Nordic Journal of Computing

BIDIRECTIONAL CONSTRUCTION OF SUFFIX TREES

Shunsuke Inenaga
Department of Informatics, Kyushu University 33, Fukuoka 812-8581, Japan
s-ine@i.kyushu-u.ac.jp

Abstract. String matching is critical in information retrieval since in many cases information is stored and manipulated as strings. Constructing and utilizing a suitable data structure for a text string, we can solve the string matching problem efficiently. Such a structure is called an *index structure*. *Suffix trees* are certainly the most widely-known and extensively-studied structure of this kind. In this paper, we present a linear-time algorithm for bidirectional construction of suffix trees. The algorithm proposed is also applicable to bidirectional construction of *directed acyclic word graphs* (*DAWGs*).

CR Classification: E.1 [Data Structures]; F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems

Key words: strings, pattern matching, data structures, suffix trees, DAWGs, linear-time algorithm

1. Introduction

String processing is of central importance to computer science, since most data available in/via computers are stored and manipulated as strings. Therefore efficient algorithms for string processing are significant and necessary. Pattern matching is the most fundamental and important problem in string processing, and it is described as follows: given a pattern string p and a text string w, examine whether or not p is a substring of w. This problem is solvable in $O(|p|)$ time by using a suitable data structure that supports indices of text w.

The most basic index structure seems to be *suffix tries*. All substrings of a given string w are recognized at nodes of the suffix trie of w. Probably the structure is the easiest to understand, but its only, however biggest, drawback is that its space requirement is $O(|w|^2)$.

This fact led the introduction of more space-economical ($O(|w|)$-spaced) structures such as *suffix trees* (see Weiner [1973], McCreight [1976], Ukkonen [1995], Gusfield [1997]), *directed acyclic word graphs* (*DAWGs*) (see Blumer *et al.* [1985], Crochemore [1986], Balík [1998]), *compact directed acyclic word graphs* (*CDAWGs*) (see Blumer *et al.* [1987], Crochemore and Vérin [1997], Inenaga *et al.* [2001b]), suffix arrays (see Manber and Myers [1993]),

Received April 2, 2003.

and some other variants. Among those, suffix trees are for sure most widely-known and extensively-studied (see Crochemore and Rytter [1994], Gusfield [1997]), perhaps because there are a 'myriad' of applications for them as shown by Apostolico [1985].

Construction of suffix trees has been studied in various contexts: Weiner [1973] invented the first algorithm that constructs suffix trees in linear time; McCreight [1976] proposed a more space-economical algorithm; Chen and Seiferas [1985] showed an efficient modification of Weiner's; Ukkonen [1995] introduced an on-line algorithm to construct suffix trees, which Giegerich and Kurtz [1997] regarded as "the most elegant"; Farach [1997] considered optimal construction of suffix trees with large alphabets; Breslauer [1998] gave a linear-time algorithm for building the suffix tree of a given trie that stores a set of strings; Inenaga *et al.* [2001a] presented an on-line algorithm that simultaneously constructs both the suffix tree of a string and the DAWG of the reversed string.

In this paper we explore *bidirectional construction* of suffix trees. Namely, the algorithm we propose allows us to update the suffix tree of a string w to the suffix tree of a string xwy, where x, y are any strings. We also show that our algorithm runs in linear time and space with respect to the length of a given string. A preliminary version of this work appears in Inenaga [2002].

Bidirectional construction of suffix trees was first considered by Stoye [1995]. His strategy was to modify the definition and structure of suffix trees so that they become more "adequate" in terms of bidirectional construction, and the resulting modified structure was named *affix trees*. His original algorithm to construct affix trees does not perform in linear time, unfortunately, but Maaß [2000] later on improved the algorithm so as to run in linear time. Another good feature of affix trees is that the affix tree of any string w also supports the indices of w^{rev}, the reversal of w. On the other hand, it is a well-known and beautiful property that the suffix tree of any string w and the DAWG of w^{rev} can share the same nodes, as well. We will show that the combination of this work and our previous work in Inenaga *et al.* [2001a] enables us to construct and update both structures in a bidirectional manner.

The size of affix trees is, of course, linear. However, for any string w the number of nodes in the suffix tree for w is less than or equal to that of the affix tree. Namely, this work contributes to reducing space requirements necessary for bidirectional construction of a data structure that supports dual indices of a given string.

2. Suffix Trees

Let Σ be a finite alphabet. An element of Σ^* is called a *string*. Strings x, y, and z are said to be a *prefix*, *substring*, and *suffix* of string $w = xyz$, respectively. The sets of prefixes, substrings, and suffixes of a string w are denoted by $\mathit{Prefix}(w)$, $\mathit{Substr}(w)$, and $\mathit{Suffix}(w)$, respectively. The length of

a string w is denoted by $|w|$. The empty string is denoted by ε, that is, $|\varepsilon| = 0$. Let $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. The i-th character of a string w is denoted by $w[i]$ for $1 \leq i \leq |w|$. The cardinality of set $S \subseteq \Sigma^*$ is denoted by $|S|$. For any string $u \in \Sigma^*$, $Su^{-1} = \{x \mid xu \in S\}$.

Let $w \in \Sigma^*$. We define an equivalence relation $\equiv_w^L$ on Σ^* by

$$x \equiv_w^L y \Leftrightarrow \mathit{Prefix}(w)x^{-1} = \mathit{Prefix}(w)y^{-1}.$$

The equivalence class of a string $x \in \Sigma^*$ with respect to $\equiv_w^L$ is denoted by $[x]_w^L$. Note that all strings not belonging to $\mathit{Substr}(w)$ form one equivalence class under $\equiv_w^L$. This equivalence class is called the *degenerate* class. All other classes are said to be *non-degenerate*.

EXAMPLE 1. Let $w = \mathtt{abcbc}$. For example, $\mathtt{b} \equiv_w^L \mathtt{bc}$ since $\mathit{Prefix}(w)\mathtt{b}^{-1} = \mathit{Prefix}(w)(\mathtt{bc})^{-1} = \{\mathtt{abc}, \mathtt{a}\}$.

All non-degenerate equivalence classes under $\equiv_w^L$ are $[\varepsilon]_w^L = \{\varepsilon\}$, $[\mathtt{a}]_w^L = \{\mathtt{abcbc}, \mathtt{abcb}, \mathtt{abc}, \mathtt{ab}, \mathtt{a}\}$, $[\mathtt{b}]_w^L = \{\mathtt{bc}, \mathtt{b}\}$, $[\mathtt{bcb}]_w^L = \{\mathtt{bcbc}, \mathtt{bcb}\}$, $[\mathtt{c}]_w^L = \{\mathtt{c}\}$, and $[\mathtt{cb}]_w^L = \{\mathtt{cbc}, \mathtt{cb}\}$.

PROPOSITION 1. (INENAGA *et al.* [2001A]) *Let $w \in \Sigma^*$ and $x, y \in \mathit{Substr}(w)$. If $x \equiv_w^L y$, then either x is a prefix of y, or vice versa.*

PROOF. By the definition of $\equiv_w^L$, we have $\mathit{Prefix}(w)x^{-1} = \mathit{Prefix}(w)y^{-1}$. There are three cases to consider:

(1) When $|x| = |y|$. Obviously, $x = y$ in this case. Thus $x \in \mathit{Prefix}(y)$ and $y \in \mathit{Prefix}(x)$.

