DNA Fragment Assembly에서 k-글자 테이블의 정렬

홍순철[0] 박근수
서울대학교 전기.컴퓨터공학부
(schong[0], kpark)@theory.snu.ac.kr

Sorting k-mer Table in DNA Fragment Assembly

Sun-Cheol Hong[0] Kunsoo Park
School of Computer Science & Engineering
Seoul National University

요 약

DNA fragment assembly 프로그램인 Phrap에서는 exact match를 찾기 위해 정렬된 k-글자 테이블 자료구조를 사용한다. 이것은 접미사 배열의 간단한 형태로서, DNA fragment assembly와 같은 응용에서는 접미사 배열보다 더 유용한 자료구조이다. 본 논문에서는 k-글자 테이블을 정렬하는 Manber-Myers, Quicksort, Radix sort 알고리즘을 살펴보고, 실험을 통해 그 중에서 가장 뛰어난 성능을 가지는 것이 Quicksort 알고리즘임을 보였다. 또한 k-글자 테이블의 정렬 문제에 있어서는, 캐쉬-메모리 아키텍쳐에 최적화되어 계산복잡도 속에 숨어있는 상수를 최소화 하는 것이 중요한 문제임을 밝힌다.

1. 서 론

Udi Manber, Gene Myers에 의해 제안된 접미사 배열[1]은 접미사 트리[2, 3]에 비해 간단한 구조와 적은 메모리 사용량으로 널리 사용되고 있다. 그러나 상황에 따라서는 접미사 배열보다 좀더 단순한 자료구조를 사용하는 것이 적합한 경우도 있다.

예를 들어, 널리 알려진 DNA fragment assembly 프로그램인 Phrap의 경우를 보면 DNA 염기열 조각들 사이의 exact match를 찾기 위해서, 접미사 배열 대신 두 접미사의 사전상 우선순위를 결정하는데 있어 단지 k 글자까지만 비교하는 정렬된 k-글자 테이블을 사용하고 있다.[4]

Phrap에서는 서로 유사한 염기열 조각들을 겹쳐 나감으로써 DNA fragment assembly를 수행한다. 그런데 모든 염기열 조각 쌍에 대해서 유사한 정도를 계산한다는 것은 비현실적이다. 그래서 Phrap에서는 염기열 조각 쌍에서 일정 길이의 exact match를 가지는 부분을 중심으로 banded Smith-Waterman 알고리즘을 사용하게 된다.[4] 이때 이 일정 길이(k) 이상의 exact match를 손쉽게 찾기 위해서 정렬된 k-글자 테이블 자료구조가 사용된다. 정렬된 k-글자 테이블 자료구조가 완성되면, 앞부분 k-글자가 동일한 접미사들이 서로 인접한 위치에 나타나게 되므로, 이런 접미사들이 텍스트에서 어디에 위치하여 어느 염기열 조각의 부분인지만 확인하면 염기열 조각 쌍 사이의 exact match 부분을 쉽게 찾을 수 있는 것이다. (실제로 Phrap에서는 단순히 exact match 부분의 길이 뿐만 아니라 exact match를 이루는 염기의 분포 상황도 고려하기 때문에 위 설명보다는 복잡한 처리를 한다.)

● 본 연구는 농업생명공학연구원 IMT-2000 출연금 정보통신 선도기술개발사업의 지원을 받았음.

정렬된 k-글자 테이블 자료구조는 접미사 배열 자료구조의 간단한 형태이므로 당연히 위에서 설명한 목적을 위해 접미사 배열 자료구조를 그냥 사용하는 것도 가능하다. 그러나 보통 DNA fragment assembly 문제에서 주어지는 텍스트의 길이는 매우 길고 반대로 k 값은 텍스트 길이에 무관하게 작은 값(Phrap에서는 k=30)으로 주어진다는 것을 고려하면, 뒤에서 설명하는 $O(Nlogk)$나 $O(kN)$의 계산복잡도를 가지는 정렬된 k-글자 테이블 자료구조 생성 알고리즘을 사용하는 것이 통상 $O(NlogN)$의 계산복잡도를 가지는 접미사 배열 생성 알고리즘을 사용하는 것에 비해서 현실적으로 매우 유리하게 된다.

접미사 배열을 만드는 알고리즘은 약간의 수정을 통해서 간단히 정렬된 k-글자 테이블을 생성하는 알고리즘으로 변경이 가능하다. 그러나 문제 상황에 맞는 간단한 방법이 높은 성능을 보여줄 수도 있다. 본 논문에서는 여러 가지 방법을 살펴본 후 결과를 분석하고 최적의 알고리즘을 제시하고자 한다.

