Computing all repeats using suffix arrays

F.Franek, W.F.Smyth, Y.Tang

Algorithms Research Group
McMaster University

We describe an algorithm that identifies *all* the repeating substrings (tandem, overlapping, and split) in a given string $\boldsymbol{x} = \boldsymbol{x}[1..n]$.

Given the suffix arrays of $\boldsymbol{x}$ and $\widehat{\boldsymbol{x}}$, the algorithm requires $\Theta(n)$ time. The output (in $\Theta(n)$ space) is either an NE array or an NE tree.

The output substrings $\boldsymbol{u}$ are *nonextendible* (NE) and thus the num-

ber of substrings output is the minimum required to identify all the repeating substrings in $\boldsymbol{x}$.

A *repeat* in a string $\boldsymbol{x}$ is a tuple

$$M_{\boldsymbol{x},\boldsymbol{u}} = (p;\, i_1, i_2, \ldots, i_r),\ \ r \geq 2,$$

where
$\boldsymbol{u} = \boldsymbol{x}[i_1..i_1{+}p{-}1] = \boldsymbol{x}[i_2..i_2{+}p{-}1] = \cdots$
$= \boldsymbol{x}[i_r..i_r{+}p{-}1]$.

$\boldsymbol{u}$ **- *repeating substring* of** $\boldsymbol{x}$
$\boldsymbol{u}$ **- *generator* of** $M_{\boldsymbol{x},\boldsymbol{u}}$
$p = |\boldsymbol{u}|$ **- *period* of** $M_{\boldsymbol{x},\boldsymbol{u}}$
r **- *exponent* of** $M_{\boldsymbol{x},\boldsymbol{u}}$

If $M_{\boldsymbol{x},\boldsymbol{u}}$ includes *all* the occurrences of $\boldsymbol{u}$ in $\boldsymbol{x}$ - then it is *complete* and denoted as $M^*_{\boldsymbol{x},\boldsymbol{u}}$.

$M_{\boldsymbol{x},\boldsymbol{u}}$ **is *left-extendible* (LE) iff** $(p{+}1;\, i_1{-}1, i_2{-}1, \ldots, i_r{-}1)$ **is a repeat.**

$M_{\boldsymbol{x},\boldsymbol{u}}$ **is *right-extendible* (RE) iff** $(p{+}1;\, i_1, i_2, \ldots, i_r)$ **is a repeat.**

If $M_{\boldsymbol{x},\boldsymbol{u}}$ **is neither LE nor RE, then it is said to be *nonextendible* (NE).**

LE or RE repeats do not need to be identified, they are implicitly "reported" as substrings of NE repeats. Thus, it suffices to compute complete NE repeats.

