선형 시간 접미사 배열 생성 알고리즘들의 비교

이성림° 박근수
서울대학교 전기,컴퓨터공학부
{sllee°, kpark}@theory.snu.ac.kr

Comparison of Linear Time Suffix Array Construction Algorithms

Sunglim Lee° Kunsoo Park
School of Computer Science & Engineering, Seoul National University

요 약

접미사 배열은 긴 문자열에 대해 효율적인 문자열 검색을 가능하게 하는 자료구조이다. 접미사 배열은 문자열의 접미사들의 사전식 정렬순서를 배열로 저장한다. 비슷한 효과를 가진 접미사 트리에 비해서 접미사 배열은 저장 공간을 적게 차지하기 때문에 생명정보과학의 염기 서열 등 큰 크기의 문자열의 처리에 더욱 유리하다. 본 논문에서는 2003년에 발표된 Ko-Aluru, K□rkk□inen-Sanders 및 기존의 Manber-Myers 등 세 개의 접미사 배열 생성 알고리즘들의 염기 서열 입력 자료에 대한 실행 시간 및 기억 장치 사용량을 실험을 통해 비교한다. 특히 Ko-Aluru와 K□rkk□inen-Sanders 알고리즘은 실행 시간 및 저장 공간의 이론적인 복잡도가 $O(n)$으로 동일하기 때문에 실험을 통해서 계산 복잡도에 숨어있는 상수를 비교한다. 실험 결과 K□rkk□inen-Sanders 알고리즘이 가장 효율적임을 보인다.

1. 서 론

1.1. 접미사 배열

접미사 배열[1]은 접미사 트리[2,3]와 함께 대표적인 문자열 검색을 위한 인덱스 자료 구조이다. 특히 최근 생명정보과학의 염기 서열이나 방대한 양의 웹 html문서의 검색 등에서 매우 긴 길이의 문자열에 대한 검색이 필요하게 됨에 따라, 접미사 트리에 비해 구조가 단순하고 기억 공간를 적게 쓰는 접미사 배열이 주목을 받게 되었다. 문자열의 길이가 n일 때 접미사 트리의 생성은 $O(n)$에 가능했으나, 접미사 배열의 생성 시간은 최근까지 $O(n\log n)$으로 접미사 트리보다 시간 복잡도가 높은 것으로 알려져 있었다. 최근에 발표된 Ko-Aluru[4], Kärkkäinen-Sanders[5] 및 Kim-Sim-Park-Park[6]의 선형 시간 접미사 배열 생성 알고리즘은 $O(n)$의 생성 시간 및 저장 공간을 사용한다.

1.2. Ko-Aluru 접미사 배열 생성 알고리즘

Ko-Aluru 알고리즘[4]은 길이 n 인 정수 알파벳(integer alphabet)의 문자열에 대해 최대 $O(n)$ 시간 및 $O(n)$ 공간을 사용해서 접미사 배열을 생성한다. Ko-Aluru 알고리즘은 대상 문자열을 1 이상의 길이를 갖는 부분 문자열들로 나누고 각 부분 문자열을 한 문자의 새로운 문자로 인코딩(encoding) 한다. 인코딩에 의해 본래 문자열에 비해 짧은 길이를 갖는 인코딩 된 문자열에 대한 접미사 배열을 재귀적으로 생성한 후 이 접미사 배열을 이용해서 본래 문자열의 접미사 배열을 구한다. 인코딩 된 문자열의 길이는 항상 본래 문자열의 길이의 1/2 이하이기 때문에, 재귀적 처리를 포함한 전체 시간 및 공간이 $O(n)$이 가능하다.

1.3. Kärkkäinen-Sanders 접미사 배열 생성 알고리즘

Kärkkäinen-Sanders 알고리즘[5]도 Ko-Aluru의 알고리즘과 마찬가지로 길이 n 인 정수 알파벳(integer alphabet)의 문자열에 대해 최대 $O(n)$ 시간 및 $O(n)$ 공간을 사용해서 접미사 배열을 생성한다. Kärkkäinen-Sanders의 알고리즘의 전체적인 형태는 인코딩과 재귀적 처리를 한다는 점에서 Ko-Aluru 알고리즘과 비슷하다. Ko-Aluru 알고리즘과의 차이점은 인코딩 방식과 그에 따른 인코딩 된 접미사 배열로부터 본래 접미사 배열을 생성하는 방법에 있다. Kärkkäinen-Sanders의 경우 항상 3 문자 길이의 부분 문자열 단위로 인코딩 하기 때문에 인코딩 된 문자열의 길이가 본래 문자열의 길이의 정확히 2/3으로, Ko-Aluru의 1/2 이하에 비해 크다. 이에 따라 재귀 함수가 불려지는 회수는 Kärkkäinen-Sanders의 알고리즘이 Ko-Aluru의 알고리즘에 비해 항상 많게 된다. 이 점에서 Ko-Aluru의 알고리즘이 더 효율적일 수 있을 것으로 보이나, 본 논문에서는 실험을 통해 이 점이 별 영향을 미치지 않음을 보인다.

