http://dx.doi.org/10.14400/JDC.2014.12.12.265

DNA 분석에 효율적인 서픽스 트리 재구성 알고리즘

최해원, 정영석, 김상진
경운대학교 컴퓨터공학과

An Efficient Suffix Tree Reconstructing Algorithm for Biological Sequence Analysis

Hae-Won Choi, Young-Seok Jung, Sang-Jin Kim
Dept. of Computer Engineering Kyungwoon University

요 약 서픽스 트리는 주어진 모든 문자열의 모든 서픽스를 트리 형태로 나타내는 자료구조로서 선형시간에 구성할 수 있으며 문자열에 대한 많은 문제를 효율적으로 해결할 수 있다. 하지만 이런 효용성에도 불구하고 서픽스 트리로 구성한 문자열을 삽입/삭제하는 경우 트리를 구성하는데 상당히 많은 시간이 소비된다. 본 논문은 이러한 문제를 해결하기 위한 서픽스 트리 재구성 알고리즘을 제안한다. 제안하는 알고리즘은 부 문자열을 삽입하는 경우와 삭제하는 경우로 나눈 다음, 발생할 수 있는 모든 경우의 수를 감안해서 설계했다. 알고리즘의 성능을 평가하기 위해서 기존의 Ukkonen 알고리즘과 비교실험 해 본 결과 서픽스 트리 재구성 시 30% 이상 시간이 절약됨을 알 수 있었다.

주제어 : 서픽스 트리, Ukkonen 알고리즘, DNA 서열, 유전체 서열

Abstract This paper introduces a new algorithms for reconstructing the suffix tree of character string, when a substring id deleted from the string or a string is inserted into the string as a substring. The algorithem has two main functions, delete-structure and insert-structure. The main objective of this algorithm is to save the time for constructing the suffix tree of an edited string, when the suffix tree of the original string is available. We tested the performance of this algorithm with some DNA sequences. This test shows that delete-reconstructing can save time when the length of the subsequence deleted is less than 30% of the original sequence, and the insert-reconstructing takes less time with regard to the length of inserted sequence.

Key Words : Suffix Tree, Ukkonen Algorithm, DNA Sequence, Biological Sequence

1. 서론

인간 유전자 프로젝트(Human genome project) 이후 많은 후속연구가 이어지면서 DNA 서열(sequence) 분석에 필요한 기법(method)들이 많이 개발되었다. 데이터 처리 관점에서 DNA 서열은 a, c, t, g 4개의 문자가 무작위로 섞여 있는 길이가 긴 문자열이다. 이러한 DNA는 알려진 바와 같이 문자열의 길이가 매우 길고 각 개인마다 일정부분 차이가 있기 때문에 서열을 분석하기 위한 효율적인 기법들이 요구되고 있다. 컴퓨터공학에서는 전

Received 12 September 2014, Revised 27 October 2014
Accepted 20 December 2014
Corresponding Author: Sang-Jin Kim
(Dept. of Computer Engineering Kyungwoon University)
Email: happychw@ikw.ac.kr

ISSN: 1738-1916

© The Society of Digital Policy & Management. All rights reserved. This is an open-access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0), which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

통적으로 문자열 처리 알고리즘이 많이 연구되어 왔는데 이들을 DNA 서열 분석에 응용할 수 있다. 그 중 응용 효율성이 높은 검증된 기법으로 서픽스 트리(suffix tree)가 있다[1-8].

서픽스 트리는 주어진 문자열의 모든 서픽스를 트리 형태로 나타내는 자료구조로써, Weiner, McCreight, 또는 Ukkonen 알고리즘에 의해 선형 시간에 구성할 수 있으며, 문자열에 관한 많은 문제를 효율적으로 해결할 수 있게 한다[9-12]. 예를 들면, 한 문자열이 다른 문자열의 부 문자열로 존재하는지의 여부를 찾는 패턴 검색(pattern matching)문제, 반복 부 문자열(substring)을 찾는 문제, 원본 문자열 오염 여부를 판별하는 문제 등을 들 수 있다. 하지만 서픽스 트리는 트리로 구성한 문자열의 극히 일부분이 제거되거나, 문자열이 첨가되는 경우에 트리를 다시 구성해야하는 문제가 야기된다. 이 문제는 Dan Gusfield가 이미 제기한 바 있으나 이를 적절히 해결하는 방법은 아직 연구에 소개되지 않았다[1].

본 논문은 주어진 문자열의 부 문자열(substring)이 제거되거나 첨가되었을 때 효율적으로 서픽스 트리를 재구성할 수 있는 서픽스 재구성 편집 알고리즘을 제시한다. 제안하는 알고리즘은 크게 두 부분으로 구성된다. 먼저, '삭제-서픽스 트리 재구성 알고리즘'은 주어진 문자열의 부 문자열이 제거되었을 때 서픽스 트리를 재구성하는 역할을 하며, '추가-서픽스 트리 재구성 알고리즘'은 다른 문자열이 추가 되었을 때 트리를 재구성하는 역할을 한다. 이러한 서픽스 트리 재구성 알고리즘은 선형 시간에 서픽스 트리를 구성할 수 있는 Ukkonen 알고리즘에 기반을 두고 있기 때문에 Ukkonen 알고리즘으로 구성한 서픽스 트리에 적용할 수 있다.

제시한 서픽스 트리 재구성 알고리즘의 평가는 DNA 서열을 표본 데이터로 서픽스 트리를 구축하고, 부 문자열을 제거하거나 다른 DNA 서열을 표본 데이타에 첨가했을 때 트리가 재구성 되는 시간을 측정하여 기존 Ukkonen 알고리즘과 비교하였다. 결과적으로 제시하는 기법의 트리 재구성 시간이 기존 Ukkonen 알고리즘보다 30%이상 단축됨을 확인 할 수 있었다. 특히 표본 데이터에 대해서 50%이하 길이의 부 문자열을 제거하거나 첨가했을 때 더 효율적이었다.

