Theory I
Algorithm Design and Analysis

(12 - Text search: suffix trees)

Prof. Dr. Th. Ottmann

Text search

Different scenarios:

Dynamic texts

- Text editors
- Symbol manipulators

Static texts

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

Properties of suffix trees

Search index

for a text σ in order to search for patterns α

Properties:

1. **Substring search** in time $O(|\alpha|)$.

2. **Queries to σ itself**, e.g.:

   Longest substring in σ occurring at least twice.

3. **Prefix search:** all positions in σ with prefix α.

Properties of suffix trees

4. **Range search:** all positions in σ in interval $[\alpha, \beta]$ with $\alpha \leq_{\text{lex}} \beta$, e.g.

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

5. **Linear complexity:**
   Required space and time for construction in $O(|\sigma|)$

Tries

Trie: tree for representing keys.

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

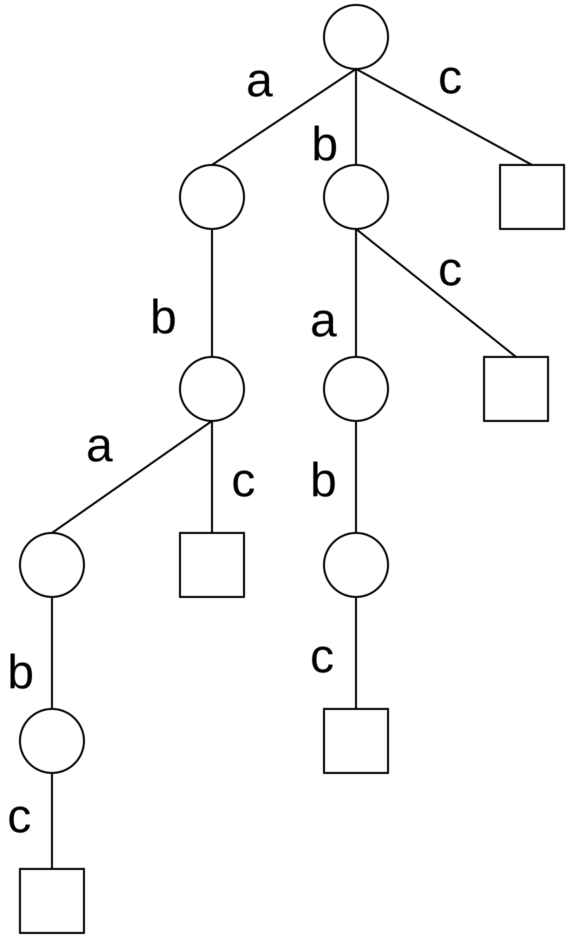
Key $\hat{=}$ String $\in \Sigma^*$

Edge of a trie T: label with a single character from Σ

Neighboring edges: different characters

Tries

Example:



Tries

Each **leaf** represents a key:

corresponds to the labeling of the edges on the path from the root to the leaf

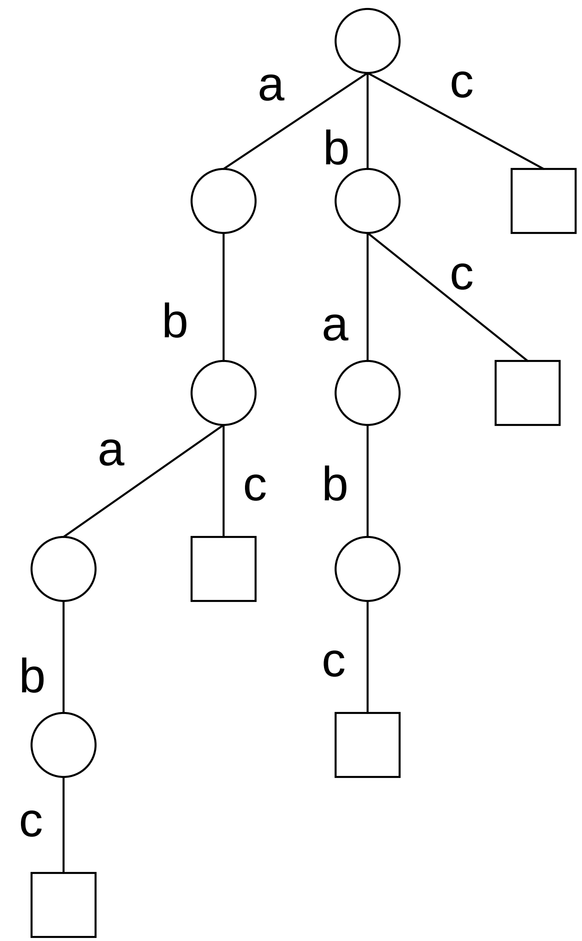
! Keys are not stored in nodes !

Suffix tries

Trie for all suffixes of a text

Example: σ = ababc

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



Suffix tries

Internal nodes of a suffix trie = substrings of σ.

