효율적인 써픽스 배열 합병 알고리즘과 응용[1)]

전정은[0] 박희진[1] 김동규[2]
[0, 2] 부산대학교 컴퓨터공학과
jejun@islab.ce.pusan.ac.kr[0], dkkim1@pusan.ac.kr[2]
[1] 한양대학교 정보통신학부
hjpark@hanyang.ac.kr

Efficient Merging Algorithms for Suffix Arrays and their Application

Jeong Eun Jeon[0], Heejin Park[1], Dong Kyue Kim[2]
[0, 2] School of Electrical and Computer Engineering, Pusan National University
[1] College of Information and Communications, Hanyang University

요 약

대표적인 인덱스 자료 구조인 써픽스 트리와 써픽스 배열은 긴 문자열에서 임의의 패턴을 검색하는 데 효율적이다. 써픽스 트리는 써픽스 배열보다 큰 공간을 차지하지만, 이미 구축된 써픽스 트리의 정보를 이용하여 쉽게 합병할 수 있다. 본 논문에서는 문자열 A와 B에 대한 써픽스 배열이 구축되어 있을 때, $A\#B\$$의 일반화된 써픽스 배열을 구축하기 위한 합병 알고리즘을 두 가지 제시하였다. 이 알고리즘을 사용하면 기존의 유전체 서열 써픽스 배열을 재사용하는 방식으로 합병하여, 빠른 시간 안에 효율적으로 합병된 써픽스 배열을 만들 수 있다. 실험 결과, 합병 알고리즘은 일반화된 써픽스 배열을 다시 구축하는 것보다 5배 정도 빠른 속도를 보였다.

1. 서론

1.1. 연구배경

써픽스 트리(suffix tree)와 써픽스 배열(suffix array)은 대표적인 인덱스 자료 구조이다. 써픽스 트리는 문자열의 모든 접미사를 압축된 트라이(trie) 형태로 나타낸 것을 말하며, McCreight[1]가 처음 소개하였다. 써픽스 배열은 문자열의 모든 접미사(suffix)를 알파벳순으로 정렬하여 그 시작위치를 배열형태로 나타낸 것이다[2]. 써픽스 배열은 써픽스 트리에 비해 공간을 적게 차지하므로, 길이가 긴 문자열의 인덱스 자료 구조에 적합하다는 장점이 있다.

생물정보학(bioinformatics)에 사용되는 유전체 서열은 1차원적 구조를 가지므로 문자열로 볼 수 있다. 특정 유전체 서열에 등장하는 패턴을 찾기 위해서 인덱스 자료 구조가 사용되며, 유전체 서열의 크기가 방대하므로 써픽스 배열을 사용하는 것이 효율적이다.

두 개의 유전체 서열 A와 B가 있을 때, 구분 문자 #과 $를 추가하여 $A\#B\$$라는 서열을 생성하여 그 상관관계를 파악할 수 있다. A와 B에 대한 써픽스 트리가 이미 존재한다면, $A\#B\$$에 대한 일반화된 써픽스 트리를 다시 구축하지 않고 두 개의 써픽스 트리를 직접 합병할 수 있어 효율적이지만, 써픽스 배열에서는 트리와 같은 합병 알고리즘이 아직 연구되지 않았다.

1.2. 연구결과

본 논문에서는 상수 알파벳에서 정의된 문자열 A와 B에 대한 써픽스 배열이 이미 구축되어 있을 때, $A\#B\$$의 써픽스 배열을 다시 구축하지 않고 직접 합병하는 선형시간 알고리즘을 두 가지 제시하였다.

실험 결과, 두 써픽스 배열을 합병하는 것은 $A\#B\$$의 써픽스 배열을 다시 구축하는 것에 비해 5배 정도 빠른 속도를 보였다. 이 알고리즘을 사용하면 큰 용량의 유전체 서열의 상관관계를 파악하기 위해 새로운 써픽스 배열을 구축할 필요 없이 두 써픽스 배열을 합병하여 효율적으로 처리할 수 있게 된다.

2. 개요

2.1 기본 정의 및 표기법

상수 알파벳 Σ 상에서 정의된 길이 n인 문자열 S가 주어져 있다고 하자. #과 $는 모든 알파벳보다 큰 문자이며, Σ 에는 나타나지 않는다고 가정한다(#은 $보다 알파벳 순서상 크다). $S[i](1 \le i \le n)$는 문자열 S의 i번째 문자를 가리킨다. 써픽스 배열은 pos와 lcp로 표현한다. $pos[i]$는 사전순서상 i번째인 접미사의 시작위치이며, $lcp[i]$는 $pos[i-1]$과 $pos[i]$의 최장 공통 접두사(longest common prefix)의 길이를 나타낸다.