본 논문의 구성은 다음과 같다. 2장에서 서픽스 트리와 Ukkonen 알고리즘에 대해서 소개하고, 3장에서 서픽스 트리에서 문자열 삽입 삭제가 일어나는 경우 효율적으로 트리를 재구성하는 기법을 제시한다. 4장에서는 제시한 알고리즘과 기존 Ukkonen 알고리즘을 비교분석한다. 마지막으로 5장에서는 결론 및 향후 연구 과제를 기술한다.

2. 관련 연구

이 장에서는 서픽스 트리의 특성과 선형 시간에 서픽스 트리를 구성할 수 있는 Ukkonen 알고리즘에 대해서 간략히 소개한다[1,9].

2.1 서픽스 트리

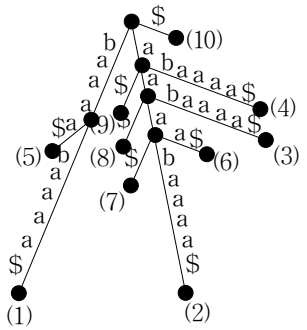
임의의 문자열 S에 대한 서픽스 트리는 S의 모든 서픽스(suffix)로 구성된 트리로 아래와 같이 두 성질을 만족한다.

(1) S의 i번째 서픽스 Y는 잎 노드 i까지 Y로 라벨(label)된 하나의 경로(path)를 가진다.
(2) S의 두 서픽스 Y와 Y'이 동일한 전치열(prefix) X를 가지고 있다면, 내부노드까지 X로 라벨된 유일한 경로가 존재한다.

[Fig. 1]은 문자열 S = *baaabaaaa*$에 대한 서픽스 트리이다. 위에서 제시한 서픽스 트리의 특징 (1)을 만족시키기 위하여 서열의 끝에 종료 문자(terminal symbol)인 '$'를 첨부한다. 잎 노드(leaf node)에 라벨한 숫자는 서픽스의 시작 위치를 나타내며 예를 들어 루트로부터 (2)번째 잎 노드까지의 라벨한 문자열은 S[2..10]이다.

2.2 Ukkonen 알고리즘

주어진 문자열 S를 서픽스 트리로 구성하는 알고리즘은 Weiner, McCreight 그리고 Ukkonen이 제안한 알고리즘이 있다[1, 9-12]. 이 알고리즘들은 선형 시간(linear time)내에 트리를 구성할 수 있다. 예를 들어 길이가 m인 문자열 S의 전치열(prefix) $S[1..i](1 \le i \le m)$에 대한 서픽스 트리를 Ti라고 가정하자. Ukkonen 알고리즘은 T_1, T_2, ...Tm 의 순서로 트리를 확장하여 S에 대한 서픽스 트리 Ti를 만든다.



[Fig. 1] Suffix Tree of S= baaabaaaa$

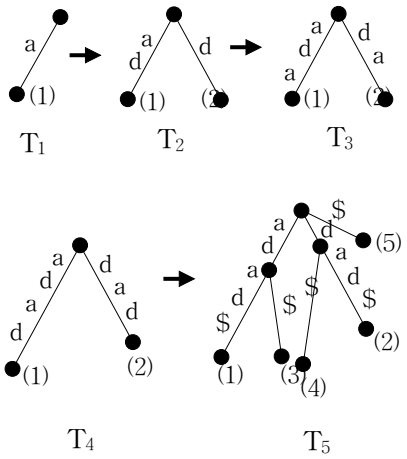
길이가 m인 문자열 S에 대한 서픽스 트리를 T라 하고 $S(i+1)$은 S의 $i+1$번째 문자라고 한다면 Ukkonen 알고리즘에 대한 의사 코드(pseudo code)는 다음과 같다.

```
Ukkonen_Suffix_Tree(T, S, m) {
    T1을 구성
    for(i= 1; i≤m-1; i++)
        for(k=1; k≤i+1; k++)
            현재 트리 Ti 에서 루트로부터 S[k..i]로 라벨
            된 지점의 끝에 S(i+1)를 추가로 확장한다;
}
```

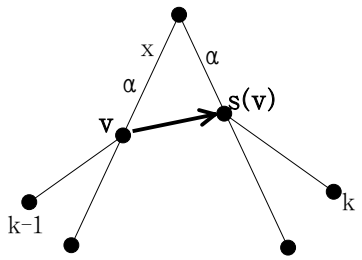
루트로부터 라벨된 $S[k..i]$의 끝에 $S(i+1)$를 확장할 때, 다음과 같은 규칙을 적용한다.

- 규칙 1: $S[k..i]$의 끝이 잎 노드이면 $S[k..i]$의 끝에 $S(i+1)$를 확장한다. 잎노드에 라벨 된 값은 바뀌지 않는다.
- 규칙 2: $S[k..i]$의 끝에서부터 시작하는 하위 경로가 존재하고 $S[k..i]$ 다음의 문자가 $S(i+1)$와 불일치(mismatch)하면, $S(i+1)$로 라벨된 잎 간선(leaf edge)을 $S[k..i]$의 끝 지점에서부터 연결함으로써 $S(i+1)$을 확장한다. 만약 $S[k..i]$의 끝이 간선의 시작과 끝 위치 사이에 존재한다면, 내부 노드를 먼저 생성하고 $S(i+1)$로 라벨한 잎 간선을 연결한다. 잎 간선의 끝인 잎 노드는 k로 라벨한다.
- 규칙 3: $S[k..i]$ 다음의 문자가 $S(i+1)$와 일치하면, $S(i+1)$를 확장하지 않는다.

[Fig. 2]는 S= *adad$*에 대한 서픽스 트리를 Ukkonen 알고리즘으로 구성하는 과정을 나타낸다. Ukkonen 알고리즘 의사 코드로 서픽스 트리를 구성할 때, 길이가 m인 문자열에 대해서 루트로부터 라벨된 문자열과 $S[k..i]$를 문자별로 비교하기 때문에, 시간 복잡도(time complexity)가 $O(m^3)$이다. 이와 같은 시간복잡도는 너무 부하가 크므로, 문자를 비교하는 횟수를 줄임으로써 서픽스 트리의 구성 시간을 줄일 필요가 있다. Ukkonen은 서픽스 링크와 스킵/카운트를 이용하여 문제를 해결한다.