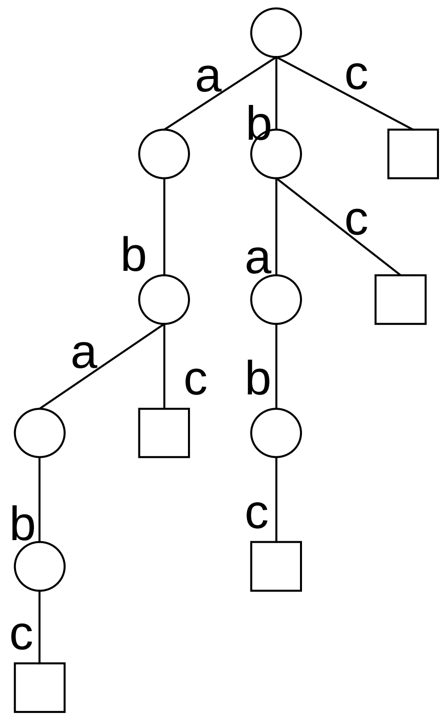
Each proper substring of σ is represented as an internal node.

Let $\sigma = a^n b^n$: $\exists$ $n^2 + 2n + 1$ different substrings = internal nodes

$\Rightarrow$ Space requirement in $O(n^2)$.

Suffix tries

A suffix trie T fulfills some of the required properties:



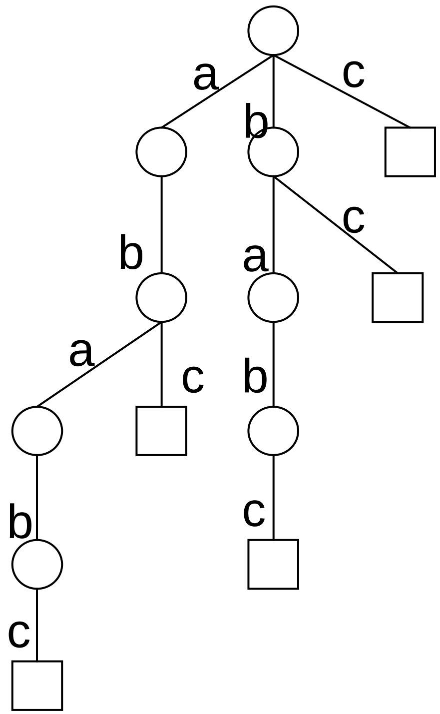
1. String matching for α: follow the path with edge labels α in T in time $O(|\alpha|)$.
   #leaves of the subtree $\triangleq$ #occurrences of α

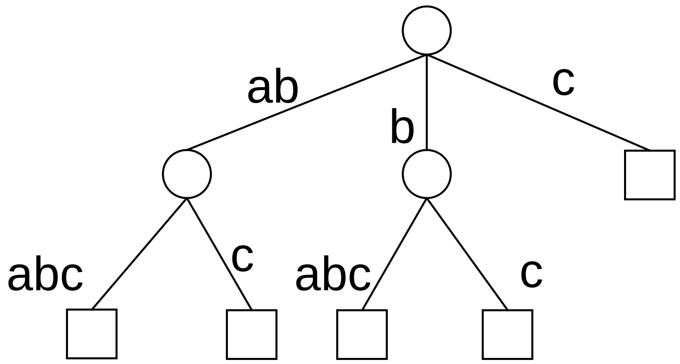
2. Longest repeated substring: internal node with the greatest depth which has at least two children.

3. Prefix search: all occurrences of strings with prefix α can be found in the subtree below the internal node corresponding to α in T.

Suffix trees

A suffix tree is created from a suffix trie by contraction of unary nodes:





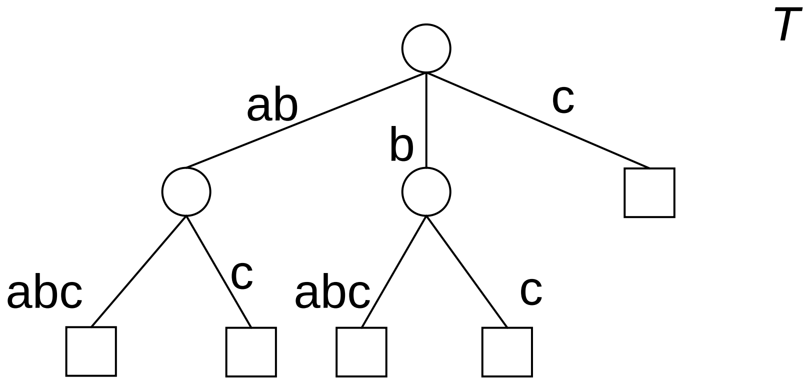
suffix tree = contracted suffix trie

Internal representation of suffix trees

Child-sibling representation

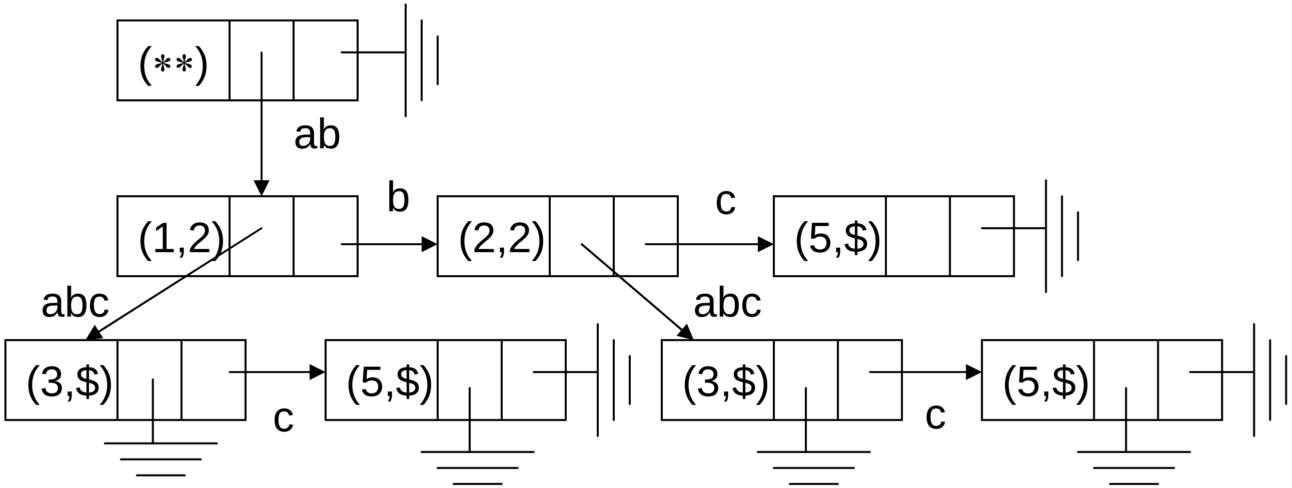
Substring: pair of numbers (i,j)

Example: σ = ababc



Internal representation of suffix trees

Example: σ = ababc



node $v = (v.w, v.o, v.sn, v.br)$

Further pointers (suffix pointers) are added later

Properties of suffix trees

(S1) No suffix is prefix of another suffix;
this holds if (last character of σ) = $ $\notin \Sigma$

Search:

(T1) edge $\widehat{=}$ 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 $n = |\sigma| \neq 1$

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

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

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

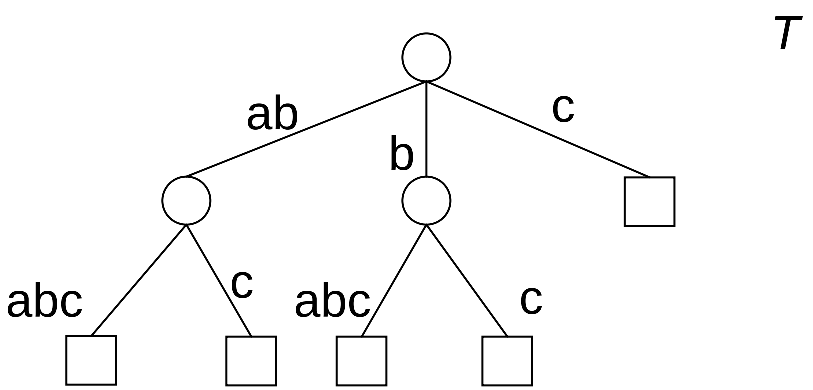
Construction of suffix trees

Definition:

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

Path: a partial path ending in a leaf

Location of a string α: node at the end of the partial path labeled with α (if it exists).

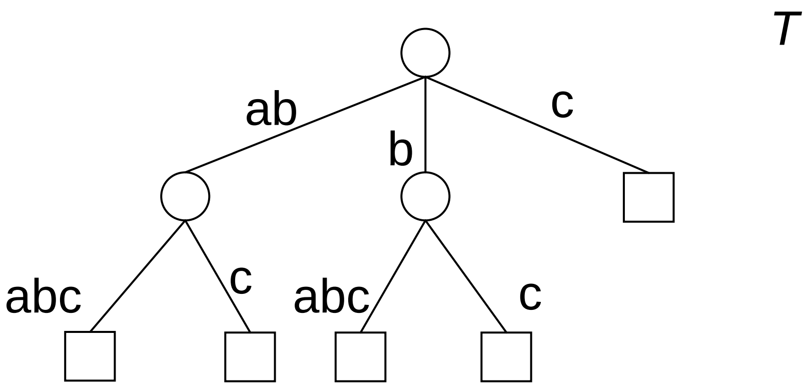


Construction of suffix trees

Extension of a string α: string with prefix α

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

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



Construction of suffix trees

Definitions:

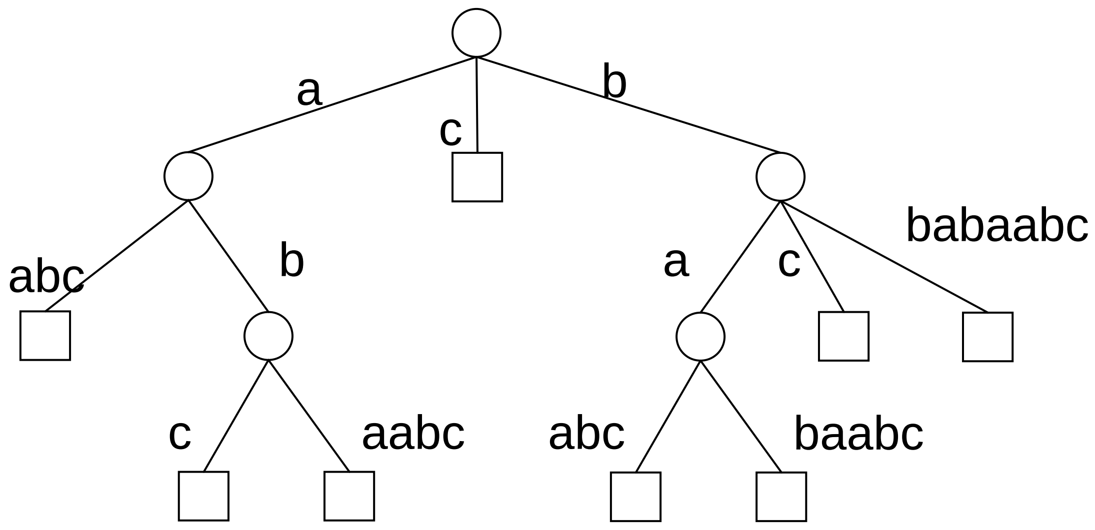
suf_i: suffix of σ starting at position i, e.g.
$suf_1 = \sigma$, $suf_n = \$$.

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

Example: σ = bbabaabc α = baa (has no location)
suf_4 = baabc
$head_4$ = ba

Construction of suffix trees

σ = bbabaabc



Naive suffix-tree construction

Begin with the empty tree T_0

Tree T_{i+1} is created from T_i by inserting suffix suf_{i+1}.

Algorithm suffix tree

Input: a text σ

Output: the suffix tree T for σ

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

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

3 insert suf_{i+1} in T_i , resulting in T_{i+1} ;

4 **end for**

Naive suffix-tree construction

In T_i all suffixes suf_j ($j < i$) already have a location.

→ $head_i$ = longest prefix of suf_i whose extended location in T_{i-1} exists.

Definition:

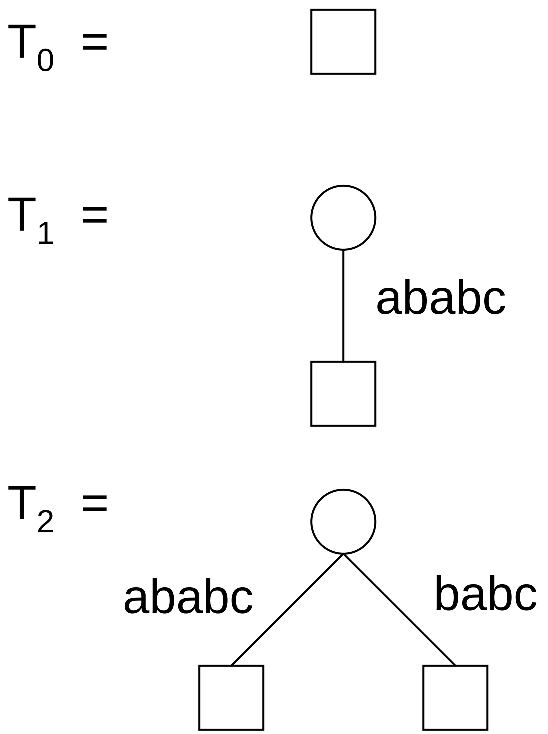
$tail_i := suf_i - head_i$, i.e. $suf_i = head_i tail_i$.

