대용량 DNA서열 처리를 위한 서픽스 트리 생성 알고리즘의 개발

(Suffix Tree Constructing Algorithm for Large DNA Sequences Analysis)

최 해 원 *
(Hae-Won Choi)

요 약 서픽스 트리는 데이터의 내부구조를 자세히 나타내고 선형시간 탐색이 가능한 효과적인 자료구조로서 DNA 서열분석 등에 유용하다. 그러나 서열을 서픽스 트리로 구축하는 경우 트리의 크기가 원본의 최소 30배 이상으로 커지므로 테라바이트(TB)급의 대용량 DNA 서열의 경우에 메모리상의 응용은 매우 어려운 문제점이 있다. 이에 본 논문에서는 디스크를 이용한 대용량 DNA의 서픽스 트리 응용기법을 제시한다. 이때 DNA 서열구조를 고려한 서픽스 트리 선형 탐색 특성 유지를 보장한다. 이를 검증하기 위하여 9G Byte의 유전자 단편 서열을 이용해 424G Byte의 서픽스 트리를 디스크에 구축한 다음, 임의의 질의 서열에 대해 KMP알고리즘과 비교한 결과 질의 응답시간에서 우수한 성능을 보였다.

핵심주제어 : 서픽스 트리, 스트링 매칭, DNA 처리, 데이터 분석, KMP

Abstract A Suffix Tree is an efficient data structure that exposes the internal structure of a string and allows efficient solutions to a wide range of complex string problems, in particular, in the area of computational biology. However, as the biological information explodes, it is impossible to construct the suffix trees in main memory. We should find an efficient technique to construct the trees in a secondary storage. In this paper, we present a method for constructing a suffix tree in a disk for large set of DNA strings using new index scheme. We also show a typical application example with a suffix tree in the disk.

Key Words : Suffix Tree, String Matching, DNA Processing, Data Analysis, KMP

1. 서 론

최근 활발히 연구되고 있는 생명공학은 미래의 새로운 성장 동력으로서 여러 산업분야에 파급효과가 매우 크다. 생명공학은 연구목적에 따라 DNA, RNA, ETS 등 여러 종류의 인간 유전체 서열(sequence)들을 다룬다. 이와 같은 다양한 유전체서열을 분석, 분리하는 도구(tool)는 전통적인 기법인 전기영동(electrophoresis)과 같은 화학적·생물학적인 기법과 함께, 최근 들어 전자, 컴퓨터 등의 IT 기술을 접목한 새로운 분석 도구가 폭넓게 사용되고 있다. 이와 같은 새로운 융합학문의 연구는 DNA Web 데이터베이스(DB)에서부터 시작하여 마이크로-어레이(micro-array), DNA 칩(DNA chip) 그리고 자

* 경운대학교 컴퓨터공학과 교수

동분석기(automatic sequencer) 등의 많은 결과물을 생산하고 있으며 유전체 서열에 효과적이고 빠른 분석을 제공함으로서 유용성을 증명하고 있다. 이 중 다양한 소프트웨어 기반의 분석툴(data analysis tool)들은 알고리즘(algorithm), 데이터 마이닝(data mining) 등의 오랜 기간 동안 연구된 컴퓨터이론들을 기반으로 하는 경우가 많다.

서픽스 트리(suffix tree) 역시 전통적인 알고리즘의 한 연구 분야로서 유전체 서열 분석에 사용된다. 서픽스 트리는 서열의 모든 서픽스를 저장한 트리를 의미하며 검색하고자 하는 패턴에 대한 선형시간(linear time) 탐색 특성을 가진다. 전통적인 서픽스 트리의 응용분야는 서열비교(sequences matching), 반복서열문제(sequences tandom Repeats problem), 서열 회문 검색(sequences palindrome searching) 등이며 스트링(string)의 내부구조를 자세히 나타내므로 분석에 효율적이고 빠른 수단을 제공하고 있다[1-5]. 하지만 서픽스 트리는 원본 데이터를 서픽스 트리로 구축하는 경우 완성된 서픽스 트리의 크기가 지나치게 커지는 응용상의 문제점이 있다. 예를 들어 100GB의 원본 데이터를 서픽스 트리로 구축하는 경우 최소 30배 이상 커지게 된다. 그러므로 기존의 서픽스 트리 연구는 메모리를 기반으로 한 상대적으로 적은 용량의 스트링(string) 응용에 집중되어 있다.

이에 비해 최근의 생명공학 연구주제는 맞춤의학을 위한 인간과 인간 사이의 염기서열 차이를 연구하는 단염기 다양성(SNP) 연구나 각 종(Species)간의 전체 염기서열 비교연구로 확대되고 있다. 이들 연구에는 각 서열간의 상대적인 상관관계를 잘 표현할 수 있고 선형검색이 가능한 서픽스 트리가 아주 적절한 분석수단이다. 문제점은 전체 염기서열 비교는 상대적으로 적은 사이즈의 서열을 다루는 것이 아니라 최소 수백 기가바이트(GB)에서 수백 테라바이트(TB) 단위의 데이터를 사용하므로 서픽스 트리를 이용하기에 매우 큰 제약이 따르는 것이다.

