써픽스 배열을 구축하는 빠른 알고리즘

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

A Fast Algorithm for Constructing Suffix Arrays

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 정렬된 모든 써픽스들의 인덱스를 저장한 자료구조이며, 긴 문자열에서 임의의 패턴을 효율적으로 검색을 할 수 있는 자료구조이다. 비슷한 자료구조인 써픽스 트리에 비해 적은 공간을 사용하기 때문에 대용량의 텍스트에 대한 처리에 더 적합하다. 본 논문에서는 써픽스 배열을 빠르게 구축하는 방법을 제안하고, 써픽스 배열 구축 알고리즘들 중에서 빠르다고 알려진 Larsson and Sadakane 알고리즘, 대표적인 선형 시간 알고리즘인 Kärkkäinen and Sanders 알고리즘 및 최근에 발표된 고정길이 문자집합에 효율적인 Kim et al. 알고리즘과 성능을 비교한다. 실험 결과 본 논문에서 제안한 알고리즘이 전반적으로 빠르게 써픽스 배열을 구축하였다.

1. 서 론

텍스트 검색 문제는 길이가 n인 텍스트에서 길이가 m인 패턴이 존재하는 모든 위치를 찾는 문제이다. 이 문제를 해결하는 방법으로 패턴을 $O(m)$시간에 전처리하여 $O(n)$시간에 검색하는 방법과 전체 텍스트에 대한 인덱스 자료구조를 $O(n)$ 시간에 구축하여 $O(m)$시간에 검색하는 두 가지 방법이 있다. 패턴의 길이에 비해 텍스트의 길이가 매우 길고, 검색이 빈번한 경우 인덱스 자료구조를 구축하는 방법이 더 적합하다.

대표적인 인덱스 자료구조는 써픽스 트리(suffix tree)와 써픽스 배열(suffix array)이다. 써픽스 트리는 텍스트의 모든 써픽스들을 압축된 트라이(trie)로 표현한 것이며, 써픽스 배열은 정렬된 모든 써픽스들의 인덱스를 리스트로 표현한 것이다. McCreight [10], Ukkonen [11], Farach [1] 등이 제시한 써픽스 트리 구축 알고리즘은 선형 시간에 구축되며, 선형 공간 복잡도를 가진다. 써픽스 배열은 써픽스 트리를 사용하여 선형 시간에 구축할 수 있지만 써픽스 트리를 사용하는 방법은 공간을 많이 차지하고, 구현이 복잡하기 때문에 비효율적이다.

Manber and Myers [9]와 Gusfield [2]는 써픽스 트리를 사용하지 않고, 텍스트에서 직접 써픽스 배열을 구축하는 알고리즘을 제시하였다. Larsson and Sadakane [7]는 Manber and Myers 알고리즘의 단점을 보완한 알고리즘을 제시하였다. 이 알고리즘들은 $O(n\log n)$시간에 써픽스 배열을 구축한다. 최근 발표된 Kärkkäinen and Sanders [3], Kim el al. [4], Ko and Aluru [6]의 알고리즘들은 써픽스 배열을 선형 시간에 구축한다. 또한 고정길이 문자집합으로 구성된 텍스트의 써픽스 배열을 $O(n\log\log n)$ 시간에 구축하는 Kim et al. [5, 12] 알고리즘이 발표되었다.

써픽스 배열의 구축 시간은 텍스트의 길이에 비례한다. 특히 DNA 스트링과 같이 길이가 매우 길고 크기가 작은 문자집합으로 구성된 텍스트는 써픽스 배열을 빠르게 구축하는 것이 매우 중요하다.

본 논문에서는 고정길이 문자집합의 크기가 작을수록 텍스트에 대한 써픽스 배열을 더 빠르게 구축하는 Kim et al. 알고리즘의 인코딩과 합병 방법을 개선하여 성능을 향상시키는 방법을 제안한다. 그리고 빠른 알고리즘으로 알려진 Larsson and Sadakane 알고리즘, 대표적인 선형 시간 알고리즘인 Kärkkäinen and Sanders 알고리즘, 고정길이 문자집합에 효율적인 Kim et al. 알고리즘을 본 논문에서 제안한 방법과 써픽스 배열 구축 시간을 비교해 본다. (알고리즘들의 객관적인 비교를 위해서 저자가 공개한 코드나 학회에 발표된 검증된 코드를 사용하였다.) 실험을 통해 본 논문에서 제안한 방법을 사용한 알고리즘이 크기가 작은 문자집합으로 구성된 텍스트의 써픽스 배열을 가장 빠르게 구축함을 보였다.

