써픽스 배열을 이용한 최장 공통 부분 스트링 계산

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

Computing Longest Common Substrings by Using Suffix Arrays

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

최장 공통 부분 스트링이란 주어진 두 개 이상의 스트링에서 가장 길게 일치하는 공통 부분 스트링을 계산하는 문제이다. 최장 공통 부분 스트링은 스트링 프로세싱이나 생물정보학 분야에서 널리 사용되고 있는 중요한 문제이지만, 현재까지 연구된 동적 프로그래밍이나 써픽스 트리를 사용한 방법은 저장 공간을 많이 차지하므로 효율적이지 못하다. 따라서 적은 저장 공간을 차지하면서도 최장 공통 부분 스트링을 빨리 구할 수 있는 알고리즘이 필요하며, 본 논문에서는 이를 위해 써픽스 배열을 도입하였다. 본 논문에서 제시한 알고리즘은 선형 시간, 공간 복잡도를 가지며, 써픽스 트리의 최하 공통 조상(LCA, Lowest Common Ancestor) 연산이나 써픽스 배열에서 사용하는 그와 비슷한 구간 최소값 질의(RMQ, Range Minima Query)를 전혀 사용하지 않으므로 매우 효율적이다.

1. 서론

최장 공통 부분 스트링(longest common substring) 문제는 두 개 이상의 스트링에서 가장 길게 일치하는 부분 스트링을 찾는 것이다[18, 3]. 최장 공통 부분 스트링 문제는 스트링 프로세싱 분야에서 매우 중요한 문제이며 생물정보학에도 널리 사용된다[15]. 예를 들어 두 파일의 유사성을 검사하거나 두 개의 DNA 서열이 주어졌을 때 이들의 생물학적 상관관계를 파악하기 위해 최장 공통 부분 스트링을 사용할 수 있다.

최장 공통 부분 스트링을 계산하는 방법은 여러 가지로 연구되었는데, 초기에는 주로 편집 거리(edit distance)를 이용한 동적 프로그래밍(dynamic programming)을 사용하였다[18,7,13]. 동적 프로그래밍은 길이가 각기 n, m인 두 스트링이 주어졌을 때, $n \times m$의 거리 테이블(distance table)을 만들어야 하므로 시간·공간상 효율적이지 못했다[18]. 오랜 기간의 연구를 통해 동적 프로그래밍의 시간 및 공간 복잡도가 많이 줄어들었으나[7,13], 여전히 최대 한계값과 최소 한계값의 차이가 크며 특히 여러 개의 스트링에 대한 최장 공통 부분 스트링을 구할 경우 각 스트링 쌍마다 거리 테이블을 만들어야 하므로 스트링 개수의 지수적 시간이 필요하다[11]. 따라서 선형 시간에 최장 공통 부분 스트링을 계산하는 방법이 연구되었고[8], 최근에는 써픽스 트리(suffix tree)를 통해 이 문제를 해결하게 되었다.

써픽스 트리[14, 17]는 스트링의 모든 써픽스를 압축된 트라이(trie)형태로 나타낸 것으로, 특정 노드의 모든 하위 노드가 공통의 접두사를 가진다. 또 모든 내부 노드에 써픽스 링크(suffix link)가 정의되어 있어[14], 쉽게 부분 스트링을 찾을 수 있다. 이러한 장점에도 불구하고, 써픽스 트리는 각 노드의 연결 관계와 써픽스 링크 등을 저장하기 위해 스트링보다 최소 5배에서 10배 정도 큰 공간을 요구하므로 DNA 서열 등 대용량의 자료에는 효율적이지 못하다. 본 논문에서는 최장 공통 부분 스트링 문제를 효율적으로 해결하기 위해 써픽스 배열을 도입하였다. 써픽스 배열은 스트링의 모든 써픽스를 사전적 순서로 정렬하여 그 시작 위치를 배열로 나타낸 것이다[12]. 배열을 이용하므로 공간을 적게 차지하며, 최근에는 압축 방법[4,5,16]이 연구되어 스트링과 비슷한 공간에 저장할 수 있게 되었다.