Aboulehoda et al.[5]은 써픽스 배열을 써픽스 트리처럼 탑다운(top-down) 방식으로 순회할 수 있도록 수정한, 향상된 써픽스 배열(enhanced suffix array)을 제시하였다. 이 정보는 *up*, *down*, *nextlIndex*의 세 가지 필드로 이루어지지만, 실제 하나의 필드에 저장할 수 있다. [그림 1]의 *cldtab*은 향상된 써픽스 배열의 예이다.

2.2 핵심 아이디어

두 개의 써픽스 배열이 주어졌을 때, 그 중 하나를 기준으로

1) 이 논문은 2003년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2003-003-D00343).

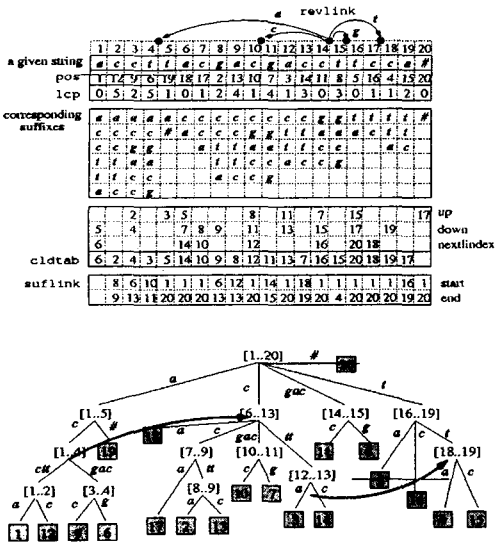

그림 1. 써픽스 배열과 써픽스 트리의 예

다른 문자열의 접미사들을 검색하여 상대적인 위치를 알 수 있다면, 두 개의 써픽스 배열을 합병할 수 있다. 써픽스 트리를 구축하는 방법 중 대표적인의 알고리즘들[1, 3, 4] 역시 기존 트리에서 검색을 통해 새로운 접미사를 삽입하는 방법을 사용하였다. 인덱스 자료 구조 상의 검색 방법은 전위 검색(forward search)와 후위검색(backward search)으로 구분할 수 있으며, 본 논문에서는 이 두 가지 검색 방법을 사용한 써픽스 배열 합병 알고리즘을 제시하고자 한다.

합병 알고리즘은 두 단계로 나누어 볼 수 있다. 첫째는 한 문자열의 써픽스 배열 상에서 다른 문자열의 접미사들을 검색하여 그 순서를 결정하는 것이며, 둘째는 이 순서를 이용하여 합병된 써픽스 배열을 계산하는 것이다. 첫 번째 단계는 검색 방법에 따라 두 가지 알고리즘으로 제시되며, 두 번째 단계는 첫 번째 단계에 의해 계산된 값을 참고로 같은 알고리즘을 사용한다.

3. 합병 알고리즘

3.1. 전위 검색(forward search) 방법

McCreight는 문자열의 접미사들을 긴 것에서부터 짧은 것의 순서로 써픽스 트리에 삽입하는 방법을 제시하였다[1]. 이 방법은 보다 긴 접미사가 트리에 삽입되어 있는 상태에서, 보다 짧은 접미사와 가장 일치하는 부분을 트리에서 찾아 그 위치에 짧은 접미사를 삽입하여 트리를 완성해나가는데, 앞에서부터 하나씩 비교한다는 점에서 전위 검색 방법이라 볼 수 있다.

전위 검색 방법을 사용하기 위해 써픽스 링크가 필요하였는데, 본 논문에서는 써픽스 배열 상에서 써픽스 링크를 정의하여 이와 같은 전위 검색 방법을 사용할 수 있도록 하였다.

써픽스 링크를 계산하기 위해서는 향상된 써픽스 배열 상에서 전위 깊이 우선 순회(preorder depth-first traverse)를 통해 최대로 일치하는 접두사를 찾아야 한다. 우선 뿌리 구간 $[1..n]$의 첫 번째 자식 구간부터 써픽스 링크를 찾는다. 전위 순회 방식이므로 부모 구간의 써픽스 링크는 이미 계산된 상태이다. 부모 구간의 접두사가 $a\alpha$이고 자식 구간의 접두사가 $a\alpha\beta$라고 하자. 부모 구간의 써픽스 링크의 접두사는 α이므로, 여기서부터 시작하여 β를 검색한다. Σ가 상수일 경우, 모든 구간에 대해 써픽스 링크를 계산하는 것은 $O(n)$시간이 걸린다.

이제 검색을 통한 순서 결정 알고리즘을 살펴보자.

