DNA 스트링에 효율적인 써픽스 배열 구축 알고리즘[1)]

조준하[O] 박희진[1] 김동규[2]
[O, 2] 부산대학교 컴퓨터공학과
jhjo@islab.ce.pusan.ac.kr[O], dkkim1@pusan.ac.kr[2]
[1] 한양대학교 정보통신학부
hjpark@hanyang.ac.kr

An Efficient Algorithm for Constructing Suffix Arrays for DNA String

Junha Jo[O], Heejin Park[1], Dong Kyue Kim[2]
[O, 2] School of Electrical and Computer Engineering, Pusan National University
[1] College of Information and Communications, Hanyang University

요 약

써픽스 배열은 텍스트의 써픽스들을 사전적 순서대로 저장하여 검색을 효율적으로 할 수 있는 자료구조이다. 생물학에서의 DNA 스트링과 같이 긴 텍스트에 대해 써픽스 배열을 이용하면 빠르게 검색할 수 있다. 써픽스 배열은 유사한 자료구조인 써픽스 트리에 비해 적은 공간을 차지하기 때문에 생물학에서 사용하는 긴 텍스트의 처리에 유리하다. 최근, 텍스트에서 바로 써픽스 배열을 선형시간에 구축하는 알고리즘들이 발표되었다. 그러나 이들 알고리즘은 정수 문자집합을 위한 알고리즘들이었다. 본 논문에서는 고정길이 문자집합에 대해 써픽스 배열을 빠르게 구축하는 알고리즘을 소개한다. 그리고 실험을 통해서 DNA 스트링과 같은 고정길이 문자집합에 대해서 다른 알고리즘들과 구축시간을 비교하여 속도 향상이 있음을 보인다.

1. 서 론

텍스트 검색에는 크게 2가지 방법이 있다. 첫 번째 방법은 패턴을 전처리 하는 방법이다. 패턴을 전처리 하는 시간은 $O(m)$이며, 이 때 검색하는 시간은 $O(n)$이다. 두 번째 방법은 텍스트 전체에 대한 인덱스 자료구조를 만드는 것이다. 전체 텍스트에 대한 인덱스 자료구조를 구축하는 시간은 $O(n)$이며, 검색하는 시간은 $O(m)$이다. DNA 스트링을 검색하기 위해서는 후자의 방법이 더 적합하다. 왜냐하면 텍스트의 길이가 패턴의 길이에 비해 훨씬 크기 때문이다.

많이 사용되는 인덱스 자료구조에는 써픽스 트리와 써픽스 배열 두 가지가 있다. 써픽스 트리는 텍스트의 모든 써픽스들을 압축된 trie로 표현하는 것이며, 써픽스 배열은 정렬된 모든 써픽스들을 리스트로 표현하는 것이다. McCreight [7], Ukkonen [9], Farach [1] 등의 써픽스 트리 구축 알고리즘은 $O(n)$시간에 구축되며, $O(n)$의 크기를 가진다. 써픽스 배열은 써픽스 트리를 순회해서 $O(n)$시간에 구축할 수 있다. 그러나 이러한 써픽스 트리는 공간을 많이 차지하고, 구현이 복잡하기 때문에 비효율적이다.

Manber and Myers [6]와 Gusfield [2]는 써픽스 배열을 사용하지 않고, 텍스트에서 직접 써픽스 배열을 $O(n\log n)$시간에 구축하는 알고리즘을 개발했다. 그리고 2003년에는 써픽스 배열을 $O(n)$시간에 구축하는 알고리즘들이 개발되었다. Kim el al. [4], Kärkkäinen and Sanders [3], Ko and Aluru [5]의 알고리즘이 그것이다.

그러나 이 알고리즘들은 정수 문자집합을 위한 알고리즘이기 때문에 DNA 스트링과 같이 문자집합의 크기가 작은 경우 빠른 속도를 보장하지 않는다. 따라서 고정길이 문자집합으로 구성된 스트링에 대해 써픽스 배열을 빠르게 구축하는 알고리즘의 개발이 필요하다.

본 논문에서는 DNA 스트링과 같이 문자집합의 크기가 고정되어 있을 때, 빠르게 써픽스 배열을 구축하는 알고리즘을 소개한다. 2장에서는 써픽스 배열을 구축하는 새로운 알고리즘을 설명할 것이며, 3장에서는 실제로 구현하여 다른 알고리즘들과 비교한 결과를 살펴볼 것이다. 마지막으로 4장에서 이 논문에 대한 결론을 내릴 것이다.