(2) When $|x| > |y|$. Let u be an arbitrary string in $\mathit{Prefix}(w)$. Assume $u = sx$ with $s \in \Sigma^*$. Then $s \in \mathit{Prefix}(w)x^{-1}$, which results in $s \in \mathit{Prefix}(w)y^{-1}$. Hence, there must exist a string $v \in \mathit{Prefix}(w)$ such that $v = sy$. By the assumption that $|x| > |y|$, we have $|u| > |v|$. From the fact that both u and v are in $\mathit{Prefix}(w)$, it is derived that $v \in \mathit{Prefix}(u)$. Consequently, $y \in \mathit{Prefix}(x)$.

(3) When $|x| < |y|$. By a similar argument to Case (2), we have $x \in \mathit{Prefix}(y)$.

□

For any string $x \in \mathit{Substr}(w)$, the longest member in $[x]_w^L$ is denoted by $\overrightarrow{x}^w$. What $\overrightarrow{x}^w$ means intuitively is that $\overrightarrow{x}^w$ is the string obtained by extending x in $[x]_w^L$ as long as possible. The following proposition states that each equivalence class in $\equiv_w^L$ other than the degenerate class has a unique longest member.

PROPOSITION 2. (INENAGA *et al.* [2001A]) *Let $w \in \Sigma^*$. For any string $x \in \mathit{Substr}(w)$, there uniquely exists a string $\alpha \in \Sigma^*$ such that $\overrightarrow{x}^w = x\alpha$.*

PROOF. Let $\overset{w}{\overrightarrow{x}} = x\alpha$ with $\alpha \in \Sigma^*$. For the contrary, assume there exists a string $\beta \in \Sigma^*$ such that $\overset{w}{\overrightarrow{x}} = x\beta$ and $\beta \neq \alpha$. By PROPOSITION 1, either $x\alpha \in \mathit{Prefix}(x\beta)$ or $x\beta \in \mathit{Prefix}(x\alpha)$ must stand, since $x\alpha \equiv_w^L x\beta$. However, neither of them actually holds since $|\alpha| = |\beta|$ and $\alpha \neq \beta$, which yields a contradiction. Hence, α is the only string satisfying $\overset{w}{\overrightarrow{x}} = x\alpha$. □

PROPOSITION 3. *Let $w \in \Sigma^*$ and $x \in \mathit{Substr}(w)$. Assume $\overset{w}{\overrightarrow{x}} = x$. Then, for any $y \in \mathit{Suffix}(x)$, $\overset{w}{\overrightarrow{y}} = y$.*

PROOF. Assume contrarily that there uniquely exists a string $\alpha \in \Sigma^+$ such that $\overset{w}{\overrightarrow{y}} = y\alpha$. Since $y \in \mathit{Suffix}(x)$, x is always followed by α in w. It implies that $\mathit{Prefix}(w)x^{-1} = \mathit{Prefix}(w)(x\alpha)^{-1}$, and therefore we have $x \equiv_w^L x\alpha$. Since $|\alpha| > 0$, $\overset{w}{\overrightarrow{x}}$ is not the longest in $[x]_w^L$, which is a contradiction. Hence, $\overset{w}{\overrightarrow{y}} = y$. □

PROPOSITION 4. *Let $w \in \Sigma^*$. For any string $x \in \mathit{Suffix}(w)$, $\overset{w}{\overrightarrow{x}} = x$.*

PROOF. By PROPOSITION 2 there uniquely exists a string $\alpha \in \Sigma^*$ such that $\overset{w}{\overrightarrow{x}} = x\alpha$. Since $x \in \mathit{Suffix}(w)$, $\alpha = \varepsilon$. □

Note that, for a string $w \in \Sigma^*$, $|\mathit{Substr}(w)| = O(|w|^2)$. For example, consider string $a^n b^n$. However, considering set $S = \{x \mid x \in \mathit{Substr}(w) \text{ and } x = \overset{w}{\overrightarrow{x}}\}$, we have $|S| = O(|w|)$. The following lemma gives a tighter upper-bound.

LEMMA 1. (BLUMER *et al.* [1985, 1987]) *Assume that $|w| > 1$. The number of the non-degenerate equivalence classes in $\equiv_w^L$ is at most $2|w| - 1$.*

A definition of the suffix tree of a string $w \in \Sigma^*$, denoted by $\mathit{STree}(w)$, on the basis of the above-mentioned equivalence classes, is following. We define it as an edge-labeled tree (V, E) with $E \subseteq V \times \Sigma^+ \times V$ where the second component of each edge represents its label. We also give a definition of the *suffix links*, kinds of failure functions, frequently utilized for time-efficient construction of suffix trees (see McCreight [1976], Ukkonen [1995]).

DEFINITION 1. *$\mathit{STree}(w)$ is the tree (V, E) such that*

$$V = \{\overset{w}{\overrightarrow{x}} \mid x \in \mathit{Substr}(w)\},$$
$$E = \{(\overset{w}{\overrightarrow{x}}, a\beta, \overset{w}{\overrightarrow{xa}}) \mid x, xa \in \mathit{Substr}(w),\ a \in \Sigma,\ \beta \in \Sigma^*,\ \overset{w}{\overrightarrow{xa}} = xa\beta,\ \overset{w}{\overrightarrow{x}} \neq \overset{w}{\overrightarrow{xa}}\},$$

and its suffix links are the set

$$F = \{(\overset{w}{\overrightarrow{ax}}, \overset{w}{\overrightarrow{x}}) \mid x, xa \in \mathit{Substr}(w),\ a \in \Sigma,\ \overset{w}{\overrightarrow{ax}} = a \cdot \overset{w}{\overrightarrow{x}}\}.$$

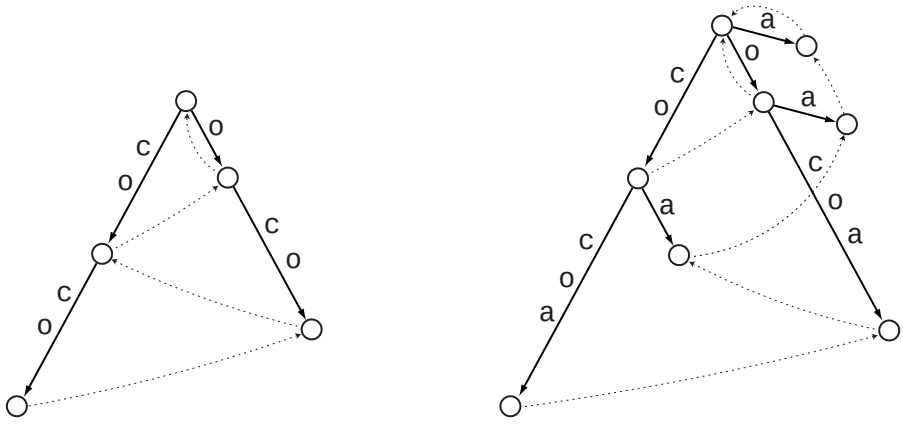

Fig. 2.1: *STree*(coco) on the left, and *STree*(cocoa) on the right. Solid arrows represent the edges, and dotted arrows denote suffix links.

The node $\overset{w}{\overrightarrow{\varepsilon}} = \varepsilon$ is called the *root* node of *STree*(w). When a node $\overset{w}{\overrightarrow{x}}$ is of out-degree zero (i.e. there is no character a such that $\overset{w}{\overrightarrow{x}} \cdot a = \mathit{Substr}(w)$), then it is said to be a *leaf* node. Each leaf node corresponds to a string in *Suffix*(w). If $x \in \mathit{Substr}(w)$ satisfies $x = \overset{w}{\overrightarrow{x}}$, x is said to be represented on an *explicit* node $\overset{w}{\overrightarrow{x}}$. If $x \neq \overset{w}{\overrightarrow{x}}$, x is said to be on an *implicit* node. *STree*(coco) and *STree*(cocoa) are displayed in Fig. 2.1.