[Fig. 2] Step of Ukkonen Algorithm



[Fig. 3] Suffix Link

2.2.1 서픽스 링크

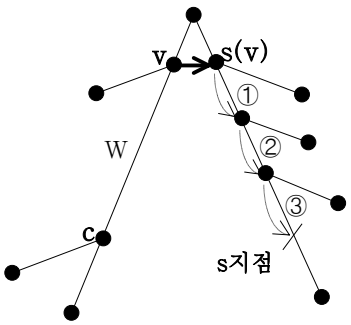
x를 문자, a를 문자열이라고 할 때, [Fig. 3]과 같이 루트로부터 노드 v와 $s(v)$까지의 경로에 각각 xa와 a로 라벨되어 있다면, v에서 $s(v)$를 가리키는 서픽스 링크를 둔

다. 만약 루트로부터 노드 v까지 x로 라벨되었다면, v에서 루트로 향하는 서픽스 링크를 둔다. k-1번째 잎 노드의 끝에 문자를 확장한 상태에서 k번째 잎 노드의 끝을 확장하고자 할 때, k-1번째 잎 노드에서부터 시작하여 v에서 서픽스 링크를 따라 $s(v)$까지 이동하므로 루트로부터 $s(v)$까지 라벨한 문자열 a를 비교할 필요가 없게 된다. 따라서 문자를 비교하는 횟수가 줄어든다.

Ukkonen 알고리즘으로 서픽스 트리를 완성했을 때, 각각의 내부 노드들은 모두 서픽스 링크가 존재한다. 임의의 노드 v에서 $s(v)$로 이동하는 서픽스 링크가 존재할 때, v의 하위 트리(subtree)에 라벨된 문자열들은 $s(v)$의 하위 트리에도 라벨되어 있다.

2.2.2 스킵/카운트(Skip/Count)

스킵/카운트는 서픽스 링크와 더불어 좀 더 빠른 시간 내에 트리를 구성하기 위하여 서픽스 링크를 통해 이동한 노드에서부터 하위 트리의 간선에 라벨된 첫 번째 문자만 비교하면서 하위 경로로 이동하는 방법으로, 문자를 비교하는 횟수가 줄어들기 때문에 시간을 줄일 수 있다. 예를 들어 [Fig. 4]에서 노드 v와 $s(v)$는 서픽스 링크로 연결되어 있으며, 노드 v에서 노드 c까지 라벨된 문자열을 W라고 할 때, $s(v)$에서부터 W로 라벨된 경로가 존재한다. 루트로부터 s지점까지 라벨된 문자열이 W와 같다면, 간선 ①, ②, ③에 라벨된 각각의 첫 번째 문자만을 비교하면서 s지점까지 이동하여 규칙 1, 2, 3에 따라 i+1번째 문자를 확장한다. 간선에 라벨된 문자열 전체를 비교하지 않기 때문에 문자를 비교하는 횟수가 줄어들게 되므로 서픽스 트리의 구성 시간을 줄일 수 있다.



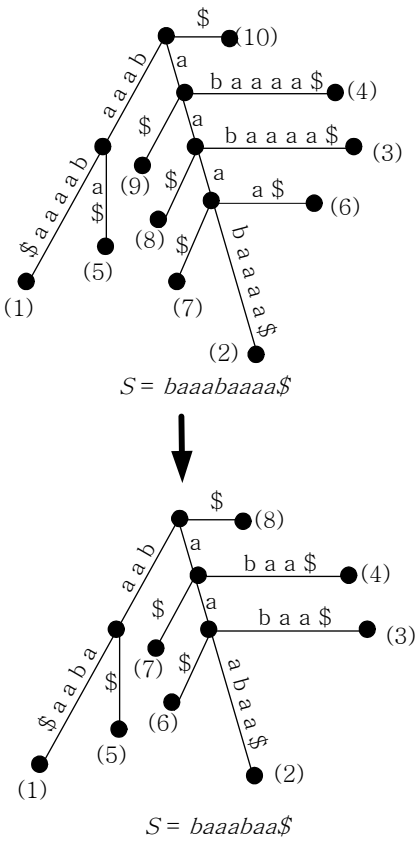
[Fig. 4] 스킵/카운트(Skip/Count)

3. 서픽스 트리 재구성 알고리즘

이 장에서는 본 논문에서 제안하는 주어진 문자열에 대해서 임의의 부 문자열(substring)이 제거되거나 추가되었을 때 서픽스 트리를 재구성하는 알고리즘을 구체적으로 기술한다. 주어진 문자열의 부 문자열이 제거되었을 때 서픽스 트리를 재구성하는 알고리즘을 삭제-서픽스 트리 재구성 알고리즘이라 하고, 문자열이 추가되었을 때 트리를 재구성하는 알고리즘을 추가-서픽스 트리 재구성 알고리즘이라한다.

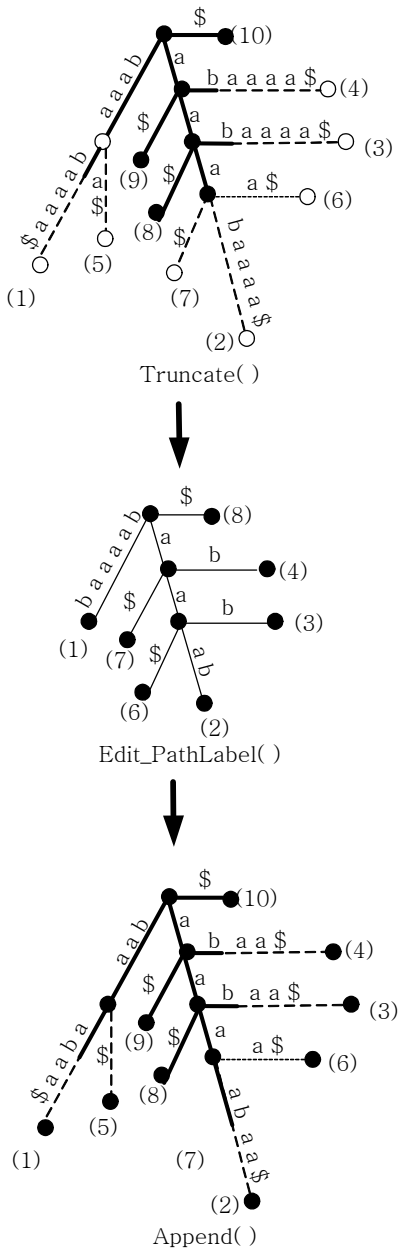
3.1 삭제-서픽스 트리 재구성 알고리즘

삭제-서픽스 트리 재구성 알고리즘은 길이가 m인 문자열 S로 만들어진 서픽스 트리에서 부 문자열 $S[i..j]$를 제거하여 길이가 $m'(=m-(j-i+1))$인 문자열 S'로 이루어진 서픽스 트리로 재구성하는 알고리즘이다.



