Definition

Let $T \in \Sigma^n$ a text. A directed tree $B_T = (V, E)$ with root r is called compact suffix tree *for T, if it satisfies the following conditions:*

1. *B_T has exactly n leaves, labeled with numbers 1 to n.*
2. *Each internal node of B_T has at least two successors.*
3. *The edges of B_T are labeled with substrings of T.*
4. *Labels of edges leaving the same node start with pairwise different symbols.*
5. *The path from the root to leaf i is labeled with $T_{i,n}$, $1 \leq i \leq n$.*

Lemma

Let $T \in \Sigma^n$ a text. A compact suffix tree B_T for T has $\mathcal{O}(n)$ nodes. Labeling all edges takes $\mathcal{O}(n \log(n))$ bits.

Beweis: n Blätter

$\leq n-1$ innere Knoten

$\Rightarrow 2n-1$ Knoten

Linear viele Kanten $\rightarrow n \cdot \#$Bits pro Markierung

Markierung ist Paar $(i,j) \in [1:n]^2$ je mit

$\log(n)$ Bits darstellbar.

Construction: (Ukkonen's algorithm)

Definition

An implicit suffix tree *is the tree resulting from the compact suffix tree for $T \cdot \$$ by*

1. *removing all occurrences $\$$ from the labels.*
2. *removing unlabeled edges (and nodes which are afterwards no longer reachable from the root) and*
3. *removing nodes with only one child (merging the incoming and the outgoing edge to one edge labeled with the concatenation of the previous labels).*

Approach: Process T symbol by symbol from left to right (online algorithm) constructing implicit suffix trees IB_k, corresponding to the prefix $T_{0,k}$. IB_0 consists only of the root. IB_1 has two nodes (root and a leaf labeled with 1), connected by an edge labeled with $T_{1,1}$.

Now we construct IB_{i+1} from IB_i, $1 \leq i \leq n-1$ as follows:

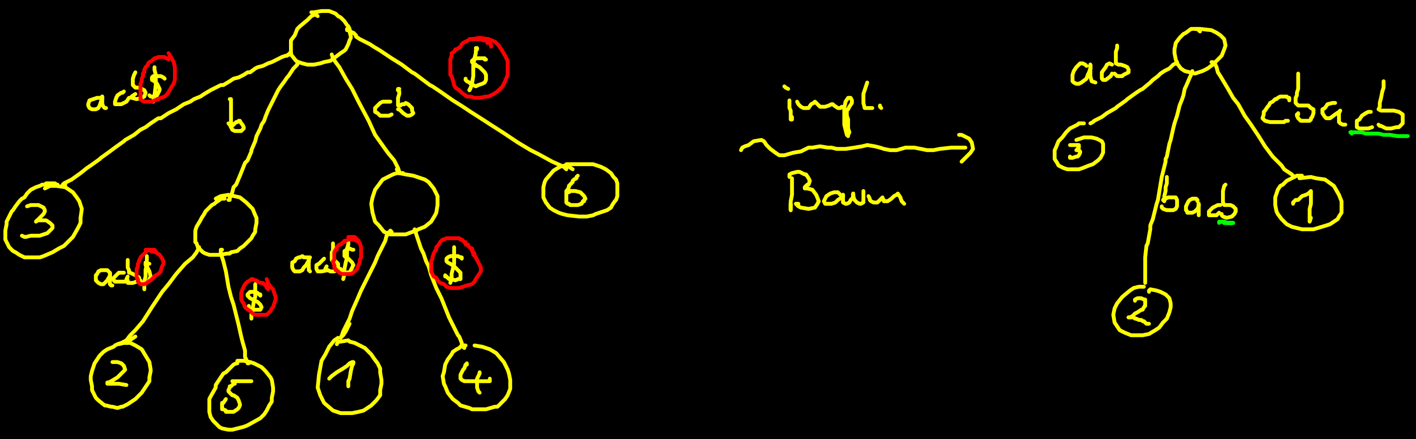
```
for $i := 1$ to $n-1$ do begin
    // Phase $i+1$
    for $j := 1$ to $i+1$ do begin
        Traverse $IB_i$ along the path $T[j..i]$;
        If necessary extend the tree at the
        position reached this way by $T[i+1]$;
        // Details follow
    end;
end;
```

Rule 1: If the path labeled $T_{j,i}$ ends in a leaf, T_{i+1} is appended to the label of the edge leading to the leaf.