It derives from Lemma 1 that:

Theorem 2.1. (McCreight [1976]) *Let $w \in \Sigma^*$. Let $\mathit{STree}(w) = (V, E)$. Assume $|w| > 1$. Then $|V| \leq 2|w| - 1$ and $|E| \leq 2|w| - 2$.*

Both algorithms by Weiner [1973] and by McCreight [1976] construct the suffix tree defined above, *STree*(w). On the other hand, the algorithm by Ukkonen [1995] constructs a slightly different version, which is modified so as to be suitable for his on-line algorithm.

As a preliminary to define the modified suffix tree, we firstly introduce a relation X_w over Σ^* such that

$$X_w = \{(x, xa) \mid x \in \mathit{Substr}(w) \text{ and } a \in \Sigma \text{ is unique s.t. } xa \in \mathit{Substr}(w)\}.$$

Let $\equiv'^L_w$ be the equivalence closure of X_w, i.e., the smallest superset of X_w that is symmetric, reflexive, and transitive.

Proposition 5. (Inenaga *et al.* [2001a]) *For any string $w \in \Sigma^*$, $\equiv^L_w$ is a refinement of $\equiv'^L_w$.*

Proof. Let x, y be any strings in *Substr*(w) and assume $x \equiv^L_w y$. According to Proposition 1, we firstly assume that $x \in \mathit{Prefix}(y)$. It follows from Proposition 2 that there uniquely exist strings $\alpha, \beta \in \Sigma^*$ such that $\overset{w}{\overrightarrow{x}} = x\alpha$

and $\overrightarrow{y}^{w} = y\beta$. Note that $\beta \in \mathit{Suffix}(\alpha)$. Let $\gamma \in \Sigma^*$ be the string satisfying $\alpha = \gamma\beta$. Then γ is the sole string such that $x\gamma = y$. By the definition of $\equiv'^{L}_{w}$, we have $x \equiv'^{L}_{w} y$. A similar argument holds in case that $y \in \mathit{Prefix}(x)$. □

COROLLARY 1. (INENAGA *et al.* [2001A]) *For any $w \in \Sigma^*$, every equivalence class under $\equiv'^{L}_{w}$ is a union of one or more equivalence classes under $\equiv^{L}_{w}$.*

The equivalence class of a string $x \in \Sigma^*$ with respect to $\equiv'^{L}_{w}$ is denoted by $[x]'^{L}_{w}$.

EXAMPLE 2. Let $w = \mathtt{abcbc}$. All equivalence classes in $\equiv'^{L}_{w}$ are $[\varepsilon]'^{L}_{w} = \{\varepsilon\}$, $[\mathtt{a}]'^{L}_{w} = \{\mathtt{abcbc}, \mathtt{abcb}, \mathtt{abc}, \mathtt{ab}, \mathtt{a}\}$, $[\mathtt{b}]'^{L}_{w} = \{\mathtt{bcbc}, \mathtt{bcb}, \mathtt{bc}, \mathtt{b}\}$, and $[\mathtt{c}]'^{L}_{w} = \{\mathtt{cbc}, \mathtt{cb}, \mathtt{c}\}$.

Note the differences between the above example and EXAMPLE 1.

The longest member of $[x]'^{L}_{w}$ is denoted by $\overrightarrow{\overrightarrow{x}}^{w}$.

The next proposition is an alternate of $\overrightarrow{\overrightarrow{(\cdot)}}^{w}$ to PROPOSITION 3 with respect to $\overrightarrow{(\cdot)}^{w}$.

PROPOSITION 6. *Let $w \in \Sigma^*$ and $x \in \mathit{Substr}(w) - \mathit{Suffix}(w)$. Assume $\overrightarrow{\overrightarrow{x}}^{w} = x$. Then, for any $y \in \mathit{Suffix}(x)$, $\overrightarrow{\overrightarrow{y}}^{w} = y$.*

PROOF. Since $\overrightarrow{\overrightarrow{x}}^{w} = x$ and $x \notin \mathit{Suffix}(w)$, there are at least two characters $a, b \in \Sigma$ such that $xa, xb \in \mathit{Substr}(w)$ and $a \neq b$. Since $y \in \mathit{Suffix}(x)$, y is also followed by both a and b in the string w. Thus $\overrightarrow{\overrightarrow{y}}^{w} = y$. □

Remark that the precondition of the above proposition slightly differs from that of PROPOSITION 3. Namely, when x is a suffix of w, this proposition does not always hold.

From here on, we explore some relationships between $\overrightarrow{(\cdot)}^{w}$ and $\overrightarrow{\overrightarrow{(\cdot)}}^{w}$.

LEMMA 2. (INENAGA *et al.* [2001A]) *Let $w \in \Sigma^*$. For any $x \in \mathit{Substr}(w)$, $\overrightarrow{x}^{w}$ is a prefix of $\overrightarrow{\overrightarrow{x}}^{w}$. If $\overrightarrow{x}^{w} \neq \overrightarrow{\overrightarrow{x}}^{w}$, then $\overrightarrow{x}^{w} \in \mathit{Suffix}(w)$.*

PROOF. We can prove that $\overrightarrow{x}^{w} \in \mathit{Prefix}(\overrightarrow{\overrightarrow{x}}^{w})$ by PROPOSITION 1 and COROLLARY 1. Now suppose $\overrightarrow{x}^{w} \neq \overrightarrow{\overrightarrow{x}}^{w}$. Let $\overrightarrow{x}^{w} = x\beta$ with $\beta \in \Sigma^+$. Supposing $\overrightarrow{\overrightarrow{x}}^{w} = x\alpha$ with $\alpha \in \Sigma^+$, we have $\beta \in \mathit{Prefix}(\alpha)$. Let $\beta\gamma = \alpha$ with $\gamma \in \Sigma^*$. By the assumption $\overrightarrow{x}^{w} \neq \overrightarrow{\overrightarrow{x}}^{w}$, we have $x\beta \not\equiv^{L}_{w} x\alpha$, although γ is the sole string that

follows $\overset{w}{\overrightarrow{x}}$ in w since $\overset{w}{\Longrightarrow}{x} = x\alpha = x\beta\gamma = \overset{w}{\overrightarrow{x}}\cdot\gamma$. This means that x is a suffix of w followed by *no* character. □

See Example 1 and Example 2 to confirm the above lemma.

Lemma 3. *Let $w \in \Sigma^*$ and $x \in Suffix(w)$. If $x \notin Prefix(y)$ for any string $y \in Substr(w) - \{x\}$, then $\overset{w}{\overrightarrow{x}} = \overset{w}{\Longrightarrow}{x}$.*

Proof. The precondition implies that there is no character $a \in \Sigma$ satisfying $xa \in Substr(w)$. Thus we have $\overset{w}{\Longrightarrow}{x} = x$. On the other hand, we obtain $\overset{w}{\overrightarrow{x}} = x$ by Proposition 4, because $x \in Suffix(w)$. Hence $\overset{w}{\overrightarrow{x}} = \overset{w}{\Longrightarrow}{x}$. □

Lemma 4. *Let $w \in \Sigma^*$ with $|w| = n$. Assume that the last character $w[n]$ is unique in w, that is, $w[n] \neq w[i]$ for any $1 \leq i \leq n-1$. Then, for any string $x \in Substr(w)$, $\overset{w}{\overrightarrow{x}} = \overset{w}{\Longrightarrow}{x}$.*

Proof. By the contraposition of the second statement of Lemma 2, if $x \notin Suffix(w)$, then $\overset{w}{\overrightarrow{x}} = \overset{w}{\Longrightarrow}{x}$. Because of the unique character $w[n]$, any suffix z of w satisfies the precondition of Lemma 3, and thus $\overset{w}{\overrightarrow{z}} = \overset{w}{\Longrightarrow}{z}$. □

We are now ready to define $STree'(w)$, which is a modified version of $STree(w)$.