[Fig. 5] Step of Restructured Suffix Tree

예를 들어 [Fig. 5]와 같이 문자열 S= *baaabaaaa$*에서 S[6..7]을 제거한 S'= *baaabaa$*의 서픽스 트리를 S에 대한 서픽스 트리를 재구성해서 만든다. 이러한 방법은 S'을 가지고 서픽스 트리를 만드는 것보다 시간이 단축되는 이점이 있다.



[Fig. 6] Step of Delete-Suffix Tree Phase

제안하는 삭제-서픽스 트리 재구성 알고리즘은 3단계로 진행된다. 먼저 문자열 S의 서픽스 트리를 S[1..i-1]까지의 서픽스와 S[j+1..m]의 서픽스로 나누는 Truncate()를 실행한다. 그 다음은 j+1이상의 모든 잎 노드(leaf node)에 대해서 루트로부터 잎 노드까지 라벨된 정보를 변경하는 Edit_PathLabel()을 실행한다. 마지막으로 루트로부터 라벨된 S[1..i-1]의 각 서픽스의 끝에 S'[i..m']을 확장하는 Append()를 실행하여 S'의 서픽스 트리를 완성한다. 삭제-서픽스 재구성 알고리즘에 대한 의사코드는 다음과 같다.

```
DelEdit_Suffix_Tree(T, S, i, j, m) {
    Truncate(T, S, i, j, m)
    Edit_PathLabel(T, i, j, m)
    Append(T, S', 1, i, m')
}
```

[Fig. 6]은 삭제-서픽스 트리 재구성 알고리즘을 이용해서 S= *baaabaaaa$*를 처리하는 단계별 과정을 보여준다.

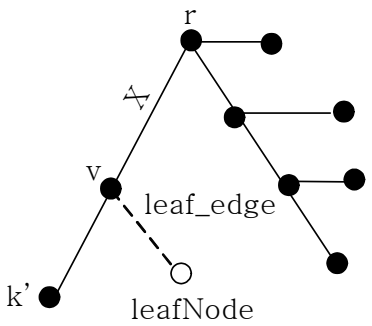
3.1.1 Truncate( ): 서픽스 트리 S를 S[1..i-1]의 서픽스와 S[j+1..m]의 서픽스로 재구성

Truncate( )는 루트로부터 S[1..i-1]와 S[j+1..m]의 서픽스로 라벨된 트리로 재구성하기 위하여 S[1..i-1]와 S[j+1..m]의 서픽스는 제거하지 않고, 나머지 문자열을 트리에서 제거하는 역할을 한다. 이는 j이하의 모든 잎 노드에 대해서 잎 노드에서부터 S[i..m]를 제거하고, 루트로부터 잎 노드까지 라벨된 문자열이 S[i..m]보다 작으면, 그 문자열 전체를 제거함으로써 가능하다. 제거할 문자열의 전치열이 S[1..i-1]의 부 문자열이거나 S[j+1..m]의 부 문자열이면 그 전치열(prefix)을 제외한 나머지 문자열을 제거한다. Truncate( ) 알고리즘의 의사코드는 다음과 같다. 아래 기술된 알고리즘에서 *delstring*은 삭제하고자 하는 문자열을 저장하는 역할을 한다.

```
Truncate(T, S, i, j, m) {
  for(leafNode = j; leafNode >= 1; leafNode--)
   if (i-1 < leafNode ≤ j)
     then delstring = 루트에서부터 leafNode까지
     라벨된 문자열;
```

```
    else delstring= S[i..m];
    if (leafNode에 대한 leaf_edge의 길이
        <= delstring의 길이)
      then { leafNode에 연결된 간선 제거;
             leafNode의 부모 노드에서 자식 노드가
             하나라면 부모 노드를 제거;
           }
    else leafNode의 leaf_edge에 라벨된 문자열에서
        S[i..m]을 제거;
}
```

예를 들어 [Fig. 7]과 같이 루트노드 r로부터 임의의 노드 v까지 라벨된 문자열을 X라 하고, 노드 v에서 *leafNode*까지 현재 현재 서픽스 트리에 저장된 문자열을 *leaf_edge*라고 가정하자. 만약 *delstring*의 길이가 *leaf_edge*의 길이보다 크다고 한다면, 이때 X는 $S[j+1..m]$의 부 문자열이거나 X의 전치열이 $S[1..j]$의 서픽스인 두 가지 경우뿐이므로 이에 대한 세부 처리과정은 다음과 같다.



[Fig. 7] Example of Deleted leaf_edge

3.1.1.1 X가 $S[j+1..m]$의 부 문자열인 경우

*leafNode*에서 점선으로 표시된 *leaf_edge*를 제거하면 $S[j+1..m]$의 부 문자열인 X는 남게된다. 문자열 X는 루트로부터 라벨되어 있으며, j이하의 잎 노드에 대해서만 *delstring*을 제거하기 때문에 결과적으로 $S[j+1..m]$의 모든 서픽스는 제거되지 않는다.

