접미사배열을 이용한 최장공통비상위문자열 찾기

조석현[O,1], 윤현철[1], 나중채[2], 심정섭[1,*]
[1]인하대학교 컴퓨터정보공학부, [2]세종대학교 컴퓨터공학과
sowoozzo@naver.com, hchyoon@gmail.com, jcna@sejong.ac.kr, jssim@inha.ac.kr

Finding the Longest Common Non-superstrings Using Suffix Arrays

Suk Hyeun Cho[O,1], Hyun Chul Yoon[1], Joong Chae Na[2], Jeong Seop Sim[1]
[1]School of Computer and Information Engineering, Inha University,
[2]Department of Computer Science and Engineering, Sejong University

문자열 집합 F가 주어질 때, 주어진 문자열 집합 F 내의 어떤 문자열도 포함하지 않는 문자열을 F에 대한 공통비상위문자열이라 하고 공통비상위문자열 중에서 가장 긴 유한길이의 문자열을 최장공통비상위문자열이라 한다. 접미사 그래프 모델 또는 접두사 그래프 모델을 이용하여 F 내의 모든 문자열들의 길이의 합에 대해 선형시간에 최장공통비상위문자열을 찾는 알고리즘들이 최근 제시되었다. 이 중 접미사 그래프 모델을 이용하는 알고리즘은 접미사트리를 이용한다. 본 논문에서는 접미사배열을 이용하여 접미사 그래프 모델을 생성함으로써 최장공통비상위문자열을 찾는 새로운 알고리즘을 제시한다.

상수 크기 문자 집합에 대한 문자열 집합 $F = \{f_1, f_2, \ldots, f_m\}$와 문자열 x를 고려해 보자. 만약 x가 모든 $f_i (1 \le i \le m)$에 대해서 상위문자열(superstring)이 아니라면, x를 F에 대한 공통비상위문자열(common nonsuperstring, 이하 CNSS로 표기)이라 한다. 만약 x가 F에 대한 공통비상위문자열 중 가장 긴 문자열이라면 x를 F에 대한 최장공통비상위문자열(LCNSS)이라 한다. LCNSS를 찾는 문제는 [1]에서 처음 소개되었다. S를 F에 속한 문자열들의 진접미사(proper suffix) 집합이라 하자. Rubinov와 Timkovsky[2]는 그래프 상의 경로가 F의 CNSS와 대응되는 S에 대한 유향 그래프 $\Gamma_S = (V, E)$를 정의했다. $\Gamma_S = (V, E)$는 다음과 같이 정의 된다. 정점 집합 V는 각 $s_i \in S (1 \le i \le n)$에 대해, v_i가 정의되고 그 외의 정점은 V 내에 존재하지 않는다. 그러면 V와 S 사이에 일대일 대응관계가 성립한다. 간선 집합 E는 $\sigma s_i (s_i \in S, \sigma \in \Sigma)$인 문자열에 대한 집합 $P(\sigma s_i) = \{x \mid x \in S \cup F, x$는 σs_i의 접두사$\}$를 정의한다. 그리고 $P(\sigma s_i)$에 속한 문자열 중 최장 문자열이 $s_j \in S$일 경우 v_j(s_j에 대응)로부터 v_i(s_i에 대응)로의 간선을 연결 해준다. 이때, 간선의 레이블은 σ이다. Γ_S가 사이클이 존재할 경우, F에 대한 LCNSS는 존재하지 않는다. 그렇지 않다면, 즉, Γ_S가 사이클이 없는 유향 그래프일 경우, Γ_S의 최장 경로는 F에 대한 LCNSS를 나타내며, $O(|V| + |E|)$ 시간에 찾을 수 있다. 따라서 LCNSS 문제를 효율적으로 해결하기 위해서는 Γ_S를 효율적으로 생성해야 한다. [3]에서는 접미사트리를 이용하여 Γ_S를 선형시간에 생성하는 알고리즘을 제시하였다. 반면 접미사 그래프 모델 Γ_S와 달리 접두사 그래프 모델을 이용하여 LCNSS를 찾는 선형시간 알고리즘이 [4]에 의해 제시되었다.