$$\overset{(S1)}{\Longrightarrow} \quad tail_i \neq \varepsilon.$$

Naive suffix-tree construction

Example: σ = ababc

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

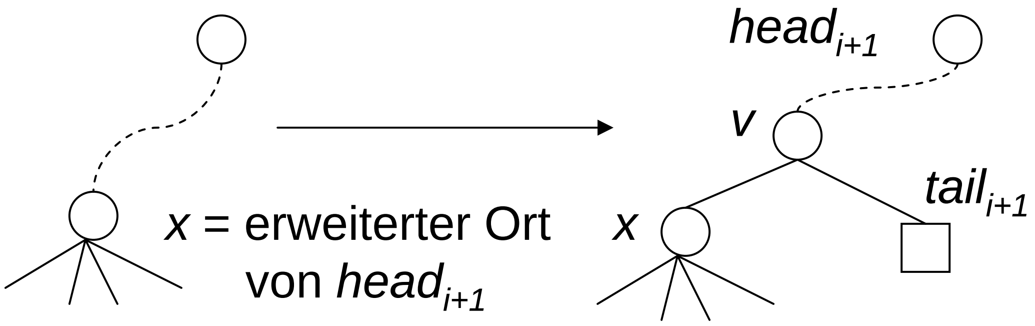


Naive suffix-tree construction

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

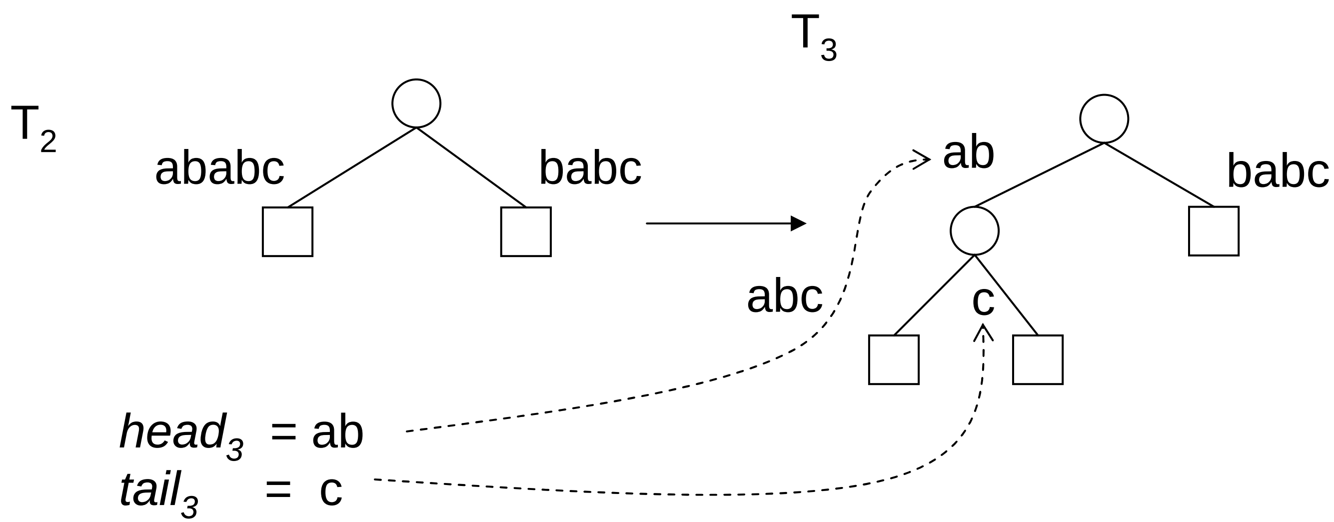
1. Determine the extended location of $head_{i+1}$ in T_i and split the last edge leading to this location into two new edges by inserting a new node.

