스트링 B-트리를 이용한 염기 서열의 k-mer 분석 시스템 구현

최정현, 진희정, 조환규
부산대학교 전자계산학과
e-mail: {jhchoi, hjjin, hgcho}@pearl.cs.pusan.ac.kr

Implementation of k-mer Analysis System for DNA Sequence Using String B-Tree

Jeong-Hyeon Choi, Hee-Jeong Jin, and Hwan-Gue Cho
Department of Computer Science
Pusan National University

요 약

최근 Human Genome Project(HGP)에서 사람의 염기 서열의 초안이 발표되었다. 생물체의 염기 서열을 분석하는 방법은 매우 많은데, 그 중 하나가 k-mer 분석이다. k-mer는 유전자의 염기 서열내의 길이가 k인 연속된 염기서열이고, k-mer 분석 방법은 염기서열이 가진 k-mer들의 빈도의 분포나 대칭성 등을 탐색하는 것이다. 그런데 유전자의 염기 서열은 대용량 텍스트이고 k가 클 때 기존의 온메모리 알고리즘으로는 처리가 불가능하므로 효율적인 자료구조와 알고리즘이 필요하다.
본 논문에서는 패턴 일치(pattern matching)에 적합하고 외부 메모리를 지원하는 스트링 B-트리(string B-tree)를 이용한 k-mer 분석 방법을 제시하고, 그것을 구현하였으며 몇 가지 실험 결과에 대하여 기술한다.

1 서론

생명 공학의 빠른 발전으로 많은 생물체의 전체 염기 서열이 밝혀지고 있다. 최근 Human Genome Project에서 사람의 염기 서열의 초안을 발표되었다[3]. 염기 서열의 기능이나 진화적 관계를 밝히기 위해서는 염기 서열을 비교 분석해야 한다. 많은 염기 서열 분석 방법 중 하나가 k-mer 분석 방법이다[6]. k-mer는 유전자의 염기 서열내의 길이가 k인 연속된 염기이고, k-mer 분석 방법은 염기서열이 가진 k-mer들의 빈도의 분포나 대칭성 등을 탐색하는 것이다.

k-mer 분석을 위해서는 모든 k-mer의 빈도를 구해야 하는데 이것은 전형적인 패턴 일치(pattern matching) 문제이다. 즉 전체 염기 서열이 텍스트이고 하나의 k-mer가 패턴이 되어 텍스트에서 패턴이 나타나는 횟수와 위치를 구하는 문제와 같다. 문자 일치에 대한 알고리즘으로는 Kruth-Morris-Pratt[8], Boyer-Moore[1], Karp-Rabin[7]의 알고리즘이 있다. 그러나 이 알고리즘들은 수행 시간이 하나의 k-mer에 대해 $O(m)$이 걸리고 존재할 수 있는 모든 k-mer의 개수는 4^k이므로 전체적으로 $O(4^k m)$이 걸린다.

염기 서열에서 k-mer들의 빈도를 가장 빠르게 구하는 방법은 Karp-Rabin[7] 알고리즘을 응용하는 방법이다. 전체 유전자 염기 서열을 S라 하고, 길이를 $|S| = m$이라 하면, $S = S[1..m]$이고, i번째 k-mer는 $S_i = S[i..i+k-1]$이다. 염기 A,C,G,T는 4진수의 숫자로 정의할 수 있는데 각각 0,1,2,3으로 정의한다. 즉 변환 함수를 g라 하면 $g(A)=0$, $g(C)=1$, $g(G)=2$, $g(T)=3$이다. i번째 k-mer를 숫자로 변환하는 함수는 다음과 같이 정의된다:

$$h(i) = g(S[i]) * 4^{k-1} + g(S[i+1]) * 4^{k-2} + ... + g(S[i+k-1])$$

즉 하나의 k-mer는 함수 h에 의해 4진수의 숫자로 변환된다. 전체 k-mer의 빈도를 위한 배열을 F라 하면 i번째 k-mer는 $F_{h(i)}$의 값을 증가시킨다. 염기 서열의 처음부터 끝까지 위의 작업을 반복함으로써 모든 k-mer의 빈도를 구할 수 있다. 이 알고리즘의 수행 시간은 $O(m)$이고 특정 패턴의 빈도에 대한 탐색 시간은 $O(1)$이다. 그러나 이 방법은 공간을 $O(4^k)$ 사용하는 단점이 있다. 실제로 12-mer는 256MB의 메모리가 필요하고, 13-mer 이상일 때는 메모리가 부족하여 수행할 수 없다.

