06 – Suffix Trees (1)

Text search

Various scenarios:

Static texts

- Literature databases
- Library systems
- Gene databases
- World Wide Web

Dynamic texts

- Text editors
- Symbol manipulators

Algorithms by Knuth, Morris & Pratt and Boyer & Moore

Properties of suffix trees

Search index

for a text σ in order to search for several patterns α.

Properties:

1. **Substring searching** in time $O(|\alpha|)$.
2. **Queries to σ itself**, e.g.:
   Longest substring of σ that occurs at least twice.
3. **Prefix search:** all positions in σ with prefix α.

Properties of suffix trees

4. **Range search:** all locations (substrings) in σ belonging to an interval
 $[\alpha, \beta]$ with $\alpha \leq_{lex} \beta$, e.g.

 abrakadabra, acacia $\in$ [abc, acc],
 abacus $\notin$ [abc, acc] .

5. **Linear complexity:**
 Space requirement and construction time in $O(|\sigma|)$.

Tries

Alphabet Σ, set S of keys, $S \subset \Sigma^*$

Key: string in Σ^*

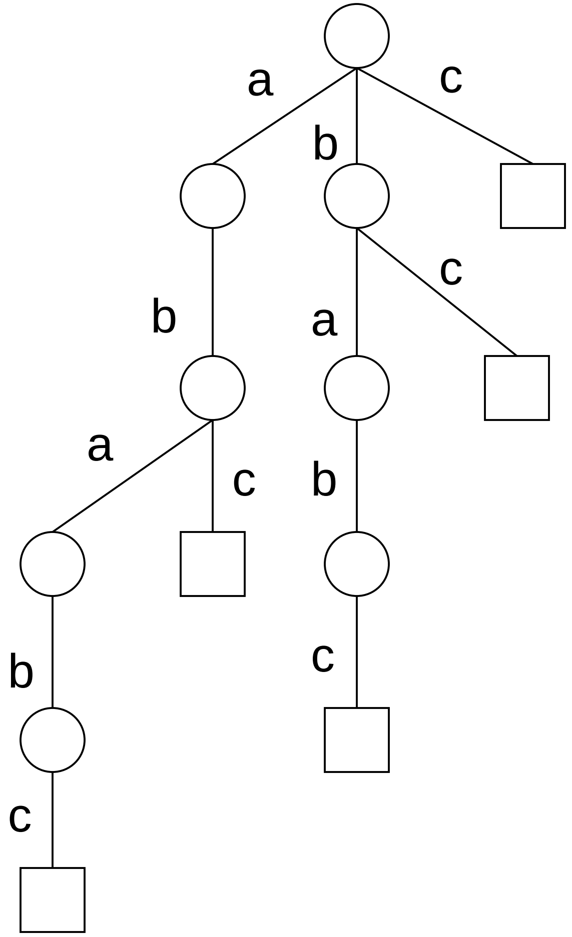
Trie: A tree representing a set of keys.

Edge of a trie T: labeled with a single character of Σ

Neighboring edges (edges that lead to different children of a node):
labeled with different characters

Tries

Example:



Tries

A **leaf** represents a key:

The corresponding key is the string consisting of the edge labels along the path from the root to the leaf.

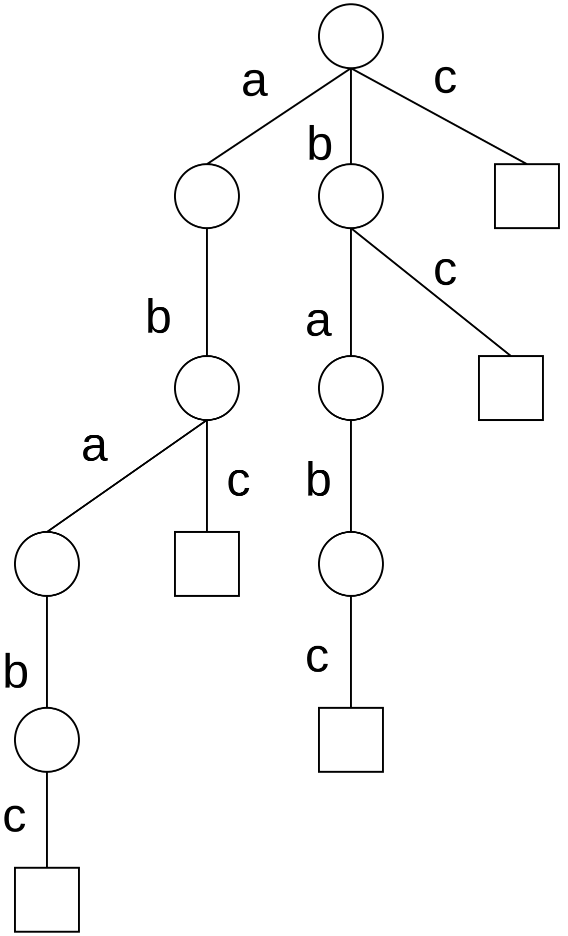
Keys are not stored in nodes!

Suffix tries

Trie representing all suffixes of a string

Example: σ = ababc

suffixes: ababc = suf_1
babc = suf_2
abc = suf_3
bc = suf_4
c = suf_5



Suffix tries

Nodes of a suffix trie $\triangleq$ substrings of σ

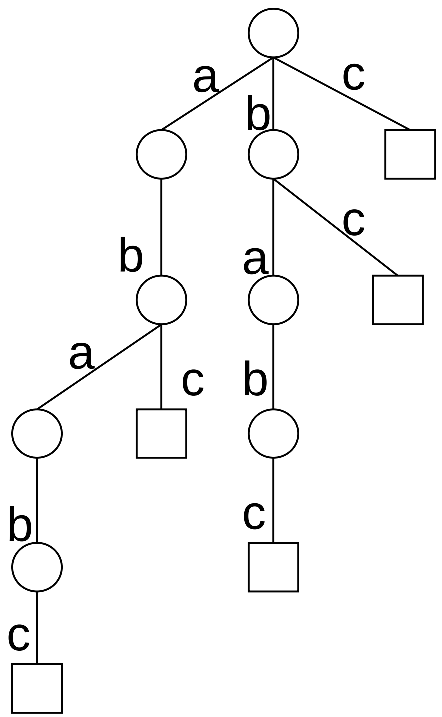
Each substring of σ is represented by a node.