2. 문제 정의

텍스트 T는 알파벳 Σ = {'A', 'C', 'G', 'T', '-'} 이루어져 있다. 'A', 'C', 'G', 'T'는 각각 아데닌, 시토신, 구아닌, 티민의 DNA 염기를 나타내는 것이고, '-'은 각각의 염기열 조각들을 구분하기 위해서 사이사이에 들어가는 구분기호로써 편의상 텍스트의 맨 첫 글자와 마지막 글자도 '-'으로 둔다. 간단히 예를 들어 'ACCTACGGACGG'인 염기열을 가지고 실험하여 'ACCTA', 'TACGGA', 'CTACG', 'ACGG' 이렇게 4개의 염기열 조각을 읽어냈다면, 주어지는 텍스트는 '-ACCTA-TACGGA-CTACG-ACGG-'가 된다. (실제 Phrap에서 사용되고 있고 본 논문의 실험에 구현된 것은 C 프로그래밍 언어에서 null 문자를 나타내는 '\0'이지만, 여기서는 편의상 '-'로 표시했다.)

본 논문에서는 알파벳 '-'을 다른 알파벳 'A', 'C', 'G', 'T'와 동등하게 취급하며, 알파벳의 사전상 우선순위는 '-' < 'A' < 'C' < 'G' < 'T' 이다. (Phrap에서는 '-'을 특수한 기호로 취급하여 앞에서 14글자 이내에 '-'이 존재하는 접미사는 정렬된 k-글자 테이블 자료구조에서 제외시켜 버린다. 그 이유는 염기열 조각의 짧은 끝부분은 exact match를 고려할 필요 자체가 없기 때문이다. 이에 반해 본 논문에서는 '-'을 역시 다른 염기 알파벳과 동등하게 간주하는 방식을 택했다. 그 이유는, DNA fragment assembly 문제에서 통상적으로 하나의 염기열 조각의 길이는 대략 평균 500 bp(base pair), 최대 900 bp 정도이므로[5] '-' 알파벳이 전체 텍스트에서 차지하는 비율이 극히 적어 전체 수행시간이나 추후 응용 과정에 큰 영향을 미치지 않을 것이기 때문이다. 또한 Phrap에서의 방식을 취하면, 여러 최적화 기법을 적용하거나 외부 메모리 알고리즘으로 확장할 경우에 있어 복잡한 문제를 야기시킬 가능성이 있다.)

● k-글자 테이블 정렬 문제

길이 N의 텍스트 $T[1...N]$와 정수 k가 주어졌을 때, 정수 배열 $S[1...N]$을 생성한다. 이때 $1 \le i < j \le N$인 임의의 정수 i, j에 대해서, T의 접미사 $T_{S[i]}$와 $T_{S[j]}$는 항상 $T_{S[i]} \le_k T_{S[j]}$의 관계를 만족한다. ($\le_k$ 기호는 두 접미사에서 앞부분 k 글자까지만 비교하였을 때의 사전상 우선순위 관계를 의미한다.)

3. 알고리즘

● Manber-Myers

Manber와 Myers가 [1]에서 제시한 접미사 배열 생성 알고리즘은 H=1부터 시작하여 매 단계에서 H 값을 2 배씩 증가시키면서, 접미사들 간에 앞에서 H 글자까지의 사전상 우선순위가 결정된 Sorted H-mer table을 유지시켜 나간다. H 값을 N 이상이 될 때까지 $O(logN)$ 단계를 반복하면 완전한 접미사 배열 자료구조를 얻을 수 있다. 따라서 H 값이 k 이상이 되는 단계까지만 수행하면 정렬된 k-글자 테이블 자료구조를 얻을 수 있고, 이 알고리즘의 계산복잡도는 $O(Nlogk)$가 된다. 원래의 알고리즘에서는 처음 Bucket sort를 사용하는데, 여기서는 다른 알고리즘들과의 간편한 비교를 위해 똑같이 Counting sort를 사용하도록 변경되었다. 본 논문의 모든 Counting sort는, 알파벳이 5가지이므로 3bit로 6글자씩 묶어 $[0, 2^{18}-1]$의 범위를 가지는 정수로 간주하여 처리하므로, Manber-Myers 알고리즘은 H 값이 6에서부터 시작하여 2 배씩 증가한다.