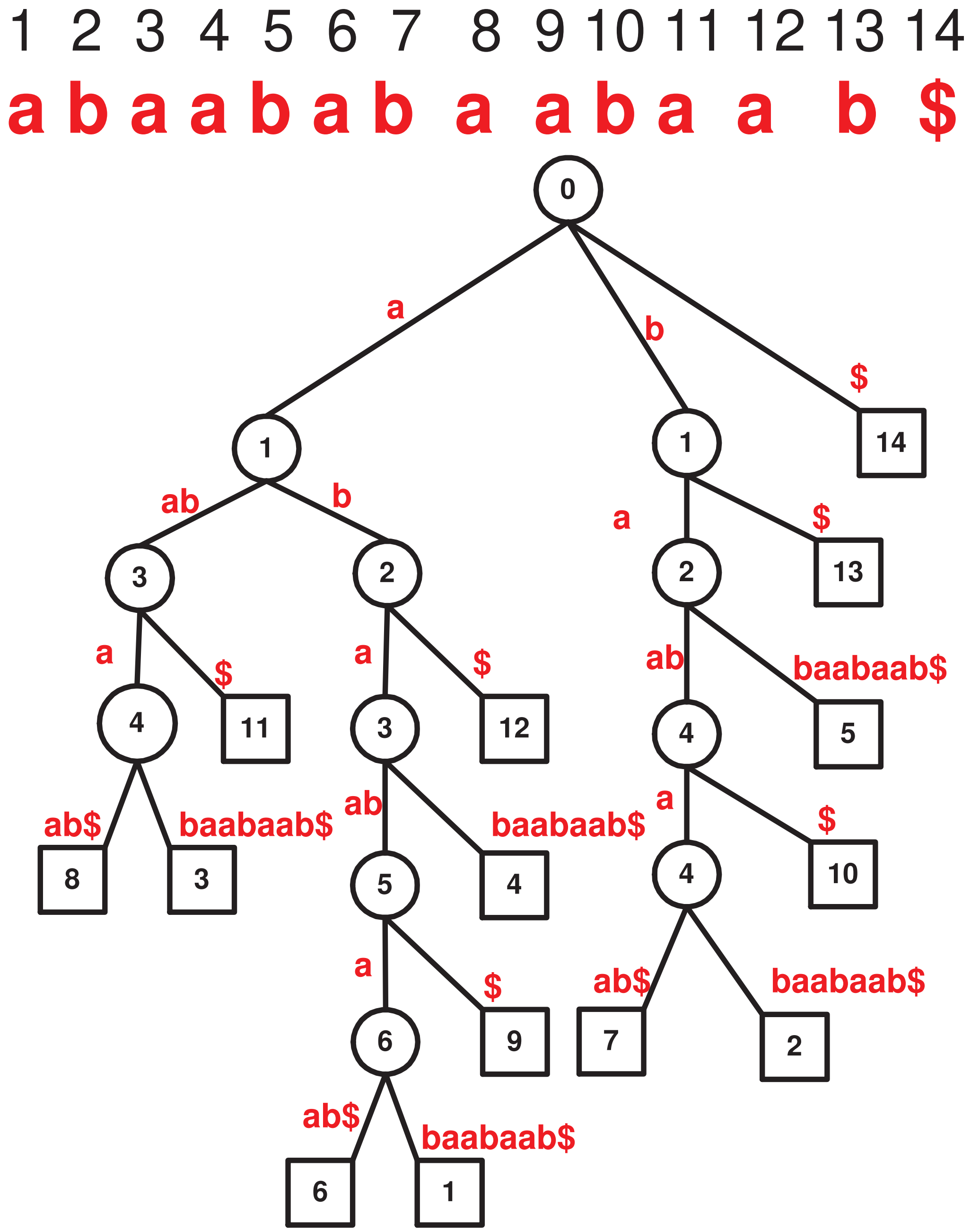
1997 *Gusfield*, 2000 *Brodal, Lyngso, Pedersen, Stoye* algorithms to compute all NE repeats given the suffix tree.

An easy observation:
$M_{\boldsymbol{x},\boldsymbol{u}}$ ***is LE iff*** $M_{\widehat{\boldsymbol{x}},\widehat{\boldsymbol{u}}}$ ***is RE.***

Thus the algorithm we are presenting:

- **computes all the complete NRE repeats of $\boldsymbol{x}$ and all the complete NRE repeats of $\widehat{\boldsymbol{x}}$;**
- **compares the complete NRE repeats of $\boldsymbol{x}$ with the complete NRE repeats of $\widehat{\boldsymbol{x}}$ to identify those that are in both lists (the complete NE repeats).**

Computing NE tree:

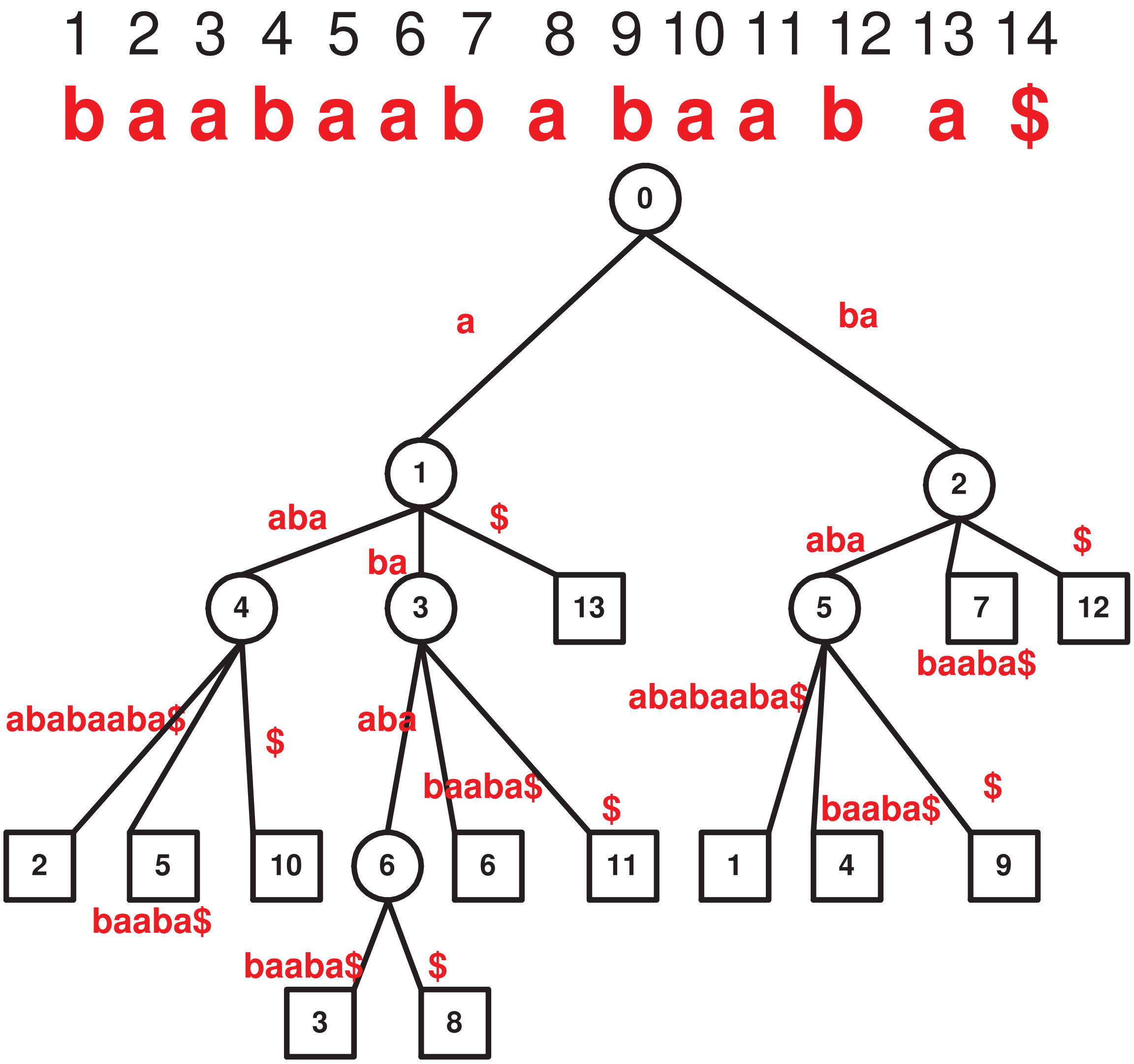

1 2 3 4 5 6 7 8 9 10 11 12 13 14
a b a a b a b a a b a a b $
0
a
b
$
1
1
14
ab
b
a
$
3
2
2
13
a
$
a
$
ab
baabaab$
4
11
3
12
4
5
ab$
baabaab$
ab
baabaab$
a
$
8
3
5
4
4
10
a
$
ab$
baabaab$
6
9
7
2
ab$
baabaab$
6
1

All round nodes in the suffix tree identify all complete NRE repeats:

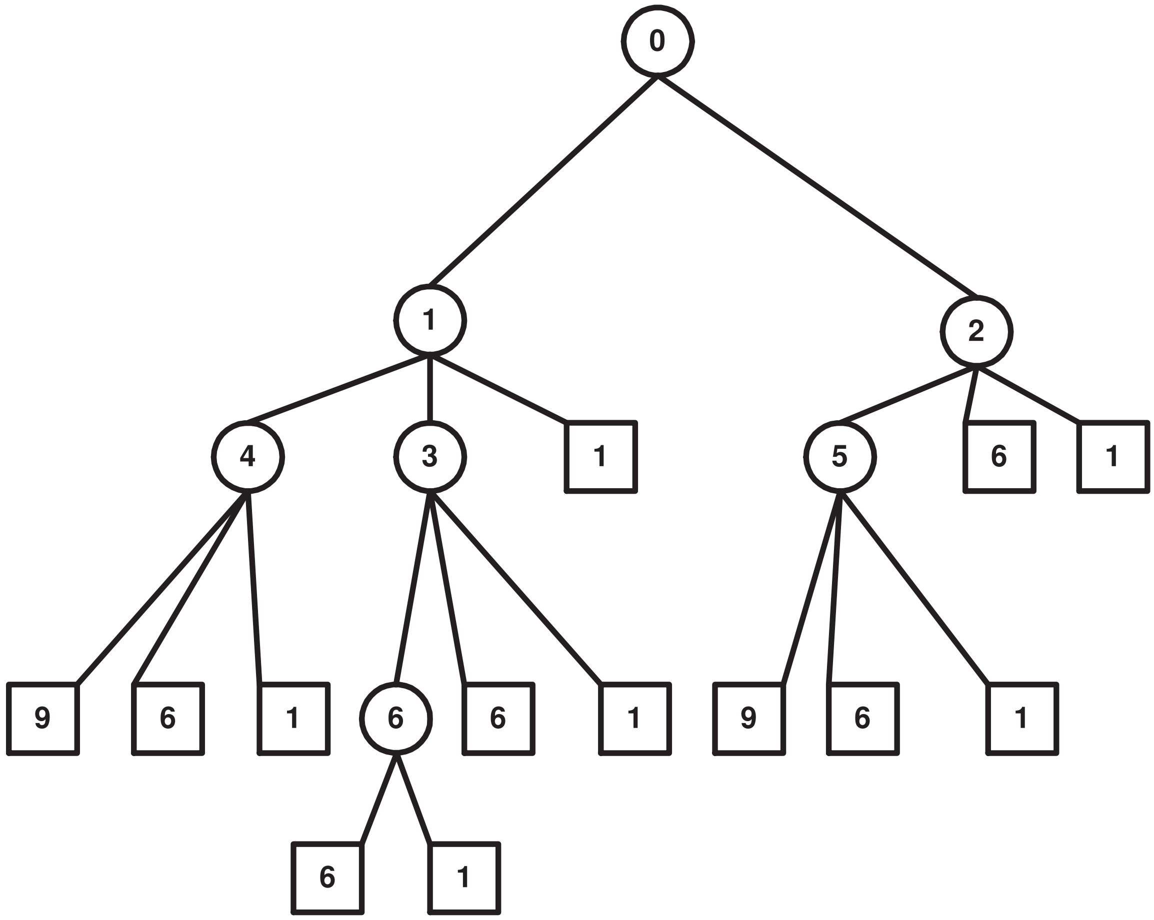
e.g. $M^*_{\boldsymbol{x},\boldsymbol{abaab}} = (5; 9, 6, 1)$ **since** $\boldsymbol{x[9..13] = x[6..10] = x[1..5]}$ $\boldsymbol{= abaab}$

It is NRE, for $\boldsymbol{x[9..14] = abaab\$}$ **while** $\boldsymbol{x[6..11] = x[1..6] = abaaba}$

Now consider the suffix tree for $\widehat{\boldsymbol{x}}$**:**



Transform a square child i of a round parent p to $n - (i + p - 2)$.