본 논문에서는 써픽스 배열을 이용하여 최장 공통 부분 스트링을 계산하는 방법을 제시한다. 이 방법은 기존의 알고리즘에 비해 적은 공간을 차지하며, 써픽스 트리에서 사용하는 최하 공통 조상(LCA, Lowest Common Ancestor) 연산이나 그에 대응하는 써픽스 배열의 구간 최소값 질의(RMQ, Range Minima Query)같은 복잡한 연산을 전혀 사용하지 않아 매우 효율적이다. 실험을 통해 써픽스 배열을 이용한 본 논문의 알고리즘이 써픽스 트리를 이용한 알고리즘보다 평균 40배 정도 빠르다는 것을 알 수 있다.

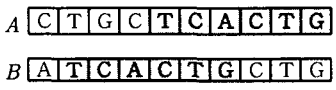
본 논문의 구성은 다음과 같다. 2장에서 문제 및 핵심 아이디어를 설명하고 3장에서 써픽스 배열을 이용한 최장 공통 부분 스트링 계산 방법을 설명한다. 그리고 4장에 기존의 알고리즘과 비교한 실험 결과를 보인 후 5장에서 결론을 맺는다.

2. 문제 정의

정의 1. 두 개의 스트링 A와 B가 주어졌을 때, A의 부분 스트링이면서 B의 부분 스트링이기도 한 가장 긴 스트링을 A와 B의 ***최장 공통 부분 스트링***이라 한다. 이 때 그 길이를 $LCS(A, B)$로 표시하며 $LCS(A, B)$를 구하는 것을 최장 공통 부분 스트링 문제라 한다.

최장 공통 부분 스트링 문제는 Gusfield[6]가 정의한 매칭 통계(matching statistics)를 계산함으로써 해결할 수 있다.

정의 2. 두 개의 스트링 A와 B가 주어졌을 때, A의 i번째 문자에서 시작하는 부분 스트링이면서 B의 부분 스트링이기도 한 가장 긴 스트링의 길이를 $MS_{A,B}(i)$라 하고, 모든 $1 \le i \le |A|$에 대해 $MS_{A,B}(i)$를 구하는 것을 ***매칭 통계***라 한다.



[그림 1] 두 스트링의 최장 공통 부분 스트링

정의에서 보는 것처럼 모든 A와 B에 대한 매칭 통계를 계산하면 쉽게 $LCS(A,B)$를 구할 수 있다. **[그림 1]**을 살펴보면, $MS_{A,B}(1)$=5(CTGCT), $MS_{A,B}(2)$=4(TGCT), ..., $MS_{A,B}(5)$=6(TCACTG)임을 알 수 있다. 음영으로 표시된 TCACTG가 두 스트링 A, B의 최장 공통 부분 스트링이며, 이는 $MS_{A,B}(i)$ 중 가장 큰 값이 나타내는 부분 스트링이다. 즉, $LCS(A,B) = \max\{MS_{A,B}(i) | 1 \le i \le |A|\}$가 성립한다. 본 논문은 이러한 성질을 이용하여 매칭 통계를 계산함으로써 최장 공통 부분 스트링 문제를 해결한다.

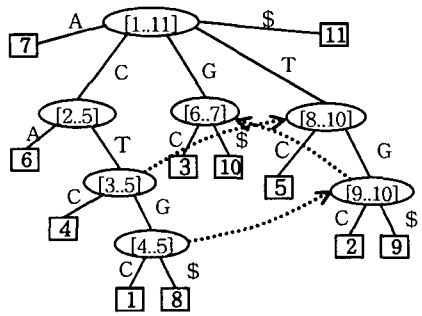
3. 써픽스 배열을 이용한 최장 공통 부분 스트링 계산 알고리즘

써픽스 배열은 써픽스 트리에 비해 적은 공간을 차지하지만, 써픽스 링크 같은 정보가 없어 매칭 통계를 계산하기 어려웠다. 그러나 Abouelhoda et al.이 향상된 써픽스 배열(enhanced suffix array)에서 써픽스 링크를 정의하고, 이를 구하는 효율적인 알고리즘을 제시하였다[1,2].