2장에서는 각 알고리즘들을 간단히 소개하고 속도 향상을 위한 방법을 설명한다. 3장에서는 실험한 결과를 보이고, 마지막으로 4장에서는 본 논문에 대한 결론을 맺는다.

2. 써픽스 배열 구축 알고리즘

2. 1. Larsson and Sadakane 알고리즘 [7]

Manber and Myers 알고리즘이 사용하는 doubling 기법을 이용한다. 각 단계에서 $O(n)$ 시간에 써픽스들을 정렬하며 단

계는 $O(\log n)$ 단계까지 증가할 수 있으므로 전체 구축 시간은 $O(n\log n)$이고 사용 공간은 $O(n)$이다. 이 알고리즘의 특징은 이미 정렬된 써픽스들은 읽지 않고, 정렬되지 않은 써픽스들만 정렬하기 때문에 Manber and Myers 알고리즘보다 빠르다.

2. 2. Kärkkäinen and Sanders 알고리즘 [3]

Kärkkäinen and Sanders 알고리즘은 재귀적 분할 정복 기법을 사용한다. 먼저 써픽스를 A, B 두 그룹으로 나눈다. A 그룹은 써픽스의 인덱스 i가 i mod 3 ≠ 1 인 써픽스이고, B 그룹은 i mod 3 = 1 인 써픽스이다. A 그룹을 인코딩하여 T'을 생성하고, T'의 써픽스 배열을 재귀적으로 구축한다. T'을 이용해서 A 그룹의 써픽스 배열을 직접 구축한다. A 그룹의 써픽스 배열을 이용해서 B 그룹의 써픽스 배열을 구축한다. A, B 그룹의 써픽스 배열을 합병해서 전체 텍스트 T의 써픽스 배열을 구축한다. 각각의 과정은 $O(n)$ 시간에 수행되며 각 재귀 단계마다 인코딩된 텍스트의 길이는 이전 텍스트 길이의 2/3가 되므로 재귀식은 $T(n) = T(2n/3) + O(n)$이다. 따라서 전체 사용 공간 및 구축 시간은 $O(n)$이다.

2. 3. Kim et al. 알고리즘 [5]

Kim et al. 알고리즘 역시 재귀적 분할 정복 기법을 사용한다. 써픽스를 A, B 두 그룹으로 나눈다. A 그룹은 홀수 써픽스이고, B 그룹은 짝수 써픽스이다. A 그룹을 먼저 인코딩하여 T'을 생성하고, T'의 써픽스 배열을 재귀적으로 구축한다. T'의 써픽스 배열로부터 A 그룹의 써픽스 배열을 직접 구축한다. A 그룹의 써픽스 배열을 이용해서 B 그룹의 써픽스 배열을 구축한다. A, B 그룹의 써픽스 배열을 합병해서 T의 써픽스 배열을 구축한다. 합병을 제외한 나머지 과정은 $O(n)$ 시간에 수행되며 각 재귀 단계마다 인코딩된 텍스트의 길이는 이전 텍스트 길이의 1/2이 되므로 재귀식은 $T(n) = T(n/2) + O(n)$이다. 합병 과정은 $O(n\log\log n)$ 시간에 수행되므로 전체 구축 시간은 $O(n\log\log n)$이고, 사용 공간은 $O(n)$이다.

2. 4. 제안한 알고리즘

Kim et al. 알고리즘은 기수 정렬(radix sorting)을 이용하여 인코딩을 한다. 기수 정렬을 이용하는 방법은 속도가 다소 느리기 때문에 문자집합의 크기가 작은 경우에는 $|\Sigma|^2$의 크기를 가지는 테이블을 이용해서 인코딩을 빠르게 수행할 수 있다. 매 재귀 단계마다 문자집합의 크기가 기하급수적으로 증가하기 때문에 DNA 스트링의 경우 3번째 재귀 단계까지 이 방법을 사용할 수 있다. 인코딩 과정은 $T(n) = T(n/2) + O(n)$ 이므로 이 방법을 이용했을 경우 1, 2, 3 재귀 단계에서 각각 50%, 75%, 87.5%의 성능향상을 기대할 수 있다.