길이 n인 문자열 A와 길이 m인 문자열 B에 대한 써픽스 배열이 이미 구축되어 있다고 가정하고, A의 써픽스 배열을 pos_A, B의 써픽스 배열을 pos_B라 하자. pos_A상에서 B의 접미사들과 가장 일치하는 부분을 검색하여 그 위치에 삽입하면 두 써픽스 배열을 합병할 수 있다. 이를 위해 $C[1..n]$라는 배열을 선언하여 A의 써픽스 배열 상에서의 B의 접미사들의 위치를 저장할 것이다. 순서는 아래와 같다.

- $C[1..n]$를 0으로 초기화한다.
- A의 써픽스 배열 pos_A에 부모-자식 테이블을 추가하여, $O(n)$ 시간에 향상된 써픽스 배열을 생성한다.
- $O(n)$시간에 pos_A에서 써픽스 링크를 계산한다.
- $B[1..m]$부터 $B[m..m]$까지 B의 접미사들을 길이가 긴 것에서부터 짧은 것 순서대로, 전위 검색을 사용하여 pos_A상의 위치를 검색한다. $B[i..m]$이 $pos_A[j-1]$과 $pos_A[j]$ 사이에 있다면, $C[j]$를 1 증가시킨다. m개를 모두 검색하는 데는 $O(m|\Sigma|)$시간이 소요된다.

합병된 써픽스 배열은 pos_G에 저장한다. $C[1..n]$이 이미 계산되어 있다면, 그 값을 차례대로 읽으며 $O(n+m)$시간에 pos_G를 구할 수 있다. $C[1]+...+C[i]$를 $p_i(1 \le i \le n)$라 정의하자. $pos_A[i]$보다 작은 B의 접미사가 p_i개 있으므로 $pos_A[i]$는 $pos_G[i+p_i]$에 저장된다. 같은 이유로 $pos_B[p_i+1..p_{i+1}]$은 $pos_G[i+p_i+1..i+p_{i+1}]$에 저장된다.

합병 알고리즘의 전체 시간복잡도를 계산해보면, 첫 번째 단계가 $O(m|\Sigma|)$, 두 번째 단계가 $O(n+m)$으로, 아래와 같은 보조정리를 얻을 수 있다.

보조정리 1. 전위 검색을 이용하면 문자열 A와 B에 대한 써픽스 배열을 $O(n+m|\Sigma|)$시간에 합병할 수 있다.

3.2. 후위 검색(backward search) 방법

Chen과 Seiferas[4]는 써픽스 링크와는 반대 개념인 역 링크를 사용하여 써픽스 트리를 구축하는 방법을 제시하였다. 짧은 접미사의 앞에 하나의 문자를 추가한 접미사의 위치를 검색하는 것은 Ferragina와 Manzini[6]가 제시한 후위 검색 방법과 유사하다.

후위 검색은 Burrow-Wheeler 변환[7]에 기반을 둔 검색 방법으로 압축된 써픽스 배열(compressed suffix array)의 검색에서도 사용되었다[8, 9]. 최근 Hon et al[10]은 후위 검색을 사용하면 써픽스 배열을 합병할 수 있음을 밝혔고, Sim et al.[11]은 후위 검색의 속도를 증가시켰다.

후위 검색 방법을 사용한 접미사 순서 결정 알고리즘의 순서

는 아래와 같다.

- $C[1..n]$을 0으로 초기화한다.
- pos_A에 대하여 후위 검색에 필요한 자료 구조를 $O(n)$시간 안에 생성한다.
- $B[m..m]$부터 $B[1..m]$까지 B의 접미사들을 길이가 짧은 것에서부터 긴 것 순서대로, pos_A에서 검색한다. $B[i..m]$이 $pos_A[j-1]$과 $pos_A[j]$ 사이에 있다면, $C[j]$를 1 증가시킨다. m개를 모두 검색하는 데는 $O(m\log|\Sigma|)$시간이 소요된다 [11].

합병된 써픽스 배열 pos_G를 구하는 것은 전위 검색 방법을 사용한 것과 동일하다. 시간복잡도를 살펴보면, 첫 번째 단계가 $O(m\log|\Sigma|)$, 두 번째 단계가 $O(n+m)$으로, 아래와 같은 보조정리를 얻을 수 있다.

보조정리 2. 후위 검색을 이용하면 문자열 A와 B에 대한 써픽스 배열을 $O(n+m\log|\Sigma|)$시간에 합병할 수 있다.