이 방법보다 더 좋은 방법은 패턴 일치를 위한 서픽스 트리와 같은 인덱스 구조를 생성한 후 각 k-mer의 횟수를 구한 것이다. 서픽스 트리 생성 시간은 $O(m)$이고 문자 검색 시간은 $O(k)$이다[5]. 그러나 서픽스 트리의 공간 사용량은 $O(m)$이지만 사람의 염색체에 대한 염기 서열이 40MB 이상의 대용량이므로 메모리 부족으로 수행할 수 없거나 가상 메모리의 사용으로 속도가 매우 저하된다. 많은 사람들이 이점을 극복하기 위해 노력하여 현재 가장 효율적인 구현은 Compact PAT-trees[2]를 이용한 방법으로 $5m$ 바이트의 공간을 사용한다.

대용량의 텍스트에 대해서는 외부 메모리의 사용이 필수적으로 역파일(inverted file), B-트리(B-tree), 프리픽스 B-트리(prefix B-tree)와 같은 변형들이 있다. Ferragina[4]는 B-트리에 기반한 스트링 B-트리(string B-tree)를 개발했다. 스트링 B-트리는 외부 메모리 자료구조에 가장 좋은 B-트리와 서픽스 배열의 특성을 조합한 패턴 일치를 위한 자료구조이다. 그것은 역파일(수정성과 atomic key), 서픽스 배열(수정성과 연속 공간), 서픽스 트리(불균형한 트리 위상), 프리픽스 B-트리(bounded-length key) 등의 제한을 모두 극복하였고, 최악의 경우에 B-트리와 같은 성능을 가지는 첫번째 외부 메모리 자료 구조이다. 페이지의 크기를 B라 하고 문자열의 길이를 m이라 하면 스트링 B-트리 생성은 $O(m \log_B m)$ 디스크 접근이 필요하고, f번 나타나는 길이 n인 문자열 P에 대한 검색은 $O((n+f)/B + \log_B m)$ 디스크 접근이 필요하다.

본 논문에서는 k-mer 분석을 위해 패턴 일치(pattern matching)를 지원하고 외부 메모리 구조에 적합한 스트링 B-트리(string B-tree)를 개선하여 사용한다.

2 스트링 B-트리(String B-Tree)

대용량 문자열의 서픽스 트리는 메모리를 많이 사용하므로 외부 메모리를 사용해야 한다. 외부 메모리를 사용할 경우 서픽스 트리의 높이가 너무 높아 디스크 입출력 시간이 많이 걸린다. 이런 단점을 해결하는 자료구조가 스트링 B-트리(string B-tree)이다. 스트링 B-트리는 기존의 B-트리를 문자열 처리에 맞게 변형시킨 것이다.

길이가 m인 문자열을 S, 서픽스들을 S_i, 이 서픽스들이 사전순으로 정렬된 집합을 $K = \{K_1, K_2, ..., K_m\}$, 노드 π에 대해 노드에 저장된 문자열의 인덱스 집합을 S_π, S_π의 가장 왼쪽 문자열 인덱스를 $L(\pi)$, S_π의 가장 오른쪽 문자열 인덱스를 $R(\pi)$라고 하자. S에 대한 스트링 B-트리 SBT는 단말 노드에 n개의 서픽스 S_i들의 인덱스가 사전 순서로 저장되고, 내부 노드 π는 $n(\pi)$개의 자식 노드 σ_1, σ_2, ..., $\sigma_{n(\pi)}$를 가지고, 각 자식 노드의 $L(\sigma)$와 $R(\sigma)$의 집합 $S_\pi = \{L(\sigma_1), R(\sigma_1), L(\sigma_2), R(\sigma_2), ..., L(\sigma_{n(\pi)}), R(\sigma_{n(\pi)})\}$를 가진다. 그림 1는 길이 30인 문자열 TTATCTAGGTTGTCTTTCCTTACGGGTTCA에 대한 스트링 B-트리이다.