● Quicksort

[6]에서 제시한 변형 Quicksort 접미사 배열 알고리즘을 k-글자 테이블 정렬 알고리즘으로 수정한 것으로, Phrap에서 현재 사용하고 있는 방법이다. 한번의 Counting sort로 접미사 앞의 6 글자에 대해서 일단 정렬하고, 그 결과에 의해 블록 단위로 다시 Quicksort를 사용하여 나머지 길이까지 정렬하는 방식이다. Phrap에서는 블록에 속하는 접미사의 수에 따라서 다시 Insertion sort와 Quicksort로 차등 적용하는 최적화 부분을 추가로 포함하고 있고, 본 논문에서의 구현도 그것을 포함시키고 있는데 약 9%의 성능 향상이 있었다.

● Radix sort

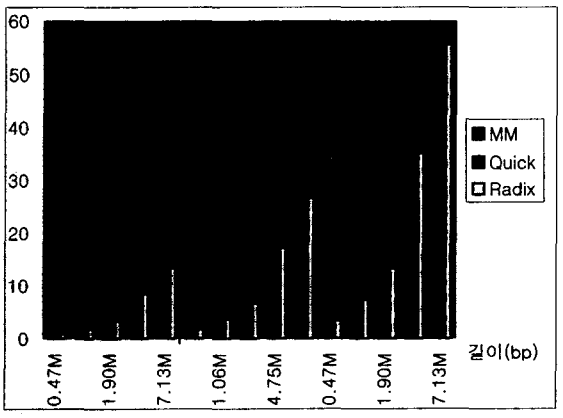
문제에서 알파벳의 수가 작고 k 값이 매우 작은 값으로 주어지는 것에 착안한 간단한 방법이다. Stable sort 방법을 사용하여 접미사의 뒤쪽 위치에서부터 비교하여 순서를 결정한다. 간단히 stable Counting sort를 사용하여 $\lceil k/6 \rceil$ 번의 단계를 수행하고, 계산복잡도는 $O(kN)$이다.

4. 실험결과 및 분석

본 실험에서는 실제 DNA fragment assembly 문제의 입력과 비슷하게 만들기 위해서, Whole-genome shotgun sequencing[5] 과정을 시뮬레이션으로 흉내내어 커버리지 x10로 만든 예제를 가지고 실험했다. 실험 환경은 펜티엄3 866Mhz x2, 1G 메모리의 워크스테이션이다.

[그림1]은 전체적인 실험결과가 표시되어 있다. 위 그림을 보면 길이가 0.47M, 1.06M, 1.90M, 4.75M, 7.13M인 텍스트 예제에 대해 각 경우에 3가지 알고리즘의 수행시간이 그래프로 그려져 있다. [그림1]의 왼쪽 부분은 k=30, 중간 부분은 k=60, 오른쪽 부분은 k=120 일 때의 실험 결과이다.

전체적으로 볼 때 Quicksort 알고리즘이 모든 경우에 있어 가장 뛰어난 성능을 보여주고 있다. Radix sort 알고리즘은 k=30 일 때에는 Quicksort 알고리즘에 비해 약간 낮지만 간단한 방식치고는 우수한 성능을 보여주고 있지만, k=60 일 때는 성능이 크 게 저하되었고, k=120 일 때는 가장 낮은 성능을 보여주었다. Manber-Myers 알고리즘은 계산복잡도 속의 큰 상수로 인해, k 값을 120 까지 증가시켰음에도 $O(Nlogk)$ 계산복잡도의 장점을 살리지 못하고, Quicksort 알고리즘에 비해 2 배 이상의 수행시간이 걸리고 있다.

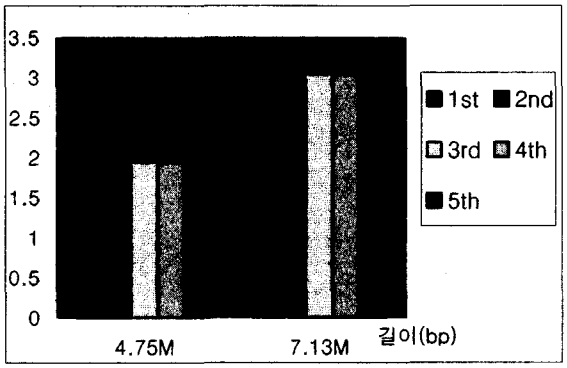


[그림1] 전체 수행결과

사실 DNA fragment assembly에서 k=120의 값은 다소 비현실적인 것으로 볼 수 있으나, 보다 일반적인 문제의 상황을 고려하고, 각 알고리즘의 성질을 뚜렷이 파악하기 위해서 실험하였다. 비교적 큰 k 값에도 불구하고 Quicksort 알고리즘이 Manber-Myers 알고리즘보다 우월한 성능을 보이고 있는 것은, 접미사 배열 알고리즘으로서의 Quicksort 알고리즘이 Manber-Myers 알고리즘을 포함하여 심지어 접미사 트리 알고리즘 보다도 뛰어난 성능을 보여주었다는 주장[6]을 설득력 있게 만들어 준다.