본 논문은 이러한 서픽스 트리의 메모리 한계성에 의한 응용 제한적인 문제점을 해결하기 위해 디스크를 이용한 응용기법을 제안한다. 제안하는 기법에서는 기존의 서픽스 트리와 달리 DNA 서열을 고려한 자료구조를 이용하여 서열정보를 노드가 아닌 에지에 저장하고 내부노드(internal node)와 잎 노드(leaf node)의 정보를 분리한 인덱스 처리를 통해 디스크 상의 위치를 상수시간에 처리할 수 있도록 한다. 그리고 생성되는 노드와 동적으로 연계되도록 서픽스 링크(suffix link)를 처리해 서픽스 트리의 선형시간 탐색 등의 중요 특성을 그대로 유지하고 저장과 검색의 효율성 높인다. 성능평가 결과 제안한 기법은 EST 서열 9GB를 이용하여 424GB의 서픽스 트리를 디스크에 구축하였으며 서픽스 트리의 특성 또한 유지할 수 있었다. 구축된 서픽스 트리를 이용하여 KMP 알고리즘과 매칭시간을 비교한 결과 서열의 크기가 커질수록 우수한 성능을 보였다. 이를 통해 대용량 DNA 서열을 서픽스 트리로 구축하는 경우에 응용제한적인 문제점을 해결할 수 있음을 보였다.

본 논문의 구성은 다음과 같다. 먼저 2장에서 서픽스 트리의 특성과 응용범위에 대해 소개하고 본 논문의 연구동기가 된 대용량 유전체 서열 응용 문제점을 분석한다. 3장에서는 본 논문에서 제안하는 대용량 DNA 서열을 디스크에 서픽스 트리로 구축하고 응용하기 위한 기법을 상세히 살펴보고 4장에서는 제안된 기법과 KMP를 이용하여 매칭 성능을 비교분석한다. 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

본 장에서는 서픽스 트리의 일반적인 특성에 대해서 알아보고 이를 대용량 DNA 서열에 적용했을 경우의 문제점에 대해 제시한다.

2.1 서픽스 트리

서픽스 트리는 스트링의 모든 서픽스를 포함하는 일종의 트리로 주어진 스트링의 내부구조를 자세히 드러내는 자료구조이다. 이러한 특성 때문에 서픽스 트리는 스트링 매칭(string matching), 최장 공동 부서열 검색(longest

common substring searching), 모든 쌍의 서픽스-프리픽스 검색(all-pairs suffix-preffix pair searching)등 복잡한 많은 스트링 문제를 선형시간에 풀 수 있는 방법을 제공한다[6-10].

일반적으로 서열비교문제(exact matching problem)를 푸는 대표적인 알고리즘은 KMP(Knuth-Morris-Pratt)와 BM(Boyer-Moore) 알고리즘이다[8]. 이러한 전통적인 알고리즘 역시 서픽스 트리를 이용한 방법과 마찬가지로 선형시간에 스트링비교가 가능하다. 하지만 알고리즘 동작방식에 차이가 있다. 즉 KMP나 BM의 경우 먼저 패턴(pattern)을 선 처리(preprocessing)한 후 텍스트(text)에서 패턴을 찾는 방식으로 동작하고 후자의 경우 이와 반대로 텍스트를 선 처리 하고 다음으로 패턴을 찾는다.

예를 들어, 길이가 각각 m과 n인 이미 알려진 텍스트 T와 알려지지 않은 패턴 P가 있다고 하자. 여기서 텍스트는 원본 스트링을 의미하고 패턴은 찾고자 하는 부분 스트링(substring)을 의미한다. KMP나 BM 알고리즘은 검색하고자 하는 새로운 패턴 P가 주어질 때마다 O(n)시간 동안 패턴을 선 처리한 후에, O(m)시간동안 텍스트에서 패턴을 찾아야 한다. 이와 달리 서픽스 트리를 이용하면 트리를 생성하기 위한 O(m) 시간동안 텍스트를 선 처리해서 트리를 생성해 두면, 검색하고자 하는 알려지지 않은 패턴 P가 주어질 때는 단지 패턴의 길이인 O(n) 시간 안에 P가 T에 존재하는지를 밝힐 수 있다.

이러한 서픽스 트리의 특성은 DNA서열정보와 같은 대용량의 원본 데이터 데이터베이스에 대해 특정한 패턴을 비교 검색하는 경우에 아주 유용하게 이용될 수 있다. 즉, 게놈프로젝트에 의해 밝혀진 DNA서열정보(T)를 한번만 서픽스 트리로 구축해 놓는다면(O(m)) 이후에 매칭 하고자 하는 부분 서열(P)은 패턴의 길이에 비례하는 시간(O(n))만 필요하기 때문에 매우 효율적인 검색이 가능하다. 이는 생명공학 연구에 소모되는 시간을 큰 폭으로 줄일 수 있음을 의미한다. 또한 서픽스 트리는 유전체 서열의 모든 서픽스를 포함하고 있으므로 상호비교연구가 중요한 유전체 분석에는 매우 중요한 도구가 될 수 있다.

2.2. 대용량 DNA 서열에 응용 시 문제점

서픽스 트리의 우수한 특성에도 불구하고 대용량 DNA서열을 서픽스 트리로 구성하기 어려운 이유 중 하나는 메모리 제한적인 문제점 때문이다. 즉, 크기가 k인 유전체 서열을 서픽스 트리로 변환하는 경우 O(k)의 메모리가 필요하지만, 여기에 내포된 상수는 적어도 30 이상이므로 원본 데이터에 비해 서픽스 트리의 용량은 30배 이상으로 커진다[2-3]. 그러므로 인간과 인간 사이의 염기서열 차이를 연구하는 단염기 다양성(SNP) 연구나 각 종(Species)간의 전체 염기서열 비교연구와 같이 수백 테라바이트(TB) 단위의 원본 데이터를 사용하는 경우에는 기존의 연구처럼 메모리를 이용해 서픽스 트리를 구현하는 것은 매우 큰 제약이 따른다.