A substring u of x is a *maximal NE repeating substring* of x if and only if

- **u occurs at least twice in x;**
- **u is not a proper substring of**

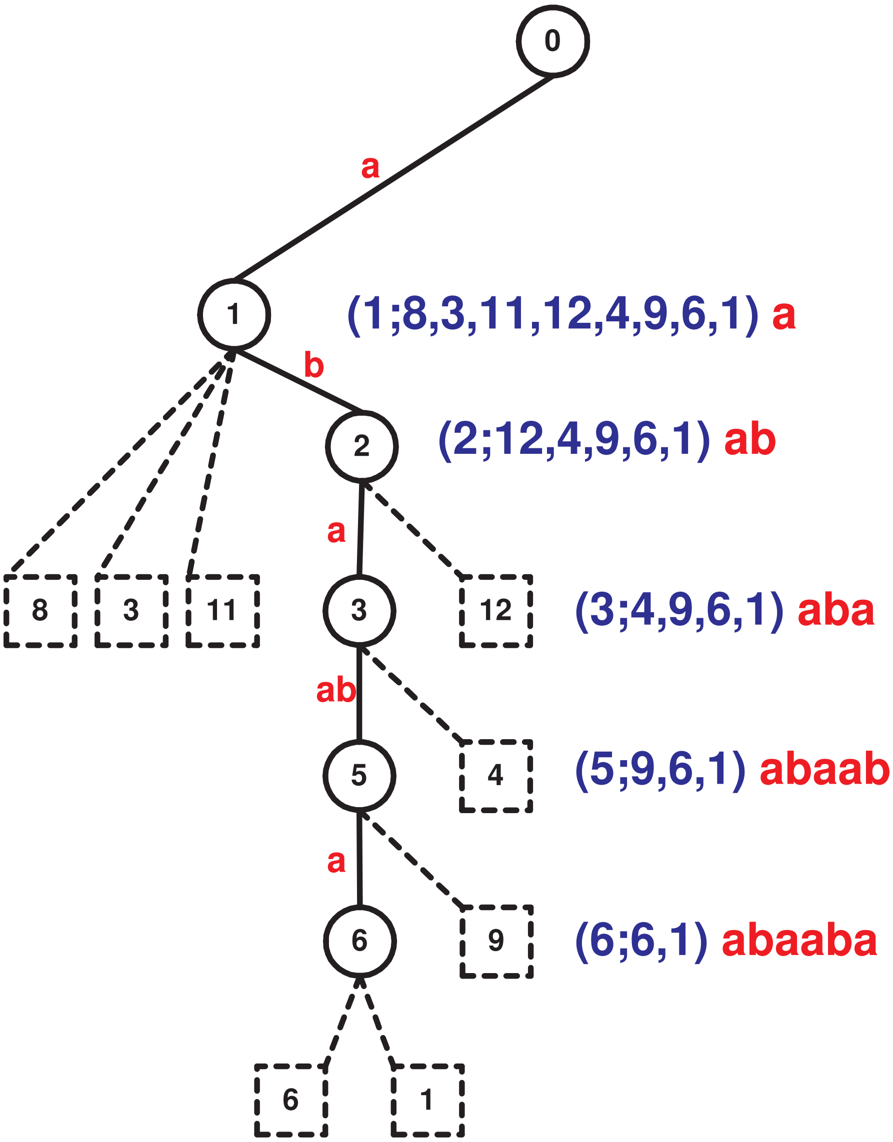
any repeating substring of $\boldsymbol{x}$.

If $\boldsymbol{u} = \boldsymbol{x}[i..i{+}p{-}1]$ ***is a maximal NE repeating substring of*** $\boldsymbol{x}$***, then the position node*** i ***occurs as a child of the lcp node*** p ***in*** $T_{\boldsymbol{x}}$***, and*** $n{-}(i{-}p{-}2)$ ***occurs as a child of*** p ***in*** $T_{\widehat{\boldsymbol{x}}}$***.***

Strategy for identifying all NE repeating substrings: just find the maximal ones (largest value of p), then locate all their ancestors in $T_{\boldsymbol{x}}$.

Note that only positions 1, 6, and 9 give rise to NLE repeats.

NE tree is the subtree of $T_{\boldsymbol{x}}$ that corresponds to these positions:


0
a
1
(1;8,3,11,12,4,9,6,1) a
b
2
(2;12,4,9,6,1) ab
a
8
3
11
3
12
(3;4,9,6,1) aba
ab
5
4
(5;9,6,1) abaab
a
6
9
(6;6,1) abaaba
6
1

Given the suffix trees $T_{\boldsymbol{x}}$ and $T_{\widehat{\boldsymbol{x}}}$ for $\boldsymbol{x}[1..n]$, the algorithm computes the NE tree $T_{\boldsymbol{x}}^{NE}$ using $O(n)$ time and $\Theta(n)$ additional space.

The approach using suffix arrays rather than suffix trees is based on similar ideas, but is combinatorially more complex.

Papers related to this topic, including these slides, can be viewed at the web site

<u>www.cas.mcmaster.ca/∼franek</u>