Let $\sigma = a^n b^n$. Then there are $n^2 + 2n + 1$ different substrings (or internal nodes).

$\Rightarrow$ space requirement is $O(n^2)$

Suffix tries

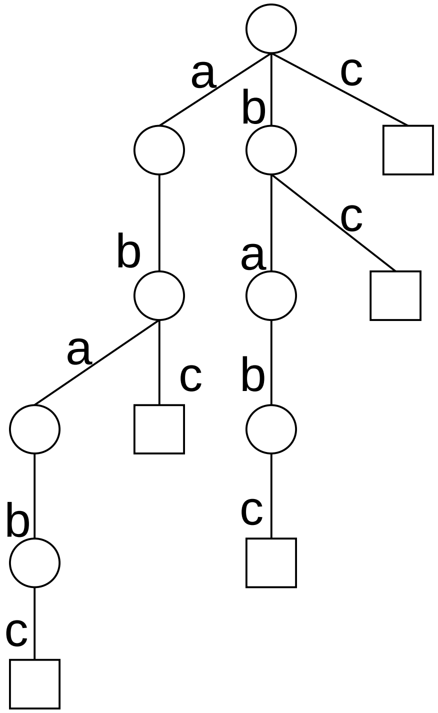
A suffix trie T satisfies some of the desired properties:

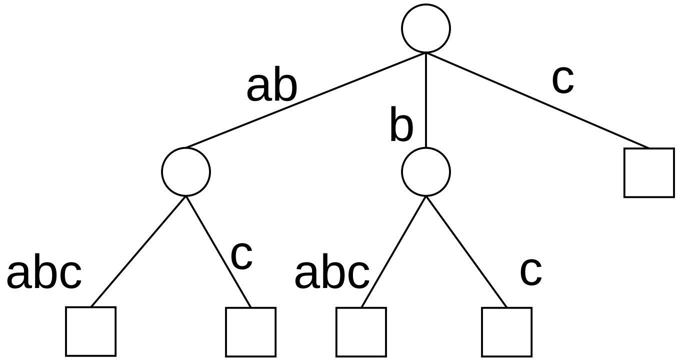


1. String matching for α: Following the path with edge labels α takes $O(|\alpha|)$ time. leaves of the subtree $\triangleq$ occurrences of α
2. Longest substring occurring at least twice: internal node with maximum depth having at least two children
3. Prefix search: All occurrences of strings with prefix α are represented by the nodes of the subtree rooted at the internal node corres-ponding to α .

Suffix trees

A suffix tree is obtained from a suffix trie by contracting unary nodes.





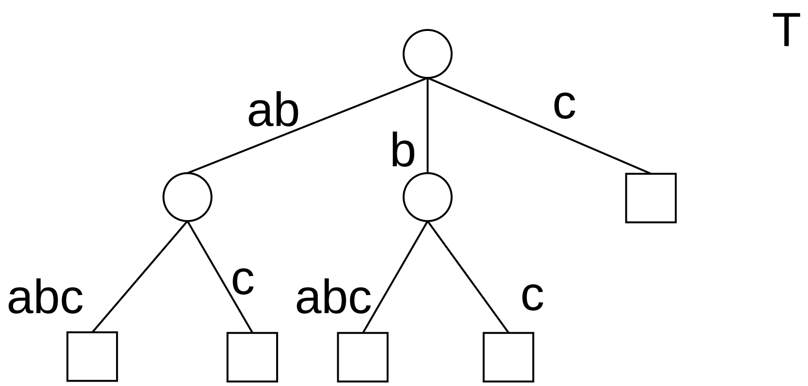
suffix tree = contracted suffix trie

Internal representation of suffix trees

Child-sibling representation

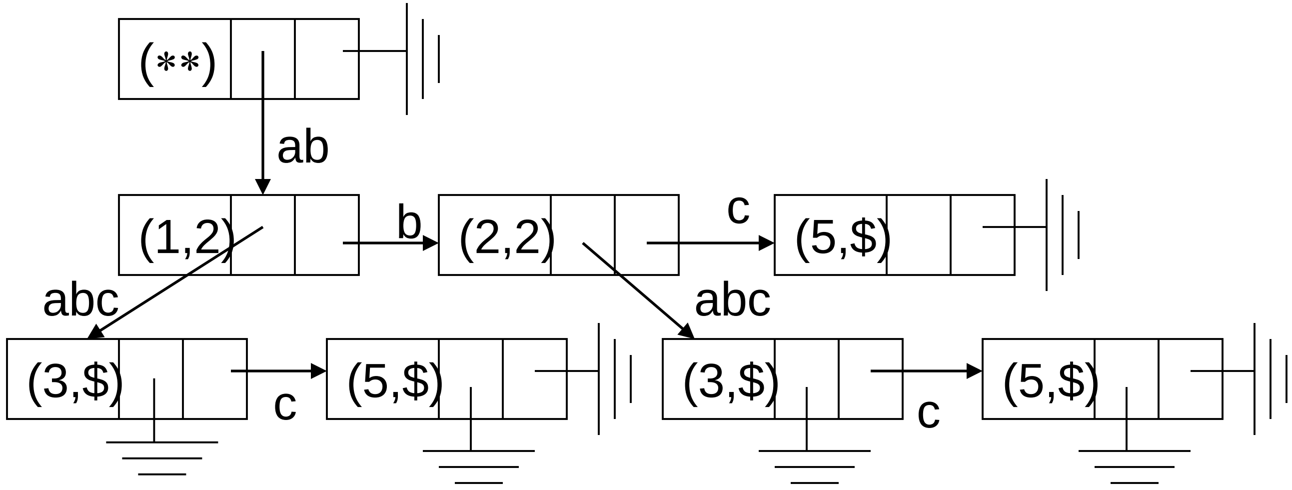
substring: pair of numbers (i,j)

Example: σ = ababc



Internal representation of suffix trees

Example: σ = ababc



node v = (v.l, v.u, v.c, v.s)

Further pointers (suffix links) are added later.