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



Naive suffix-tree construction

Example: σ = ababc



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 σ_{w.u} = σ_{j+1}
    k := w.u – 1;
    while k < w.o and σ_{k+1} = σ_{j+1} do
        k := k +1; j := j + 1
    end while
```

Naive suffix-tree construction

if $k = w.o$ **then** $v := w$

until k $< w.o$ or w = nil

/* v is the contracted location of $head_{i+1}$ */

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

Running time for suffix insertion: O( )

Total time for naive suffix-tree construction: O( )

The algorithm *M*

(Mc Creight, 1976)

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

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

Analysis of algorithm *M*

Theorem 1

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

Remark:

Ukkonen (1992) found an $O(n)$ **on-line** algorithm for the construction of suffix trees, i.e. after each step i, the resulting structure is a correct suffix tree for $t_1 \ldots t_i$ (where $\sigma = t_1 \ldots t_n$).

Suffix tree: application

Usage of suffix tree *T*:

1 Search for string α: follow the path with edge labeling α in *T* in time $O(|\alpha|)$.

leaves of the subtree $\hat{=}$ occurrences of α

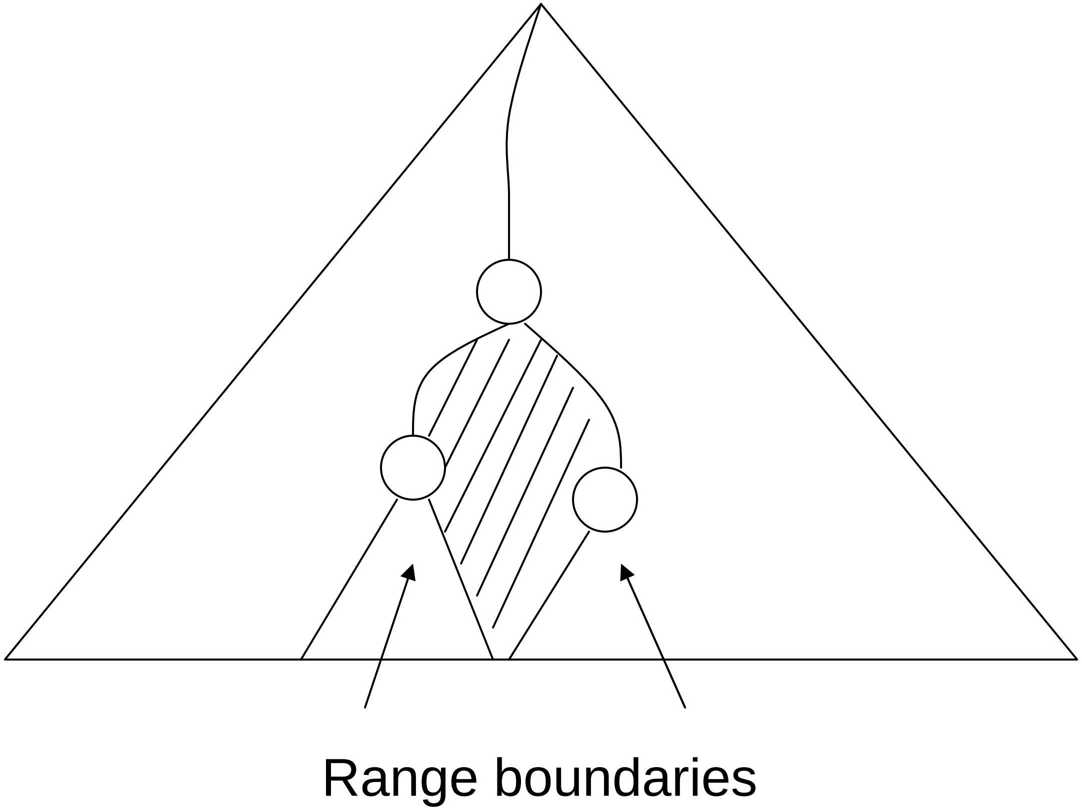
2 Search for longest repeated substring:

Find the location of a substring with the greatest weighted depth that is an internal node

3 Prefix search: All occurrences of strings with prefix α can be found in the subtree below the „location“ of α in *T*.

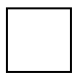