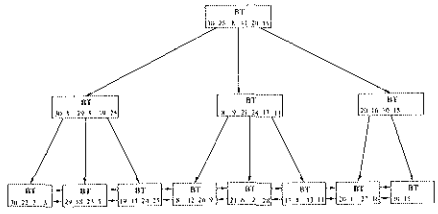


그림 1: 스트링 B-트리의 예

스트링 B-트리에서 빈번히 발생하는 연산 중 하나가 어떤 문자열이 노드 π의 문자열 인덱스 집합 S_π에서 사전순으로 따질 때 어느 위치에 들어가는 지를 검색하는 것이다. 이것은 이진 탐색으로 해결될 수 있지만 문자열 비교라는 점과 문자열이 외부 메모리에 존재할 경우를 위해 Blind Trie(BT)를 이용한다. Blind trie(BT)는 compacted trie와 Patricia 종류에 속하며 스트링 B-트리의 각 노드에 있는 문자열 인덱스의 집합 S_π로 만들어 지는 자료구조이다.

일반 문자열에 대한 스트링 B-트리가 아닌 하나의 문자열의 서픽스들에 대한 스트링 B-트리를 빠르게 생성하기 위해서는 *succ pointer*가 필요하다. 서픽스 S_i의 *succ pointer*는 S_{i+1}를 포함하는 단말 노드에 대한 포인터이고, $succ^{-1}$ *pointer*는 S_{i-1}를 포함하는 단말 노드에 대한 포인터이다.

스트링 B-트리의 단말 노드 π는 다음과 같은 구조를 가진다.

- $n(\pi)$개의 문자열 인덱스의 집합 $S_\pi = \{i | S_i$ is the suffix$\}$.
- 문자열 인덱스 집합 S_π에 대한 BT(blind trie)
- 단말 노드들을 이중 연결 리스트(double linked list)로 만들기 위한 $next(\pi)$와 $prev(\pi)$ 포인터.
- $lcp(R(prev(\pi)), L(\pi))$와 $lcp(R(\pi), L(next(\pi)))$.
- $parent(\pi)$ 포인터
- S_π 문자열에 대한 $succ$와 $succ^{-1}$ 포인터 집합.

스트링 B-트리의 내부 노드 π는 다음의 정보를 가진다.

- $n(\pi)$개의 자식 노드 포인터 집합 $C_\pi = \sigma_1, \sigma_2, ..., \sigma_{n(\pi)}$
- $2n(\pi)$개의 문자열 인덱스 집합 $S_\pi = L(\sigma_1), R(\sigma_1), ..., L(\sigma_{n(\pi)}), R(\sigma_{n(\pi)})$.
- 문자열 인덱스 집합 S_π에 대한 BT(blind trie)
- $parent(\pi)$ 포인터

여기서 $lcp(S_1, S_2)$란 문자열 S_1과 S_2 사이의 longest common prefix를 말한다.

3 k-mer 분석 시스템

3.1 스트링 B-트리의 확장

효율적인 검색을 위해 스트링 B-트리의 단말 노드에 인접한 서픽스들의 lcp 정보를 저장한다. 즉, $n(\pi)$개의 문자열 인덱스 집합 S_π에 대해 $n(\pi)$개의 $LCP_\pi = \{i | lcp(S_i, S_{i+1})\}$이 저장된다. 이것을 이용하면 Ferragina가 사용했던 패턴 일치시에 구했던 패턴과 일치하는 오른쪽 경계를 구하지 않고 단말 노드를 따라가면서 알 수가 있으므로 속도가 향상된다.

유전자 염기서열이 정적이라는 사실을 이용하면 Ferragina[4]가 제안한 방법보다 더 빠르게 스트링 B-트리를 생성할 수 있다. 이 방법은 염기서열의 서픽스들을 정렬하여 정렬 리스트와 lcp 리스트를 구한다[9]. 그런 다음 정렬 리스트에 해당하는 서픽스들을 순서대로 단말 노드에 추가하고, 생성된 단말 노드의 부모 노드인 내부 노드를 만든다. 같은 높이의 노드가 하나(루트 노드)가 될 때까지 계속 내부 노드를 만든다. 또한 이 방법은 공간을 컴팩트하게 사용한다.

3.2 k-mer의 빈도 계산

스트링 B-트리에서 k-mer의 빈도를 구하는 것은 Ferragina[4]가 제안한 패턴 일치 알고리즘을 사용하면 된다. 그러나 스트링 B-트리에서 모든 k-mer의 빈도를 구할 때 위의 방법은 반복적인 루트 노드에서 단말 노드로의 순회와 빈도가 0인 k-mer에 대해서도 수행을 하므로 효율적이지 못하고 시간이 많이 걸린다.

