제31회 한국정보처리학회 춘계학술발표대회 논문집 제16권 제1호 (2009. 4)

절단 접미사 트리를 생성하는 새로운 알고리즘

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

A New Algorithm for Constructing the Truncated Suffix Tree

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

요 약

절단 접미사 트리(truncated suffix tree)는 접미사 트리의 절단 버전으로, 주어진 문자열의 부분 문자열 중 일정 길이 이하인 것들만을 표현하는 자료구조이다. 절단 접미사 트리는 일정 길이 이하의 문자열들만을 고려하는 응용에 유용한데, 특히 LZ77 압축과 같이 온라인 생성 알고리즘이 필요한 응용들도 있다. 본 논문에서는 절단 접미사 트리를 온라인으로 생성하는 새로운 알고리즘을 제시한다.

1. 서론

접미사 트리(*suffix tree*)는 주어진 문자열(string)의 모든 접미사(suffix)를 표현하는 압축(compacted) trie [1,2,3]로 문자열 알고리즘 분야에서 기본적인 자로 구조이면서, 텍스트 처리, 압축, 비전, 계산 분자 생물학 등 다양한 분야에 활용된다. 이중에서 데이터 압축은 접미사 트리의 중요한 응용 분야로 특히 Ziv-Lempel 압축 기법 [4,5]과 같은 치환(substitution) 압축 기법에 유용하다.

접미사 트리를 Ziv-Lempel 압축 기법에 활용하기 위한 많은 연구들이 진행되었다. Rodeh, Pratt, Even [6]은 2개 혹은 3개의 접미사 트리를 이용하여 선형시간에 LZ77 [4] 압축을 구현하는 방법을 제시했고, Fiala와 Greene [7]는 접미사 트리를 생성하는 McCreight 알고리즘 [2]을 변형하여 역시 선형 시간에 LZ77을 구현하는 방법을 제시하였다. Larson [8]은 Fiala와 Greene의 결과가 LZ77에서 사용되는 이동창(sliding window)의 왼쪽 끝은 제어가 가능하지만, 오른쪽은 제어가 불가능하다는 점을 지적하고, Ukkonen 알고리즘 [3]을 사용하여 이 문제를 해결할 수 있음을 보였다. 위의 알고리즘들은 모두 접미사 트리를 활용하는데, 접미사 트리는 LZ77에서 필요한 기능보다 더 막강한 기능을 제공하지만 공간은 많이 사용한다. 이러한 공간 낭비를 막기 위해 절단 접미사 트리가 제안되었다.

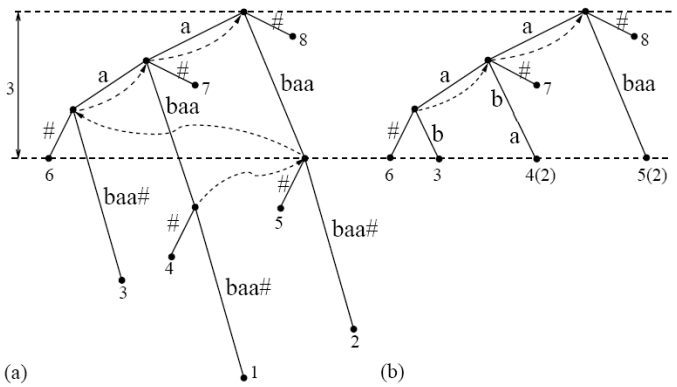
절단 접미사 트리(truncated suffix tree) [9]는 접미사 트리의 아래 부분을 잘라놓은 형태로, 접미사 트리가 주어진 문자열의 모든 부분 문자열을 저장하는데 반해, 절단 접미사 트리는 일정 길이 k 이하의 부분 문자열만을 저장하여 공간이 절약된다는 장점이 있다. 절단 접미사 트리를 생성하는 가장 간단한 방법은 접미사 트리를 생성한 후 길이가 k 이상인 부분을 잘라내는 것이지만, 이 방법은 절단 접미사의 공간 효율성에 대한 이점을 잘 활용하지 못한다. 그래서 접미사 트리를 중간에 거치지 않고, 절단 접미사 트리를 직접 생성하는 오프라인(off-line) 알고리즘과 온라인(on-line) 알고리즘이 제시되었고, 이 생성 알고리즘들을 LZ77에 활용하는 방법이 제시되었다 [9].