3. 디스크를 이용한 서픽스 트리 생성

본 장에서는 서픽스 트리를 대용량 DNA 서열에 적용했을 경우 발생하는 문제점을 해결하기 위해 디스크를 이용한 서픽스 트리 응용기법을 제시한다. 이를 위해서 DNA 서열구조를 분석하고 이를 이용하여 서픽스 트리의 선형시간 탐색 등의 특성을 유지하기 적합하도록 자료구조와 처리 기법을 제시한다.

3.1 자료구조

디스크를 이용하여 DNA 서열을 서픽스 트리로 구축할 때 유의할 점은 선형시간 탐색 등의 서픽스 트리 특성은 그대로 유지가 되어야 한다는 것이다. 또한 DNA가 a, c, t, g의 유클레오티드로 구성되는 구조 역시 고려할 필요가 있다. 하지만 기존의 서픽스 트리 구현기법은 메모리를 참조하는 포인터를 이용하므로 새로운 자료구조와 저장방식이 필요하다. 따라서 본 논문에서는 서픽스 트리 구조를 내부노드와 잎노드 그리고 에지로 구분하고, 디스크의 저장 효

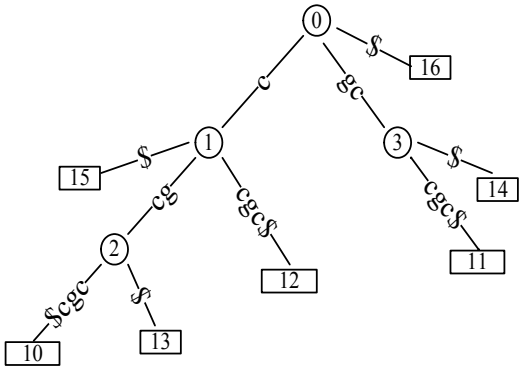
율성과 서픽스 링크와 같은 서픽스 트리의 고유 특징을 쉽게 유지하기 위해서 염기서열의 정보를 노드보다는 에지에 저장하도록 한다. 즉, 에지는 DNA 서열의 정보를 저장하는 역할을 하고 내부노드와 잎 노드는 에지를 저장한 디스크 상의 위치를 저장하는 인덱스 역할과 서픽스 링크를 유지하는 역할을 한다. 그러므로 내부노드와 잎 노드는 별도의 자료구조 없이 노드번호 형식으로만 존재하고 에지는 4byte의 다섯 개의 필드로 구성된다. 그림 2는 에지의 각 필드 명칭을 보여준다.

startNode	firstIndex	lastIndex	seqNum	endNode

<그림 1> 에지의 자료구조

에지 자료구조에서 첫 번째와 마지막 필드는 에지가 연결된 시작 노드 번호와 마지막 노드 번호를 나타낸다. 만약 *startnode* 필드가 0 이면 트리의 루트(root)를 의미한다. 두 번째와 세 번째 필드는 서열의 첫 번째 인덱스와 마지막 인덱스를 나타낸다. 마지막으로 *seqNum* 필드는 DNA 서열의 번호를 나타낸다. 에지 자료구조의 5개의 필드 중에서 *startnode*와 *endnode* 그리고 *firstindex* 3개 필드는 인덱싱을 위한 키로 사용된다. 즉 *startnode*와 서열의 *firstindex*로 키를 만들고 *endnode*와 상수인 *BACK*으로 또 하나의 키를 만든다. 전자를 전방키(forward key)라하고 후자를 후방키(backward key)라 한다. 전방키는 루트에서부터 잎 노드 쪽으로 순회(traverse)할 때 사용되고, 후방키는 반대의 경우에 사용된다. 두 키에 의해 트리에서 DFS(depth first search)나 BFS(breadth first search)가 가능하다.

제시하는 자료구조에서 노드는 데이터를 저장하는 것이 아니라 에지의 첫 번째 문자와 결합하여 디스크 저장위치를 접근(access)하는 인덱스 역할을 한다. 이는 서픽스 트리에서 각 노드에서 분기된 모든 에지의 첫 번째 문자는 서로 다른 특성을 이용하는 것이다. 노드의 이러한 역할을 위해 각 노드는 노드번호를 가지며 내부 노드와 잎 노드 번호는 서로 틀리게 구성한다. 이는 잎 노드와 관련된 정보는 내부 노드의 것보다 훨씬 적기 때문에 저장용량을 줄이고 인덱스 역할을 쉽게 하도록 하기 위함이다.



<그림 2> 노드와 에지로 구성된 서픽스 트리

그림 2는 제시한 노드와 에지의 자료구조를 바탕으로 구성된 서픽스 트리를 보여준다. 그림 2에서 보듯이 각 노드에서 분기된 모든 에지의 첫 번째 문자는 서로 틀리며 이를 이용해 키를 조합할 수 있다. 예를 들어 노드 1번에서 노드 번호와 에지의 첫 번째 문자의 쌍인 '1*$*'와 '1*g*' 그리고 '1*c*'는 디스크를 접근하기 위한 키 역할을 수행한다.

3.2 디스크에 서픽스 트리 구축

제시된 자료구조를 이용해서 디스크에 서픽스 트리를 구축할 때 유의할 점은 선형시간 탐색과 같은 서픽스 트리의 주요 특성을 유지하는 것이다. 이를 위하여 본 장에서는 서열의 정보를 저장하고 있는 에지에 대한 처리와 서픽스 링크에 대한 처리 그리고 서픽스 트리를 디스크에 저장하고 접근하기 위한 인덱스 처리 기법을 제시한다.