Definition 2. *$STree'(w)$ is the tree (V, E) such that*

$$V = \{\overset{w}{\Longrightarrow}{x} \mid x \in Substr(w)\},$$

$$E = \{(\overset{w}{\Longrightarrow}{x}, a\beta, \overset{w}{\Longrightarrow}{xa}) \mid x, xa \in Substr(w),\ a \in \Sigma,\ \beta \in \Sigma^*,\ \overset{w}{\Longrightarrow}{xa} = xa\beta,\ \overset{w}{\Longrightarrow}{x} \neq \overset{w}{\Longrightarrow}{xa}\},$$

and its suffix links are the set

$$F = \{(\overset{w}{\Longrightarrow}{ax}, \overset{w}{\Longrightarrow}{x}) \mid x, xa \in Substr(w),\ a \in \Sigma,\ \overset{w}{\Longrightarrow}{ax} = a\cdot\overset{w}{\Longrightarrow}{x}\}.$$

Remark that $STree'(w)$ can be obtained by replacing $\overset{w}{\overrightarrow{(\cdot)}}$ in $STree(w)$ with $\overset{w}{\Longrightarrow}(\cdot)$. $STree'(\texttt{coco})$ and $STree'(\texttt{cocoa})$ are shown in Fig. 2.2.

We have the next corollary deriving from Lemma 4.

Corollary 2. *Let $w \in \Sigma^*$ with $|w| = n$. Assume that the last character $w[n]$ is unique in w, that is, $w[n] \neq w[i]$ for any $1 \leq i \leq n-1$. Then, $STree(w) = STree'(w)$.*

Indeed, $STree(\texttt{cocoa})$ in Fig. 2.1 and $STree'(\texttt{cocoa})$ in Fig. 2.2 are the same, where the last character `a` is unique in string `cocoa`.

According to Corollary 2, using an end-marker \$ that occurs nowhere in w, we have $STree(w\$) = STree'(w\$)$ for any $w \in \Sigma^*$.

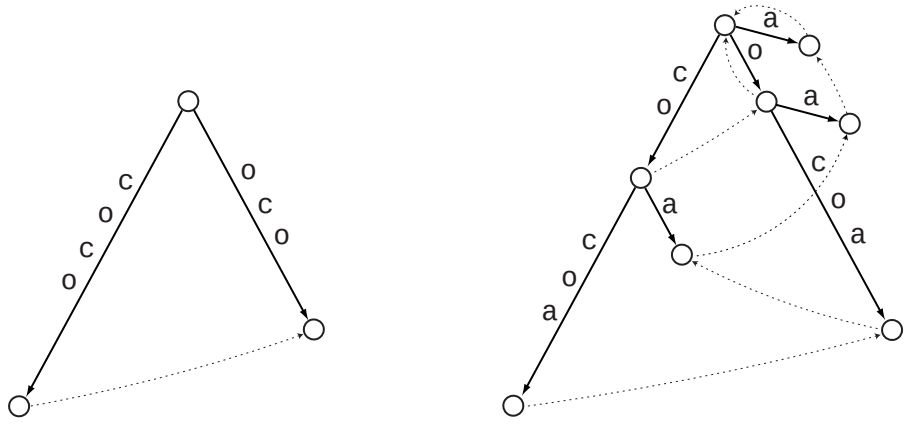

Fig. 2.2: *STree′*(coco) on the left, and *STree′*(cocoa) on the right. Solid arrows represent the edges, and dotted arrows denote suffix links.

3. Bidirectional Construction of Suffix Trees

3.1 Right Extension

Assume that we have $STree'(w)$ with some $w \in \Sigma^*$. Now we consider updating it into $STree'(wa)$ with $a \in \Sigma$, by inserting the suffixes of wa into $STree'(w)$. Ukkonen [1995] achieved the following result.

THEOREM 3.1. (UKKONEN [1995]) *For any $a \in \Sigma$ and $w \in \Sigma^*$, $STree'(w)$ can be updated to $STree'(wa)$ in amortized constant time.*

The construction of $STree'$(cocoa) with right extension is shown in Fig. 3.3.

Here we only recall essence of Ukkonen's algorithm together with some supporting lemmas and propositions.

Let y be the longest string in $Substr(w) \cap Suffix(wa)$. Then y is called the *longest repeated suffix* of wa and denoted by $LRS(wa)$. Since every string $x \in Suffix(y)$ belongs to $Substr(w)$, we do not need to newly insert any x into $STree'(w)$.

LEMMA 5. *Let $a \in \Sigma$ and $w \in \Sigma^*$. Let $y = LRS(w)$. For any string $x \in Suffix(w) - Suffix(y)$, $\overset{wa}{\Longrightarrow} x = \overset{w}{\Longrightarrow} x \cdot a$.*

PROOF. Since $y = LRS(w)$, any string $x \in Suffix(w) - Suffix(y)$ appears only once in w as a suffix of w, and is therefore $\overset{w}{\Longrightarrow} x = x$. Also, x is followed only by a in wa, and thus $\overset{wa}{\Longrightarrow} x = xa$. □

The above lemma implies that a leaf node of $STree'(w)$ is also a leaf node in $STree'(wa)$. Thus we need no explicit maintenance for leaf nodes. Namely,