스트링 B-트리의 모든 단말 노드에는 서픽스들이 사전순으로 정렬되어 있고 인접한 서픽스들의 lcp 정보가 있기 때문에 단말 노드를 순회하면서 k-mer의 빈도를 계산하는 것이 더 빠르다. k를 구하고자 하는 k-mer이라 하고 F를 해당 k-mer의 빈도가 저장되는 배열이라 할때 알고리즘 1은 빈도가 0이 아닌 모든 k-mer들의 빈도를 구한다.

알고리즘 1 단말 순회를 이용한 모든 k-mer 빈도 구하기

```
SB-kmer
  Input: k // k-mer
  Output: F // 빈도 배열
    count = 0;
    pat = 첫번째 k-mer;
    node = 첫번째 단말노드;
    while node ≠ NULL;
        node를 메모리에 로드한다.;
        for i from 0 to N_leaf-1
            s = node의 i번째 서픽스;
            lcp = node의 i번째 lcp;
            if lcp > k then
                count = count + 1;
            else
                if length(s) ≥ k then
                    F에 count 추가;
                    pat = s[0]...s[k-1];
                end if
                count = 1;
            end if
        end for
        node = next(node);
    end while
```

4 실험 결과

본 논문에서 제시한 시스템은 CPU가 300MHz (R12000), 메모리가 512MB인 SGI *Octane*에서 C++와 Open Inventor 라이브러리를 사용하여 구현하였다. 실험에 사용한 데이터는 길이 m인 랜덤 데이터와 NCBI에 있는 생물체의 전체 염기 서열(whole genome)이다. 시간에 관련된 실험 결과는 timex 프로그램을 이용하여 측정한 시간이다.

4.1 스트링 B-트리의 생성 시간

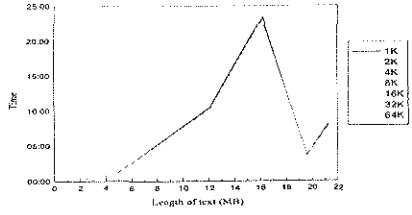
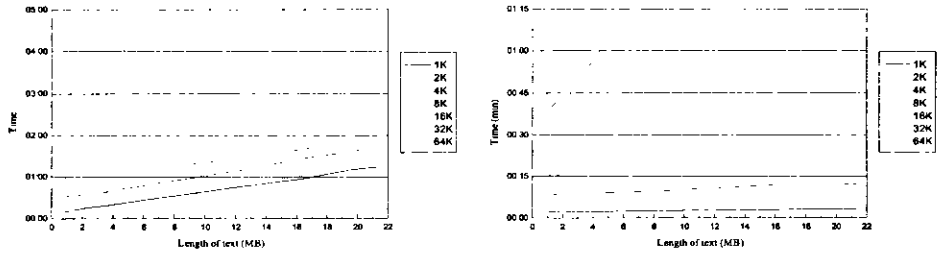


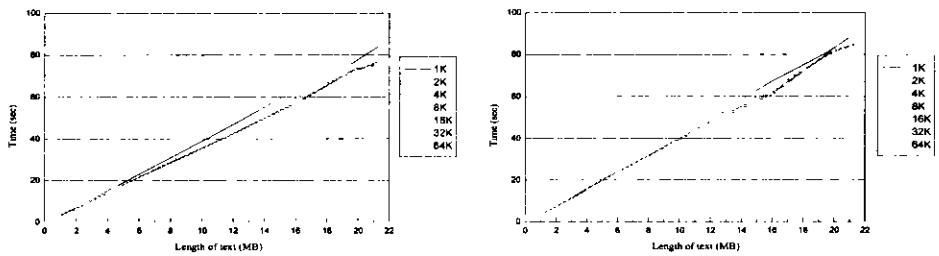
그림 2: 6종의 염기 서열에 대한 스트링 B-트리 생성 시간

그림 2는 6종의 염기 서열에 대한 스트링 B-트리의 생성 시간을 보여준다. 염기 서열의 길이가 약 16MB인 예쁜 꼬마 선충(*C. elegans*)에 대한 생성 시간은 25분 걸린다. 생성 시간이 서열의 길이별로 차이가 나는 이유는 서픽스들의 정렬 리스트에서 인접한 서픽스들의 lcp들이 차이가 나기 때문이다. lcp들이 작을 수록 더 빠르게 스트링 B-트리를 생성할 수 있다. 페이지의 크기는 생성 시간이나 파일 크기에 별로 영향을 미치지 않는다.



(a) 6-mer (b) 8-mer

그림 3: Ferragina의 방법으로 k-mer의 빈도 계산



(a) 6-mer (b) 20-mer

그림 4: 단말노드 순회 방법으로 k-mer의 빈도 계산

4.2 k-mer의 빈도 계산

그림 3이 Ferragina가 개발한 방법으로 계산할 때 걸리는 시간이고, 그림 4가 본 시스템에서 제안한 방법으로 계산할 때 걸리는 시간이다. 그림에서 볼 수 있듯이 4MB의 텍스트에 대해 전자의 방법은 k가 6에서 8로 증가할 때 수행 시간이 4분에서 65분으로 증가하고 페이지의 크기에 영향을 많이 받지만 후자의 방법은 k가 6에서 20으로 증가하더라도 시간이 거의 증가하지 않는다.

5 결론 및 향후 연구과제

본 논문에서는 생물체의 염기 서열 분석과 진화적 관계를 밝히는 데 사용되는 분석 방법 가운데 하나인 k-mer를 이용하는 분석 시스템을 제시하였다. k-mer의 빈도를 구하기 위해 염기 서열의 길이와 메모리량에 따라 Karp-Rabin 알고리즘, 서픽스 트리(suffix tree), 스트링 B-트리(string B-tree)를 사용할 수 있다.

Karp-Rabin[7]의 알고리즘을 응용한 방법은 단순하고 속도가 빠른 알고리즘이지만 13-mer 이상에 대해서는 메모리 부족으로 수행할 수가 없다. 13-mer 이상의 k-mer 분석을 하기 위해 외부 메모리에 효율적인 스트링 B-트리를 사용해야 한다.

Ferragina[4]가 제안한 스트링 B-트리 생성 방법은 서픽스들을 순서대로 노드에 추가해 나가는 삽입 정렬 방식이지만 본 논문에서는 먼저 서픽스들에 대해 정렬을 한 다음 노드를 만들어 가는 상향식 알고리즘을 사용하여 생성 시간을 단축시켰다. 스트링 B-트리 생성 시간은 염기 서열의 길이에 의존하며 $O(m/B)$ 디스크 접근이 필요하다. 이것보다 더 중요한 요소가 정렬된 서픽스 리스트에서 인접한 서픽스들 간의 lcp이다. lcp가 크면 클 수록 생성 시간은 더 걸린다. 검색은 텍스트의 길이 m, 검색 문자열의 길이 n, 빈도 f, 페이지 크기 B에 의존하며 $O((n+f)/B + \log_B m)$ 디스크 접근이 필요하다.

모든 k-mer의 빈도 계산은 Ferragina의 검색 알고리즘을 전체 k-mer에 대해 수행하는 것보다 단말 노드를 순회하면서 존재하는 k-mer의 빈도를 계산하는 것이 더 빠르다. 이 때 수행 시간에 영향을 미치는 것은 k와 m이고, 시간 복잡도는 $O(m/B)$의 디스크 접근과 $O(km)$의 문자 비교이다. 스트링 B-트리의 자료구조를 확장하여 단말노드에 lcp 정보를 저장함으로써 $O(m)$의 문자 비교만 필요하므로 k에 의존하지 않고 k-mer의 빈도를 구할 수 있다.

감사의 글

본 연구는 한국과학재단 목적기초연구(2000-1-30300-012-2) 지원으로 수행되었음.

참고문헌

[1] R. S. Boyer and J. S. Moore. A fast string matching algorithm. *Comm. ACM*, 20:762-772, 1977.

[2] David R. Clark and J. Ian Munro. Efficient suffix trees on secondary storage. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 383-391, 1996.

[3] J. C. Venter *et. al.* The sequecne of the human genome. *Science*, 291:1304-1351, Feb 2001.

[4] P. Ferragina and R. Grossi. The string B-tree: A new data structure for string search in external memory and its application. *Journal of ACM*, 46(2):236-280, 1999.

[5] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences.* Addison wesley, 1998.

[6] S. Karlin, L. Brocchieri, J. Mrazek, A. M. Campbell, and A. M. Spormann. A chimeric prokaryotic ancestry of mitochondria and primitive eukaryotes. *Proc Natl Acad Sci U S A*, 96(16):9190-9195, Aug 1999.

[7] R. Karp and M. Rabin. Efficient randomized pattern matching algorithms. *IBM J. Res. Development*, 21:249-260, 1987.

[8] D. E. Knuth, J. H. Morris, and V. B. Pratt. Fast pattern matching. *Algorithmica*, 6:323-350, 1977.

[9] U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935-948, 1993.