Rule 2: If the path does **not** end in a leaf and there is no possibility to continue it with T_{i+1}, a new edge to a new leaf is created and labeled with T_{i+1}. The leaf is labeled j. If $T_{j,i}$ ends amidst an edge, additionally a new internal node has to be created at the respective position.

Rule 3: If the path can be continued with T_{i+1} nothing is done.

Example: $T = cbacb\$$



Caution: Nested loops + traversal along $T_{j,i}$ $\Rightarrow$ running time cubic in length of text.

Tricks:
1.) If for given j Rule 3 is applied for the first time:
$\Rightarrow$ The path labeled $T_{j,i}$ can be continued with T_{i+1}, created when inserting word w.
$\Rightarrow$ Suffixes of w inserted in an earlier phase guarantee existence of a continuation of $T_{j',i}$, $j' > j$ with T_{i+1}.

$\Rightarrow$ Rule 3 implies termination of the current phase.

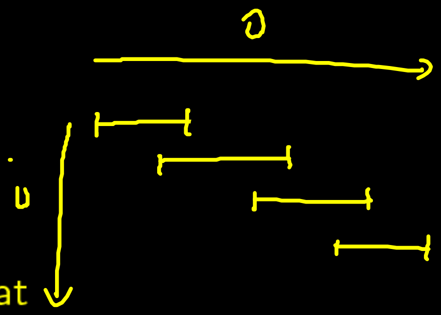
2.) A leaf always remains a leaf. If labeled j it represents all suffixes of T starting at position j. Whenever we insert $T_{j,i+1}$ in a later phase, we reach leaf j and apply rule 1. With 1.) it follows that:

- Each phase $i+1$ starts with a sequence of extensions (beginning with $j = 1$), in which only rules 1 and 2 are applied.
- Let j_i denote the last extension in phase i in which rule 1 or rule 2 is applied. As each application of rule 2 generates a new leaf $j_i \leq j_{i+1}$ holds. Thus the initial sequence of applications of rules 1 and 2 can't become shorter in later phases.

$\Rightarrow$ All extensions for $j \in [1 : j_i]$ in phase $i+1$ will be by rule 1 because either there was already a leaf labeled j in phase i (in this case only its edge label is extended with T_{i+1}) or in phase i the second rule has been applied for $T_{j,i}$ (we consider $j \in [1 : j_i]$, so rule 3 is ruled out for extension j) so now there exists a leaf labeled j for which the edge label is extended.

$\Rightarrow$ We don't have to do extensions $1..j_i$ **explicitly** if we mark the leaves' edges with (p, e), e a global symbol meaning current end of text. (e is set to $i+1$ in phase $i+1$.)

3.) For $j \in [j_i + 1, i + 1]$ use rule 2 or 3.



- rule 3 $\Rightarrow j_{i+1} := j - 1$ (i.e. $j = j_{i+1} + 1$); terminate.
- phase terminated by different rule $\Rightarrow j_{i+1} := i + 1$.

Observation: Two consecutive phases have at most (WC is that rule 3 ends phase) one j in common, for which both do **explicit** extensions:

Phase 2: compute explicit extensions for $j = j_1 + 1 \dots j_2 + 1$,
Phase 3: compute explicit extensions for $j = j_2 + 1 \dots j_3 + 1$,
...
Phase i-1: compute explicit extensions for $j = j_{i-2} + 1 \dots j_{i-1} + 1$,
Phase i: compute explicit extensions for $j = j_{i-1} + 1 \dots j_i + 1$.