1.4. Kim-Sim-Park-Park 접미사 배열 생성 알고리즘

Kim-Sim-Park-Park 알고리즘[6]도 Ko-Aluru 알고리즘이나 Kärkkäinen-Sanders 알고리즘과 마찬가지로 인코딩 된 문자열을 재귀적으로 처리해서 길이 n 인 정수 알파벳(integer alphabet)의 문자열에 대해 최대 $O(n)$ 시간 및 $O(n)$ 공간을 사용해서 접미사 배열을 생성한다. Kim-Sim-Park-Park 알고리즘은 입력 문자열을 even 문자열 및 odd 문자열로 나누어 한 쪽을 인코딩해서 재귀적으로 접미사 배열을 생성하므로, 인코딩 된 문자열의 길이가 본래 문자열의 길이의 정확히 1/2이다. Kim-Sim-Park-Park 알고리즘은 Ko-Aluru 알고리즘이나 Kärkkäinen-Sanders 알고리즘에 비해 복잡하고 비효율적이므로 본 논문의 비교에서는 제외한다.

1.5. Manber-Myers 접미사 배열 생성 알고리즘

Manber-Myers 알고리즘[1]은 일반 알파벳(general alphabet)의 문자열에 대해 최대 $O(n\log n)$ 시간 및 $O(n)$ 공간을 사용해서 접미사 배열을 생성한다. 대상 문자열의 특성에 따라 최악의 경우 $O(n\log n)$ 시간이 소요되지만, 평균적인 입력을 가정할 경우 평균 $O(n)$ 시간이 걸린다.[1] 본 논문에서는 Manber-Myers 알고리즘의 최대 실행 시간이 $O(n\log n)$으로 최대 실행 시간이 $O(n)$ 인 Ko-Aluru 알고리즘 및 Kärkkäinen-Sanders 알고리즘에 비해 이론적으로 불리하지만, 생명정보과학의 염기 서열을 입력 문자열로 하는 실험 결과 Manber-Myers 알고리즘이 평균 실행 시간인 $O(n)$에 근접하면서 여전히 실용적으로 효과적인 알고리즘 임을 보인다.

1.6. 본 실험의 의미

Ko-Aluru, Kärkkäinen-Sanders 및 Manber-Myers의 세 접미사 배열 생성 알고리즘은 위 설명과 같이 최대 혹은 평균 $O(n)$ 시간과 최대 $O(n)$ 공간을 사용한다. 동일한 이론적 복잡도를 가지고 있기 때문에 실험을 통해서 상수 배 이내에서 세 알고리즘의 공간 및 시간 효율성을 비교한다.

접미사 배열 생성 알고리즘의 상수 배의 공간 및 시간 소비의 차이도 현재 생명정보과학의 염기 서열 처리 등에서는 중요하다. NCBI에 등록되어 있는 염기 서열들을 보면 수 십 메가바이트에 이르는 것들을 볼 수 있는데, 현재 상용 컴퓨터의 메모리가 보통 수 천 메가바이트 정도임을 감안할 때 접미사 배열이 입력 문자열에 대해 바이트당 10 바이트를 쓰는 지 20 바이트를 쓰는 지는 관련 소프트웨어의 개발에서 중요한 요소가 된다.

2. 실험 설정

2.1. 알고리즘의 소스 코드

알고리즘은 모두 ANSI C/C++ 프로그래밍 언어로 구현되어 있다. 가장 효율적인 구현을 보장하기 위해 각 접미사 배열 생성 논문 저자의 소스 코드를 이용했고, 가능하지 않은 경우에는 본 논문에서 직접 구현했다.

각 알고리즘의 소스 코드에 대한 정보는 아래와 같다.

■ Ko-Aluru 알고리즘

본 논문에서 직접 구현했다. 설계상의 작업 공간(working-space)은 입력 문자열의 길이가 n 일 때, 약 40n 바이트 가량이다. 공간보다 속도 중심으로 구현하여 여러 번 사용되는 자료구조를 매 번 다시 계산하지 않고 한 번 계산해서 공간을 차지하고 있다. Ko-Aluru의 논문에 명시되어 있지 않은 부분 문자열의 정렬은 부록에 설명한 방법으로 구현되었다.