2. 써픽스 배열 구축 알고리즘

문자집합을 Σ라고 하자. 문자집합 Σ로 구성된 텍스트를 T라고 하고, 텍스트의 길이를 n으로 표시한다. 텍스트 T에서 i $(1 \le i \le n)$번째 문자를 $T[i]$라고 하고, 텍스트의 마지막 문자 $T[n]$은 Σ의 어떤 문자보다 사전적인 순서가 빠르고, 텍스트 안에 없는 특별한 문자 #으로 나타낸다. 텍스트 T의 써픽스 배열을 SA_T라고 한다. 텍스트 T에서 홀수 번째 위치에서 시작하는 써픽스를 홀수(짝수) 써픽스라 하고, 모든 홀수(짝수) 써픽스들의 정렬된 리스트를 홀수(짝수) 써픽스 배열이라고 한다. 홀수 (짝수) 써픽스 배열은 $SA_o(SA_e)$ 로 나타낸다.

2.1 전체 알고리즘의 개요

1. 인접한 두 개의 문자($T[2i-1]$, $T[2i]$)의 짝을 이용해서 T를 T'로 부호화한다. T'는 T의 크기의 절반이 된다.
2. 부호화된 텍스트 T'의 $SA_{T'}$를 재귀적으로 구축한다.
3. $SA_{T'}$을 이용해서 T의 홀수 써픽스 배열 SA_o를 구축한다.
4. SA_o를 이용해서 T의 짝수 써픽스 배열 SA_e를 구축한다.

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

5. 홀수 써픽스 배열 SA_o와 짝수 써픽스 배열 SA_e를 합병하여 SA_T를 구축한다.

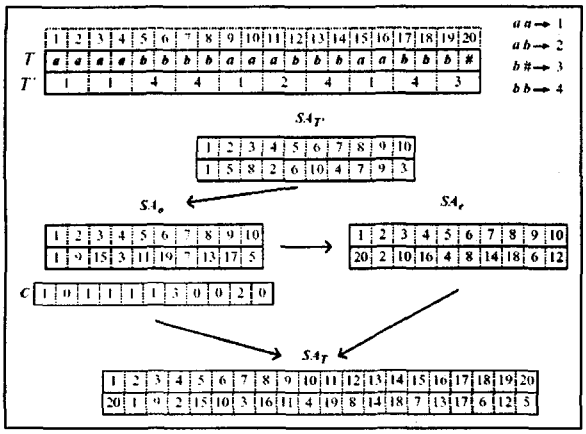


그림 1. 써픽스 배열 구축 알고리즘의 예.

2.2 합병 알고리즘

후위 검색은 $P(=p_1p_2p_3\cdots p_m)$를 뒤에서부터 한 문자씩 읽으면서 써픽스 배열 SA_T에서의 위치를 찾아 나가는 것이다. p_m의 위치를 찾고, 그 다음은 $p_{m-1}p_m$의 위치를 찾고, 이러한 과정을 반복하면서 $p_1p_2p_3\cdots p_m$까지의 위치를 찾아간다. 후위 검색을 하게 되면 패턴 P의 모든 써픽스들이 텍스트 T에 어떤 곳에 위치하는지를 알 수 있다.

후위 검색을 하기 위한 자료구조는 Sim et al. [8]의 알고리즘의 방법을 사용하였고, 이 알고리즘은 $O(m\log|\Sigma|)$의 검색 시간이 걸리며, 사용공간은 $O(n)$이다.

짝수와 홀수 써픽스 배열을 합병하는 과정은 두 단계로 이루어지며 이 과정에서 하나의 저장 공간을 필요로 한다. 이 저장 공간을 배열 $C[1..n/2+1]$라고 하자.

1. 인접한 두 개의 문자($T[2k]$, $T[2k+1]$)의 짝을 이용해서 T''으로 부호화한다. T''과 SA_o를 이용해서 후위검색을 하여, 홀수 써픽스 $SA_o[i-1]$보다 크고, $SA_o[i]$보다 작은 짝수 써픽스들의 개수를 계산한다. 그 결과를 $C[i]$에 저장한다. $C[n/2+1]$에는 $SA_o[n/2]$보다 큰 짝수 써픽스들의 개수를 저장한다.
2. C를 이용해서 홀수 써픽스 배열 SA_o와 짝수 써픽스 배열 SA_e를 합병하여 써픽스 배열 SA_T를 만든다. $C[i]$에 저장된 개수만큼의 짝수 써픽스들은 홀수 써픽스 $SA_o[i]$보다 사전적 순서가 빠른 것이므로 SA_T에 먼저 저장한다. 그리고, 홀수 써픽스 $SA_o[i]$를 SA_T에 저장한다. 이 과정을 계속 반복해서 홀수 써픽스 배열과 짝수 써픽스 배열을 SA_T에 저장하면 써픽스 배열은 완성된다.