Properties of suffix trees

(S1) No suffix of σ is prefix of another suffix.

This holds if the last character of σ is $\$ \notin \Sigma$.

Search:

(T1) edge $\triangleq$ non-empty substring of σ.

(T2) neighboring edges:

corresponding substrings start with different characters

Properties of suffix trees

Size

(T3) each internal node ($\neq$ root) has at least two children

(T4) leaf $\triangleq$ (non-empty) suffix of σ.

Let $n = |\sigma| > 1$.

$\overset{(T4)}{\Longrightarrow}$ number of leaves $= n$

$\overset{(T3)}{\Longrightarrow}$ number of internal nodes $\leq n-1$

$\Rightarrow$ space requirement in $O(n)$

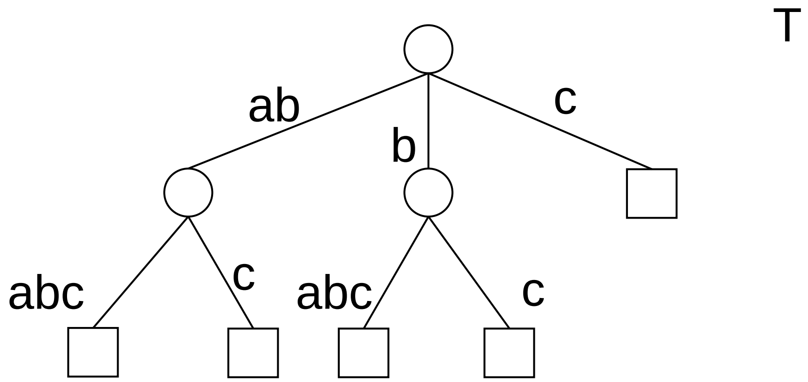
Construction of suffix trees

Definitions

Partial path: Path from the root to a node in *T.*

Path: A partial path ending at a leaf.

Location of a string α : Node where the partial path corresponding to α ends (if it exists).



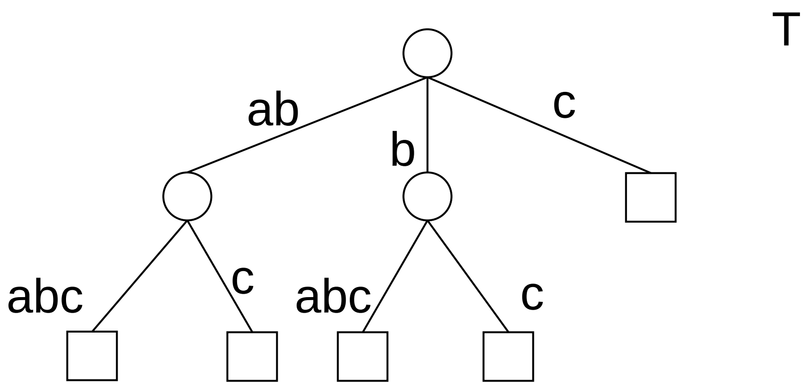
α = bab (has no location)

Construction of suffix trees

Extension of a string α : string with prefix α

Extended location of a string α: location of the shortest extension of α whose location is defined

Contracted location of a string α: location of the longest prefix of α whose location is defined



α = bab (has no location)

Construction of suffix trees

Definitions:

suf_i : suffix of σ beginning at position i, e.g. $suf_1 = \sigma$, $suf_n = \$$.

$head_i$: longest prefix of suf_i that is also a prefix of suf_j for some $j < i$.

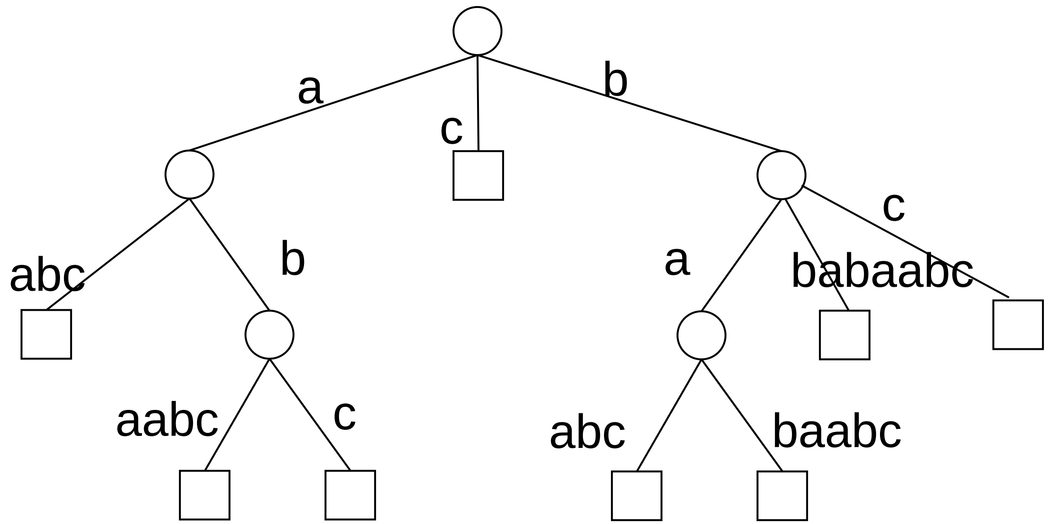
Example: σ = bbabaabc

suf_4 = baabc

$head_4$ = ba

Construction of suffix trees

σ = bbabaabc



Naive suffix tree construction

Start with the empty tree T_0 .

The tree T_{i+1} is constructed from T_i by inserting the suffix suf_{i+1}.

Algorithm *suffix-tree*

Input: string σ

Output: suffix tree T for σ

1 $n := |\sigma|$; $T_0 := \emptyset$;

2 **for** $i := 0$ **to** $n - 1$**do**

3 insert suf_{i+1} into T_i, store the result in T_{i+1} ;

4 **endfor;**

Naive suffix tree construction

All suffixes suf_j with $j \leq i$ have a location in T_i .