■ Kärkkäinen-Sanders 알고리즘

Kärkkäinen-Sanders 접미사 배열 생성 알고리즘의 저자의 하나인 Sanders의 소스 코드를 사용하였다. 그 소스 코드는 Kärkkäinen-Sanders 논문에서 지정한 인터넷 URL에 게시되어 있다.[1]

■ Manber-Myers 알고리즘

Manber-Myers 접미사 배열 생성 알고리즘의 저자들인 Manber과 Myers가 1990년 작성한 소스 코드를 이용하였다.[2] 접미사 배열의 검색 속도를 높이기 위해 사용하는 LCP(Longest Common Prefix)의 생성을 병행하는 code.c 소스 코드와 LCP 생성을 하지 않는 buck.c 소스 코드 중 buck.c를 사용하였다. 본 실험에서는 LCP를 제외한 접미사 배열의 생성 시간 및 공간만을 비교하기 때문이다.

2.2. 입력 자료

입력 자료는 NCBI에 등록되어 있는 생물의 염기 서열을 이용하였다. 세균이나 인간의 염색체 등의 시퀀싱이 완료된 자료를 이용하였다. 알파벳 크기는 염기 서열에서 A, G, C, T에 해당하는 4개 이고, 길이는 최대 16 메가 바이트이다.

2.3. 작동 환경

실험에 사용된 운영체제는 LINUX (RedHat 7.0)이고 프로세서는 Intel Pentium4 Xeon 2.4GHz Dual에 주 기억 장치는 2Gbyte인 하드웨어가 사용되었다. 모든 실험에서 프로그램이 항상 주 기억 장치에서만 실행됨을 확인하였다. 본 실험은 보조 기억 장치를 고려하지 않고, 임의 접근이 가능한 주 기억 장치에서의 알고리즘의 동작을 비교한다.

2.4. 측 정

알고리즘의 실행 시간은 입력 파일을 읽어 들이는 시간과 결과를 출력하는 시간을 제외한 알고리즘 수행 시간을 측정하였다. 소스 코드 내부에 운영 체제의 시스템 시간을 읽는 코드를 추가하여 시간을 측정하였다. 시스템 시간의 정밀도는 0.001초이다. 5번 이상씩 측정하여 일정한 값이 나오는 것을 확인하였다. 시간이 높게 나오는 측정치는 멀티태스킹의 영향으로 보고 무시하였다.

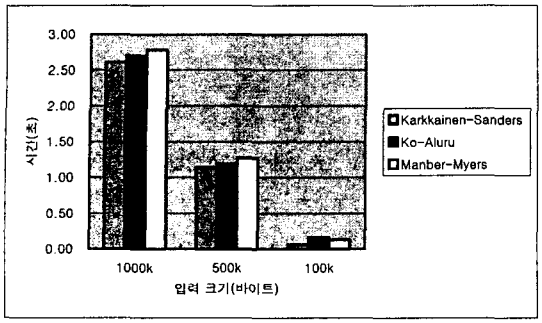
3. 실험 결과

3.1. 시간 및 공간 비교

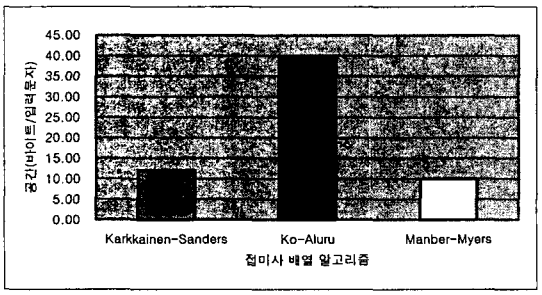
실험 결과 [그림 1]과 같이 세 개의 알고리즘의 실행 시간이 큰 차이가 나지 않는 가운데, Kärkkäinen-Sanders 알고리즘이 가장 빨랐다. [그림 2]의 기억 공간 비교에서 보듯이 Ko-Aluru 알고리즘의 경우 속도 최적화를 위해 기억 공간을 많이 차지하고, Manber-Myers 알고리즘의 경우 평균 시간은 $O(n)$ 이지만, 이론적 최대 시간이 $O(n\log n)$으로 큰 것을 감안하면, Kärkkäinen-Sanders 알고리즘이 가장 효과적인 접미사 배열 생성 알고리즘이라 할 수 있다.