3.2.1 에지 처리

앞서 제시한 자료구조에서 설명한 바와 같이 서열의 실제적인 정보는 에지에 저장한다. 이러한 에지정보는 하나의 작은 서열에서 시작하여 점차 서열의 수가 늘어남과 동시에 복잡하게 상호 연계된다. 이들 정보를 빠짐없이 디스크에

저장하기 위해 본 논문에서 제시하는 기본 아이디어는 한번 생성된 에지는 수정만 일어나지 제거되지 않고 항상 존재하도록 만드는 것이다. 즉 에지가 만들어지는 순서대로 생성된 에지를 디스크의 파일에 순차적으로 저장하고 참조위치를 인덱스 테이블이 저장하는 것이다. 이를 통해 새로 생성되는 에지에 대한 정보는 쉽게 추가되고 관리된다.

그러나 서픽스 트리의 유용한 특성을 유지하는 핵심은 단순히 생성되는 서열을 디스크에 저장하는 것에서 더 나아가 기존의 서열에 연계되어 추가 생성되는 서열과 업데이트 되는 서열 정보를 동적으로 저장하는 것이다. 이를 위해서는 기본적인 아이디어 외에 추가적인 기법이 필요하다. 이에 제시하는 기법은 노드를 생성할 때 에지가 생성되는 순서와 일치하게 노드번호를 차례대로 부여하고 에지가 분기되는 경우의 수를 고려하여 서열을 저장하는 것이다. 에지를 처리하기 위한 알고리즘은 그림 3과 같다.

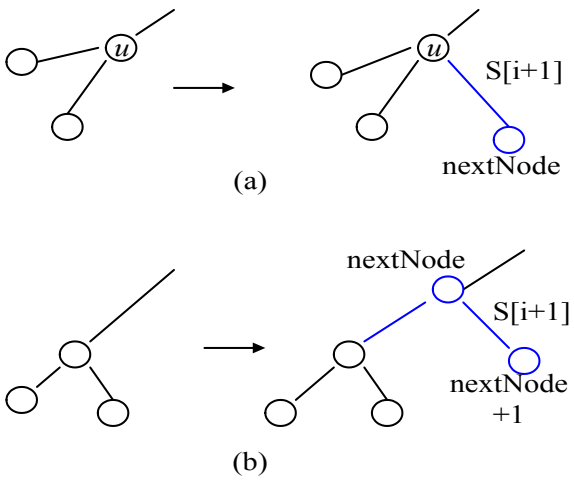
새로운 서열을 서픽스 트리에 추가할 때 서열의 실제적인 정보를 저장하는 에지는 기존의 에지에서 분기(branching)되는 경우와 새롭게 분할생성(splitting)되는 두 가지 경우가 있다. 그림 3에서 라인 2~11은 분기를 처리하는 과정이고 라인 12~15는 분할생성을 처리하는 과정이다. 에지의 분기는 새로운 노드가 생성되는 것이 아니라 기존에 존재하는 임의의 내부노드를 기점으로 분기되며 결과적으로 하나의 새로운 에지와 잎 노드가 생성된다. 이와 같은 분기는 생성되는 에지 자료구조의 *startnode* 필드는 이미 존재하는 노드번호를 입력하고 *endnode* 필드는 새로운 노드 번호를 부여하도록 처리한다. 이때 주의할 것은 만약 새로운 노드 번호가 *nextnode*라는 변수에 저장되어 있다면 *endnode* 필드 입력은 *nextnode*의 값이 되고 *nextnode*의 값은 1이 증가하도록 설정하는 것이다. 이를 통해 추가되는 서열의 에지 정보가 기존에 저장된 서픽스 트리의 에지 정보를 삭제(overwriting)하는 것을 방지할 수 있다.

Algorithm for Edge Processing

```
1:  make_edge(sequence) {
2:   if(branch=1) {
3:    then make_branch( ) {
4:         startnode = old_node_number;
5:         if(nextnode=1){
6:           endnode = nextnode_num;
7:           nextnode_num=nextnode_num+1;
8:          }
9:         endnode = new_node_number;
10:     }
11:   }
12:  else {
13:    startnode = nextnode_num;
14:    endnode = nextnode_num+1;
15:   }
16: }
```

<그림 3> 에지 처리 알고리즘

분기에 비해 분할생성은 기존의 존재하는 에지에서 새로운 내부노드가 생성되면서 에지가 추가로 만들어지는 경우이다. 이때 새로 생긴 에지의 *startnode* 필드는 *nextnode*의 값을 가지고, *endnode* 필드는 *nextnode*+1 이 되도록 설정하면 된다. 그림 4는 에지가 생성되기 직전의 모습과 에지가 생성된 후의 처리를 보여준다. 그림 4 (a)는 에지가 분기되는 경우이며 그림 4 (b)는 새로운 내부노드가 만들어지면서 에지가 분할생성 되는 경우를 나타낸다.



<그림 4> 에지 생성 (a)분기 (b)분할생성

3.2.2 서픽스 링크 처리

서픽스 링크는 서픽스 트리에서 노드와 노드 사이를 연결함으로서 선형시간 탐색 특성을 제공하는 중요한 역할을 한다. 그러므로 이미 서픽스트리에 저장된 노드와 새로 생성되는 노드 사이의 정보교환이 필수적이다. 이와 같은 정보교환을 기존의 연구에서는 메모리를 기반으로 포인터를 사용하여 처리하였다. 그러나 본 논문에서 주목하고 있는 메모리에서 처리하기 힘든 대용량 DNA데이터는 디스크를 기반으로 하므로 기존의 방법을 그대로 사용할 수 없다.