→ $head_{i+1}$ = longest prefix of suf_{i+1} that is a prefix of suf_j, with $j < i+1$

Definition:

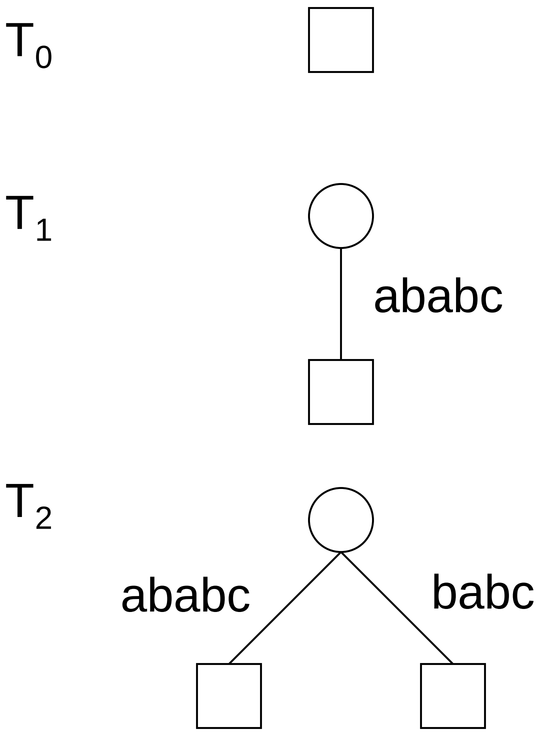
$tail_{i+1} := suf_{i+1} - head_{i+1}$ i.e. $suf_{i+1} = head_{i+1}\, tail_{i+1}$.

$$\overset{(S1)}{\Longrightarrow} \quad tail_{i+1} \neq \varepsilon$$

Naive suffix tree construction

Example: σ = ababc

suf_3 = abc
$head_3$ = ab
$tail_3$ = c

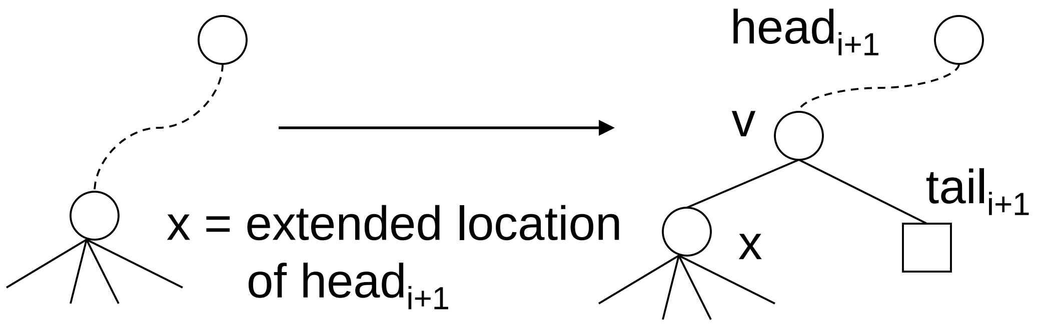


Naive suffix tree construction

T_{i+1} can be constructed from T_i as follows:

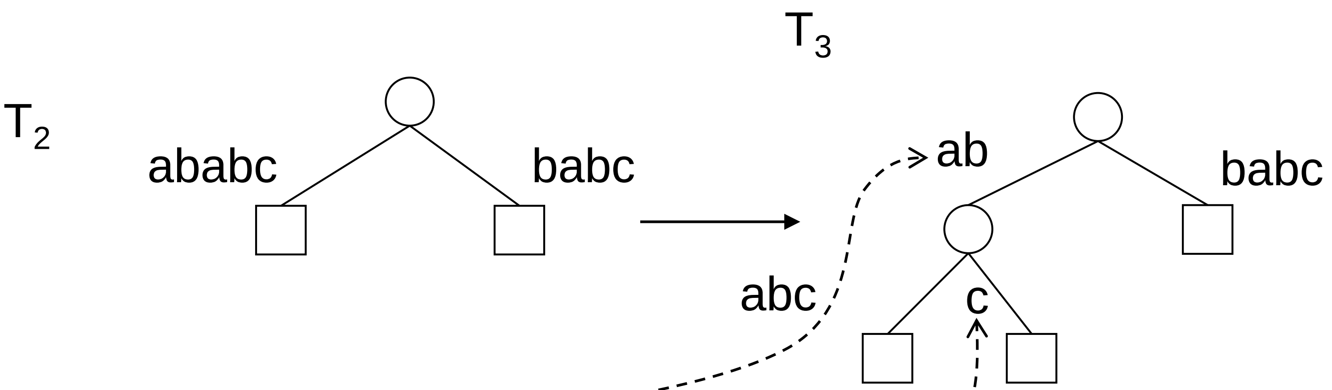
1. Determine the extended location of $head_{i+1}$ in T_i and split the last edge leading to this location into two new edges by inserting a new node.

2. Insert a new leaf as location for suf_{i+1} .



Naive suffix tree construction

Example: σ = ababc



$head_3$ = ab
$tail_3$ = c

Naive suffix tree construction

Algorithm *suffix-insertion*

Input: tree T_i and suffix suf_{i+1}

Output: tree T_{i+1}

$v :=$ root of T_i ;
$j := i$;
repeat
    find child w of v with $\sigma_{w.l} = \sigma_{j+1}$;
    if $w \neq$ nil **then**
        $k := w.l$; $j := j + 1$;
        while $k < w.u$ **and** $\sigma_{k+1} = \sigma_{j+1}$ **do**
            $k := k + 1$; $j := j + 1$;
        endwhile;
    endif;

Naive suffix tree construction

    if $k = w.u$ **then** $v := w$;

until $k < w.u$ or w = nil; /* v is the contracted location of $head_{i+1}$ */

insert the location of $head_{i+1}$ and $tail_{i+1}$ below v into T_i ;

Running time of *suffix-insertion* : $O(n\text{-}i)$

Total time required for the naive construction: $O(n^2)$

The algorithm MCC

(McCreight, 1976)

Idea: Extended location of $head_{i+1}$ in T_i is determined in constant amortized time. (Additional information required!)