향상된 써픽스 배열은 기존의 써픽스 배열에 부모-자식 정보를 저장한 child 테이블과 써픽스 링크를 저장한 slink 테이블을

index	1	2	3	4	5	6	7	8	9	10	11
pos	7	6	4	1	8	3	10	5	2	9	11
lcp	0	0	1	2	3	0	1	0	1	2	0
child	2	6	4	5	3	8	7	11	10	9	
slink				8,10	9,10					6,7	

(a) 향상된 써픽스 배열과 써픽스 링크



(b) 구간 트리와 써픽스 링크

[그림 2] 향상된 써픽스 배열과 구간 트리

추가한 것이다. child의 정보로 써픽스 트리와 동일한 형태인 구간 트리(interval tree)를 표현하며, 이 구간 트리에서 써픽스 트리에서와 같은 써픽스 링크를 정의할 수 있다. 특정 구간의 써픽스 링크는 그 구간의 공통 접두사가 $a\alpha$일 때, α를 공통 접두사로 갖는 구간이며, slink에 그 구간의 시작과 끝 위치를 저장한다. **[그림 2]**에 스트링 CTGCTCACTG$에 대한 향상된 써픽스 배열과 그에 대응하는 구간 트리를 나타내었다. **[그림 2]**의 (a)와 (b)를 비교해보면 써픽스 배열에서도 정확한 써픽스 링크를 저장하고 있음을 알 수 있다.

문자집합 Σ에서 정의된 길이가 n인 스트링 A가 주어져 있다. A의 써픽스 배열에서 써픽스 링크를 계산하는 데는 $O(n \log n)$ 시간이 소요되었으나[2], Kim et al.[9, 19]이 구간 최소값 질의(range minima query)같은 복잡한 자료구조를 사용하지 않고서도 $O(n)$시간에 계산할 수 있는 새로운 알고리즘을 고안하였다. 자식 노드를 연결 리스트 형태로 가지고 있는 향상된 써픽스 배열은 특정 문자를 나타내는 자식 노드를 찾는데 $O(|\Sigma|)$시간이 소요되므로 $|\Sigma|$가 커질수록 패턴 검색 속도가 현저히 느려지는 문제가 있었다. 그러나 최근 Kim et al.[10]이 $O(\log|\Sigma|)$에 자식 노드를 찾는 방법을 제시하여 써픽스 트리를 이용한 것보다 빠른 속도로 패턴을 검색할 수 있게 되었다.

길이가 n인 스트링 A와 길이가 m인 스트링 B의 최장 공통 부분 문자열을 계산하기 위해서는 B의 모든 써픽스가 A의 부분 스트링과 얼마나 일치하는지 알아야 한다. 기본적인 방법은 B의 써픽스들을 각각 독립된 스트링으로 가정하여 하나씩 A에서 검색하는 것이지만, 이 경우 $O(nm)$시간이 소요된다. 본 논문에서는 써픽스 링크를 이용하여 좀 더 빠르게 검색하는 방법을 사용한다.

써픽스 배열을 이용하여 A,B의 최장 공통 부분 스트링을 구하는 방법은 다음의 단계를 거친다.

- 스트링 A에 대한 향상된 써픽스 배열을 구축하고 써픽스 링크를 계산한다.
- 스트링 B의 모든 써픽스를 A의 향상된 써픽스 배열에서 검색하여 각각의 매칭 통계를 기록한다.
- 기록된 매칭 통계에서 가장 큰 값을 찾는다.