[1] http://www.mpi-sb.mpg.de/~sanders/programs/suffix/

[2] Copyright (c) 1990 Arizona Board of Regents for the University of Arizona, Authors: Gene Myers & Udi Manber



[그림 1] 알고리즘의 실행 시간 비교



[그림 2] 알고리즘의 사용 저장 공간 비교

3.2. Sanders의 재귀 회수 단축 기법

Sanders는 Kärkkäinen-Sanders 알고리즘을 구현할 때 알파벳의 크기가 문자열 크기와 같아지면 더 이상 재귀적으로 처리하지 않고 단순 정렬로 재귀를 끝냄으로써 재귀 회수를 줄이는 기법을 이용하였다. 이는 알고리즘이 재귀적으로 불릴 때 마다 알파벳의 크기가 기하급수적으로 증가한다는 점과, 알파벳의 크기와 문자열의 크기가 같아지면 문자열의 각 문자가 유일하기 때문에 문자열에 대한 단순한 사전순 정렬로 접미사 배열을 구할 수 있다는 점을 이용한 것이다. 실제로 이 기법을 적용하면 1000k 데이터의 경우 재귀 함수 호출 회수가 17회에서 4회로 크게 줄어들어 Kärkkäinen-Sanders 알고리즘의 경우 전체 속도가 20% 가량 빨라진다. 하지만 Ko-Aluru 알고리즘의 경우는 이 기법을 적용해도 3% 미만의 차이 밖에 나지 않는데, 이는 Kärkkäinen-Sanders의 인코딩이 문자열을 2/3으로 줄이는 반면 Ko-Aluru는 1/2 이하로 줄이는 데서 오는 차이이다. (0.66^4=0.20, 0.5^4=0.06)

4. 결 론

가장 효과적인 선형 시간 접미사 배열 생성 알고리즘은 Kärkkäinen-Sanders의 알고리즘이다.

인코딩에 의한 문자열 축소 비율이 1/2 미만인 Ko-Aluru의 알고리즘보다 2/3인 Kärkkäinen-Sanders 알고리즘이 더 효율적인 것에서도 알 수 있듯이, Ko-Aluru나 Kärkkäinen-Sanders 알고리즘과 비슷한 부류의 인코딩과 재귀적 처리를 이용한 선형 접미사 배열 생성 알고리즘의 경우 인코딩 비율에 따른 재귀 함수 호출 회수는 성능에 영향을 거의 미치지 못하며, 인코딩 및 머지(merge) 과정이 얼마나 단순하게 처리되는지가 압도적으로 중요한 요인이 된다.

참고 문헌

[1] Udi Manber, Gene Myers, "Suffix array: A new method for on-line string searches", In Proc. of the 1st ACM-SIAM Symposium on Discrete Algorithms, 319-327, 1990.

[2] E. McCreight, "A space-economical suffix tree construction algorithm", Journal of the ACM, 23(2):262-272, 1976.

[3] E. Ukkonen, "On-Line Construction of Suffix Trees", Algorithmica, 14(3):249-260, 1995.

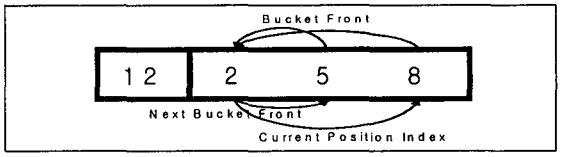
[4] Pang Ko and Srinivas Aluru. "Space efficient linear time construction of suffix arrays", In Proc. 14th Annual Symposium on Combinatorial Pattern Matching. Springer, June 2003.

[5] Juha Kärkkäinen and Peter Sanders. "Simple linear work suffix array construction", In Proc. 13th International Conference on Automata, Languages and Programming. Springer, 2003.

[6] D. K. Kim, J. S. Sim, H. Park, and K. Park. Linear-time construction of suffix arrays. In Proc. 14th Annual Symposium on Combinatorial Pattern Matching. Springer, June 2003

부록. 가변 길이 부분 문자열 정렬 구현

Ko-Aluru 접미사 배열 생성 알고리즘에서는 입력 문자열을 가변 길이의 부분 문자열로 나누어 각 부분 문자열을 하나의 문자로 인코딩 하는 데, 이 때 가변 길이 부분 문자열들을 사전순으로 정렬할 필요가 있다. Ko-Aluru의 논문에는 한 문자 씩 반복적으로 버킷팅(bucketing) 한다[1] 라고 간단히 나와 있어서, Ko-Aluru 논문 내용을 바탕으로 [그림 3]과 같이 세 종류의 링크를 갖는 배열을 이용해서 구현하였다.



[그림 3] 부분 문자열 정렬을 위한 구조

알고리즘이 진행되는 동안 버킷(Bucket)이 점점 세분화 되어 새로운 버켓 경계가 생겨나기 때문에, 각 원소가 자신이 속한 버켓의 가장 앞 원소를 가리키는 링크를 가지고 있으며, 가장 앞 원소는 새로운 버켓 경계를 가리키는 링크를 가지고 있어 참조될 때 마다 각 원소의 링크를 갱신해준다. 버켓 내의 위치를 가리키는 인덱스도 버켓의 가장 앞 원소가 가지고 있다.

[1] Ko-Aluru 논문[4] 6쪽 아래에서 2번째 문단 참조