$\Rightarrow |T|$ many phases and $j_i \leq n$ imply at most $2n$ explicit extensions.

```
Create tree $IB_1$;
$j[1] := 1$; // leaf 1 already exists
for $i$:= 1 to $n-1$ do begin
    // Phase $i+1$
    $e := i+1$; // all implicit extensions
    $j[i+1] := i+1$; // no application of rule 3
    for $j$:= $j[i]+1$ to $i+1$ do begin
        Traverse $IB_i$ along path $T[j..i]$;
        If necessary, extend $IB_i$ by $T[i+1]$;
        if (rule 3 was used) then begin
            $j[i+1] := j-1$;
            End phase $i+1$;
        end;
    end;
end;
```

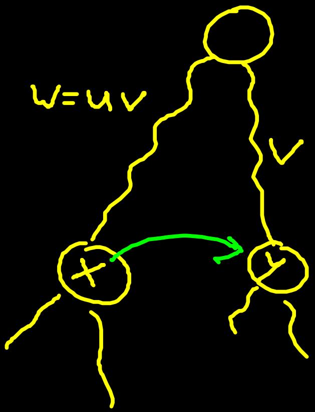
Example: $T = acacag$.

4.) Speed up traversal of edges labeled with more than one character by only evaluating the first character. The position at which to continue in the *traversal word* is determined from the indices saved for the edge. (We already noticed that in phase $i+1$ each word $T_{j,i}$ is present in the tree.)

5.) Add utility links:

Definition

Let $w = u \cdot v$, $u \in \Sigma$, $v \in \Sigma^\star$, and IB an implicit suffix tree. If IB contains an internal node x, reached from the root by w and a node y with path-label v, the suffix link of x points to y.

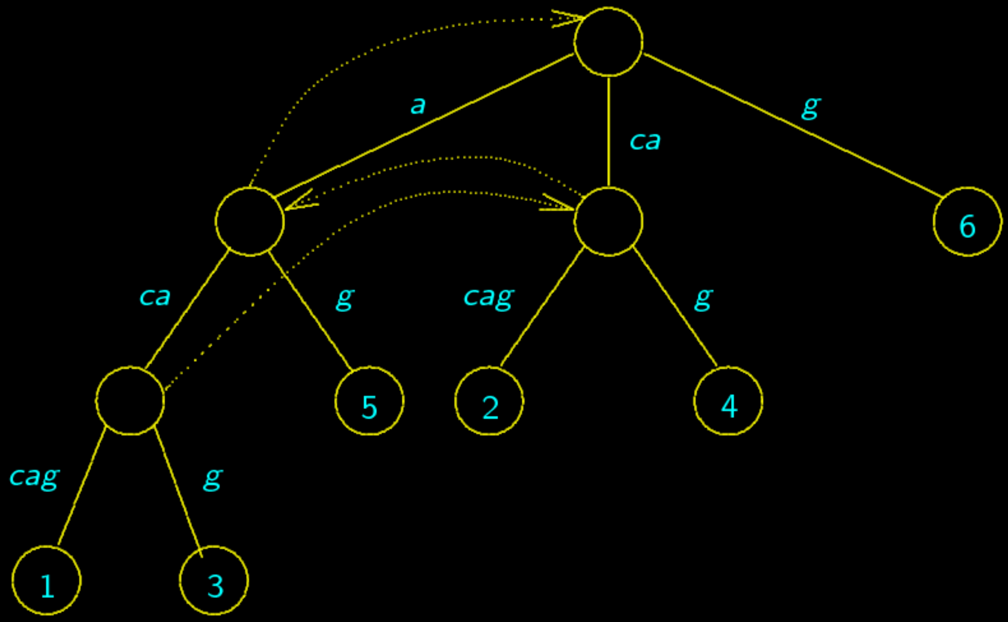


Advantage: The places of explicit extensions can be found without traversing the tree (starting at the root) each time.
If in phase $i+1$ the extension for $T_{j,i+1}$ has to be done, the tree has to be traversed along $T_{j,i}$ and the existence of T_{i+1} at the position x reached this way has to be checked.
The next extension then considers $T_{j+1,i+1}$. The node reached by $T_{j+1,i}$ is found via the suffix link of x (if there is an internal node).
The same holds for the next extension (a suffix link leads possibly directly to the node reached by $T_{j+2,i}$) and so on.

x no internal node: Return to y, the last node on the way to x (we can always save this node) and follow this node's suffix link.
$\Rightarrow$ The node reached thusly is reached by a prefix of $T_{j+1,i}$.From there we have to continue using the symbols *between* y and x.