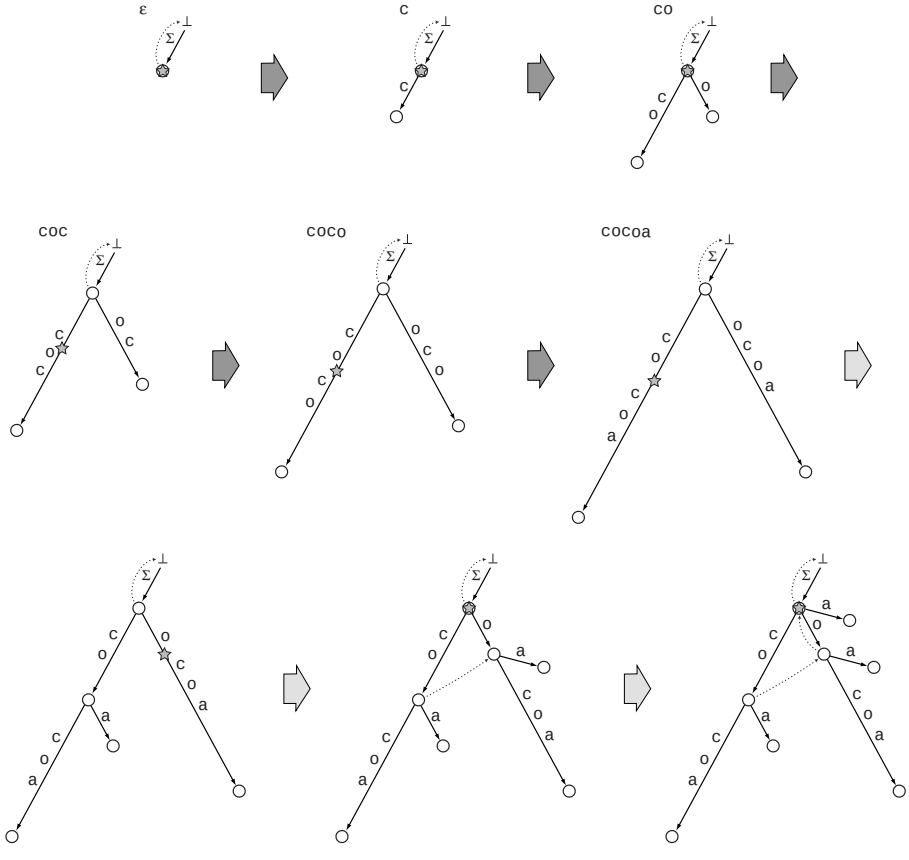

Fig. 3.3: The construction of $STree'(w)$ with right extension, where $w = \texttt{cocoa}$. The star mark represents the longest repeated suffix of each suffix tree. The $\perp$ node corresponds to the eliminator symbol ξ. The Σ symbol represents *any* character in the alphabet. The suffix links of all leaf nodes are omitted here, because they are not really necessary in the algorithm.

we can insert all strings of $Suffix(w) - Suffix(y)$ into $STree'(w)$ *automatically* (for more detail, see Ukkonen [1995]). Now we can focus only on the suffixes of $LRS(wa)$.

PROPOSITION 7. *Let $a \in \Sigma$ and $w \in \Sigma^*$. Let $y = LRS(w)$ and $z = LRS(wa)$. For any string $x \in Suffix(y) - Suffix(z)a^{-1}$, $\overset{wa}{\Longrightarrow} x = x$.*

PROOF. Since $x \in Suffix(y)$, x has appeared at least twice in w. Because $x \notin Suffix(z)a^{-1}$, $xa \notin Suffix(z)$. These facts imply the existence of $b \in \Sigma$ such that $xb \in Substr(w)$ and $b \neq a$. Consequently, we have $\overset{wa}{\Longrightarrow} x = x$. □

The above proposition implies that if $x \in Suffix(y)$, $\overset{w}{\Longrightarrow} x \neq x$ ($\overset{w}{\Longrightarrow} x$ is implicit in $STree'(w)$), and $\overset{wa}{\Longrightarrow} x = x$ ($\overset{wa}{\Longrightarrow} x$ will be explicit in $STree'(wa)$), a new explicit

node $\overset{wa}{\Longrightarrow} x = x$ has to be created in the update of $STree'(w)$ to $STree'(wa)$. Plus, a new leaf node $\overset{wa}{\Longrightarrow} xa = xa$ is created with the new edge $(\overset{wa}{\Longrightarrow} x, a, \overset{wa}{\Longrightarrow} xa)$. Now the next question is how to detect where to create the new explicit node $\overset{aw}{\Longrightarrow} x$ in the suffix tree.

We here define the *eliminator* ξ for any character $a \in \Sigma$ by

$$a\xi = \xi a = \varepsilon$$

and $|\xi| = -1$. Moreover, we define that $\xi \in \mathit{Prefix}(\varepsilon)$ and $\xi \in \mathit{Suffix}(\varepsilon)$, but $\xi \notin \mathit{Prefix}(x)$ and $\xi \notin \mathit{Suffix}(x)$ for any $x \in \Sigma^+$. The symbol ξ corresponds to the auxiliary node $\perp$ introduced by Ukkonen [1995]. Owing to the introduction of ξ, we can establish the following lemma.

LEMMA 6. *Let $a \in \Sigma$ and $w \in \Sigma^*$. Let $y = \mathit{LRS}(w)$ and $z = \mathit{LRS}(wa)$. Assume $x \in \mathit{Suffix}(y) - \mathit{Suffix}(z)a^{-1}$. Suppose t is the longest string in $\mathit{Prefix}(x)$ such that $\overset{w}{\Longrightarrow} t = t$. Let $x' = \mathit{Suffix}(x)$ with $|x'| + 1 = |x|$, and $t' = \mathit{Suffix}(t)$ with $|t'| + 1 = |t|$. For the string $\alpha \in \Sigma^*$ such that $t\alpha = x$, $t'\alpha = x'$.*

Notice that we can reach string x' via the suffix link of the node for t in $STree'(w)$ and along the path spelling out α from the node for t' (recall DEFINITION 2). For instance, see the 1st and 2nd phases for `cocoa` in Fig. 3.3. After creating new explicit nodes $\overset{w}{\Longrightarrow} \mathtt{co}$ and $\overset{w}{\Longrightarrow} \mathtt{coa}$, the star mark goes backward to the parent node of $\overset{w}{\Longrightarrow} \mathtt{co}$, which is the root node $\overset{w}{\Longrightarrow} \varepsilon$. Then it moves to $\perp$ node via the suffix link of $\overset{w}{\Longrightarrow} \varepsilon$, and goes down along edges with spelling out `co`. The star mark is now on the location for `o`, where a new explicit node will be created in the next phase. This operation is continued until the star mark reaches $\mathit{LRS}(\mathtt{cocoa})$. Ukkonen [1995] proved that the amortized cost of this operation is constant, on the assumption that every edge label α of $STree'(w)$ is actually implemented by a pair (i, j) of integers such that the substring of w beginning at position i and ending at position j is α.

3.2 Left Extension

Weiner [1973] proposed an algorithm to construct $STree(aw)$ by updating $STree(w)$ with $a \in \Sigma$ in amortized constant time. On the other hand, what we treat in this section is the conversion of $STree'(w)$ into $STree'(aw)$. From here on we delve in what happens to $STree'(w)$ when updated to $STree'(aw)$.

LEMMA 7. *Let $a \in \Sigma$ and $w \in \Sigma^*$. For any string $x \in \mathit{Substr}(w) - \mathit{Prefix}(aw)$, $\overset{w}{\Longrightarrow} x = \overset{aw}{\Longrightarrow} x$.*

PROOF. Since $x \notin \mathit{Prefix}(aw)$, there is no new occurrence of x in aw. Thus we have $[x]'^{L}_{w} = [x]'^{L}_{aw}$. □

The above lemma ensures that any implicit node of $\mathit{STree}'(w)$ does not become explicit in $\mathit{STree}'(aw)$ if it is not associated with any prefix of aw.

Now we turn our attention to the strings in $\mathit{Prefix}(aw)$. Basically, we have to insert prefixes of aw into $\mathit{STree}'(w)$ in order to obtain $\mathit{STree}'(aw)$. However, no strings in set $\mathit{Substr}(w) \cap \mathit{Prefix}(aw)$ need to be newly inserted since they are already in $\mathit{STree}'(w)$. Let x be the longest string in set $\mathit{Substr}(w) \cap \mathit{Prefix}(aw)$. Then x is called the *longest repeated prefix* of aw and denoted by $\mathit{LRP}(aw)$. In updating $\mathit{STree}'(w)$ to $\mathit{STree}'(aw)$, we have to insert all prefixes of aw that are longer than $\mathit{LRP}(aw)$, into $\mathit{STree}'(w)$.