두 번째 단계에서 매칭 통계는 다음과 같이 계산한다. B의 부분 스트링을 $B[a..b]$라 표현할 때, A의 향상된 써픽스 배열에서 $B[i..m]$을 검색하는 경우를 생각해보자. 구간 트리를 따라 구간 $[u..v]$까지 검색한 후, $B[j..j](i \le j \le m)$에서 오류(mismatch)가 발생했다고 가정한다. 이 때, $[u..v]$의 공통 접두사는 $B[i..j-1]$이다. $[u..v]$의 써픽스 링크가 $[p..q]$이면, 구간 $[p..q]$의 공통 접두사는 $B[i+1..j-1]$일 것이다. 이는 다음 써픽스 $B[i+1..m]$의 접두사이기도 하므로, 다시 루트부터 검색할 필요 없이 구간 $[p..q]$에서부터 $B[j..m]$을 검색한다. 이렇게 진행하면 한 번 비교한 문자를 다시 비교하지 않으므로 선형 시간에 모든 써픽스를 검색할 수 있다.

예를 들어 **[그림 2]**의 향상된 써픽스 배열을 이용하여 스트링 A=CTGCTCACTG와 B=CTGA의 매칭 통계를 구해보자. i가 1일 때 구간 $[4..5]$에서 오류가 발생하므로 $MS_{A,B}(1)$=CTG이다. 이 때 루트인 $[1..11]$에서 다시 TGA를 검색하면

```
procedure MatchingStatistics(B, ESA_A)
1: start = 1, end = n
2: j = 1
3: for i = 1 to m
4:   j = Search (B[j..n], ESA_A[start, end])
5:   MS_{A,B}(i) = j − i
6:   (start, end) = suffix link of ESA_A[start, end]
7: end for
```

[그림 3] 써픽스 배열을 이용한 매칭 통계 알고리즘

시간이 많이 걸리지만, [4..5]의 써픽스 링크를 이용하면 TG까지 일치하는 구간인 [9..10]을 상수시간에 찾을 수 있으므로 문자 A부터 다시 검색하면 된다.

A의 향상된 써픽스 배열 ESA_A와 B가 주어졌을 때 A와 B의 매칭 통계를 구하는 알고리즘의 의사코드가 **[그림 3]**에 나타나있다. 4번째 줄의 ***Search***는 $ESA_A[start..end]$부터 스트링 $B[j..n]$을 검색하면서 $start$와 end를 현재까지 검색한 구간으로 갱신하고, 오류가 발생할 경우 그 위치를 돌려주는 함수이다.

두 개 이상의 스트링에서의 최장 공통 부분 스트링 계산 역시 선형시간에 수행할 수 있다. k개의 스트링 $S_1, S_2, ..., S_k$가 있을 때, 특정 스트링 $S_a(1 \le a \le k)$에 대한 향상된 써픽스 배열을 구축한 후 모든 스트링 $S_i(1 \le i \le k, i \ne a)$를 각각 검색하여 매칭 통계를 구해 $LCS(S_a, S_i)$를 기록한다. 끝으로 기록된 값들 중 최소값을 구하면 k개 스트링의 최장 공통 부분 스트링의 길이를 얻을 수 있다. 이 방법은 모든 스트링의 길이를 더한 시간이 소요된다.

4. 실험결과

본 논문에서 제시한 알고리즘의 성능 측정을 위해 기존의 써픽스 트리를 이용한 방법과 향상된 써픽스 배열을 이용한 방법으로 최장 공통 부분 스트링을 구하는 실험을 하였다. 써픽스 트리는 Ukkonen[17]의 방법에 따라 구축하였으며, 저장 공간을 줄이기 위해 연결 리스트(linked list)를 사용하였다. 실험은 Pentium-4 2.8Ghz CPU에 2GB의 메모리를 장착한 컴퓨터에서 수행하였다. **[표 1]**은 문자 집합 Σ의 크기가 각각 2, 4, 64, 128일 때, 스트링 A, B의 길이를 1M, 5M, 10M로 변경하며 수행시간을 측정한 결과이다.