4. 응용 및 실험

4.1. 합병 알고리즘의 응용

두 문자열 A, B에 대한 써픽스 배열이 이미 구축되어 있어도, $A\#B\$$라는 일반화된 문자열에 대해서는 새로이 써픽스 배열을 구축해야 했다. 그러나 본 논문에서 제시한 합병 알고리즘을 사용하면 일반화된 써픽스 배열을 효율적으로 구축할 수 있게 된다. 이렇게 일반화된 써픽스 배열은 유전체 서열 간 상관관계를 찾는 것에 응용될 수 있다. 두 유전체 서열 A, B의 상관관계를 얻기 위해서는 $A\#B\$$에 대한 써픽스 배열을 구축하여 유사한 부분을 찾아낼 수 있는데, 이 때 A, B에 대한 써픽스 배열은 이미 구축되어 있으므로, 본 논문의 알고리즘을 사용하여 짧은 시간 안에 두 자료 구조를 합병할 수 있다.

4.2. 실험 및 결과

합병 알고리즘의 효율성을 검증하기 위하여, 본 논문의 써픽스 배열 합병 알고리즘의 수행 시간과 $A\#B\$$의 써픽스 배열을 다시 구축하는[12] 시간을 비교해보았다. A와 B의 길이는 같다고 가정하여, 각기 1M, 5M, 10M, 30M 크기의 임의생성 문자열을 대상으로, 알파벳의 크기를 2, 4, 64, 128로 변화시키며, Pentium 4 2.8Ghz, 2GB RAM의 컴퓨터상에서 실험하였다.

[그림 2]에서 보는 것처럼 두 개의 써픽스 배열을 합병하는 것이 연결된 문자열에 대한 써픽스 배열을 다시 생성하는 것보다 5배 정도 빠르다. 전위 검색 방법을 사용한 합병 알고리즘은 알파벳이 작을 때(2, 4), 후위 검색 방법을 사용한 합병 알고리즘은 알파벳이 클 때(64, 128) 빠른 속도를 보였다.

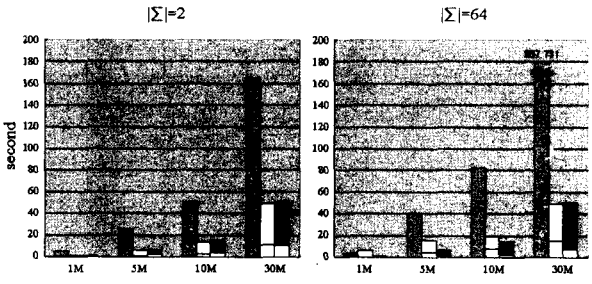

그림 2. 실험결과

5. 결론

본 논문에서는 알파벳이 고정되어 있을 때, 이미 구축된 두 써픽스 배열을 직접 합병하여 일반화된 써픽스 배열을 구축할 수 있는 효율적인 알고리즘을 두 가지 제시하였다. 이 알고리즘들은 유전체 서열간의 상관관계를 파악하는 데 필요한 일반화된 써픽스 배열을 빠른 시간 안에 만들어 준다.

참고문헌

[1] E. McCreight, A space-economical suffix tree construction algorithm, J. Assoc. Comput. Mach. 23, 262-272, 1976

[2] U. Manber and G. Myers, Suffix arrays: A new method for on-line string searches, SIAM J. Comput. 22, 935-938, 1993

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

[4] M. Chen and J. Seiferas, Efficient and elegant subword tree construction, In A. Apostolico and Z. Galil, editors, Combinatorial Algorithms on Words, NATO ASI Series F: Computer and System Sciences, 1985

[5] M. Abouelhoda, E. Ohlebusch, and S. Kurtz, Optimal exact string matching based on suffix array, Symp. on String Processing and Information Retrieval, 31-43, 2002

[6] P. Ferragina and G. Manzini, Opportunistic data structures with applications, IEEE Symp. Found. Computer Science, 269-278, 2001

[7] M. Burrows and D. Wheeler, A block sorting lossless data compression algorithm, Technical Report 124, Digital Equipment Corporation, 1994

[8] R. Grossi and J. Vitter, Compressed suffix arrays and suffix trees with applications to text indexing and string matching, ACM Symp. Theory of Computing, 397-406, 2000

[9] K. Sadakane, Succinct representations of lcp Information and improvements in the compressed suffix arrays, ACM-SIAM Symp. on Discrete Algorithms, 225-232, 2002

[10] W. Hon, K. Sadakane and W. Sung, Breaking a time-and-space barrier in constructing full-text indices, IEEE Symp. Found. Computer Science, 251-260, 2993

[11] J. Sim, D. Kim, H. Park and K. Park, Linear-time search in suffix array, Australasian Workshop on Combinatorial Algorithms, 2003

[12] J. Kärkkäien and P. Sanders, Simple linear work suffix array construction, Int. Colloq. Automata Languages and Programming, 943-955, 2003