T'의 문자집합을 Σ'라고 하면, 첫 번째 단계의 시간 복잡도는 $O(n\log|\Sigma'|)$이고, 두 번째 단계는 $O(n)$이므로 전체 합병 알고리즘의 시간 복잡도는 $O(n\log|\Sigma'|)$이다. i번째 재귀 단계에서의 텍스트를 T^i라 하고, 이 때의 문자집합을 Σ^i라고 하면, 여기서 다음과 같은 따름정리를 얻을 수 있다.

따름정리 1. i번째 단계의 시간 복잡도는 $O(|T^i|\log|\Sigma^{i+1}|)$이다.

2.3 알고리즘의 시간복잡도

이 알고리즘은 합병을 제외하면 $O(n)$의 시간 복잡도를 가진다. 따라서 모든 재귀 단계에서 합병에 필요한 시간을 고려하면 전체 알고리즘의 시간 복잡도를 구할 수 있다. 시간 복잡도의 분석은 $\log\log n$ 번째까지의 재귀 단계 부분과 $\log\log n$ 이후의 재귀 단계부분으로 나누어서 분석한다.

보조정리 1. $\log\log n$ 번째 재귀 단계까지의 시간 복잡도는 $O(\log\log n)$이다.

증명. i번째 재귀 단계의 시간 복잡도는 따름정리 1에 의해서 $O(|T^i|\log|\Sigma^{i+1}|)$이다. 이 때, $|T^i|$는 $|T^{i-1}|$의 절반이고, $|\Sigma^i|$는 $|\Sigma^{i-1}|$의 제곱만큼 증가할 수 있으므로 $O(|T^i|\log|\Sigma^{i+1}|) = O(n/2^{i-1}\cdot\log|\Sigma|^{2^i}) = O(n\cdot\log|\Sigma|)$가 된다. $|\Sigma|$는 고정된 크기이므로 $O(n)$이다. 따라서 $\log\log n$ 번째까지의 재귀 단계의 시간 복잡도는 $O(\log\log n)$이다.

보조정리 2. $\log\log n$ 이후의 모든 재귀 단계의 시간 복잡도는 $O(n)$이다.

증명. $i\,(i > \log\log n)$ 단계에서의 시간 복잡도는 따름정리 1에 의해서 $O(|T^i|\log|\Sigma^{i+1}|)$이다. $|T^i|$는 $|T^{i-1}|$의 절반이고, $|\Sigma^{i+1}|$은 $|\Sigma|^{2^i}$까지 증가할 수 있으므로 $O(n/2^{i-1}\cdot\log|\Sigma|^{2^i})$가 된다. 이 때 $|\Sigma|$는 $|T^i|$보다 클 수 없으므로 $O(n/2^{i-1}\cdot\log(n/2^i))$가 된다. i를 $\log\log n+j$로 나타내면 $O(n/2^{i-1}\cdot\log(n/2^i)) = O(n/(2^{j-1}\cdot\log n)\cdot\log(n/2^{\log\log n+j}))$이다. 그리고 $\log n \geq \log(n/2^{\log\log n+j})$이므로 결국 $O(n/2^{j-1})$이 된다. 따라서 $O(n+n/2+n/4+\cdots) = O(n)$이다.

보조정리 1과 2에 의해서 다음의 정리를 얻을 수 있다.

정리 1. 이 알고리즘의 시간 복잡도는 $O(\log\log n)$이다.

3. 써픽스 배열 구축 알고리즘 구현 및 비교 결과

3.1 비교 알고리즘과 소스 코드

Ko and Aluru 알고리즘, Kärkkäinen and Sanders 알고리즘과 본 논문의 알고리즘을 비교하였다. 객관적으로 비교하기 위해 각 알고리즘들의 저자가 공개한 소스코드를 이용했고, 그렇지 않은 경우 직접 구현해서 비교했다.

- Ko and Aluru 알고리즘
  공개된 소스코드가 없어 논문에 기술된 그대로 구현했다.
- Kärkkäinen and Sanders 알고리즘
  Kärkkäinen and Sanders 논문에 기술된 Sanders의 인터넷 URL을 참조해서 구한 소스를 사용했다.

3.2 실험 환경

실험에 사용된 시스템의 사양은 다음과 같다.

