Affix Trees

Moritz G. Maaß, June 2000

http://www.informatik.tu-muenchen.de/~maass

maass@informatik.tu-muenchen.de

Agenda

1. Introduction
2. Construction of Suffix Trees
3. Construction Affix Trees
4. Complexity
Important Suffix Tree Properties

- Representation of repeated substrings
- Right branching substrings are represented by branching nodes
- Each tree position represents a unique string
- Moving down in the tree extends the string, moving towards the root shortens the string.
Representation of Tree Positions with Reference Pairs

```
root
```

```
5  +  ε
3  +  ε
2  +  ab
```

```
  root
```

```
  +  a
```

```
1
2
3
4
5
6
7
8
9
```
- Two ways of “shortening” the represented substring abab at the front to come to the position of bab
- Suffix links operate at the front of the represented tree, while edges operate at the end
Reverse Tree

Linear Bidirectional On-Line Construction of Affix Trees

Moritz G. Maaß, June 2000
Definition 1 (right branching and left branching). A substring $w$ of $t$ is right branching (left-branching), if there $w$ occurs at two different positions in $t$ with distinct succeeding (preceding) letters ($w$ is r.b. in $t$, if $\exists x, y \in \Sigma, u, v, u', v' \in \Sigma^* . t = uwxv \land t = u'wyv' \land x \neq y$).

Definition 2 ($\Sigma^+$-tree). A $\Sigma^+$-tree $T$ is a rooted, directed tree with edge labels from $\Sigma^+$. For each $a \in \Sigma$, every node in $T$ has at most one outgoing edge whose label starts with $a$.

Definition 3 (path($n$)). If $n$ is a node in $\Sigma^+$-tree $T$, then path($n$) is the string built by concatenating all edge labels from the root to $n$. It is a unique identifier for the tree position.

Definition 4 (words($T$)). A string $u$ is in words($T$), if there is a node $n$ in $T$ s.t. $\exists v \in \Sigma^* . uv = \text{path}(n)$.
Definition 5 (Suffix tree). A suffix tree of string $t$ is a $\Sigma^+$-tree with $\text{words}(T) = \{u \mid u$ is a substring of $t\}$.

Definition 6 (Suffix Link). A suffix link is an auxiliary edge from node $n$ to node $m$ where $m$ is the node s.t. $\text{path}(m)$ is the longest proper suffix of $\text{path}(n)$ represented by a node in $T$. 
**Reverse Tree and Affix Trees**

**Definition 7 (Reverse tree $T^{-1}$).** The reverse tree $T^{-1}$ of a $\Sigma^+$-tree $T$ augmented with suffix links is defined as the tree that is formed by the suffix links of $T$, where the direction of each link is reversed, but the label is kept.

**Definition 8 (Affix tree).** An affix tree $T$ of a string $t$ is a $\Sigma^+$-tree s.t.

- $\text{words}(T) = \{u\mid u$ is a substring of $t\}$ and
- $\text{words}(T^{-1}) = \{u\mid u$ is a substring of $t^{-1}\}$.
Affix Trees

Linear Bidirectional On-Line Construction of Affix Trees

Moritz G. Maaß, June 2000
Previous Work

- Weiner, McCreight: linear suffix tree construction
- Ukkonen: linear on-line suffix tree construction, reference pairs, open edges
- Giegerich and Kurtz: relationship between suffix tree and its reverse tree through suffix links
- Stoye: affix tree data structure
- Blumer et al.: DAWG, c-DAWG with suffix links invariant under reversal
1. Introduction

2. Construction of Suffix Trees

3. Construction Affix Trees

4. Complexity
On-Line Suffix Tree Construction

CST(bbabaaba)

CST(bbabaabab)
Anti-On-Line Suffix Tree Construction

CST(abaababb)  
CST(babaababb)
Complexity of Suffix Tree Construction

Lemma 1. Ukkonen’s algorithm constructs $CST(t)$ on-line in time $O(|t|)$.

Lemma 2. With the additional information of knowing the length of the active prefix for any suffix $s$ of $t$ before inserting it, it takes $O(|t|)$ time to construct $CST(t)$ in an anti-on-line manner.
1. Introduction

2. Construction of Suffix Trees

3. Construction Affix Trees

4. Complexity
The Problem in Constructing Affix Trees
The Problem in Constructing Affix Trees (continued)
The Solution: Paths
The Solution: Paths (continued)
Additional Steps in Affix Tree Construction

● Updating Paths:

**Lemma 3.** The prefix parent of the active suffix leaf is (also) a prefix node.

● Keeping track of the active suffix point, the active prefix point, the active suffix leaf, and the active prefix leaf:

**Lemma 4.** The active prefix will grow in the iteration from $t$ to $ta$, iff the new active suffix of $ta$ is represented by a prefix leaf in $CAT(t)$.

● Deleting Nodes
Summary of all Steps

1. Remove the suffix link from the active suffix link to $s$.

2. Lengthen the text, thereby lengthening all open edges.

3. Insert the prefix node for $t$ as suffix parent of $\overline{ta}$ and link it to $s$.

4. Insert relevant suffixes and update suffix links.

5. Make the location of the new active suffix $\alpha(ta)$ explicit and add a suffix link from the new active suffix link to it.

6. Update the active prefix, possibly deleting a node.
Example of a Single Iteration
1. Introduction

2. Construction of Suffix Trees

3. Construction Affix Trees

4. Complexity
Complexity of Affix Tree Construction

**Theorem 1.** \( \text{CAT}(t) \) can be constructed in an on-line manner from left to right or from right to left in time \( O(|t|) \).

**Theorem 2.** Bidirectional construction of affix trees has linear time complexity.
Growth of the active suffix in a reverse iteration adds node-accounted part.

Insertion of suffix nodes in a reverse iteration is only relevant to node-accounted part.

node-accounted part cannot be nested.
Conclusion

- Affix trees are a natural extension of suffix trees.
- Construction can be done in linear time, on-line and bidirectional.
- Affix tree augmented by paths behave like suffix trees.
- The view can be switched from the suffix to the prefix tree at any time.
- Right branching and left branching substrings represented in one structure.