를 end-위치라 정의했을 때, 시각 p에서 문자열 S_q의 active-위치는 전 시각 $p-1$에서 문자열 S_q의 end위치로부터 쉽게 찾을 수 있다. 시각 p에서 문자열 S_q를 처리하여 $T_{p,q-1}$을 $T_{p,q}$로 변형하는 알고리즘의 개요는 다음과 같다.

transform($T_{p,q-1}$) // $T_{p,q}$생성 함수
1: 시각 $p-1$에서의 s_q의 end-위치로부터
   접미사 x의 위치 (시각 p의 active-위치) 찾기;
2: while x의 위치 $\neq$ 시각 p의 end-위치 do
3: 접미사 x를 처리;
4: $x \leftarrow$ 길이가 하나 짧은 접미사;//접미사 링크 이용
5: p의 end-위치를 다음 시각 $p+1$을 위해 저장

본 논문의 알고리즘과 하나의 문자열에 대한 Ukkonen의 알고리즘의 차이점은 다음과 같다. 문자열 S_q에 대한 처리를 생각해보자. Ukkonen의 알고리즘에서는 시각 $p-1$에서 $p-1$번째 문자를 처리한 후, 바로 다음 시각 p에 p번째 문자를 처리한다. 하지만 문자열 집합에 대한 본 알고리즘은 문자 $S_q[p-1]$을 처리하고 다음 문자 $S_q[p]$를 처리하는 사이에 다른 문자열들을 처리하게 되어 트리의 모양이 달라지므로 이러한 상황을 고려해야한다. 그러나 이러한 상황이 특별한 처리를 요구하지는 않는다.

본 알고리즘의 정확성과 시간 복잡도는 Ukkonen 알고리즘의 정확성과 시간 복잡도로부터 도출된다. Ukkonen 알고리즘은 하나의 문자열에 대한 접미사 트리를 선형 시간에 올바르게 생성한다. 본 알고리즘에서도 하나의 문자열만을 고려했을 때, Ukkonen의 알고리즘과 동일하고, 차이는 여러 개의 문자열들에 대한 알고리즘이 동시에 진행되면서 중간에 노드들이 생긴다는 점인데, 이 노드들은 Ukkonen 알고리즘의 정확성이나 시간 복잡도에 영향을 주지 않는다. 따라서 본 알고리즘은 선형시간에 접미사 트리를 올바르게 생성한다.

4. 결론

본 논문에서는 문자열들의 집합에 대한 일반화된 접미사 트리를 선형시간에 동반적이고 온라인으로 생성하는 알고리즘을 제안하였다. 본 논문에서 제시한 알고리즘은 하나의 문자열에 대한 접미사 트리를 온라인으로 생성하는 Ukkonen의 알고리즘을 여러개의 문자열을 처리할 수 있도록 확장한 것으로, 문자열 집합의 모든 문자열 길이의 합에 비례하는 시간에 수행된다. 본 알고리즘은 기존의 역방향 방식 생성 알고리즘과 달리 정방향 방식으로 생성하기 때문에 개념적으로 더 이해하기 쉽다는 장점이 있다.

참고문헌

[1] P. Weiner. Linear pattern matching algorithms. In Proceedings of the 14th IEEE Symposium on Switching and Automata Theory, pages 1-11, 1973.

[2] E. M. McCreight. A space-economical suffix tree construction algorithm. Journal of the ACM, 23(2): 262-272, 1976.

[3] E. Ukkonen. On-line construction of suffix trees. Algorithmica, 14(3): 249-260, 1995.

[4] J.C. Na, R. Giancarlo, and K. Park, On-line construction of two-dimensional suffix trees in $O(n^2 \log n)$ time, Algorithmica 48(2): 173-186, 2007.

[5] D. Gusfield, Algorithms on Strings, Trees, and Sequences, Cambridge Univ. Press, 1997.