Kim et al. 알고리즘은 합병시 배열 C를 사용하여 홀수 써픽스 배열과 짝수 써픽스 배열을 합병한다. 배열 C는 정렬된 홀수 써픽스 배열의 인접한 두 써픽스 사이에 포함될 짝수 써픽스들의 개수를 의미한다. 배열 C를 계산하기 위해서는 인코딩된 텍스트의 문자집합의 크기를 가지는 구간내에서 이진 검색을 해야 한다. 문자집합의 크기가 작은 경우에는 이진검색을 하지 않고 스캔하면서 값을 구하는 방법을 사용할 수 있다. 이렇게 하면 이진 검색에 필요한 자료구조를 만들지 않아도 되고, 스캔하는 시간도 이진 검색에 비해 느리지 않기 때문에 속도를 향상시킬 수 있다. DNA 스트링은 첫 번째 재귀 단계에서 인코딩된 텍스트의 문자집합은 크기가 16이므로 이 방법을 사용할 수 있었지만 두 번째 재귀 단계부터는 문자집합의 크기가 커져서 이 방법을 사용할 수 없었다.

3. 성능 비교

3. 1. 실험 환경

실험에 사용한 시스템은 [표 1]과 같다.

[표 1] 실험에 사용된 시스템

CPU	Intel Pentium IV 2.8G
RAM	DDR 2.0G
O/S	Microsoft Windows 2000 Server

각 알고리즘들의 소스 코드에 대한 정보는 다음과 같다.

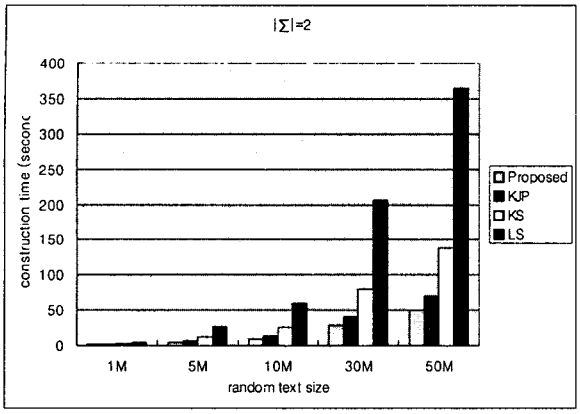
- Larsson and Sadakane 알고리즘
  - 논문의 저자 중 한명인 Larsson의 코드를 사용하였다.
  http://www.larsson.dogma.net/research.html/
- Kärkkäinen and Sanders 알고리즘
  - 논문의 저자 중 한명인 Sanders의 코드를 사용하였다.
  http://www.mpi-sb.mpg.de/~sanders/programs/suffix/
- Kim et al. 알고리즘
  - 논문의 저자가 구현한 코드를 사용하였다.
- 본 논문에서 제안한 알고리즘
  - Kim et al. 알고리즘을 수정하여 인코딩과 합병을 빠르게 하는 방법을 직접 구현하여 사용하였다.

실험에 사용된 텍스트는 문자집합의 크기가 2, 4 인 랜덤 스트링과 DNA 스트링을 사용하였다. 랜덤 스트링의 경우 길이를 1M, 5M, 10M, 30M, 50M로 하여 실험하였다. DNA 스트링은 A, C, G, T의 4개의 문자로 구성되어 있고, NCBI에 등록되어 있는 것들을 사용하였다. 모든 실험은 주 기억 장치만을 사용하였고, 각 실험은 10번씩 반복하여 얻은 평균값을 1/1000초 단위로 측정하였다.

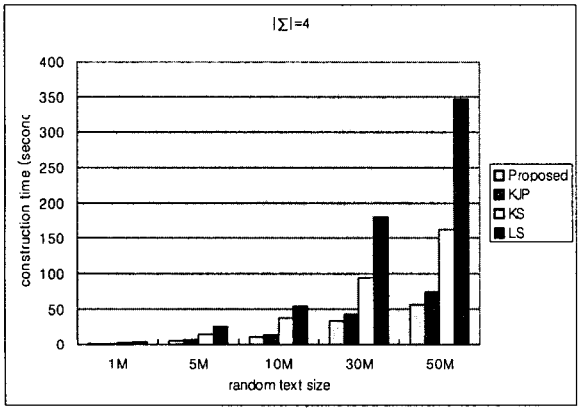
3. 2. 실험 결과

[그림 1]과 [그림 2]는 문자집합의 크기가 2와 4인 랜덤 스트링에 대한 써픽스 배열 구축 속도를 측정한 결과이다. LS는 Larsson and Sadakane, KS는 Kärkkäinen and Sanders, KJP는 Kim et al., Proposed는 본 논문에서 제안한 방법을 사용한 알고리즘을 의미한다.