[표 1] 실험에 사용된 시스템 사양

CPU	Intel Pentium4 2.8GHz
RAM	Samsung DDR 2G
O/S	MS Windows 2000 professional

실험 시 주 기억 장치에서만 실행되게 하여, 보조 기억 장치를 이용하지 않았다. 파일에서 읽고 쓰는 부분을 제외한 순수한 알고리즘만의 실행시간을 1/1000 초 단위로 측정하였으며 각 경우에 대해서 10번 이상 실험하여 얻은 값들의 평균을 기록하였다.

3. 3 입력 자료

입력 자료는 NCBI에 등록되어 있는 DNA 스트링을 이용하였다. 문자집합의 크기는 DNA 스트링에서 사용되는 A, C, G, T에 해당하는 4이다. 사용된 DNA 스트링은 다음과 같다.

[표 2] 실험에 사용한 DNA 스트링

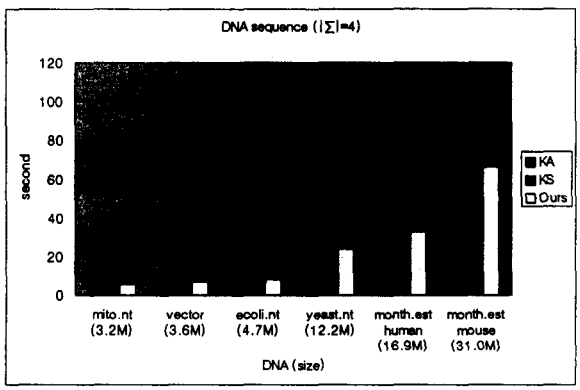
DNA 스트링	DNA 스트링의 크기
mito.nt	3.2M
vector	3.6M
ecoli.nt	4.7M
yeast.nt	12.2M
month.est human	16.9M
month.est mouse	31.0M

위의 파일들에 포함되어 있는 주석들은 모두 제거한 파일을 이용해서 실험하였다.

3. 4 실험 결과

[표 3] DNA 스트링의 써픽스 배열 구축 시간 (단위 : 밀리초)

DNA	mito.nt	vector	ecoli.nt	yeast.nt	month. est human	month. est mouse
KA03	8.234	7.781	11.750	32.609	41.094	87.656
KS03	9.531	10.062	14.640	40.734	53.125	101.756
Ours	5.500	6.828	8.453	23.875	32.969	66.031



[그림 2] 표 3의 그래프

표와 그래프에서 볼 수 있듯이 DNA 스트링을 이용해서 써픽스 배열을 구축하는 시간이 다른 알고리즘에 비해서 빠르다는 것을 알 수가 있다. Ko and Aluru 알고리즘에 비해서 평균 25% 정도 빠르며, Karkkainen and Sanders 알고리즘에 비해서 평균 38% 정도 빠르다.

4. 결 론

지금까지 발표된 선형 써픽스 배열 구축 알고리즘은 정수 문자집합에 대한 알고리즘들이었다. 이러한 알고리즘들은 DNA 스트링과 같이 스트링의 길이가 매우 길고, 스트링을 구성하는 문자집합의 크기가 스트링의 길이에 비해 매우 작게 고정되어 있는 경우에 효율적이지 못하다. 따라서 본 논문에서는 고정길이 문자집합에 대해 효율적으로 써픽스 배열을 구축하는 알고리즘을 소개하였다. 알고리즘을 실제로 구현하여 비교해 본 결과 앞서 설명한 것처럼 본 논문의 알고리즘은 DNA 스트링과 같은 고정길이 문자집합으로 구성된 스트링에 대해서 다른 선형 써픽스 배열 구축 알고리즘보다 25% - 38% 정도 더 빠르게 써픽스 배열을 구축하였다.

최근 써픽스 배열의 사용공간을 줄이는 방법이 활발히 연구되고 있다. 써픽스 배열에 필요한 공간을 $O(n)$ bit로 만들어서 구축하는 것이다. 본 논문에서는 써픽스 배열의 빠른 구축에 초점을 두었기 때문에, 사용공간은 $O(n\log n)$ bit이다. $O(n)$ bit를 사용하는 빠르고 효율적인 써픽스 배열 구축 알고리즘의 개발이 필요할 것이다.

참고문헌

[1] M. Farach, Optimal suffix tree construction with large alphabets, IEEE Symp. Found. Computer Science, 137-143, 1997.

[2] D. Gusfield, An "Increment-by-one" approach to suffix arrays and trees, manuscript, 1990.

[3] J. Kärkkäinen and P. Sanders, Simpler linear work suffix array construction, Int. Colloq. Automata Languages and Programming, 943-955, 2003.

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

[5] P. Ko and S. Aluru, Space-efficient linear time construction of suffix arrays, Symp. Combinatorial Pattern Matching, 200-210, 2003.

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

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

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

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