본 논문에서 서픽스 링크를 처리하기 위한 기본적인 방법은 노드와 동적으로 연동하는 배열을 사용하는 것이다. 즉 저장된 노드와 생성되는 노드를 고려한 배열을 구성하고 노드 처리 시 인덱스를 순차적으로 저장한다. 여기에 부가적으로 내부노드와 잎노드를 분리해서 서픽스 링크를 처리함으로서 상수시간에 각 노드의 정보를 구분할 수 있도록 한다. 이에 대한 근거는 서픽스 트리에서 모든 내부 노드는 하나의 서픽스 링크를 가지고 상대적으로 잎 노드는 서픽스 링크를 가지지 않기 때문이다. 이에 대한 자세한 처리과정을 그림 5에서 보여주고 있다.

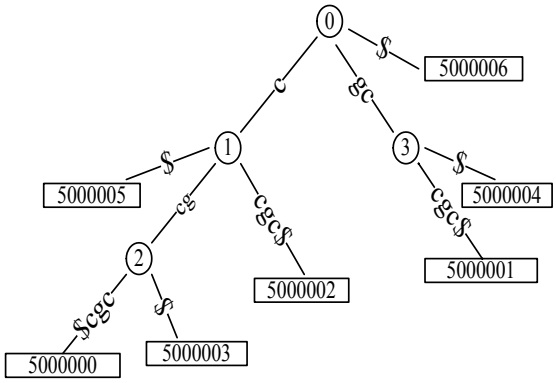
그림 5에서 서픽스 링크를 처리할 때 내부 노드 번호와 잎 노드의 노드 번호를 완전히 분리시켰다. 즉 내부 노드는 0에서부터 시작하여 노드 번호를 부여하고 잎 노드는 내부 노드 번호와 충분히 차이가 큰 임의의 상수에서부터 순차적으로 노드 번호를 부여하는 것이다. 이와 같이 순차적인 노드 번호를 기억하는 두 개의 변수를 각각 *next_internal_node*와 *next_leaf_node*로 둔다면, 앞서 제시한 에지의 분기와 분할생성의 두 가지 경우로 나눠서 처리가 가능하다.

Algorithm for suffixlink Processing

```
1:  make_suffixLink(sequence) {
2:   if(branch=1) {
3:    then make_internal( )  {
4:         next_leafnode = endnode;
5:         next_leafnode +=1;
6:        }
7:   else make_leaf( ) {
8:      next_internal_node =startnode;
9:      next_leafnode =endnode;
10:    next_internal_node +=1;
11:    next_leafnode +=1;
12:  }
```

<그림 5> 서픽스링크 처리 알고리즘

그림 5의 라인 2~6은 에지가 분기되는 경우에 서픽스 링크의 처리를 나타내고 라인 7~11은 분할생성되는 경우의 처리를 나타낸다. 분기의 경우 해당 에지의 *endnode* 필드를 *next_leaf_node*로 한 후 *next_leaf_node*의 값을 1 증가시킴으로서 처리 할 수 있다. 이에 비해 한 개의 내부노드와 한 개의 잎 노드가 생성되는 분할생성의 경우에는 해당 에지의 *startnode* 필드를 *next_internal_node*로 하고 *endnode* 필드를 *next_leaf_node*로 한 후, *next_internal_node* 와 *next_leaf_node*의 값을 모두 1씩 증가시키면 된다. 이렇게 함으로써 내부 노드와 잎 노드는 임의의 정수를 기점으로 완전히 구분된다. 이와 같은 노드 처리 방식을 이용하면 내부 노드 수만큼의 배열만 있으면 공간 낭비 없이 서픽스 링크를 저장 할 수 있다. 일반적으로 트리에서 전체 서열의 길이를 i 라 하면 최대 $2i-1$개의 에지가 존재할 수 있기 때문에 전체 가능한 내부 노드의 수 역시 최대 i 이다. 그러므로 i개의 서픽스 링크를 위한 배열만 있으면 된다. 반면에 현재 노드가 잎 노드인지 아닌지를 판별하기 위해서 기존의 서픽스 트리 생성 방식대로 노드가 생성된다면, 현 노드에 연결된 모든 자식노드의 포인터(pointer)가 널(null)인지를 판별하기 위하여 $O(|\Sigma|)$의 시간이 필요하다. 그러나 본 논문에서 제시된 기법에서는 단지 노드 번호가 정해진 임의의 정수보다 큰지만 비교하면 상수 시간에 알 수 있으므로 매우 효율적이다. 그림 6은 내부노드와 잎 노드의 노드번호를 분리하는 임의의 정수를 5000000이라 했을 때 이를 기준으로 구분된 서픽스 트리를 나타낸다. 내부노드번호는 0부터 5000000 미만의 값을 가지고 잎 노드 번호는 5000000 이상의 값을 가지게 된다.