본 논문에서는 접미사배열을 이용하여 Γ_S를 생성하는 알고리즘을 제시한다. 제시된 알고리즘은 문자열 집합 F에 속한 모든 문자열의 길이의 합을 $\|F\|$라 했을 때, 상수 크기 문자 집합 Σ에 대해, $O(\|F\|)$ 시간이 걸린다. 본 논문에 제시된 알고리즘은 Γ_S를 생성하기 위해 일반화접미사배열을 이용한다는 점이 [3]에서 제시된 알고리즘과의 가장 큰 차이점이다. 따라서 여기에서는 Γ_S를 생성하는 새로운 알고리즘에 대해 설명하기로 한다.

본 논문에서 제시되는 Γ_S 생성 알고리즘은 총 세 단계로 구성된다. 첫 번째 단계에서는 skew 알고리즘[5]을 이용하여 일반화접미사배열(GSA)을 생성한다. 두 번째 단계에서는 Γ_S의 정점 집합 V를 생성한다. 세 번째 단계에서는 Γ_S의 간선 집합 E를 생성하기 위해 필요한 최근접접두사(nearest prefix)와 link함수를 계산하고, Γ_S의 간선을 연결시켜 Γ_S를 완성한다.

GSA는 문자열 집합이 주어졌을 때, 각 문자열의 접미사들을 사전순으로 정렬한 배열이며 접미사들의 위치를 저장한다. GSA는 Γ_S를 생성하는데 기반이 되는 정보를 제공하며, GSA를 생성하기 위해 F에 속한 문자열 끝에 종결문자를 붙인 F′ 을 이용한다. 종결 문자 $\$_i (1 \le i \le m)$는 사전순으로 Σ내의 어떤 문자보다 빠르다고 가정한다. 또한 서로 다른 임의의 두 종결문자 $\$_i$, $\$_j$에 대해서 $i < j$일 경우 $\$_i$가 $\$_j$보다 빠르다고 가정한다. $F = \{f_1, f_2, \ldots, f_m\}$일 때, $F' = \{f_1\$_1, f_2\$_2, \ldots, f_m\$_m\}$ 이라 하자. 이 때 F′ 에 대한 GSA는 두 정수의 쌍 <i, j>를 저장하며, 이는 F′의

* 이 논문은 2009년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임 (2010-0028134).

문자열 f'_i의 접미사 $f'_i[j..|f'_i|]$를 나타낸다. 접미사배열은 문자열의 길이에 대해 선형시간에 구할 수 있다[5-7]. 본 논문에서는 그 중 [5]에 제시된 skew 알고리즘을 이용하여 F′에 대한 GSA를 O(‖F‖) 시간에 구하며, GSA에 대한 최장공통접두사(longest common prefix, 이하 LCP로 표기) 배열을 [8]에 제시된 알고리즘을 이용하여 O(‖F‖) 시간에 구한다.

앞으로 특별한 언급이 없다면, GSA의 원소가 나타내는 접미사는 종결문자를 제외한 접미사를 의미한다. 또한 GSA 상의 접미사라 함은 GSA의 각 원소가 나타내는 접미사를 의미하며, 접미사 α의 GSA 상의 인덱스라 함은 GSA[k]가 나타내는 접미사가 α일 때 이 때의 인덱스 k를 의미한다. 대표접미사 $R(\alpha)$는 α의 GSA 상의 인덱스 중 가장 작은 인덱스 값을 가지며, GSA 상에서 접미사 α를 대표하는 GSA의 원소를 의미한다.

Γ_S의 정점 v_i는 $R(\alpha)$이면서 $\alpha \notin F$인 GSA의 원소에 대해서 생성한다. 한편 GSA 상에서 인접한 두 문자열이 동일한 문자열일 경우 두 문자열간의 길이가 같고, 두 문자열간의 LCP 값이 두 문자열의 길이와 같다는 사실을 알 수 있다. 그리고 이를 통해 GSA의 원소가 $R(\alpha)$인지를 확인 할 수 있다. 또한 α가 F에 속할 경우 GSA[$R(\alpha)$] 값은 항상 <i, 1>이므로 상수시간에 F에 대한 포함 여부를 확인할 수 있다.