LEMMA 8. *Let $a \in \Sigma$ and $w \in \Sigma^*$. For any $x = \mathit{Prefix}(aw) - \mathit{Substr}(w)$, $\overset{aw}{\Longrightarrow} x = aw$.*

PROOF. String x is a prefix of aw which is longer than $\mathit{LRP}(aw)$. This implies that there is no occurrence of x in w. Therefore x appears in aw exactly once as a prefix of aw, meaning that there exists a unique character that follows x in aw. Hence $\overset{aw}{\Longrightarrow} x = aw$. □

The above lemma means that, simply by adding the new leaf node $\overset{aw}{\Longrightarrow} aw = aw$ to $\mathit{STree}'(w)$, we can obtain $\mathit{STree}'(aw)$. Moreover, the in-coming edge of the leaf node $\overset{aw}{\Longrightarrow} aw$ will be inserted from the node that corresponds to $\mathit{LRP}(aw)$. We now clarify what happens to $\mathit{LRP}(aw)$ when the new prefixes of aw are inserted to $\mathit{STree}'(w)$.

PROPOSITION 8. *Let $a \in \Sigma$ and $w \in \Sigma^*$. Let $x = \mathit{LRP}(aw)$ and $y = \mathit{LRS}(w)$. If $x \notin \mathit{Suffix}(w) - \mathit{Suffix}(y)$, then $\overset{aw}{\Longrightarrow} x = x$. Otherwise, $\overset{aw}{\Longrightarrow} x = aw$.*

PROOF. We first consider the case that $x \notin \mathit{Suffix}(w) - \mathit{Suffix}(y)$. Recall that x is the *longest* string in $\mathit{Substr}(w) \cap \mathit{Prefix}(aw)$. Moreover, $x \notin \mathit{Suffix}(w) - \mathit{Suffix}(y)$. Hence, there exist two characters $b, c \in \Sigma$ such that $xb, xc \in \mathit{Substr}(aw)$ and $b \neq c$. Thus we have $\overset{aw}{\Longrightarrow} x = x$.

Now we consider the second case, $x \in \mathit{Suffix}(w) - \mathit{Suffix}(y)$. Here, x occurs only once in w as its suffix. Thus $\overset{w}{\Longrightarrow} x = x$. On the other hand, by the definition of $\mathit{LRP}(aw)$, we obtain $x \in \mathit{Prefix}(aw) - \{aw\}$. Therefore, there uniquely exists a character $d \in \Sigma$ which follows x in aw. Hence we have $\overset{aw}{\Longrightarrow} x = aw$. □

The above proposition implies that if $\mathit{LRP}(aw)$ does not correspond to a leaf node of $\mathit{STree}'(w)$, it will be represented by an explicit node in $\mathit{STree}'(aw)$, and otherwise, it becomes implicit in $\mathit{STree}'(aw)$ (see the 3rd and 4th steps

of Fig. 3.6 to be shown later on). We stress that this characterizes a difference between $STree'(w)$ and $STree(w)$. More concretely, Weiner's original algorithm constructs $STree(aw)$ on the basis of the next proposition.

PROPOSITION 9. *For any $a \in \Sigma$ and $w \in \Sigma^*$, if $x = LRP(aw)$, then $\overset{aw}{\Longrightarrow} x = x$.*

Now the next question is how to locate $LRP(aw)$ in $STree'(w)$. Our idea is similar to Weiner's strategy for constructing $STree(w)$. Let y be the longest element in set $Prefix(w) \cup \{\xi\}$ such that $ay \in Substr(w)$. Then y is called the *base* of aw and denoted by $Base(aw)$.

LEMMA 9. (WEINER [1973]) *Let $a \in \Sigma$ and $w \in \Sigma^*$. If $y = Base(aw)$, then $ay = LRP(aw)$.*

PROOF. Assume contrarily that y' is the string such that $ay' = LRP(aw)$ and $|y'| > |y|$. By the definition of $LRP(aw)$, we have $ay' \in Prefix(aw)$, which yields $y' \in Prefix(w)$. It, however, contradicts the precondition that $y = Base(aw)$ since $|y'| > |y|$. □

According to the above lemma, $Base(aw)$ can be a clue to locating $LRP(aw)$ in $STree'(w)$.

Let z be the longest element in set $Prefix(w) \cup \{\xi\}$ such that $\overset{w}{\Longrightarrow} az = az$. Then z is called the *bridge* of aw and denoted by $Bridge(aw)$.

LEMMA 10. *Let $a \in \Sigma$ and $w \in \Sigma^*$. If $x = LRP(w)$, $y = Base(aw)$ and $z = Bridge(aw)$, then $y \in Prefix(x)$ and $z \in Prefix(y)$.*

PROOF. By LEMMA 9 we have $ay = LRP(aw)$. It is easy to see that $|LRP(aw)| \leq |LRP(w)| + 1$, which implies $|y| \leq |x|$. Now we obtain $y \in Prefix(x)$. It can be readily shown that $az \in Prefix(ay)$, since $ay = LRP(aw)$. Thus we have $z \in Prefix(y)$. □

Let $y = Base(aw)$ and $z = Bridge(aw)$. Assume $\gamma \in \Sigma^*$ is the string satisfying $z\gamma = y$. Then, we have $az\gamma = LRP(aw)$ by LEMMA 9 and LEMMA 10.

The detection of $LRP(aw)$ in $STree'(w)$ is illustrated in Fig. 3.4. We start from $LRP(w)$ and then go up the path backward until encountering $Bridge(aw)$. We move to the node for az and go down the path spelling out γ, and now we are at the location for $LRP(aw)$. Finally we insert a new edge labeled with $\beta\alpha$ from the location for $LRP(aw)$ due to LEMMA 8, and the resulting structure is $STree'(aw)$. The dashed arrow from $Bridge(aw) = z$ to az is the *labeled reversed suffix link* of z. The set F' of the links of $STree'(w)$ is defined as follows.

$$F' = \{(\overset{w}{\Longrightarrow} x, a, \overset{w}{\Longrightarrow} ax) \mid x, ax \in Substr(w), a \in \Sigma, \text{ and } \overset{w}{\Longrightarrow} ax = a \cdot \overset{w}{\Longrightarrow} x\}.$$

Observe that there is a one-to-one correspondence between F and F' for $STree'(w)$ (see DEFINITION 1).

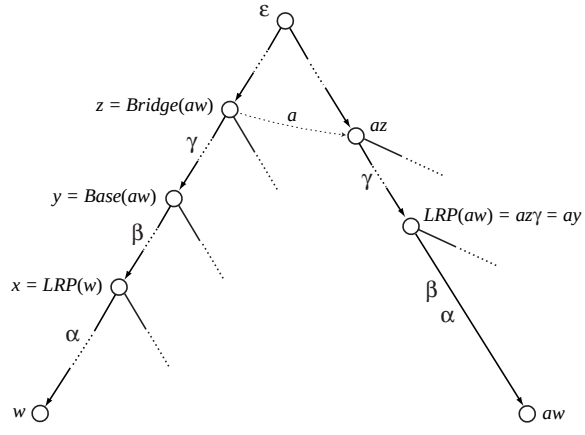

Fig. 3.4: In $STree'(w)$ we start from $LRP(w)$ and go up until $Bridge(w)$. Then we move to az and go down along the path spelling out γ. We are now on the location for $LRP(aw)$, and from there we insert a new edge labeled with $\beta\alpha$. Now all prefixes of aw are inserted, we have $STree'(aw)$.

In order that we can find $Base(aw)$ efficiently, we maintain a table for each explicit node, as well as Weiner's algorithm. For every explicit node this table can be computed in constant time and space for any fixed alphabet. Note that, however, $Base(w)$ can sometimes be associated with an implicit node in $STree'(w)$. The following lemma shows a property of $Base(w)$ when it is implicit in $STree'(w)$.