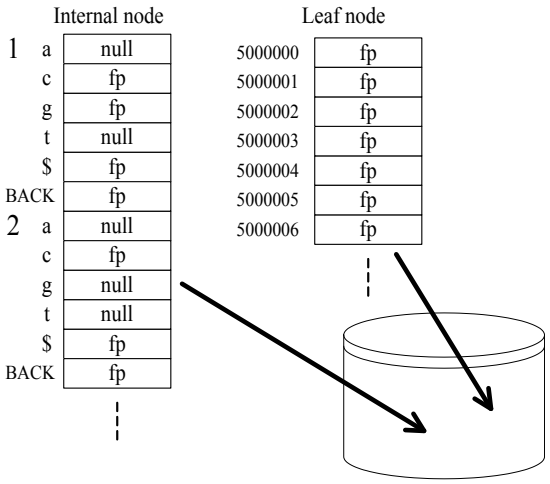
<그림 6> 내부노드번호와 잎노드 번호로 구분된 서픽스 트리

3.2.3 인덱스 처리

앞서 제시한 자료구조를 토대로 에지 처리와 서픽스 링크처리가 완료된 후 그 결과로서 *edge_file*과 *internal_node_file* 그리고 *leaf_node_file* 파일이 생성된다. 여기서 *edge_file*은 DNA 서열의 모든 서픽스 서열 정보를 저장한다. 그리고 *internal_node_file*과 *leaf_node_file*은 각각 내부노드와 잎 노드에 대한 정보를 저장하면서 에지가 저장되어 있는 파일 상의 위치를 인덱스 하는 기준점이 된다. 본 절에서는 내부노드와 잎 노드에 인덱스 정보를 저장하고 디스크 상의 위치를 상수시간에 처리하기 위한 과정을 상세히 제시한다.

제시한 자료구조에서 임의의 내부 노드와 DNA서열을 이용해서 만들 수 있는 키의 조합은 알파벳 *a*, *c*, *g*, *t*에 종결 문자 *$* 그리고 역순위 검색에 사용하는 *BACK*의 여섯 가지이다. 이를 기반으로 첫 번째 내부노드에서부터 시작해서 마지막 내부노드까지 각 노드에 대해, 만약 키에 해당하는 에지가 존재한다면 그 파일 상의 위치를 저장하고 키에 해당하는 에지가 존재하지 않는다면 '널'(null)을 저장한다. 예를 들어 현재 노드 1과 알파벳 '*a*' 로 만든 키에 해당하는 에지가 존재한다면 파일 상의 위치를 저장하고 만약에 없다면 '널'을 저장한다. 만약 파일 상의 위치를 저장하는데 4 바이트가 필요하다고 가정하면, 디스크에 저장된 총 내부노드 파일의 크기는 (마지막 내부노드-첫 번째 내부노드+ 1) × (4×6) 바이트가 된다. 잎 노드에 대한 인덱스 정보처리 역시 내부노드와 유사하며, 이를 통해 파일의 크기와 위치를 알 수 있다. 차이점은 잎 노드의 경우 가능한 키의 조합이 BACK 하나뿐인 것이다. 그러므로 디스크에 저장된 총 잎 노드 파일의 크기는 (마지막 잎 노드 - 첫 번째 잎 노드 +1) × 4 바이트이다.

이러한 과정의 결과로 *internal_node_file*과 *leaf_node_file*에 인덱스 정보가 저장된다. 이러한 인덱스 정보는 모두 노드 번호순으로 순차적으로 저장되었기 때문에 에지의 디스크 상의 위치는 상수시간에 계산가능하다. 예를 들어 노드 번호 *k*와 첫 번째 알파벳 '*c*'가 가리키는 에지의 디스크 상의 위치는 다음과 같다. 만약 *k*가 내부 노드라면 *internal_node_file*의 $k \times (4\times6) +$ '*c*' × 4 바이트에 에지가 저장되어 있고, 잎 노드라면 *leaf_node_file* 의 $(k - \text{LEAF}) \times 4$ 바이트에 에지가 저장되어 있다.



<그림 7> 내부노드와 잎노드의 디스크 저장

그림 7은 에지에 대한 인덱스 정보가 내부노드와 잎 노드에 저장되는 예를 나타낸다. 예를 들어, 그림 7에서 노드 번호가 2이고 첫 번째 문자가 "*c*" 일 경우 에지의 위치는 *internal_node_file*의 52 번째의 값에 저장되어 있다. 이와 같이 *internal_node_file*의 널(null)로 저장된 인덱스들은 추가되는 DNA 서열에 의해 분기가 발생할 경우 갱신되기 때문에 점차로 없어진다. 이는 제안하는 기법에서 인덱스 정보처리의 효율성을 높이기 위해 새로운 내부 노드

를 만들어지지 않고 이전에 존재하는 내부 노드를 갱신하도록 설정했기 때문이다. 그리고 *internal_node_file*와 *leaf_node_file*를 이용한 인덱스 정보는 생명공학 연구의 응용 특성에 따라 일정한 노드 범위를 재구성하여 데이터 범위를 조절할 수 있는 장점도 있다.

4. 실험 및 분석

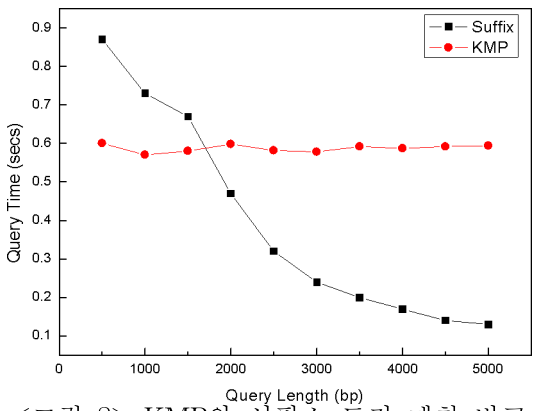
본 장에서는 지금까지 제안된 기법을 실제로 구현하여 몇 가지 실험을 통해 성능을 분석한다. 실험에 사용된 시스템 환경은 4GB의 주 메모리와 2.5GHz 듀얼(dual) 제온 CPU 그리고 1.5TB의 단일 하드디스크를 가진 서버급 컴퓨터이며 자바 언어로 구현하였다. 실험에 사용된 데이터는 DNA 분석에 널리 응용되는 발현된 유전자 단편인 EST(expressed sequence tags)중에서 와일드 카드(wild card)가 없는 30,000~50,000 베이스(base) 길이의 약 100,000개 서열을 사용하였으며, 원본 데이터의 양은 약 9G Byte이다. 본 논문에서 제안한 기법을 이용해서 디스크에 서픽스 트리를 구축하는데 소비된 시간은 약 7시간이며, 생성된 서픽스 트리의 크기는 표 1과 같다.