When the extended location of $head_{i+1}$ in T_i has been found:
Creating a new node and splitting an edge takes O(1) time.

Theorem 1

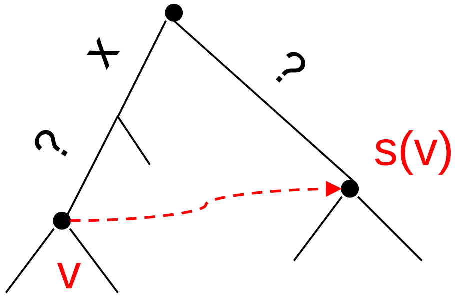
Algorithm MCC constructs a suffix tree for σ with $|\sigma|$ leaves and at most $|\sigma|$ - 1 internal nodes in time $O(|\sigma|)$.

Suffix links

Definition:

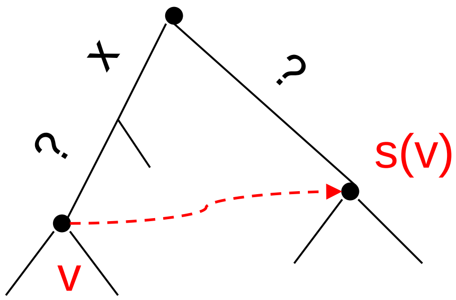
Let *x*? be an arbitrary string where *x* is a single character and ? some (possibly empty) substring.
For an internal node *v* with edge labels *x*? the following holds:

> If there exists a node *s*(*v*) with edge label ?, then there is a pointer from *v* to *s*(*v*) that is called a suffix link.



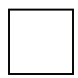
Suffix links

The idea is as follows:
By following the suffix links, we do not have to start each search for a splitting point at the root node. Instead, we can use the suffix links in order to determine these nodes more efficiently, i.e. in constant amortized time.