세 번째 단계에서 사용될 기호에 대해서 알아보자. Len 배열은 접미사의 길이를 나타내는 배열로서, Len[k]는 GSA[k]가 나타내는 접미사의 길이를 저장한다. $str(v_i)$는 $v_i(\in V)$에 대응되는 문자열 $s_i(\in S)$를 나타낸다. 또한 $ver(s_i)$는 s_i에 대응되는 정점 v_i를 나타낸다. $link(v_i, \sigma)$는 다음과 같이 정의된다. $str(v_i)$에 대한 $P(\sigma s_i)$의 최장 문자열이 s_j일 경우에는 $ver(s_j)$를 반환하고, $f_j(\in F)$일 경우에는 null을 반환한다.최근접접두사 배열 npx는 link함수를 구하기 위해 사용되며, npx[k]는 GSA[k]가 나타내는 접미사 β의 진접두사 중 GSA 상에 존재하는 가장 긴 접두사 α의 $R(\alpha)$ 값으로 정의된다. 또한 α는 $S \cup F$에 속한 문자열 중 β의 최장진접두사(longest proper prefix)이기도 하다.

npx[k]는 스택을 이용하여 구할 수 있다. 스택에 GSA의 $R(\alpha)$인 원소에 대해, 두번째 원소부터 차례로 삽입한다. 이 때 스택에 새로 삽입하려는 (GSA의 원소가 나타내는) 접미사 β의 LCP 값 스택의 top이 나타내는 접미사 α의 길이보다 크거나 같다면, β의 최근접접미사는 α가 되고, npk[k]는 $R(\alpha)$값으로 정의된다. α의 길이가 β의 LCP 값보다 크다면, 스택의 top이 가리키는 원소는 제거되고 다음 top이 나타내는 접미사에 대해 다시 최근접접두사 여부를 판단한다. 스택은 초기값으로 GSA[1]을 가진다.

$link(v_i, \sigma)$ 함수를 계산하는 단계는 두 가지 경우로 나누어 생각할 수 있다. 우선 임의의 σ에 대해 $\sigma\, str(v_i)$가 GSA 상에 존재할 경우 link 함수의 정의에 따라 $link(v_i, \sigma)$는 $\sigma\, str(v_i) \in F$일 경우 null 값을 $\sigma\, str(v_i) \in S$일 경우 $ver(\sigma\, str(v_i))$ 값을 가진다. 두 번째 경우는 $\sigma\, str(v_i)$가 GSA 상에 존재 하지 않을 경우이다. $str(v_i)$의 대표접미사 $R(str(v_i))$의 npx값을 k (= npx[$R(str(v_i))$])라 하고, GSA[k]가 나타내는 접미사를 α라 하자. 이 때 $link(v_i, \sigma)$는 $ver(\alpha)$의 $link(ver(\alpha), \sigma)$함수 값을 가져온다.

link함수를 계산 한 후, Γ_S 그래프 내의 모든 정점 v_i에 대해 $link(v_i, \sigma) = v_j$일 때, v_j로부터 v_i로 향하는 간선을 연결시켜 Γ_S를 완성한다.

참 고 문 헌

[1] V. G. Timkovsky, Complexity of common subsequence and supersequence problems and related problems, *Cybernetics and Systems Analysis* 25, 5, 565-580, 1990.

[2] A. R. Rubinov, and V. G. Timkovsky, String noninclusion optimization problems, *SIAM Journal on Discrete Mathematics* 11, 3, 456-467, 1998.

[3] J.C. Na, D.K. Kim, J.S. Sim, Finding the longest common nonsuperstring in linear time, *Information Processing Letters*, 1066-1070, 2009

[4] 최시원, 이도경, 김동규, 나중채, 심정섭, “최장공통비상위문자열을 찾는 새로운 알고리즘”, 정보과학회논문지, 67-71, 2009.

[5] J. Karkkainen, and P. Sanders, Simple linear work suffix array construction, *In Proc. 30th International Colloquium on Automata, Languages and Programming*, 943-955, 2003.

[6] D. K. Kim, J. S. Sim, H. Park, and K. Park. Linear-time construction of suffix arrays. *Combinatorial Pattern Matching*. 186-199, 2003.

[7] P. Ko and S. Aluru. Space efficient linear time construction of suffix arrays. *Combinatorial Pattern Matching*. 203-210, 2003

[8] T. Kasai, G. H. Lee, H. Arimura, S. Arikawa, and K. S. Park, Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications, *Combinatorial Pattern Matching*, 181-192, 2001