LEMMA 11. *Let $a \in \Sigma$ and $w \in \Sigma^*$. Let $y = Base(aw)$. If $y \neq \overset{w}{\Longrightarrow} y$, then $y = LRS(w)$.*

PROOF. Since $ay = LRP(aw)$, y appears at least twice in w. We now consider the following three cases.

(1) All occurrences of y in w are followed by same character b. In this case, string yb turns out to be a prefix of w that is longer than y and appears more than once in w. It means that $y \neq Base(w)$, which is a contradiction.

(2) There exist at least two distinct characters b, c such that $yb, yc \in Substr(w)$. In this case, $\overset{w}{\Longrightarrow} y = y$, a contradiction.

(3) One occurrence of y in w is followed by *no* character. This implies that y is a suffix of w.

Therefore only the third case is possible. This case, we have $y \in Suffix(w)$ and $y \in Prefix(w)$, which implies that y is the longest string satisfying the condition. Therefore, $y = LRS(w)$. □

Since $y = LRS(w)$, it is guaranteed that $\overset{w}{\Longrightarrow} ay = ay$. We hereby regard y as $Bridge(w)$ and maintain the labeled reversed suffix link of $LRS(w)$, which is

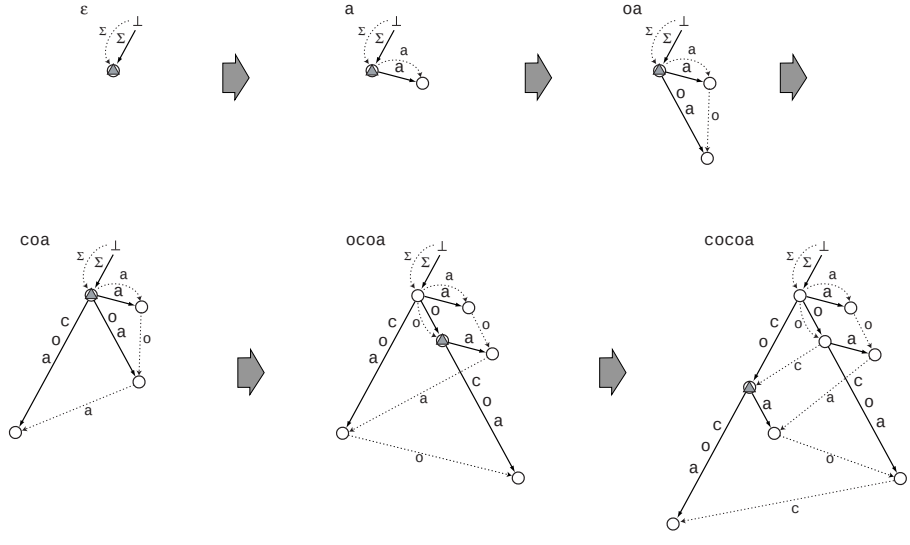

Fig. 3.5: The construction of $STree'(w)$ with left extension, where $w = \texttt{cocoa}$. The triangle mark represents the longest repeated suffix of each suffix tree. The $\perp$ node corresponds to the eliminator symbol ξ. The Σ symbol represents *any* character in the alphabet.

always associated with the shortest leaf node of $STree'(w)$.

By a similar argument to Weiner [1973], it can be established that the amortized amount of time needed for the detection of $LRP(aw)$ in $STree'(w)$ is constant, again on the assumption that every edge label is implemented by a pair of integers.

We now have the following theorem.

THEOREM 3.2. *For any $a \in \Sigma$ and $w \in \Sigma^*$, $STree'(w)$ can be updated to $STree'(aw)$ in amortized constant time.*

Fig. 3.5 shows the construction of $STree'(\texttt{cocoa})$ with left extension.

3.3 Mutual Influences

Here, we consider mutual influences between Right Extension and Left Extension. The next lemma shows what happens to $LRP(w)$ when $STree'(w)$ is updated to $STree'(wa)$.

LEMMA 12. *Let $a \in \Sigma$ and $w \in \Sigma^*$. Assume $LRP(w) = LRS(w)$. Let $x = LRS(w)$. If $xa \in Prefix(w)$, then $LRP(wa) = xa$.*

PROOF. Since $xa \in Prefix(w)$, $LRS(wa) = xa$. Thus $xa = LRP(wa)$. □

This lemma shows when and where $LRP(wa)$ moves from the location of $LRP(w)$ according to the character a newly added to the right of w (see the

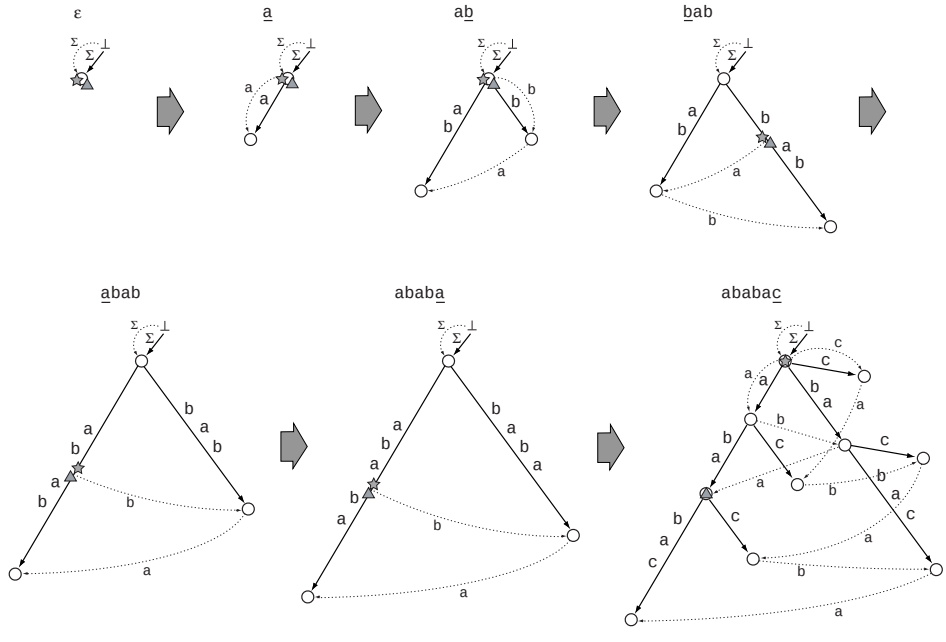

Fig. 3.6: A bidirectional construction of $STree'(w)$ with $w =$ `ababac`. Solid arrows represent edges, and dotted arrows denote labeled reversed suffix links. On Right Extension, the labeled reversed suffix links are used for another direction, that is, as "normal" suffix links. In each suffix tree, the triangle (star, respectively) indicates the location of the longest repeated prefix (suffix, respectively). The character newly added in each step is underlined.

5th step in Fig. 3.6). Examining the precondition, "if $xa \in \mathit{Prefix}(w)$", is feasible in $O(|\Sigma|)$ time, which is regarded as $O(1)$ if Σ is a fixed alphabet.

The following lemma stands in contrast to Lemma 12.

LEMMA 13. *Let $a \in \Sigma$ and $w \in \Sigma^*$. Assume $LRP(w) = LRS(w)$. Let $x = LRP(w)$. If $ax \in \mathit{Suffix}(w)$, then $LRS(aw) = ax$.*

This lemma shows when and where $LRS(aw)$ moves from the location of $LRS(w)$ according to the character a newly added to the left of w. Examining the precondition, "if $ax \in \mathit{Suffix}(w)$", is also possible in $O(|\Sigma|)$ time, and moving from $LRS(w)$ to $LRS(aw)$ is possible in constant time by the use of the labeled reversed suffix link of $LRS(w)$ (see the 3rd and 4th steps of Fig. 3.6).