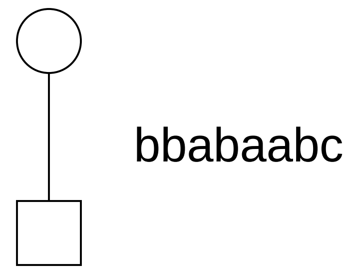


Suffix tree: example

T_0



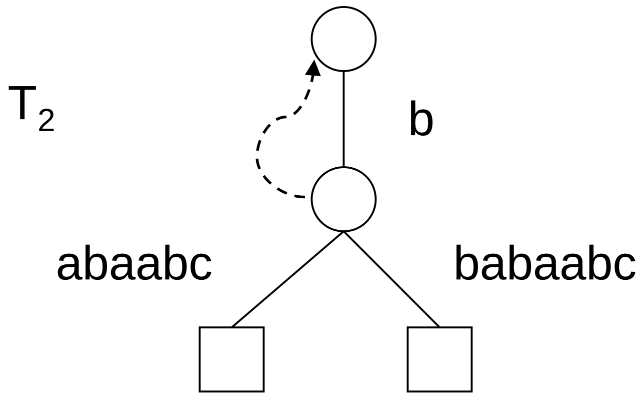
T_1

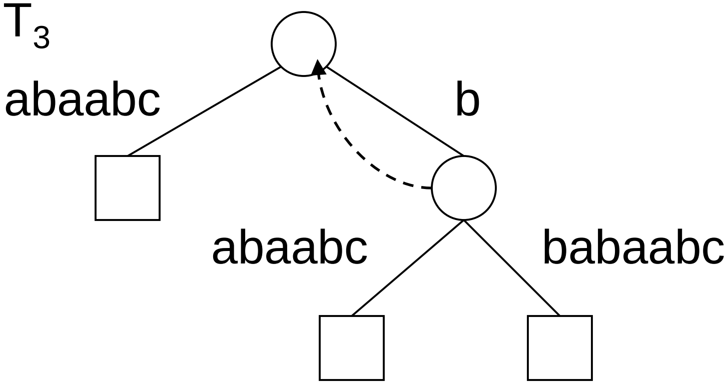


suf_1 = bbabaabc

suf_2 = babaabc
$head_2$ = b

Suffix tree: example

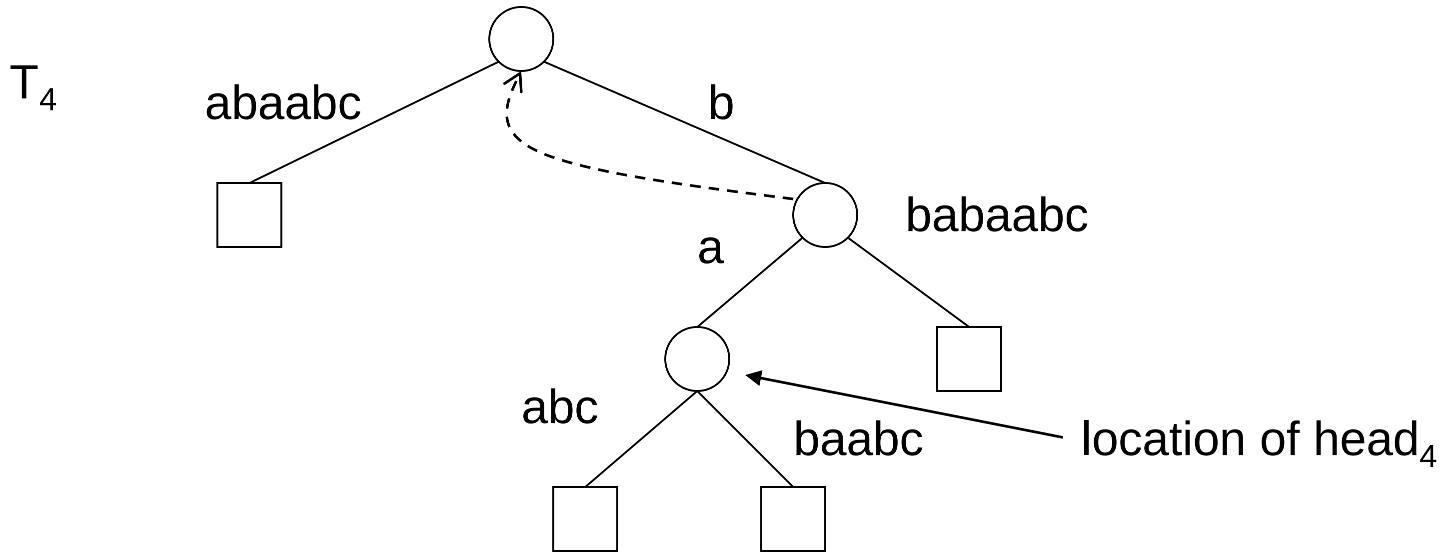




suf_3 = abaabc
$head_3 = \varepsilon$

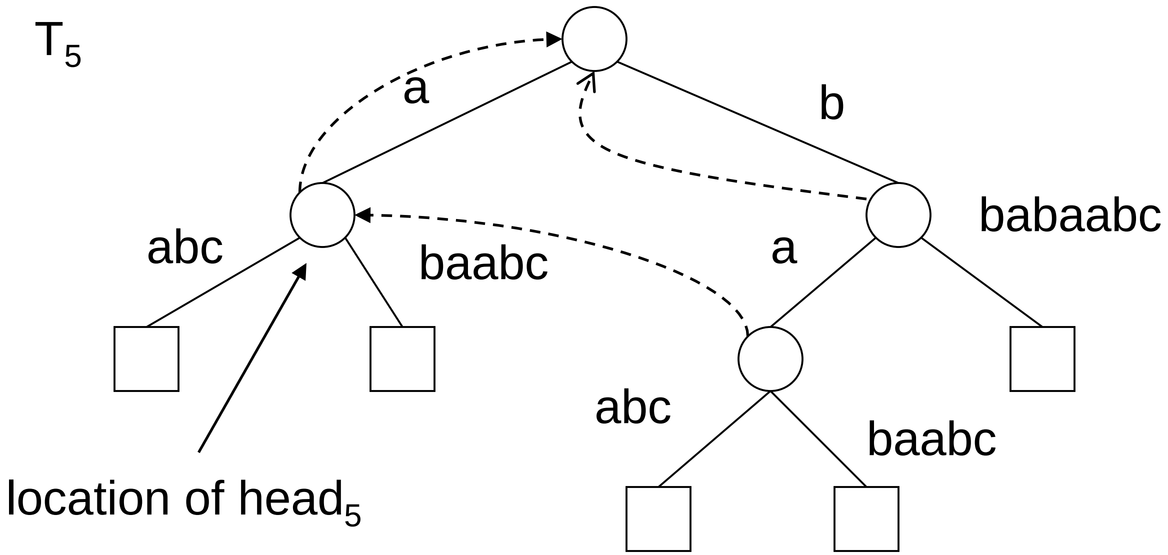
suf_4 = baabc
$head_4$ = ba

Suffix tree: example



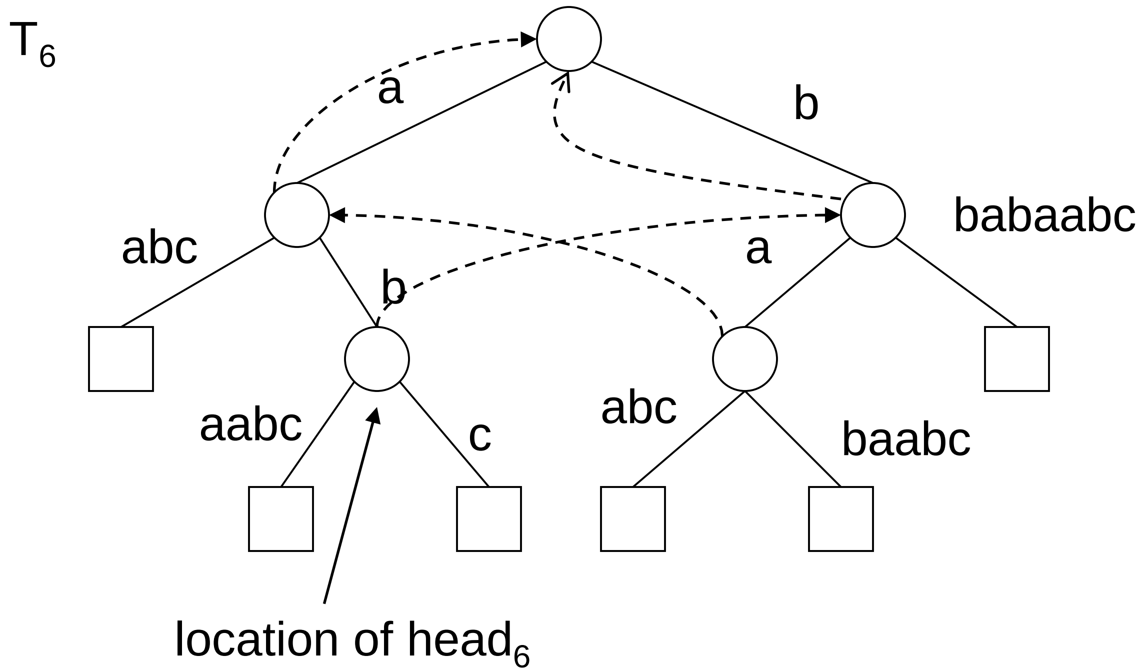
suf_5 = aabc
$head_5$ = a

Suffix tree: example



suf_6 = abc