Larsson and Sadakane 알고리즘은 알고리즘의 시간복잡도의 영향이 나타나 텍스트가 커질수록 구축 속도가 느려진다. 랜덤 스트링의 경우 최장 공통 접두사(longest common prefix)의 평균값이 크기 때문에 Larsson and Sadakane 알고리즘은



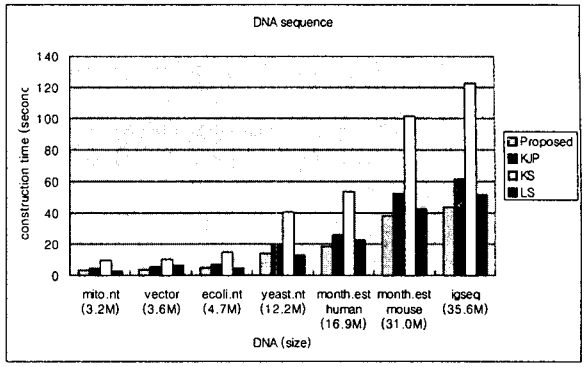
[그림 1] |Σ|=2인 랜덤 스트링에 대한 써픽스 배열 구축 속도



[그림 2] |Σ|=4인 랜덤 스트링에 대한 써픽스 배열 구축 속도

느리다. Kärkkäinen and Sanders 알고리즘은 선형 시간 알고리즘이지만 $O(n\log\log n)$ 시간 알고리즘인 Kim et al. 알고리즘보다 느렸다. 그 이유는 $\log\log n$은 실제 데이터에서 매우 작은 값이므로 시간 복잡도 $O(n)$에 숨은 상수와 비교할 수 있기 때문이다. 본 논문에서 제안한 방법을 사용한 경우 Kim et al. 알고리즘보다 더 빠르게 써픽스 배열을 구축하였으며, 텍스트의 크기가 작은 일부 경우를 제외하면 가장 빠르게 써픽스 배열을 구축하였다.

[그림 3]은 DNA 스트링에 대한 써픽스 배열 구축 속도를 측정한 결과이다. Kärkkäinen and Sanders 알고리즘이 써픽스 배열을 가장 느리게 구축하는 것을 볼 수 있는데 이것은 3개의 문자를 인코딩하기 때문에 시간 복잡도 $O(n)$에 숨은 상수가 크기 때문일 것이다. 몇몇 DNA 스트링의 경우 $O(n\log n)$ 알고리즘인 Larsson and Sadakane 알고리즘이 DNA 스트링에 대한 써픽스 배열을 가장 빠르게 구축하였다. 그 이유는 이런 DNA 스트링은 랜덤 스트링과는 달리 써픽스들의 최장 공통 접두사의 평균값이 작아서 써픽스의 처음 몇 개의 문자들만을 정렬하면 전체 써픽스를 정렬할 수 있기 때문이다. 본 논문에서 제안한 알고리즘은 Kim et al. 알고리즘보다 더 빠르게 써픽스 배열을 구축하고 있고, DNA 스트링의 크기가 커질수록



[그림 3] DNA 스트링에 대한 써픽스 배열 구축 속도

써픽스 배열을 빠르게 구축한다. 본 논문에서 제안한 방법이 전체적으로 가장 좋은 성능을 보이고 있다.

4. 결 론

본 논문에서는 문자집합의 크기가 작은 경우 Kim et al. 알고리즘의 인코딩과 합병을 개선하는 방법을 제안하였다. 이 방법을 이용하여 써픽스 배열을 구축한 결과 Kim et al. 알고리즘에 비해 구축 시간이 70% 정도로 단축되었다. 다른 알고리즘들과 비교한 경우에도 랜덤 스트링과 DNA 스트링에 대한 실험에서 가장 빠르게 써픽스 배열을 구축하였다.

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] D. Kim, J. Jo, H. Park, A fast algorithm for constructing suffix arrays for fixed-size alphabet, Workshop on Efficient and Experimental Algorithms, 301-314, 2004.

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

[7] N. Larsson and K. Sadakane, Faster suffix sorting, Manuscript, 1999.

[8] S. Lee and K. Park, Efficient implementations of suffix array construction algorithms, Australasian Workshop on Combinatorial Algorithms, 64-72, 2004.

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

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

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

[12] 조준하, 박희진, 김동규, DNA 스트링에 효율적인 써픽스 배열 구축 알고리즘, 정보과학회 학술지, 봄 학술발표논문집(A) 31권, 1호, 961-963, 2004.