As a result of discussion, we finally obtain the following:

THEOREM 3.3. *For any string $w \in \Sigma^*$, $STree'(w)$ can be constructed in bidirectional manner and in $O(|w|)$ time.*

A bidirectional construction of $STree'(w)$ with $w =$ `ababac` is displayed in Fig. 3.6.

4. Concluding Remarks

We introduced an algorithm for bidirectional construction of suffix trees, which performs in linear time. This is a counterpart of the algorithm of Stoye [1995] for bidirectional construction of affix trees. We stress that our new algorithm requires less space than Stoye's.

An interesting fact is that the tables for finding $Base(w)$ used in Weiner [1973] correspond to the edges of $DAWG(w^{rev})$ (see Crochemore and Rytter [1994]). This implies that our algorithm is also able to update a DAWG to the *right* direction. On the other hand, in Inenaga *et al.* [2001a] we presented a linear-time algorithm that constructs not only $STree'(w)$ but also $DAWG(w^{rev})$ in a left-to-right on-line manner, which is based on the algorithm by Ukkonen [1995]. This algorithm enables us to update a DAWG to the *left* direction. Therefore, the algorithm of this paper turns out to be adaptive to bidirectional construction of DAWGs.

An interesting open problem is whether or not linear-time bidirectional construction of CDAWGs is possible. It can be done in amortized constant time to convert $CDAWG(w)$ into $CDAWG(wa)$ by the use of the algorithm of Inenaga *et al.* [2001b]. However, as is mentioned in Inenaga *et al.* [2002a], we conjecture that the conversion of $CDAWG(w)$ to $CDAWG(aw)$ would not be possible in (amortized) constant time (also see Inenaga *et al.* [2002b]). Still, there might remain a possibility to construct CDAWGs in a right-to-left on-line manner. That is, a chunk of characters x (namely a string x) are at once appended to the left of the current string w, so that updating $CDAWG(w)$ to $CDAWG(xw)$ can be done in amortized constant time (this case we would not obtain $CDAWG(vw)$ for any proper suffix v of x excepting $v = \varepsilon$). However, we are unsure if such convenient selection of the length of x is really possible or not.

Acknowledgements

The author wishes to thank Prof. Ayumi Shinohara and Prof. Masayuki Takeda. Daily fruitful and enthusiastic discussion with them led the author to the inspiration for this work.

References

Apostolico, Alberto. 1985. The Myriad Virtues of Subword Trees. In *Combinatorial Algorithm on Words*, Volume 12 of *NATO Advanced Science Institutes, Series F*. Springer-Verlag, 85–96.

Balík, Miroslav. 1998. Implementation of DAWG. In *Proc. The Prague Stringology Club Workshop '98 (PSCW'98)*. Czech Technical University.

Blumer, Anselm, Blumer, Janet, Haussler, David, Ehrenfeucht, Andrzej, Chen, M. T., and Seiferas, Joel. 1985. The Smallest Automaton Recognizing the Subwords of a Text. *Theoretical Computer Science 40*, 31–55.

Blumer, Anselm, Blumer, Janet, Haussler, David, McConnell, Ross, and Ehrenfeucht, Andrzej. 1987. Complete Inverted Files for Efficient Text Retrieval and Analysis. *Journal of the ACM 34*, 3, 578–595.

Breslauer, Dany. 1998. The suffix tree of a tree and minimizing sequential transducers. *Theoretical Computer Science 191*, 131–144.

Chen, M. T. and Seiferas, Joel. 1985. Efficient and Elegant Subword Tree construction. In *Combinatorial Algorithm on Words*, Volume 12 of *NATO Advanced Science Institutes, Series F*. Springer-Verlag, 97–107.

Crochemore, Maxime. 1986. Transducers and Repetitions. *Theoretical Computer Science 45*, 63–86.

Crochemore, Maxime and Rytter, Wojciech. 1994. *Text Algorithms*. Oxford University Press, New York.

Crochemore, Maxime and Vérin, Renaud. 1997. On Compact Directed Acyclic Word Graphs. In *Structures in Logic and Computer Science*, Volume 1261 of *Lecture Notes in Computer Science*. Springer-Verlag, 192–211.

Farach, M. 1997. Optimal suffix tree construction with large alphabets. In *Proc. The 38th Annual Symposium on Foundations of Computer Science (FOCS'97)*. IEEE Computer Society, 137–143.

Giegerich, Robert and Kurtz, Stefan. 1997. From Ukkonen to McCreight and Weiner: A Unifying View of Linear-Time Suffix Tree Construction. *Algorithmica 19*, 3, 331–353.

Gusfield, Dan. 1997. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, New York.

Inenaga, Shunsuke. 2002. Bidirectional Construction of Suffix Trees. In *Proc. The Prague Stringology Conference '02 (PSC'02)*. Czech Technical University, 75–87.

Inenaga, Shunsuke, Hoshino, Hiromasa, Shinohara, Ayumi, Takeda, Masayuki, and Arikawa, Setsuo. 2001a. On-Line Construction of Symmetric Compact Directed Acyclic Word Graphs. In *Proc. of 8th International Symposium on String Processing and Information Retrieval (SPIRE'01)*. IEEE Computer Society, 96–110.

Inenaga, Shunsuke, Hoshino, Hiromasa, Shinohara, Ayumi, Takeda, Masayuki, Arikawa, Setsuo, Mauri, Giancarlo, and Pavesi, Giulio. 2001b. On-Line Construction of Compact Directed Acyclic Word Graphs. In *Proc. 12th Annual Symposium on Combinatorial Pattern Matching (CPM'01)*, Volume 2089 of *Lecture Notes in Computer Science*. Springer-Verlag, 169–180.

Inenaga, Shunsuke, Shinohara, Ayumi, Takeda, Masayuki, and Arikawa, Setsuo. 2002a. Compact Directed Acyclic Graphs for a Sliding Window. In *Proc. of 9th International Symposium on String Processing and Information Retrieval (SPIRE'02)*, Volume 2476 of *Lecture Notes in Computer Science*. Springer-Verlag, 310–324.

Inenaga, Shunsuke, Shinohara, Ayumi, Takeda, Masayuki, Bannai, Hideo, and Arikawa, Setsuo. 2002b. Space-Economical Construction of Index Structures for All Suffixes of a String. In *Proc. of 27th International Symposium on Mathematical Foundation of Computer Science (MFCS'02)*, Volume 2420 of *Lecture Notes in Computer Science*. Springer-Verlag, 341–352.

Maass, Moritz G. 2000. Linear Bidirectional On-Line Construction of Affix Trees. In *Proc. 11th Annual Symposium on Combinatorial Pattern Matching (CPM'00)*, Volume 1848 of *Lecture Notes in Computer Science*. Springer-Verlag, 320–334.

Manber, Udi and Myers, Gene. 1993. Suffix arrays: A new method for on-line string searches. *SIAM J. Compt. 22*, 5, 935–948.

McCreight, Edward M. 1976. A space-economical suffix tree construction algorithm. *Journal of the ACM 23*, 2, 262–272.

Stoye, Jens. 1995. Affixbäume. Master's thesis, Universität Bielefeld.

Ukkonen, Esko. 1995. On-line Construction of Suffix Trees. *Algorithmica 14*, 3, 249–260.

Weiner, Peter. 1973. Linear pattern matching algorithms. In *Proc. 14th Annual Symposium on Switching and Automata Theory*, 1–11.