<표 1>디스크에 생성된 서픽스트리 파일

디스크에 생성된 서픽스 트리 파일	크기 (GB)	비율 (%)
edge_file	271.36	64
internal_node_file	119.56	28.2
leaf_node_file	33	7.8
전체	424	100

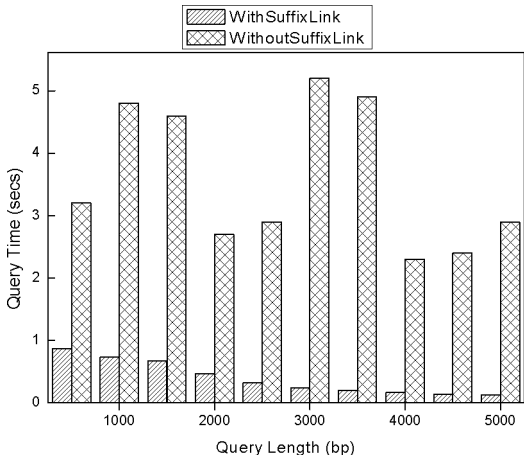
표 1에서 *edge_file*와 *leaf_node_file*은 전체 파일 중 각각 64%와 7.8%를 차지하며 널(null) 값을 가지지 않도록 자료구조를 구성하였으므로 낭비되는 공간이 전혀 없다. 반면에 *internal_node_file*의 경우는 자료구조상 사용하지 않는 널(null) 부분이 존재하는데, 대략 *internal_node_file*의 10%를 차지한다. 결과적으로 디스크에 생성된 서픽스 트리 파일의 합은 424GB이다. 원본 EST의 크기는 9GB이므로 약 47배로 용량이 커졌음을 알 수 있다. 이와 같은 용량의 증가는 서열의 중복 정도에 따라 달라질 수 있지만 최소 30배 이상이다[2-3]. 더구나 실험에 사용한 EST 파일은 발현된 유전자 단편의 극히 일부분에 지나지 않는다. 만약 인간의 개인적인 유전 특성인 단염기 다양성(SNP) 연구나 각 종(Species)간의 전체 염기서열 비교연구와 같이 수백 테라바이트(TB) 단위의 원본 데이터를 사용하는 경우에는 제안하는 디스크를 이용하는 서픽스 트리 구성 기법이 유용한 생물학적 연구수단이 될 것이다.

본 논문에서 제안한 기법의 성능을 분석하기 위해 디스크에 구현된 서픽스 트리를 이용하여 임의의 DNA 서열 매칭(matching)에 적용하여 보았다. 비교하기 위해 사용된 알고리즘은 검색에 일반적으로 널리 사용되는 KMP 알고리즘으로서 $O(m+n)$ 시간 복잡도를 가진다. 그림 8은 KMP 알고리즘과 제안한 기법의 DNA서열 매칭을 비교한 결과이다. x축은 질의서열(Query Sequence)의 길이를 베이스(base) 단위로 나타내고 y축은 주어진 질의서열에 대해 매칭에 성공한 시간을 나타낸다. 그림 8에서 볼 수 있듯이 서픽스 트리를 이용할 경우의 질의 시간은 질의 길이가 늘어날수록 감소한다. 이는 서픽스 트리의 특성 상 서열이 짧은 경우 분기가 많이 일어나기 때문이다. 이에 비해 KMP 알고리즘을 사용할 경우는 질의 서열, 즉 패턴(n)의 길이에 상관없이 거의 소요시간이 일정하다. 하지만 질의 서열의 길이가 1700베이스를 넘어서는 순간부터 본 논문에서 제안한 기법이 우수한 성능을 보임을 알 수 있다.



<그림 8> KMP와 서픽스 트리 매칭 비교

매우 많은 양의 DNA 데이터를 서픽스 트리를 구축하는 것은 상당한 시간을 소비한다. 하지만 DNA 서열과 같이 데이터 변동이 거의 없는 데이터인 경우는 한번만 구축해 놓는다면 이후에 매칭 하고자 하는 부분서열(Subsequence)은 패턴의 길이 n에 비례하는 시간(O(n))만 필요하기 때문에 선형시간 검색이 가능하므로 응용 측면에서 매우 효율적이다. 이와 같이 특성을 보장하는 것은 서픽스 링크에 의한 중복정보 감소와 검색시간 단축의 효과 때문이다. 그러므로 제안하는 기법에서 서픽스 링크가 정상적으로 동작하는지 여부는 매우 중요하다. 그림 9는 본 논문에서 제안한 기법에서 서픽스 링크를 사용한 경우와 서픽스 링크를 제거한 경우를 비교한 결과이다.



<그림 9> 서픽스 링크 사용유무에 따른 매칭 비교

그림 9의 x축은 질의서열의 길이를 베이스 단위로 나타내고 y축은 주어진 질의서열에 대해 매칭에 성공한 시간을 나타낸다. 그림 9에서 서픽스 링크를 가진 기법의 경우 질의 시간은 질의 길이가 늘어날수록 감소한다. 하지만 서픽스 링크가 없는 기법의 경우에는 질의 시간이 상대적으로 많이 소모되면서 매우 불규칙적이다. 이는 서픽스 링크가 제거되면 내부노드 분기에 대한 정보가 소실되기 때문이다. 그러므로 본 논문에서 제안한 기법은 서픽스 링크의 특성을 유지하면서 대용량 DNA 서열을 디스크에 구현하여 매칭 등에 충분히 응용할 수 있음을 알 수 있다.