[그림2]는 Radix sort 알고리즘에서 (k=30일 때) 각 stable Counting sort 단계의 수행시간을 나타내고 있는데, 이 결과는 중요한 점을 시사하고 있다. 두 번째 이후 stable Counting sort 단계에서부터, 정렬된 k-글자 테이블에 나타난 위치 순서대로 접미사들을 액세스하는 것은 높은 빈도의 랜덤 메모리 I/O를 발생시켜 캐쉬-메모리 아키텍쳐에서 효율성을 떨어뜨려 수행시간이 크게 증가했음을 잘 나타내고 있는 것이다. [그림1]에서

k 값이 증가함에 따라 Radix sort의 성능이 급격히 낮아진 것은 [그림2]에서 직접적인 원인을 찾을 수 있으며, Manber-Myers 알고리즘이 실망스러운 성능을 보이는 것과 또 반대로 작게 나눠진 블록 단위로 정렬을 수행하는 Quicksort 알고리즘이 가장 뛰어난 성능을 보이는 것도 비슷한 맥락에서 설명할 수 있다.



[그림2] Radix sort의 각 단계별 수행시간 (k=30)

5. 외부 메모리 알고리즘으로의 확장

지금까지는 모두 내부 메모리만으로 정렬된 k-글자 테이블을 생성 가능한 경우로 한정했다. 이것은 아주 긴 길이의 텍스트를 다루는 문제 상황을 제대로 반영하지 못한 것 같지만 그렇지는 않다. 왜냐하면, 현재 하드웨어 기술의 발전으로 기가급의 메모리를 장착하여 100M 정도 길이의 텍스트를 내부 메모리만 사용하여 처리 가능하게끔 하는 것이 현실적으로 어렵지 않고, 또한 외부 메모리 접미사 배열 생성 알고리즘인 BaezaYates-Gonnet-Snider 알고리즘[7]에서처럼 일정 길이의 텍스트로 나누어 정렬하고 그 결과를 점진적으로 합치는 방법을 통해 외부 메모리 알고리즘으로 쉽게 확장 가능하다.

BGS와 같은 방식으로 외부 메모리 알고리즘으로 확장하게 된다면, 메모리에 읽어 들인 텍스트 부분을 정렬하는 내부 정렬 알고리즘의 성능도 중요하지만, 내부 정렬 알고리즘이 사용하는 메모리 양이 더욱 중요하게 된다. 이 메모리 양에 의해 결국 한번에 정렬할 수 있는 텍스트 부분의 최대 길이가 결정되고, 이 길이가 크면 클수록 그만큼 결과를 점진적으로 합치는 과정의 횟수가 줄어들기 때문이다. 특히 이 과정은 주로 디스크 I/O와 관계되기 때문에, 이 과정의 횟수에 따라 전체 수행시간이 크게 좌우되게 된다.

BGS와 같은 방식으로 외부 메모리 알고리즘으로 확장하게 되는 경우에 있어서도 Quicksort 방법이 가장 유리하다. 처음의 Counting sort를 위한 1M byte 정도의 메모리와 k-글자 테이블을 저장하기 위한 $4N$의 메모리만을 요구하기 때문이다

6. 결론

정렬된 k-글자 테이블 생성 문제는 접미사 배열 생성 문제의 간단한 형태로, DNA fragment assembly와 같은 긴 길이의 텍스트를 다루는 경우에 있어서는 캐쉬-메모리 아키텍처를 고려하여 계산복잡도 속의 상수 값을 최소화하는 것이 중요한 문제임을 지적하고, Quicksort 알고리즘이 그에 맞추어 뛰어난 성능을 보임을 실험으로 밝혔다.

또한 외부 메모리 알고리즘으로의 확장하는 간단한 방법을 제시하고, 그 경우에 있어서도 Quicksort 알고리즘이 유리함을 밝혔다.

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] *Phrap Document & Phrap Source Code*

[5] Gene Myers, "Whole-Genome DNA Sequencing", 1999.

[6] M. Burrows, D.J. Wheeler, "A Block-sorting Lossless Data Compression Algorithm", Tech. Report 124, 1994.

[7] Andreas Crauser, Paolo Ferragina, "On Constructing Suffix Arrays in External Memory", ESA, 224-235, 1999.