3.1.1.2 X의 전치열이 $S[1..j]$의 서픽스인 경우

만약 X의 전치열이 $S[1..j]$의 서픽스라면, 노드 v의 하위 트리에 존재하는 모든 잎 노드들은 j이하의 잎 노드이다. [Fig. 7]에서 X의 전치열을 $S[1..j]$의 서픽스인 $S[k'..j]$라고 할 때, Truncate()는 내림차순으로 진행되므로 *leafNode*에서 *leaf_edge*를 제거하면, 노드 v는 자식 노드가 k'번째 잎 노드뿐이므로 제거된다. 그 다음에 k'번째 잎 노드에 대한 *delstring*이 제거되기 때문에, X의 전치열이 $S[1..i-1]$의 서픽스라면 X의 전치열은 제거되지 않고, $S[i..j]$의 서픽스라면 제거되어 결과적으로 루트로부터 $S[1..i-1]$의 서픽스만 남게된다. 그러므로 *leafNode*에서는 *leaf_edge*를 제거한다.

따라서 j이하의 잎 노드에 대해서 *leaf_edge*가 *delstring*보다 크면 *delstring*을 제거하고, *leaf_edge*의 길이가 *delstring*보다 작으면 *leaf_edge*를 제거할 때, 루트로부터 $S[1..i-1]$와 $S[j+1..m]$의 각 서픽스로 라벨된 트리로 재구성된다.

3.1.2 Edit_PathLabel( ): j+1이상의 모든 잎 노드에 대해서 루트로부터 잎 노드까지 라벨된 정보를 변경

길이가 m인 문자열 S에서 부 문자열 $S[i..j]$를 제거한 후 변경된 길이 $m'(= m-(j-i+1))$의 문자열 S'가 될 때, $S[1..i-1]$의 서픽스는 $S'[1..i-1]$의 서픽스와 같다. $S[j+1..m]$은 $S'[i..m']$과 문자열이 서로 같으나 위치는 서로 다르므로, S'의 서픽스 트리를 구성하기 위하여 루트로부터 $S[j+1..m]$의 서픽스로 라벨된 경로의 간선과 잎 노드의 정보는 $S'[i..m']$의 정보로 변경해야한다. 따라서 $j+1$이상의 잎 노드에서부터 루트까지 라벨된 정보를 변경하기 위하여 삭제-서픽스 트리 재구성 알고리즘 2단계인 Edit_PathLabel( )를 실행한다.

단, $S'[i..m']$의 전치열이 $S'[1..i-1]$의 서픽스일 수 있으므로, 간선에 라벨된 값이 $S'[1..i-1]$의 부 문자열에 해당하거나, 한번 정보가 바뀌진 간선인 경우에는 변경하지 않는다. *leafNode* > j 인 *leafNode*에서 루트까지의 경로에 있는 간선에 (p, q)로 라벨되어 있을 때 Edit_PathLabel( )의 알고리즘은 다음과 같다.

```
Edit_PathLabel(T, i, j, m) {
    for(leafNode = j+1; leafNode <= m; leafNode ++)
      leafNode에 라벨된 값을 변경;
      (p, q)로 라벨 되어 있는 각 edge에 대해서
```

if not (변경된 간선이거나 q <= i-1)
then p, q 값을 변경;
}

3.1.3 Append( ): 루트로부터 라벨된 S[1..i-1]의 각 서픽스의 끝에 S'[i..m']을 확장하여 S'의 서픽스 트리를 완성

앞서 기술된 삭제-서픽스 트리 재구성 알고리즘의 두 단계를 마친 서픽스 트리는 루트로부터 $S'[1..i-1]$와 $S'[i..m']$의 서픽스로 구성된 트리이다. Append( )는 S'의 서픽스 트리를 완성하는 마지막 단계이다. Append( )의 의사코드는 다음과 같다.

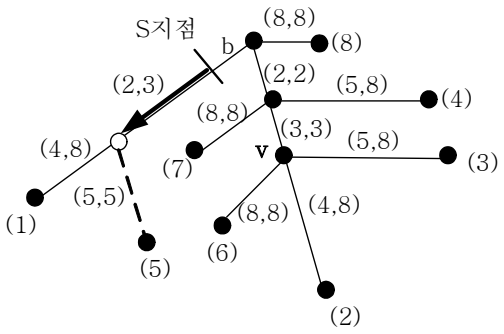
Append(T, S', 1. i, m') {
for(k= 1; k<= i-1; k++)
루트로부터 라벨된 $S'[k..i-1]$의 끝에 $S'[i..m']$을 확장;
}

삭제-서픽스 트리 재구성 알고리즘은 Ukkonen 알고리즘으로 구성한 서픽스 트리를 대상으로 편집하기 때문에 기존의 서픽스 트리에 존재하는 서픽스 링크를 이용할 수 있다. 그래서 루트로부터 라벨된 $S'[k..i-1]$의 끝지점을 s라 할 때, s까지 서픽스 링크와 스킵/카운트 방법으로 이동하고 $S'[i..m']$을 확장한다.

$S'[i..m']$을 확장하는 방법은 s에서부터 시작하는 하위 경로의 유무에 따라 달라진다. 만약 s에서부터 시작하는 하위 경로가 없으면 s의 끝에 $S'[i..m']$을 바로 확장할 수 있다. 그렇지 않고 s에서부터 하위 경로가 존재한다면, 이 하위 경로에 라벨된 문자열은 $S'[1..i-1]$의 서픽스 또는 $S'[i..m']$의 서픽스이거나 $S'[1..i-1]$의 서픽스에 $S'[i..m']$의 서픽스가 연결된 문자열이다. 왜냐하면 앞서 삭제-서픽스 트리 재구성 알고리즘 1, 2단계에 의해 $S'[1..i-1]$의 서픽스와 $S'[i..m']$의 서픽스로만 구성된 트리에서 $S'[1..i-1]$의 서픽스에 $S'[i..m']$을 확장하기 때문에 하위 경로에 라벨된 문자열은 위의 두 경우만 존재한다.