$head_6$ = ab

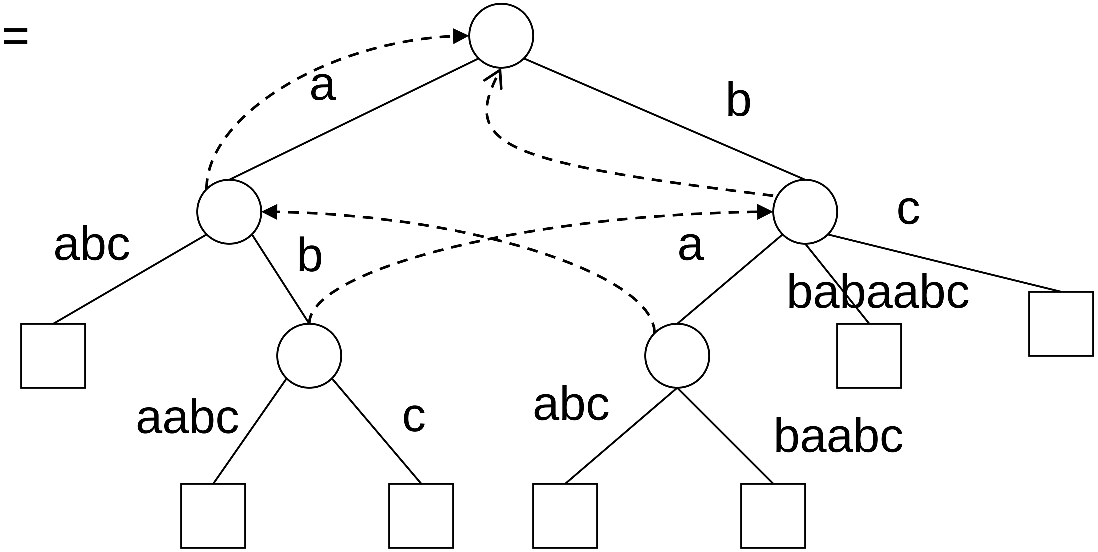
Suffix tree: example



suf_7 = bc
$head_7$ = b

Suffix tree: example

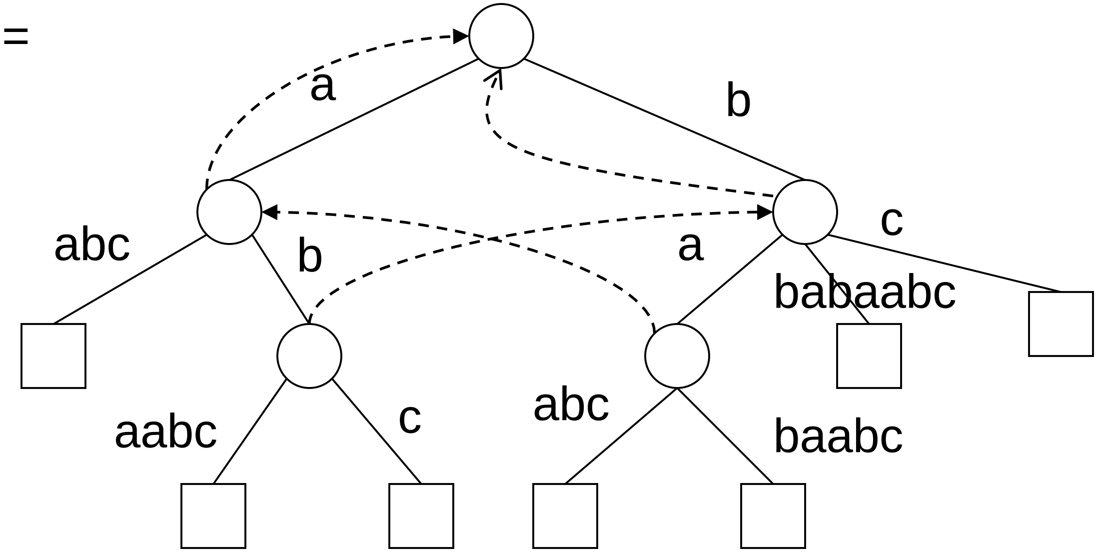
T_7 =



suf_8 = c

Suffix tree: example

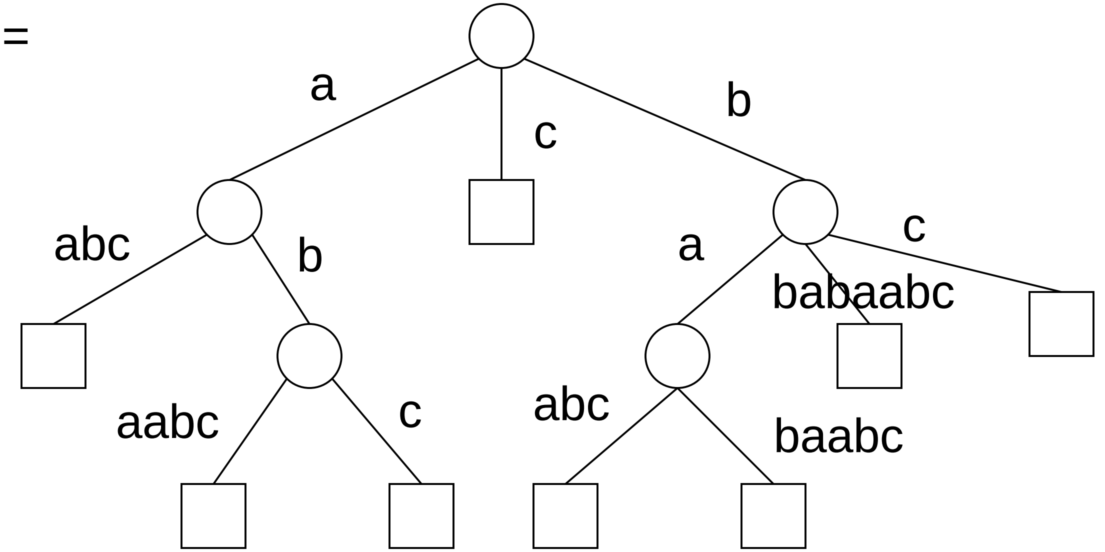
$T_7 =$



$suf_8 = c$

Suffix tree: example

$T_8 =$



The algorithm MCC

Iteration $i + 1$: Given T_i, construct T_{i+1}:

Invariant: In T_i all internal nodes have a suffix link, except for the internal node possibly inserted into T_i in iteration i.