Suffix tree: application

4 Range query for [α, β] :

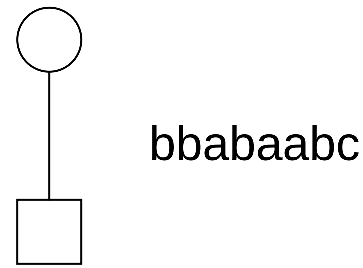


Suffix tree: example

T_0 =



T_1 =

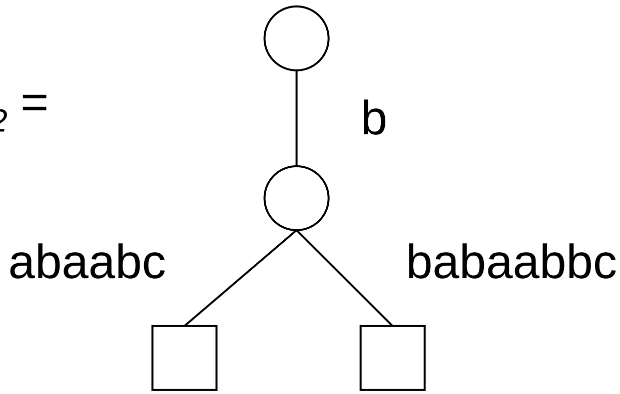


suf_1 = bbabaabc

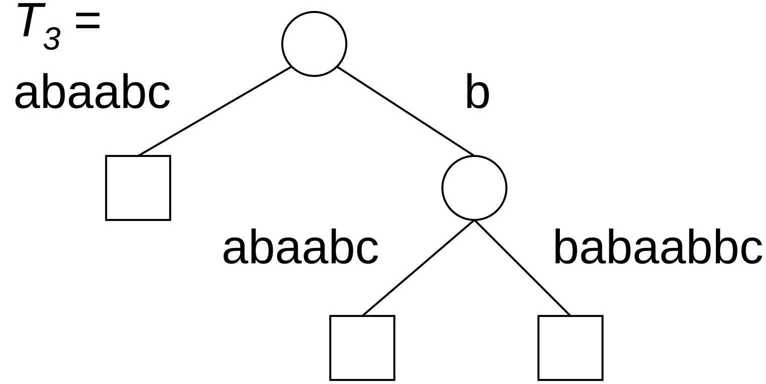
suf_2 = babaabc
$head_2$ = b

Suffix tree: example

$T_2 =$



$T_3 =$

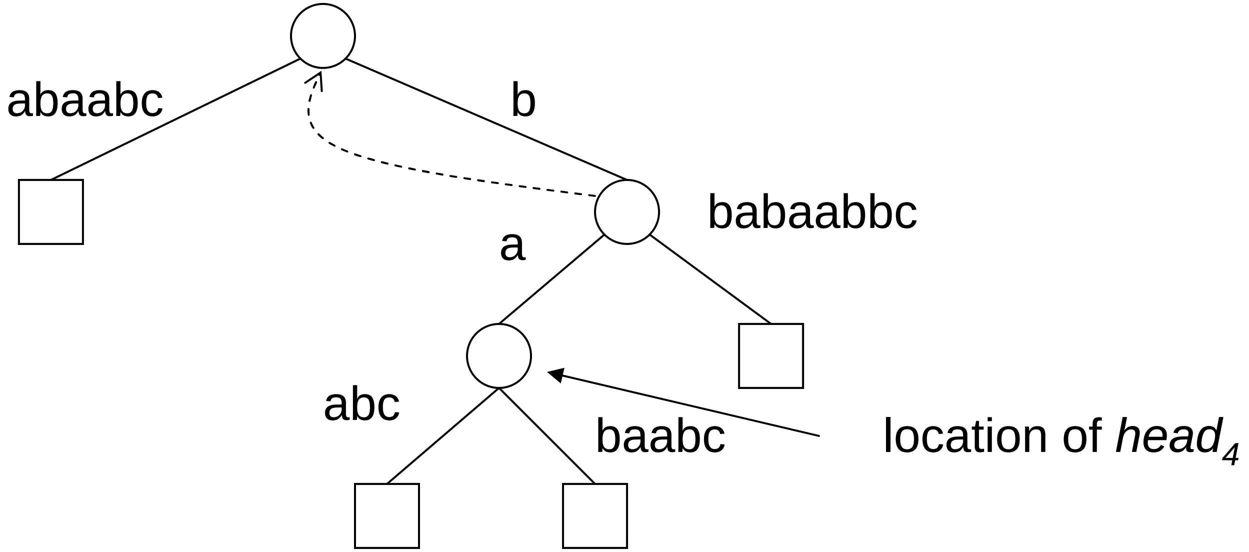


suf_3 = abaabc
$head_3 = \varepsilon$

suf_4 = baabc
$head_4$ = ba

Suffix tree: example

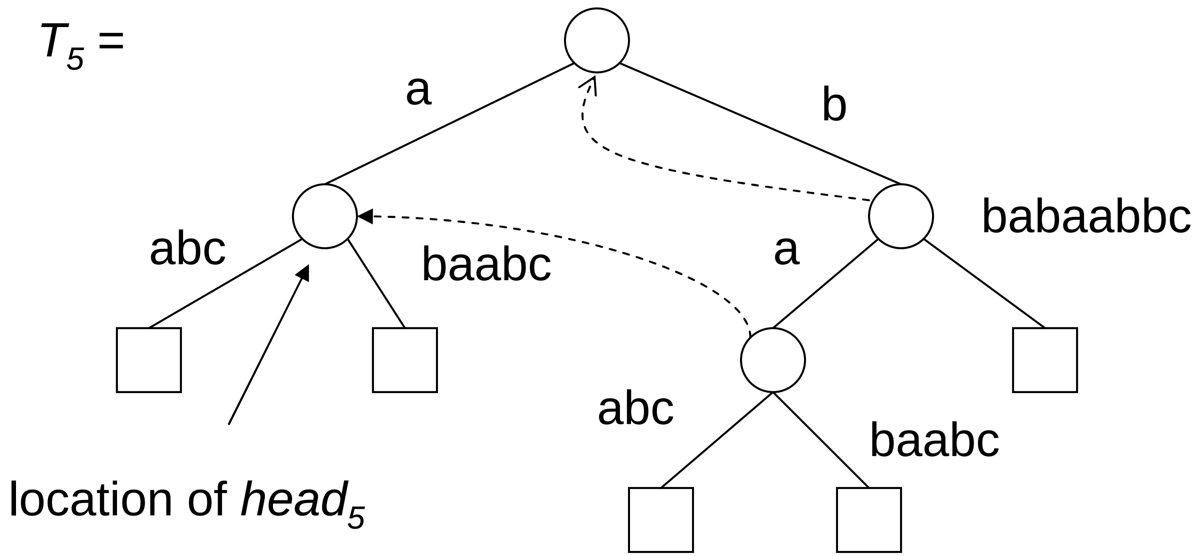
T_4 =



suf_5 = aabc
$head_5$ = a

Suffix tree: example

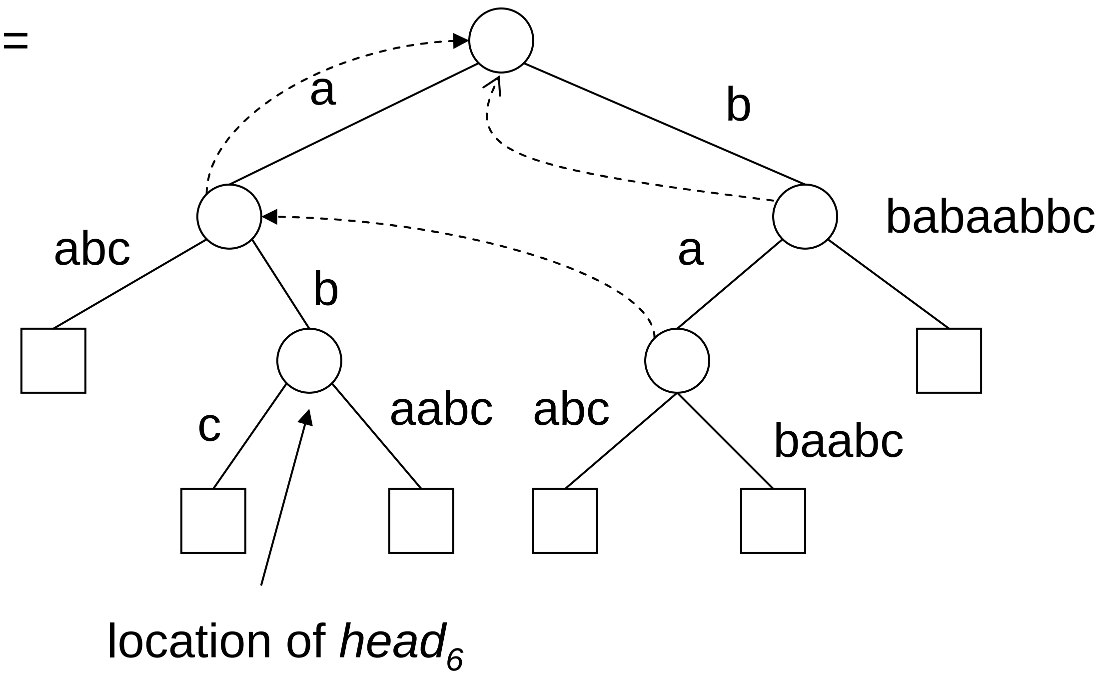
T_5 =



suf_6 = abc
$head_6$ = ab

Suffix tree: example

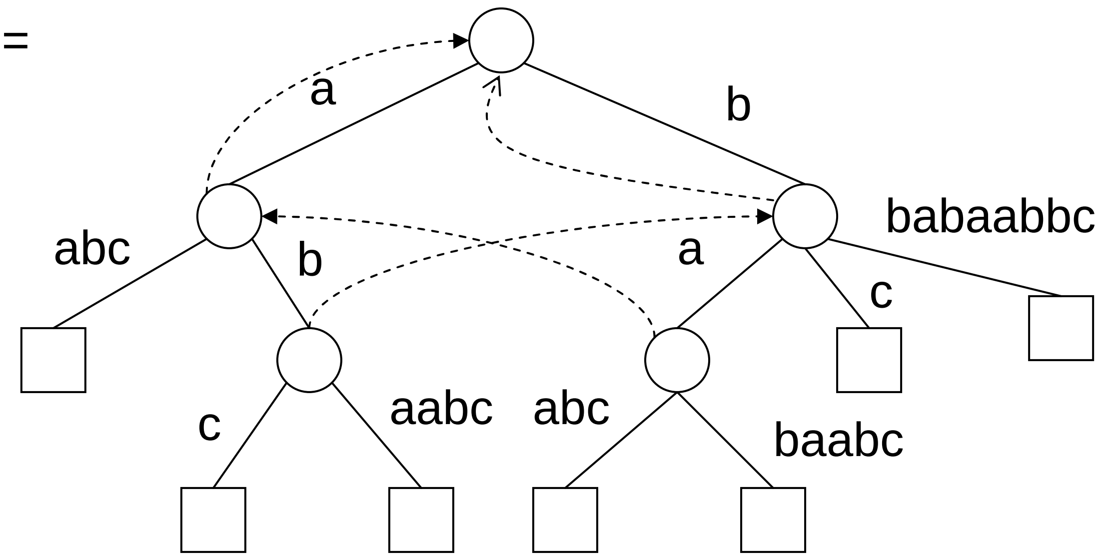
T_6 =



suf_7 = bc
$head_7$ = b

Suffix tree: example

$T_7 =$



$suf_8 = \text{c}$

Suffix tree: example

$T_8 =$