$S'[1..i-1]$의 서픽스를 문자열 X라 하고, $S'[i..m']$의 서픽스를 문자열 Y라고 할 때 X, Y로 라벨된 경로는 루트에서부터 존재하고, s지점에서 확장할 문자열인 $S'[i..m']$도 루트에서부터 경로가 경로가 만들어진다. 그래서 s에서부터 라벨된 문자열이 X 또는 Y라 한다면, 그 문자열과 $S'[i..m']$의 가장 긴 공통 전치문자열(prefix)을 루트로부터 찾을 수 있다. s지점에서부터의 하위 경로는 하나 이상이므로, 공통된 전치열을 찾아서 이동하기 위하여 스킵/카운트를 적용하며, 불일치하는 지점에서 확장한다.

s지점에서부터 문자열 XY로 라벨된 경로가 존재할 때는 X와 Y로 나누어 스킵/카운트를 이용한다. 우선 X와 확장할 문자열의 가장 긴 공통된 전치열을 루트로부터 찾아서 그 길이만큼 스킵/카운트로 이동하고, 불일치하면 확장한다. 만약 X의 길이만큼 s지점에서 스킵/카운트로 이동했다면, Y와 나머지 확장할 문자열의 가장 긴 공통된 전치열을 루트에서부터 찾아서 그 길이만큼 역시 스킵/카운트로 이동하며, 불일치할 때 확장한다.



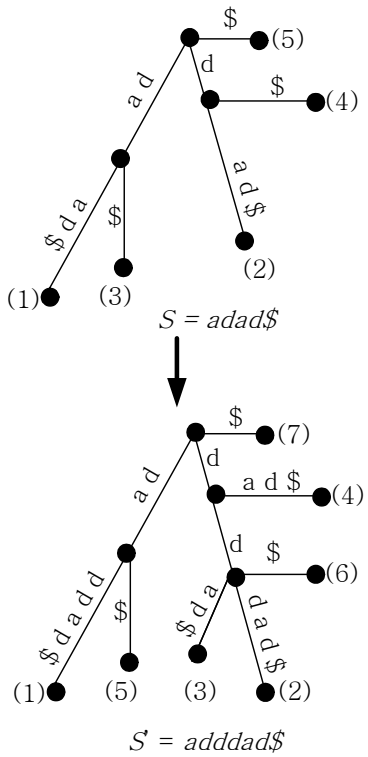
[Fig. 8] Example of append( )

[Fig. 8]은 append( )의 예시로서, 삭제-서픽스 트리 재구성 알고리즘을 나타낸 [Fig. 6]의 append( ) 단계 서픽스 트리에서 잎 노드 번호 (1), (2), (3), (4)를 확장 한 후 (5)번 잎 노드를 확장하는 과정을 나타낸다. (5)번째 잎 노드를 확장하기 위하여 루트로부터 $S'[5..5]= b$가 라벨된 지점인 s까지 이동한다. s지점에서부터 $S'[6..8]$을 확장해야하는데, s지점에서의 하위 경로가 $S'[2..8]=S'[2..5]S'[6..8]$로 라벨되어 있다. $S'[2..5]$와 $S'[6..8]$은 루트로부터 라벨되어 있으므로, 우선 확장할 문자열 $S'[6..8]$과 $S'[2..5]$의 가장 긴 공통된 전치열을 루트로부터 찾는다. 루트에서 노드 v까지가 가장 긴 공통된 전치열이므로 그 길이만큼 스킵/카운트로 이동한 후, 비교결과 $가 불일치하므로 트리를 확장한다. s지점에서부터 하위 경로에 라벨된 문자열은 위의 경우에서 살펴본 바대로 루트에서부터 존재하기 때문에 스킵/카운트를

이용하여 루트로부터 라벨된 $S'[1..i-1]$의 각 서픽스의 끝에 $S'[i..m']$을 확장할 수 있으며, Append()가 종료되면 S'에 대한 서픽스 트리가 완성된다.

3.2 추가-서픽스 트리 재구성 알고리즘

완성된 서픽스 트리에 문자열을 추가하는 경우 서픽스 트리를 새로 만드는 방법을 사용한다면 문자열의 길이에 비례해서 많은 시간을 소비하게 되어 상당한 부하(overhead)가 된다. 이와 같은 문제점을 해결하기 위해서 구성된 서픽스 트리에 문자열을 추가하는 경우에 적용할 수 있는 추가-서픽스 트리 재구성 알고리즘을 제시한다. 예를 들어 [Fig. 9]와 같이 문자열 S= *adad$*의 서픽스 트리에서 S의 인덱스 2번부터 H=*dd*를 첨가한 S'= *adddad$*인 서픽스 트리로 재구성 할 수 있다.



[Fig. 9] Step of Insert-Suffix Tree Phase

추가-서픽스 트리 재구성 알고리즘 5단계로 실행된다. 먼저 1단계와 2단계는 삭제-서픽스 트리 재구성 알고리즘에서 제시한 Truncate( ), Edit_PathLabel( )를 이용한다. 3단계는 Ukkonen 알고리즘을 구현한 Ukkonen_Suffix_Tree( )를 사용해서 추가하려는 문자열 H를 서픽스 트리에 추가한다. 4단계는 앞장에서 설명한 Append( )를 이용해서 루트로부터 라벨된 문자열 H의 각 서픽스의 끝에 $S'[i+n+1..m']$을 확장한다. 마지막 5단계는 Append( )를 이용해서 루트로부터 라벨된 $S'[1..i]$의 각 서픽스의 끝에 $HS'[i+n+1..m]$을 추가해서 서픽스 트리를 확장한다. 길이가 m인 문자열 S에서 인덱스 i와 i+1사이에 길이 n인 문자열 H를 첨가하여 문자열 S'를 만들고 이를 이용해서 서픽스 트리 T'을 만든다. 추가-서픽스 트리 재구성 알고리즘의 의사 코드는 다음과 같다.

```
InsEdit_Suffix_Tree(T, S, i, i+1, m) {
    Truncate(T, S, i, i+1, m)
    Edit_PathLabel(T, i, i+1, m)
    Ukkonen_Suffix_Tree(T, H, n)
    Append(T, S', i+1, i+n+1, m')
    Append(T, S', 1, i+1, m')
}
```