본 논문에서는 절단 접미사 트리를 생성하는 새로운 온라인 알고리즘을 제시한다. 기존에 제시된 온라인 알고리즘에서는 주어진 문자열의 부분 문자열을 나타내는 절단 접미사 트리의 노드가 알고리즘이 진행되면서 변경될 수 있다는 성질이 있다. 이 성질은 접미사 트리가 활용되는 응용에 따라서 문제가 되기도 한다. 본 논문에서 제시하는 알고리즘에서는 부분 문자열을 나타내는 단말노드가 한번 정해지면, 알고리즘이 진행되면서 바뀌지 않고 고정된다. 본 알고리즘은 접미사 트리를 생성하는 Weiner [1]의 알고리즘과 유사하다. Weiner의 알고리즘에서는 문자열이 맨 뒤에서부터 입력되는 온라인 상황을 가정하는데, 본 논문에서도 이는 똑같이 적용된다.

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

2. 절단 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. 위치 i에서 시작하는 S의 접미사는 $S(i)$로 표기한다. 양의 상수



(그림 1) (a) $abaabaa\#$의 접미사 트리와
(b) 절단 접미사 트리(k=3) [9]

k에 대해서, k-*인자*(factor)는 길이가 k인 S의 부분 문자열 또는 길이가 k보다 작은 S의 접미사로 정의한다. 위치 i에서 시작하는 k-인자를 $k\text{-}fac_i$로 표시할 때, $i \leqq n-k$이면 $k\text{-}fac_i$는 $S[i..i+k-1]$이고, 그렇지 않으면 $S[i..n]$이다.

문자열 S의 절단 접미사 트리는 S의 모든 k-인자를 표현하는 경로 압축(path-compressed) trie이다. 절단 접미사 트리의 각 간선은 S의 부분 문자열을 나타내고, 이를 간선의 *레이블*(*label*)이라 칭한다. 루트 이외의 각 내부(internal) 노드는 두 개 이상의 자식을 가진고, 한 노드에서 나가는(outgoing) 임의의 두 간선의 label은 같은 문자로 시작하지 않는다. 각 단말노드 w에 대해서, w는 반드시 S의 k-인자중 하나를 나타내고, 반대로 S의 한 k-인자를 나타내는 단말노드가 반드시 존재한다. 접미사와 단말노드 사이에 일대일 대응이 존재하는 접미사 트리와 달리 절단 접미사 트리에서는 하나의 단말노드가 여러 k-인자를 표현할 수도 있다. 전체 단말노드의 개수는 많아야 n이므로 전체 노드의 개수는 $2n$보다 작다. 그림 1은 문자열 $abaabaa\#$의 접미사 트리와 절단 접미사 트리를 보여준다. 각 단말 노드의 번호는 문자열 S에서 접미사와 k-인자가 시작하는 위치를 나타낸다.

절단 접미사 트리를 선형시간에 생성하는 기존의 알고리즘을 소개한다[9]. 오프라인(off-line)과 온라인(on-line), 두 개의 알고리즘이 개발되어있다. 오프라인 알고리즘은 시작 위치가 빠른 순으로 k-인자들을 삽입하는 방식으로 접미사 트리를 생성하는 McCreight의 알고리즘 [2]과 유사하다. 온라인 알고리즘은 문자열 S의 각 문자가 처음부터 하나씩 입력된다고 가정했을 때, 현재까지 입력된 문자열에 대한 절단 접미사 트리를 생성하는 방식으로 접미사 트리를 생성하는 Ukkonen의 알고리즘 [3]과 유사하다. 온라인 알고리즘에 의해 생성된 트리는 현재까지 입력된 문자열의 k-인자들을 표현하는데, 이후에 입력되는 문자들을 처리하는 과정에서 기존의 k-인자들을 표현하는 노드의 위치가 변경될 수 있다는 문제점이 있다. 즉, S의 j번째 문자까지 입력되었을 때, $k\text{-}fac_i$를 나타내는 단말 노드가 w라고 했을 때, 다음 문자 $j+1$번째 문자가 입력된 후에, $k\text{-}fac_i$를 나타내는 단말 노드가 w'으로 바뀔 수 있다. 이는 온라인 상황에서 k-인자를 나타내는 단말 노드가 고정되어야 하는 응용에서 문제가 될 수 있다.