[표 1] 써픽스 트리와 써픽스 배열을 이용한 최장 공통 부분 스트링 알고리즘의 수행 시간 비교

$\lvert\Sigma\rvert$	써픽스 트리			써픽스 배열		
	1M	5M	10M	1M	5M	10M
2	0.94	5.51	147.67	0.26	1.32	2.89
4	1.98	11.19	253.25	0.33	1.35	2.85
64	17.45	115.31	314.36	1.29	3.86	4.04
128	25.22	203.70	649.31	1.90	7.36	8.78

연결 리스트로 구현한 써픽스 트리는 특정 문자를 갖는 자식 노드를 찾는데 $O(|\Sigma|)$시간이 소요되므로 $|\Sigma|$가 커질수록 속도가 현저히 떨어진다. 반면, 본 논문에서 제시한 알고리즘은 $O(\log|\Sigma|)$시간에 같은 작업을 할 수 있으므로 $|\Sigma|$와 길이의 변화에 민감하지 않다. 문자 집합과 스트링의 길이가 작을 때는 약 2배 정도의 속도 향상이 있으나, 문자 집합이 크고 스트링 길이가 길 경우에는 최대 80배 정도의 차이를 보인다.

5. 결론

본 논문은 최장 공통 부분 스트링을 계산하는 효율적인 알고리즘을 제시하였다. 지금까지 최장 공통 부분 스트링을 구하는 문제는 동적 프로그래밍이나 써픽스 트리를 사용하여 해결하였으나 시간과 공간적인 면에서 비효율적이었다. 본 논문에서는 써픽스 트리에 비해 적은 공간을 차지하는 써픽스 배열을 이용하여 빠른 시간에 최장 공통 부분 스트링을 계산하였다. 실험결과 써픽스 트리를 이용한 기존의 알고리즘보다 평균 40배 정도 성능이 향상되었다.

6. 참고문헌

[1] 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

[2] M. Abouelhoda, S. Kurtz, E. Ohlebusch, Replacing suffix trees with enhanced suffix arrays, *J. Discrete Algorithms*, 53-86, 2004

[3] V. Chvátal and D. Sankoff, Longest Common subsequences of two random sequences, *J. Applied Probability*, 12, 306-315, 1975

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

[5] 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

[6] D. Gusfield, Algorithms on Strings, Trees, and Sequences, Computer Science and Computational Biology, Press Syndicate of the University of Cambridge, 1997

[7] D. Hirschberg, Algorithms for the longest common subsequence problem, *J. the Assoc. Comput. Mach.*, 24(4), 664-675, 1977

[8] L. Hui, Color set size problem with applications to string matching, *In Proc. of Combinatorial Pattern Matching*, LNCS 644, 230-243, 1992

[9] D. K. Kim, J. E. Jeon, H. Park, An Efficient index data structure with the capabilities of suffix trees and suffix arrays for alphabets of non-negligible size, *Symp. on String Processing and Information Retrieval*, 2004

[10] D. K. Kim, J. E. Jeon, H. Park, Merging suffix arrays in Linear time for Integer Alphabets, *In Proc. of AWOCA*, 129-140, 2004

[11] D. Maier, The complexity of some problems on subsequences and supersequences, *J. Association for Computing Machinery*, 25(2), 322-336, 1978

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

[13] W. Masek and M. Paterson, A faster algorithm computing string edit distances, *J. Computer and System Sciences*, 20(1), 18-31, 1980

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

[15] S. Needleman, C. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, *J. Mol. BioInformatics*, 48(3), 443-453, 1970

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

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

[18] R. Wagner and M. Fischer, The string-to-string correction Problem, *J. ACM*, 21, 168-173, 1974

[19] 전정은, 박희진, 김동규, 효율적인 써픽스 배열 합병 알고리즘과 응용, 정보과학회 학술지, 봄 학술발표논문집(A) 31권, 1호, 973-975, 2004.