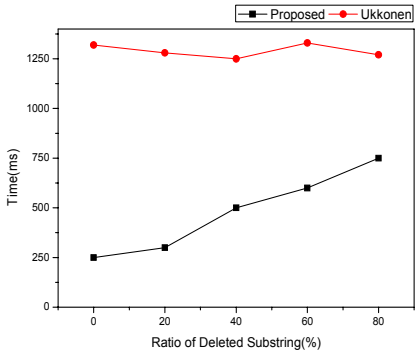
4. 알고리즘 평가

이 장에서는 서픽스 트리 재구성 알고리즘의 성능을 평가한다. 평가에 쓰인 표본 데이터는 DNA 유전자 서열인 Drosophila Realse 4 유전자 서열을 300kbp로 잘라서 사용하였다[13]. 추가-서픽스 트리 재구성 알고리즘에서 첨가할 문자열은 DNA 서열인 HMR195를 사용하였다[14, 15]. HMR195는 서열이 길이가 짧은 대신 종류가 많기 때문에 5만, 10만, 15만, 20만, 25만, 30만 길이의 서열로 조정하였다. 평가 방법은 표본 데이터로 만들어진 서픽스 트리에 서로 다른 길이의 부 문자열을 제거하는 경우와 HMR195 서열을 추가하는 경우로 나눠서 실험했으며, 각 실험에서 본 논문에서 제시한 알고리즘과 기존의 Ukkonen 알고리즘을 이용하여 서픽스 트리를 재구성하는 시간을 비교 측정하였다.

4.1 삭제-서픽스 트리 재구성 알고리즘 평가

표본 데이터를 이용해서 구성된 서픽스 트리에서 특정 위치에 있는 서로 다른 길이의 부 문자열을 제거하는

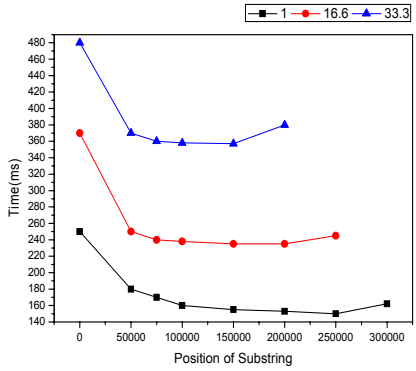
경우, 제안한 알고리즘을 이용해서 서픽스 트리를 재구성하는 데 필요한 시간과 기존의 Ukkonen 알고리즘으로 서픽스 트리를 재구성 하는 필요한 시간을 비교해 보았다. [Fig. 10]은 제거한 부 문자열의 길이 비율(%)에 따라 나타난 서픽스 트리의 구성 시간 변화를 그래프로 나타낸 것이다. 본 논문에서 제안한 삭제-서픽스 트리 재구성 알고리즘은(*Proposed*)이 기존의 Ukkonen알고리즘(*Ukkonen*)에 비해서 서픽스 트리를 재구성하는데 필요한 시간이 현저히 적음을 알 수 있다. 제거한 부 문자열의 비율이 80%가 되더라도 제안한 알고리즘의 시간이 평균 30%이상 절약된다. 이와 같은 시간절약은 DNA서열의 삽입, 삭제 실험이 빈번한 바이오 부분 응용에 매우 유용하다.



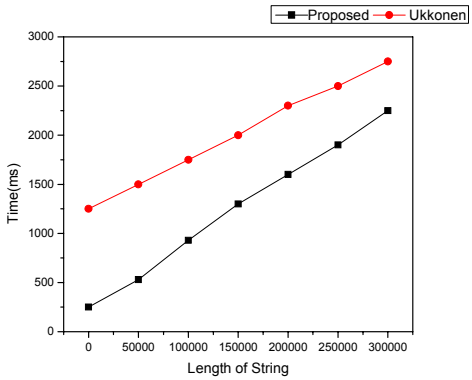
[Fig. 10] Comparison of Reconstructing Time at Sequence Delete

다음은 제안한 삭제-서픽스 트리 재구성 알고리즘에서 삭제하는 문자열의 위치가 트리 재구성 시간에 미치는 영향을 알아보았다. [Fig. 11]은 표본 데이터로 만든 서픽스 트리에서 각각 5만, 10만, 15만, 20만, 25만, 30만 번째 위치에서 표본 데이터의 1%(부 문자열 길이: 3000)와 16.6%(길이: 50,000) 그리고 33.3%(길이: 100,000)길이에 해당하는 부 문자열을 삭제할 때 필요한 시간을 나타낸다.

그 결과 제거하는 문자열의 위치가 1에서 5만번째 사이일 때 시간의 변화가 크게 나타났는데, 삭제-서픽스 트리 편집 알고리즘의 두 번째 단계인 Edit_PathLabel( )에서 비교적 많은 시간이 소요되었기 때문이다. 이와 같은 실험 결과를 통해서 삭제-서픽스 트리 편집 알고리즘은 전체 문자열 대비 50% 이하 길이의 부 문자열이 제거되었을 경우에 보다 효과적으로 사용할 수 있음을 알 수 있다.



[Fig. 11] Comparison of Reconstructing Time at Position of Deleted Substring



[Fig. 12] Comparison of Reconstructing Time at Sequence Insert

4.2 추가-서픽스 트리 재구성 알고리즘 평가

추가-서픽스 트리 재구성 알고리즘의 평가는 표본 데이터의 각 위치에 따라 첨가하는 HMR195의 길이를 달리하여 제안한 알고리즘의 시간(*Proposed*)을 측정하고, 이를 기존의 Ukkonen의 알고리즘으로 서픽스 트리를 새로 구성했을 때의 시간(*Ukkonen*)과 비교하였다.

[Fig. 12]는 표본 데이타에서 HMR195를 길이별로 첨가했을 때에 대한 서픽스 트리의 구성 시간을 그래프로 나타낸 것으로, *Proposed*와 *Ukkonen* 모두 첨가한 HMR195의 길이에 비례해서 시간이 증가한다. 하지만 제안한 알고리즘이 기존 알고리즘 보다 시간이 더 단축됨을 알 수 있다.