5. 결 론

본 논문은 서픽스 트리의 메모리 응용중심의 문제점을 해결하기 위해 대용량 서열을 디스크에 서픽스 트리로 구축하고 응용하기 위한 기법을 제시했다. 생명공학 연구에 중요한 도구로 사용될 수 있는 서픽스 트리는 패턴에 대한 선형시간 탐색 특성을 가지고 서열의 내부구조를 자세히 나타내므로 서열분석에 효율적이고 빠른 수단을 제공하고 있다. 하지만 서픽스 트리는 원본 데이터를 서픽스 트리로 구축하는 경우 완성된 서픽스 트리의 크기가 지나치게 커지는 응용상의 문제점이 있었다. 그러므로 기존의 서픽스 트리 응용분야는 메모리 기반의 상대적으로 작은 용량의 서열에 집중되었다. 그러나 최근 생명공학의 이슈인 맞춤의학을 위한 단염기 다양성(SNP) 연구나 각 종간의 전체 염기서열 비교연구와 같이 수백 기가바이트(GB)에서 테라바이트(TB)의 대용량 데이터를 사용하는 경우 서픽스 트리와 같은 자세한 서열간의 상관관계를 나타내줄 도구가 절실하게 필요함에도 불구하고 실제로 적용하기는 매우 어려운 문제점이 있었다.

본 논문에서 제안한 디스크를 이용한 서픽스 트리 응용 기법은 기존의 서픽스 트리와 달리 DNA 서열을 고려한 자료구조를 이용하여 서열정보를 노드가 아닌 에지에 저장하고 내부노드와 잎 노드의 정보를 분리한 인덱스 처리를 통해 디스크 상의 위치를 상수시간에 처리할 수 있도록 하였다. 그리고 생성되는 노드와 동적으로 연계되도록 서픽스 링크(suffix link)를 처리해 서픽스 트리의 선형시간 탐색 등의 중요 특성을 그대로 유지하고 저장과 검색의 효율성을 높였다.

제안한 기법의 성능 분석을 위해 발현된 유전자 단편인 EST중에서 30,000~50,000 베이스 길이의 약 9GB 용량의 100,000개 서열을 사용하여 실험하였다. 그 결과 424GB 용량의 서픽스

트리를 디스크에 생성하였으며 이를 이용하여 KMP 알고리즘과 질의 서열 검색을 비교하였다. 그 결과 제안한 기법은 질의 서열의 길이가 길어질수록 우수한 성능을 보임을 알 수 있었다. 또한 디스크에 저장된 서픽스 트리가 선형 시간 탐색 특성을 유지하고 있는 지 확인하기 위하여 서픽스 링크를 제거한 기법과 서픽스 링크를 유지한 기법을 비교해 보았다. 그 결과 제안한 기법은 서픽스 트리의 선형시간 탐색성능을 그대로 유지함을 확인하였다.

참 고 문 헌

[1] Dan Gusfild, *Algorithms on Strings, Trees, and Sequence : Computer Science and Computational Biology*, CAMBRIDGE University Press, 2002.

[2] Juha Karkkainen and Esko Ukkonen, "Sparse Suffix Tree," *COCOON*, pp.219-233, 1996.

[3] E. Ukkonen, "On-Line Construction of Suffix Trees," *Algorithmica*, vol. 14, pp.249-260, 1995.

[4] Hardison,R.C., Oeltjen,J. and Miller,W., "Long human-mouse sequence alignments reveal novel regulatory elements: a reason to sequence the mouse genome," *Genome Res.*, vol. 7, pp.959 - 966, 1998.

[5] Shimuzu,N., Roe,B.A., Chissoe,S. et al., "The DNA sequence of human chromosome 22," *Nature*, vol. 402, pp.489 - 495, 1999.

[6] Mark Nelson, "Fast String Searching With Suffix Trees," *Dr. Dobb's Journal*, 1996.

[7] Graham A. Stephen, *String Search*, University College of North Wales, 2003.

[8] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms*, The MIT Press, 2005.

[9] Schwartz,S., Zhang,Z., Frazer,K.A., Smit,A., Riemer,C., Bouck,J., Gibbs,R., Hardison,R. and Miller,W., "A web server for aligning two genomic DNA sequences," *Genome Res.*, vol. 10, pp.577 - 586, 2004.

[10] Batzoglou,S., Pachter,L., Mesirov,J.P., Berger,B. and Lander,E.S. , "Human and mouse gene structure: comparative analysis and application to exon prediction," *Genome Res.*, vol. 10, pp.950 - 958, 2005.

최 해 원 (Hae-Won Choi)

- 1996년 2월 : 경일대학교 컴퓨터학과(공학사)
- 2000년 2월 : 경북대학교 컴퓨터공학과(공학석사)
- 2009년 2월 : 경북대학교 컴퓨터공학과(공학박사)
- 2005년 9월~현재 : 경운대학교 컴퓨터공학과 교수
- 관심분야 : 유비쿼터스 컴퓨팅, 알고리즘

논 문 접 수 일 : 2009년 8월 18일
1차수정완료일 : 2009년 10월 22일
2차수정완료일 : 2010년 2월 2일
게 재 확 정 일 : 2010년 2월 20일