3. 생성 알고리즘

본 절에서는 절단 접미사 트리를 생성하는 새로운 온라인 알고리즘을 제시한다. 본 논문에서는 문자열 S가 맨 끝 문자부터 온라인으로 입력되는 상황을 고려한다. 즉, 본 논문에서 제시하는 알고리즘이 정방향 온라인 알고리즘이 아니고, 역방향 온라인 알고리즘이지만, 접미사 트리가 활용되는 많은 응용에서 이는 큰 문제가 되지 않는다.

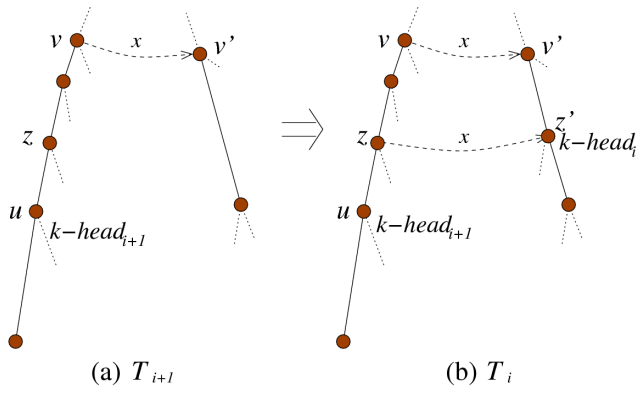
본 알고리즘에서는 각 노드에 크기가 Σ인 링크 벡터(link vector)를 저장한다. 노드 v에서 문자 c에 의한 링크 벡터를 $L(v,c)$라 표기하자. 노드 v에 의해 표현되는 문자열을 α라 했을 때, $c\alpha$를 표현하는 노드 u가 존재하면, $L(v,c)$는 u를 가리키고, 그렇지 않으면 null 값을 가진다. 다음으로 $k\text{-}head_i$를 정의한다. $k\text{-}head_i$는 $k\text{-}fac_i$의 접두사인 동시에 다른 위치 $j(j>i)$에서 시작하는 k-인자의 접두사인 가장 긴 문자열이다. $k\text{-}head_i$를 표현하는 노드는 정의에 의해서 반드시 존재한다.

알고리즘은 총 n 단계로 구성되고, 역순으로 진행된다. 즉, 단계 n부터 시작해 단계가 감소하여 마지막에 단계 1이 수행된다. 처음 단계 n의 시작 시 트리에는 오직 루트 노드만 존재한다. (이 트리를 T_{n+1}라 표기하자.) 다음 맨 끝의 $k\text{-}fac_n$부터 $k\text{-}fac_{n-1}$, ..., 순으로 차례로 트리에 삽입되고, 마지막에 $k\text{-}fac_1$이 삽입된다. 단계 i에서 $k\text{-}fac_i$까지 삽입된 트리를 T_i라 표기하면, 트리는 T_{n+1}, T_n, T_{n-1}, 순으로 생성되어 최종적으로 S의 절단 접미사 트리인 T_1이 생성된다.