Consequence: Assuming each internal node has a suffix link, an **explicit** extension has constant amortized running time.

Example: Extend tree by $T_{j,i+1} = acag$ and its suffixes:



Lemma

If a new internal node x with path label $u\alpha$, $u \in \Sigma$, is created by extension j of phase $i+1$ then either α already ends at an internal node of the current tree or this node is created at the next extension (extension $j+1$ of phase $i+1$).

Proof: New internal node $\Leftrightarrow$ rule 2 $\Leftrightarrow$ path $T_{j,i}$ ends amidst an edge label with next character c not equal T_{i+1}.
$\Rightarrow$ Earlier phase inserted $T_{j,i} \cdot c$.
$\Rightarrow$ The same phase afterwards processed $T_{j+1,i} \cdot c$.
$\Rightarrow$ $\exists$ path $\mathcal{P}$ for $T_{j+1,i}$ in the tree.
a) $\mathcal{P}$ can only be continued with c:
$\Rightarrow$ Extension by $T_{j+1,i+1}$ creates an internal node at the position considered.
b) $\mathcal{P}$ can be continued with various symbols:
$\Rightarrow$ At the position considered an internal node must be present. $\square$

Corollary

For Ukkonen's algorithm we can assure that each internal node created has a suffix link after the following extension. This requires constant extra time.

Proof: Induction: IB_1 has no internal nodes relevant for suffix links (at the root $u = \varepsilon$).
Assumption: Assumption holds after phase i.
Lemma 2 $\Rightarrow$ The target node of a node created in the j-th extension of phase $i+1$ will be present after the $(j+1)$-th extension of the same phase. (This gives a simple method to create suffix links: We remember nodes created by rule 2 and add the suffix link after reaching or creating the target node during the next extension.)
Since it is impossible that the last extension of a phase (considering the single character T_{i+1}) creates a new internal node all new internal nodes will have a suffix link after the $(i+1)$-th phase. $\square$

Running time: At most $2 \cdot |T|$ explicit extensions with constant amortized cost lead to running time linear in $|T|$.

Compact suffix tree: Add a $|T|+1$-th phase to the algorithm with $T' = T \cdot \$$. Afterward replace e with $|T'|$ by a tree traversal (linear time)

$\Rightarrow$ compact suffix tree for T'.

Applications

String-Matching:

Text T fixed, String P varies (suffix tree reasonable only in this case!).
$\Rightarrow$ Running time in $\mathcal{O}(|P| + k)$ for k the number of occurrences of P in T.
(By assumption the subtree reached via P has k leaves and thus at most $2k-1$ nodes and can be traversed in time $\mathcal{O}(k)$.)

Set-Matching: Sequential Search of all P_i in the suffix tree has the same running time bound as the Aho-Corasick algorithm.

But: Aho-Corasick creates search term tree of size $\mathcal{O}(m)$, $m := \sum_{1 \le i \le N} |P_i|$, in time $\mathcal{O}(m)$, and searches in time $\mathcal{O}(n)$, $n := |T|$.
Suffix tree has size $\mathcal{O}(n)$, construction time $\mathcal{O}(n)$ and search time $\mathcal{O}(m)$.
$\Rightarrow$ If all P_i together are longer than the text, the suffix tree solution needs less space but more time (preprocessing ignored). If the set of the strings is shorter than the text, the Aho-Crasick needs less space but more time.
$\Rightarrow$ We observe a place/time-*trade-off*, as no solution is superior in place and time consumption at the same time.