5. 결론

Ukkonen 알고리즘을 비롯한 기존의 서픽스 트리 응용연구는 부 문자열의 삽입 삭제가 발생하는 경우 서픽스 트리를 재 생성하기 위한 시간 낭비가 심한 문제점이 있었다. 더구나 DNA 데이터와 같이 대용량 문자열인 경우 길이에 비례해서 비효율성이 더 증가한다. 본 논문은 이러한 문제를 개선 할 수 있도록 트리 재구성 시간을 절약할 수 있는 서픽스 트리 재구성 알고리즘을 제시하였다. 제안하는 알고리즘은 부 문자열을 삽입하는 경우와 삭제하는 경우로 나눈 다음, 삽입/삭제 시 발생할 수 있는 경우의 수를 감안하여 알고리즘을 설계하였다.

제안하는 알고리즘을 평가하기 위해서 DNA 서열로 서픽스 트리를 구성하고 상이한 길이의 문자열을 삽입/삭제하는 실험을 통해 기존 알고리즘과 비교하였다. 그 결과 부 문자열을 제거하는 경우에 제안한 알고리즘의 시간이 평균 30%이상 절약됨을 알 수 있었다. 또한 문자열을 삽입하는 경우에도 제안한 알고리즘이 Ukkonen 알고리즘으로 트리를 새로 구성할 때보다 시간이 단축됨을 알 수 있었다.

본 논문이 제안한 서픽스 트리 재구성 알고리즘은 DNA 서열의 분석뿐만 아니라 RNA, 단백질 등 다른 생물학적 서열(biological sequence)에 대해서도 적용가능하다. 추가적으로 실제 DNA 서열은 그 일부가 제거되거나 첨가될 뿐만 아니라, 역위, 치환(displacement), 중복(duplication)등 다양한 돌연변이에 의해 변경된다. 그러므로 향후에는 DNA의 다양한 돌연변이를 감안한 서픽스 트리 재구성 알고리즘 연구가 요구된다.

REFERENCES

[1] D. Gusfield, Algorithm on String, Tree, and Sequence, Cambridge University Press, pp. 87- 107, 1997.

[2] David W., Bioinformatics, sequences and Genome Analysis, MOUNT Press, 2001.

[3] Younshin Oh, Dinh Truong Nguyen, Identification of 1,531 cSNPs from Full-length Enriched cDNA Libraries of the Korean Native Pig Using in Silico Analysis, Genomics & Informatics, vol. 7, no. 2, 2009, pp. 65-84.

[4] Josue Samayoa1, Fitnat H. Yildiz and Kevin Karplus, Identification of prokaryotic small proteins using a comparative genomic approach, Bioinformatics, vol.27, no.13, 2011, pp. 1765-1771.

[5] Chan Park, Ji-Seong Jeong, Design and Implementation of Bio-Medical Data Measurement System through the Stereo Microscope, Korea Contents Association KISTI-KOCON ICCC2009, November, vol.7, no.2, 2009, pp. 357-360.

[6] Young-Ohk Song, Sung-young Kim and Duk-Jin Chang, Design of the System and Algorithm for the Pattern Analysis of the Bio-Data, Korea Contents Association, November, vol.10, no.8, 2008, pp. 104-110.

[7] Audry P. G., Alan M.M., Conservation and Evolution of Cis-Regulatory Systems in Ascomycete Fungi, PLOS Biology, vol. 2, no. 12, 2004, pp. 398-405.

[8] Ketil Malde, Eivind Coward and Inge Jonassen, Fast sequence clustring using suffix array algorithm, Bioinformatics, 2003, pp. 1221-1226.

[9] E. Ukkonen. On-line construction of suffix-trees. Algorithmica, 1995, pp. 249-67.

[10] E.M. McCreight, A Space-Economical Suffix Tree Construction Algorithm. Journal of the ACM, vol. 23, no.2, 1976, pp. 262-272.

[11] R.A.Gibbs and S.Kurtz, From Ukkonen to McCreight and Weiner: A unifying view of linear-time suffix tree construction, Algorithmica. 1997.

[12 William S.Klug and Michael R.Cummings, Genetics. Sixth Edition, pp. 251-281.

[13] Uwe Ohler, Promoter Prediction on a Genomic scale-the Adh Experience, Preprint from Genomes Res. 2000, pp. 539-542.

[14] Ogasawara, j. and Morishita, S. Fast and Sensitive Algorithm for Aligning Ests to Human Genome. Bioinfomatics Conference, Proceedings. IEEE Computer Society, 2002, pp. 43-53.

[15] Mark Nelson, “Fast String Searching With Suffix Trees”, Dr. Dobb's Journal, 1996.

최 해 원(Choi, Hae Won)

- 1996년 2월 : 경일대학교 컴퓨터공학과(공학사)
- 2000년 2월 : 경북대학교 컴퓨터공학과(공학석사)
- 2009년 2월 : 경북대학교 컴퓨터공학과(공학박사)
- 2006년 3월 ~ 현재 : 경운대학교 컴퓨터공학과 조교수
- 관심분야 : 알고리즘, 유비쿼터스 컴퓨팅
- E-Mail : happy9950@ikw.ac.kr

정 영 석(Jung, Young-Seok)

- 1995년 2월 : 영남대학교 컴퓨터학과(공학사)
- 1998년 2월 : 영남대학교 컴퓨터공학과(공학석사)
- 2003년 2월 : 영남대학교 컴퓨터공학과(공학박사)
- 2000년 3월 ~ 현재 : 경운대학교 컴퓨터공학과 교수
- 관심분야 : Ad Hoc Network, 컴퓨터네트워크 프로토콜
- E-Mail : ysjung@ikw.ac.kr

김 상 진(Kim, Sang-Jin)

- 1994년 2월 : 계명대학교 컴퓨터공학과 (공학사)
- 1996년 2월 : 경북대학교 컴퓨터공학과 (공학석사)
- 2000년 8월 : 경북대학교 컴퓨터공학과 (공학박사)
- 1999년 9월 ~ 현재 : 경운대학교 컴퓨터공학과 교수
- 관심분야 : 알고리즘, 게임이론
- E-Mail : sjkim@ikw.ac.kr