다음으로 단계 i에서 트리 T_{i+1}에 $k\text{-}fac_i$를 삽입하여 T_i를 생성하는 과정을 설명한다. (그림 2 참조)

1. 트리 T_{i+1}에서 경로 레이블이 $k\text{-}head_i$인 위치를 찾는다. 이 위치는 단계 $i+1$에서 $k\text{-}fac_{i+1}$를 삽입할 때의 정보를 이용하여 찾을 수 있다. T_{i+1}에서 $k\text{-}head_{i+1}$를 나타내는 노드를 u라 하고, $S[i]$를 x라 놓자. 노드 u에서부터 루트로 올라가면서, 문자 x에 의한 링크 벡터가 null이 아닌 u에서 가장 가까운 노드 v를 찾는다. 링크 $L(v,x)$의 값을 v'이라 하자. 노드 v'에서부터 아래로 내려오면서 $k\text{-}head_i$의 위치를 찾는다.
2. $k\text{-}head_i$의 위치에 $k\text{-}head_i$를 나타내는 노드 z'을 생성하고, 경로(v,u)상에서 $k\text{-}head_i$에서 맨 앞 문자 x를 뺀 문자열을 나타내는 노드 z을 찾은 후, $L(z,x)=z'$으로 설정한다.

본 논문의 알고리즘은 접미사 트리를 생성하는 Weiner의 알고리즘 [1]과 유사한데, 차이점은 다음과 같다. 접미사 트리에서는 2번째 과정에서 $k\text{-}head_i$의 위치를 찾기 위



(그림 2) 단계 i에서의 알고리즘

해서 문자를 비교할 때 불일치(mismatch)가 일어날 때까지 반복되지만, 절단 접미사 트리에서는 길이가 k로 제한되기 때문에, 단말 노드에 도달하면 더 이상 비교하지 않고 끝난다. 이 경우 $k-head_i$를 나타내는 노드는 마지막에 도달한 단말노드가 된다.

본 알고리즘의 정확성과 시간 복잡도는 Weiner 알고리즘의 정확성과 시간 복잡도로부터 도출된다. Weiner 알고리즘은 접미사 트리를 선형시간에 올바르게 생성한다. 본 알고리즘은 비교의 길이가 k로 제한된다는 점을 제외하고는 Weiner의 알고리즘과 동일한데, 이는 Weiner 알고리즘의 정확성과 시간 복잡도에 영향을 주지 않는다. 따라서 본 알고리즘은 선형시간에 절단 접미사 트리를 올바르게 생성한다.

4. 결론

본 논문에서는 주어진 문자열의 절단 접미사 트리를 생성하는 새로운 온라인 알고리즘을 제시하였다. 기존의 온라인 알고리즘에서는 트리의 노드가 나타내는 문자열이 알고리즘이 진행되면서 변경될 수 있다. 본 논문에서는 접미사 트리를 생성하는 Weiner의 알고리즘을 변형하여, 트리의 노드가 타나내는 문자열이 고정되도록 하였다. 하지만, 문자열을 역방향으로 처리하는 Weiner의 알고리즘의 단점도 똑같이 가지고 있다. 문자열을 정방향으로 처리하면서도 k-인자들을 표현하는 노드들이 고정되는 알고리즘 개발은 향후 연구 과제로 제안한다.

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. Ziv, A. Lempel, A universal algorithm for sequential data compression, IEEE Trans. Inform. Theory IEEE 23 (3): 337-343, 1977.

[5] J. Ziv, A. Lempel, Compression of individual sequences via variable-rate coding, IEEE Trans. Inform. Theory, IEEE 24 (5): 530-536, 1978.

[6] M. Rodeh, V.R. Pratt, S. Even, Linear algorithm for data compression via string matching, J. ACM, 28 (1): 16-24, 1981.

[7] E.R. Fiala, D.H. Greene, Data compression with 7nite windows, Comm. ACM 32 (4): 490-505, 1989.

[8] N.J. Larsson, Extended application of suffix trees to data compression, Proc. Data Compression Conf., pp. 190-199, 1996.

[9] J.C. Na, A. Apostolico, C.S. Iliopoulos, and K. Park, Truncated suffix trees and their application to data compression, Theoretical Computer Science 304/1-3, 87-101, 2003.