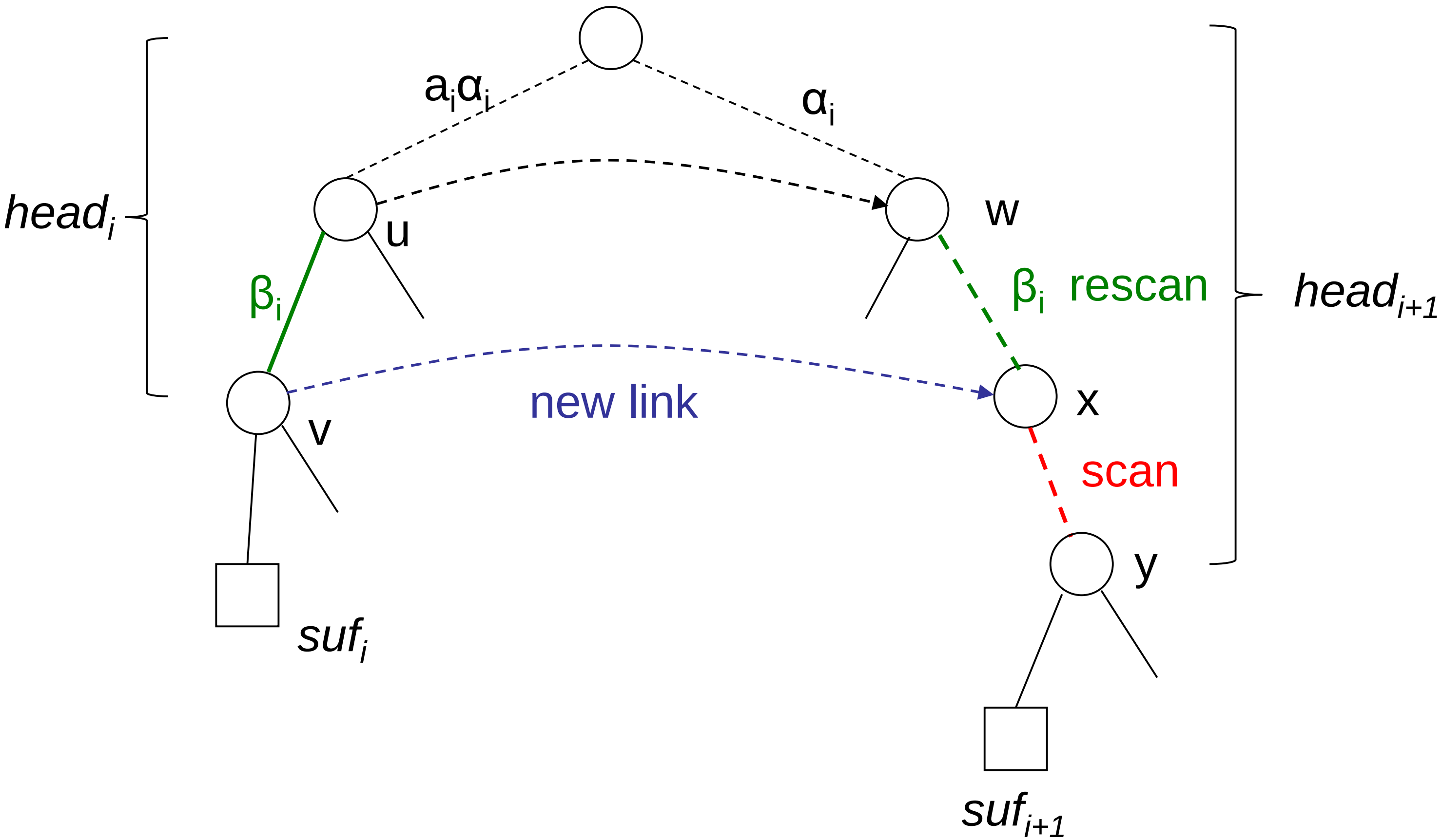
Lemma: If $a\gamma$ has a location in T_i, so does γ in T_{i+1}.

Proof: Note that a string α has a location in T_i if and only if there exist two suffixes suf_j and suf_k, where $1 \leq j \neq k \leq i$, such that α is the longest common prefix of suf_j and suf_k.

Thus if $a\gamma$ is the longest common prefix of suf_j and suf_k, with $1 \leq j \neq k \leq i$, then γ is the longest common prefix of suf_{j+1} and suf_{k+1}, where $1 \leq j+1 \leq i+1$ and $1 \leq k+1 \leq i+1$.

Hence γ has a location in T_{i+1}.

Iteration i +1



MCC traverses the suffix link of the nearest ancestor of suf_i having such a link. Then it identifies $head_{i+1}$, using rescan and scan operations, and sets a new suffix link if required.

Analysis

Iteration i +1:

γ_i = longest prefix of suf_i having a location with suffix link in T_i.

Cost rescan: It is not necessary to scan all the characters of β_i. Since β_i is in the tree, starting at node w, it suffices to traverse the respective edges by inspecting the edge labels. Thus the cost is poportional to number of edges traversed. Whenever an edge is fully traversed, the edge label adds to γ_{i+1}. The rescan starts at a string of length $|\gamma_i|$-1. Therefore the cost is a constant factor times $|\gamma_{i+1}| - (|\gamma_i|-1) +1$.

Cost scan: Proportional to number of character comparisons. The scan starts at a string of length $|head_i|$-1. Thus the cost is a constant factor times $|head_{i+1}| - (|head_i|-1) + 1$.

Analysis

Summation over all iterations

$\Sigma_{0\leq i\leq n-1} (|\gamma_{i+1}| - (|\gamma_i|-1) +1) = |\gamma_n| - |\gamma_0| + 2n \leq 3n$

$\Sigma_{0\leq i\leq n-1} (|head_{i+1}| - (|\mathrm{head}_i|-1) + 1) = |head_n| - |head_0| +2n \leq 3n$

Suffix tree: application

Usage of a suffix tree T:

1 Search for a string α:
Follow the path with edge labels α (takes $O(|\alpha|)$ time).
leaves of the subtree $\triangleq$ occurrences of α

2 Search for the longest substring occurring at least twice:
Find the location of a substring with maximum weighted depth that is an internal node.

3 Prefix search:
All occurrences of strings with prefix α are represented by the nodes of the subtree rooted at the extended location of α in T.

Suffix tree: application

4 Range search for $[\alpha, \beta]$: