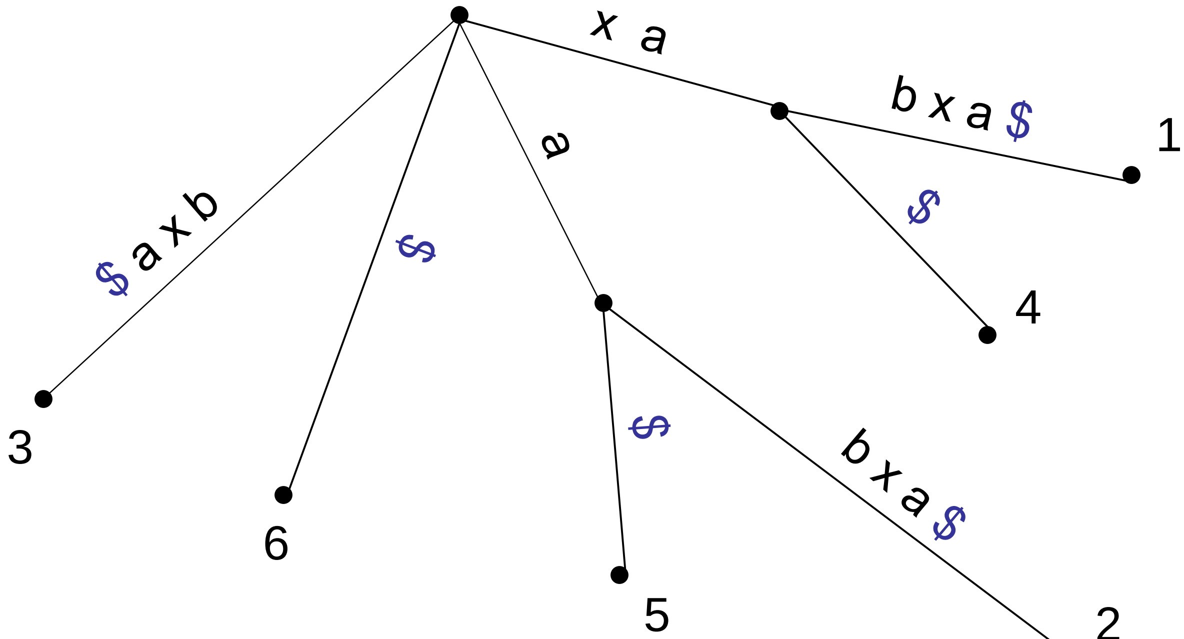
07 – Suffix Trees (2)

Suffix tree

t = x a b x a $
1 2 3 4 5 6



Ukkonen's algorithm: implicit suffix trees

Definition: An *implicit suffix tree* is a tree obtained from the suffix tree for $t\$$ by

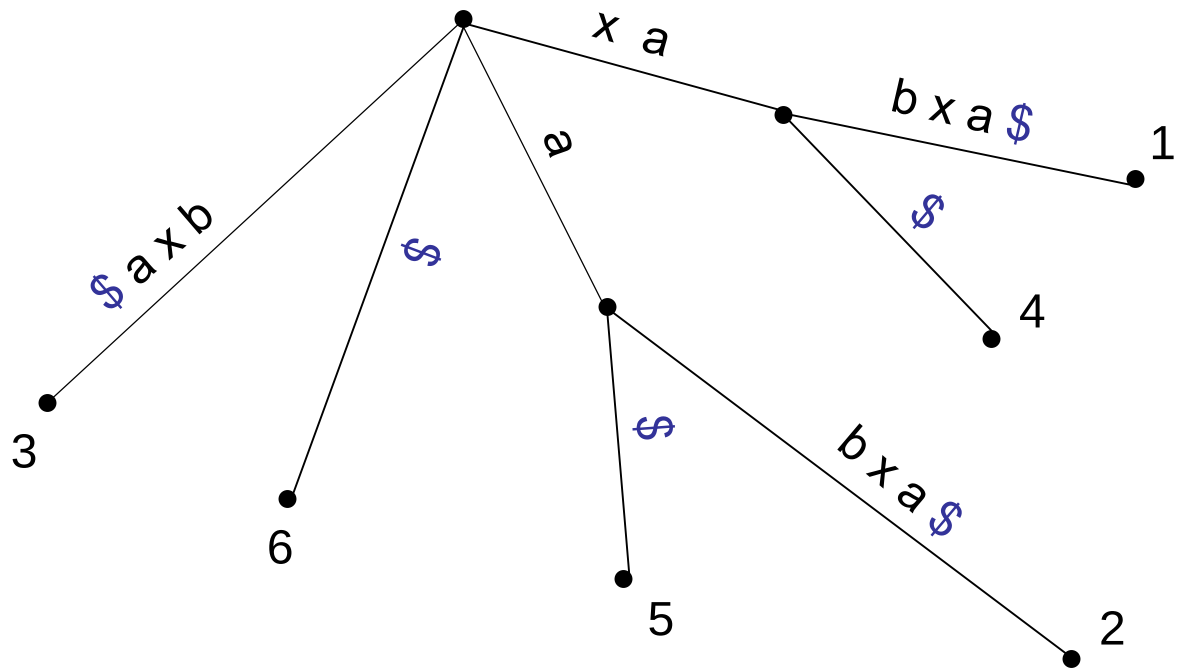
(1) deleting every copy of \$ from the edge labels,

(2) deleting edges that have no label,

(3) deleting unary nodes.

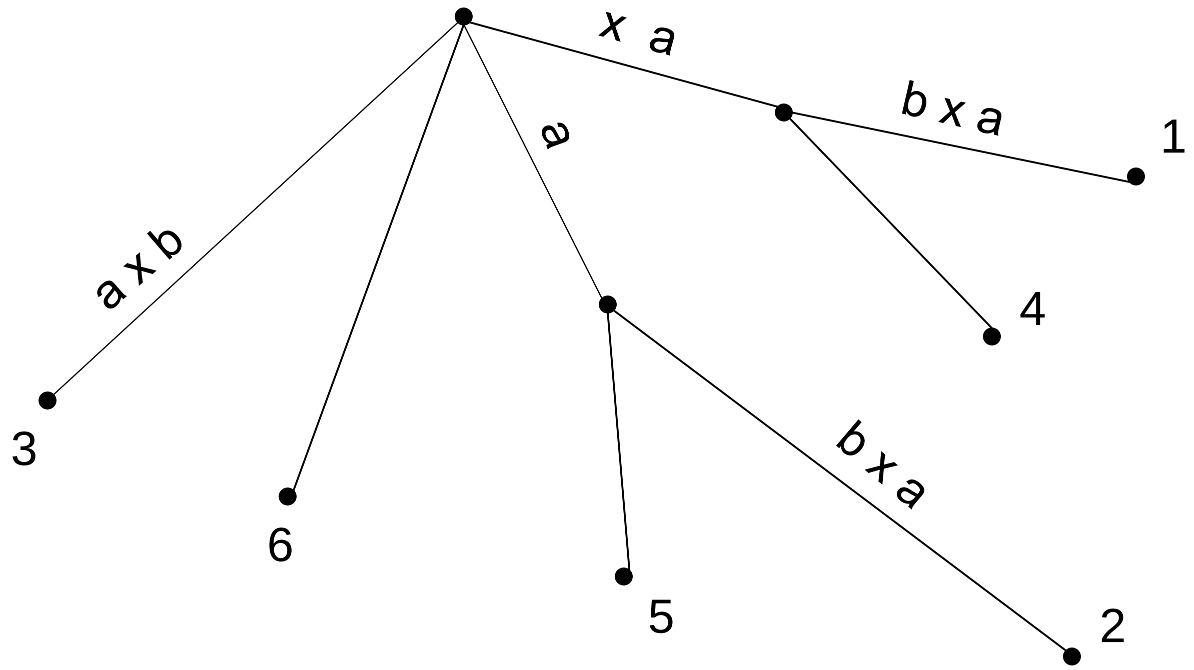
Ukkonen's algorithm: implicit suffix trees

t = x a b x a $
1 2 3 4 5 6



Ukkonen's algorithm: implicit suffix trees

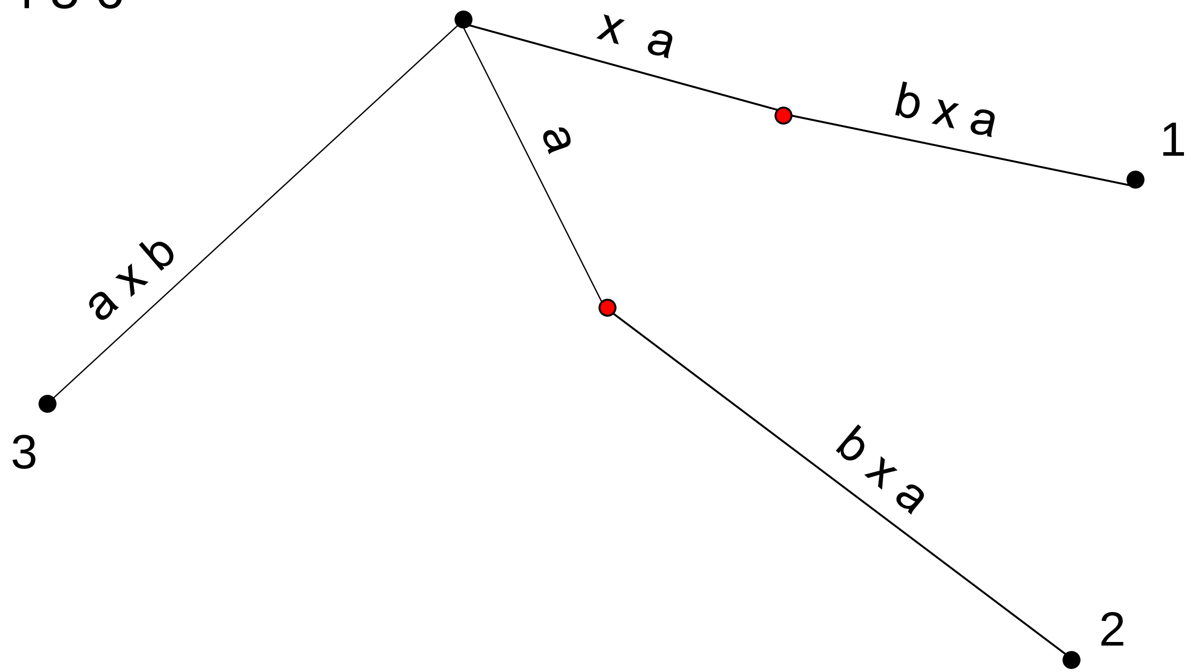
(1) deleting $ from the edge labels



Ukkonen's algorithm: implicit suffix trees

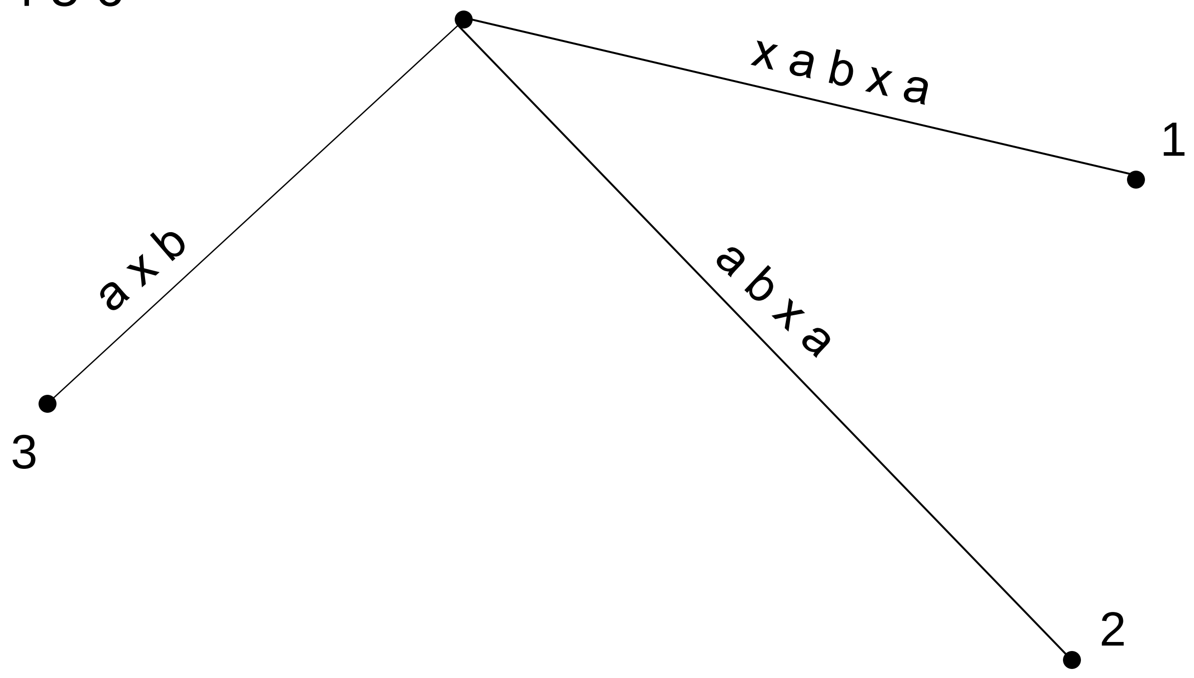
(2) deleting edges that have no label

t = x a b x a $
    1 2 3 4 5 6



(3) deleting unary nodes

t = x a b x a $
1 2 3 4 5 6



Ukkonen's algorithm

Let $t = t_1 t_2 t_3 \ldots t_n$.

Ukk is an online algorithm: The suffix tree $ST(t)$ is constructed step by step by constructing a sequence of implicit suffix trees for the prefixes of t:

$$ST(\varepsilon),\ ST(t_1),\ ST(t_1 t_2),\ \ldots,\ ST(t_1 t_2 \ldots t_n)$$

$ST(\varepsilon)$ is the empty implicit suffix tree, consisting of the root only.

Ukkonen's algorithm

This is an *online* approach in the sense that in each step, the implicit suffix tree for a prefix of t is created without knowledge of the rest of the input string t.

Since the algorithm reads the input string character by character from left to right, it works *incrementally*.

Ukkonen's algorithm

Incremental construction of an implicit suffix tree:

Induction basis: $ST(\varepsilon)$ consists of the root only.

Induction step: $ST(t_1 \ldots. t_i)$ is extended to $ST(t_1 \ldots t_i t_{i+1})$ for all $i < n$.

Let T_i denote the implicit suffix tree for $t[1...i]$.

- First, we construct T_1: This tree has a single edge labeled with character t_1.
- In phase i+1, we construct tree T_{i+1} from T_i.
- We iterate for $i = 1,\ldots,n-1$.

Ukkonen's algorithm

Pseudo code for Ukk:

Construct tree T_1.

for i = 1 **to** n–1 **do**

begin {phase i+1}

    for j = 1 **to** i +1 **do**

        begin {extension j}

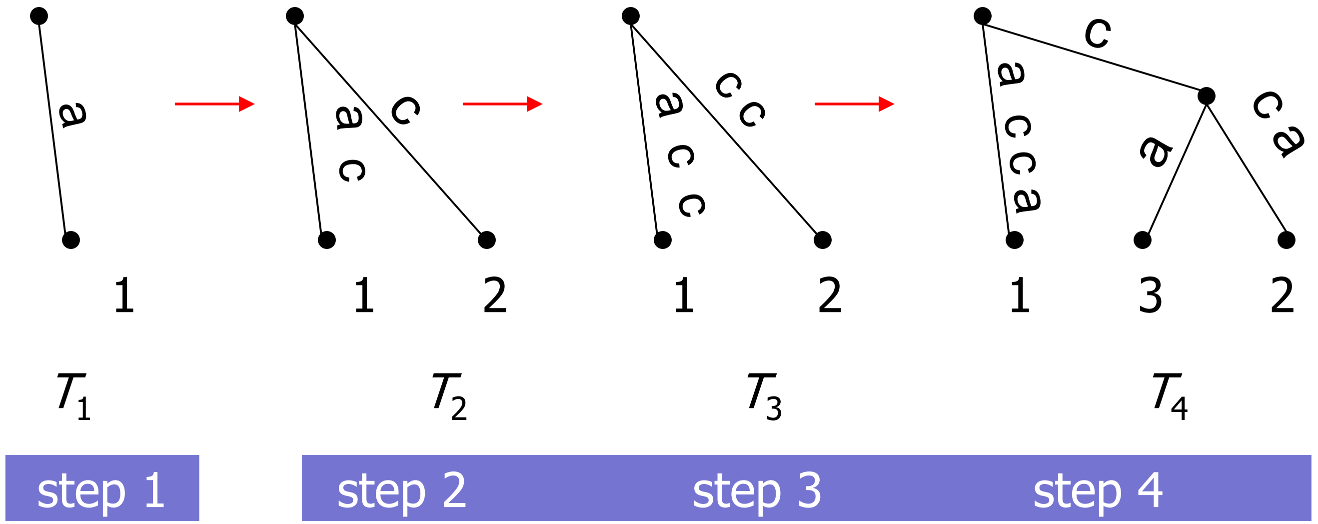
            In the current tree find the end of the path from the root labeled $t[j \dots i]$. If necessary, extend that path by adding character $t[i+1]$, thus ensuring that string $t[j \dots i+1]$ is in the tree.

        end;

end;

Ukkonen's algorithm

t = a c c a $



Ukkonen's algorithm

- In phase i+1, string t[1...i+1] is first inserted into the tree, followed by strings t[2...i+1], t[3...i+1],... (in extensions 1,2,3,...)
- In extension j of phase i+1, the end of the path from the root labeled with substring t[j...i] is determined. Then this substring is extended by adding the character t[i+1] to its end (unless t[i+1] already appears there).
- Extension i+1 of phase i+1 inserts the single character string t[i+1] into the tree (unless it is already there).

Ukk: Suffix extension rules

Extension j (in phase i+1) results from applying one of the following rules:

Rule 1: If the path $t[j...i]$ ends at a leaf, character $t[i+1]$ is added to the end of the label on that leaf edge.

Rule 2: If no path from the end of string $t[j...i]$ starts with character $t\ [i+1]$, then a new leaf edge labeled with character $t[i+1]$ is created. A new internal node will also be created there if $t[j...i]$ ends inside an edge. (This is the only extension that increases the number of leaves! The new leaf represents the suffix starting at position j.)

Rule 3: If some path from the end of string $t[j\ ...i]$ starts with character $t[i+1]$, then string $t[j…i\ +1]$ is already in the current tree, so we do nothing.

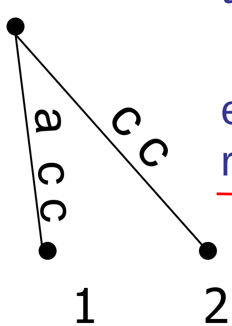
Ukkonen's algorithm

t = a c c a $

t [1...3] = acc

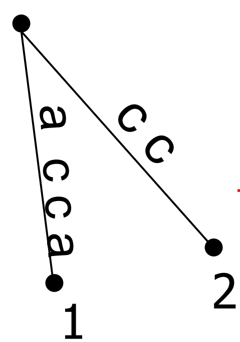
t [1...4] = acca

T_3



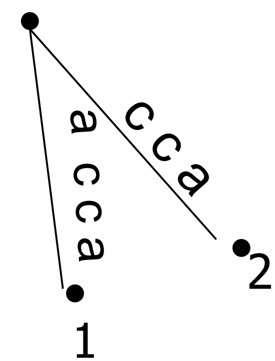
t[1..4] = acca

extend suffix 1
rule 1



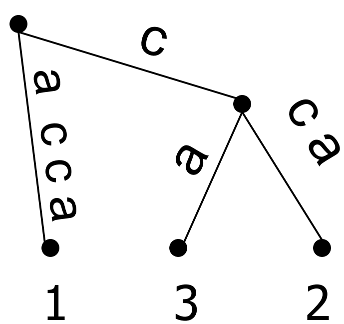
t[2..4] = cca

extend suffix 2
rule 1



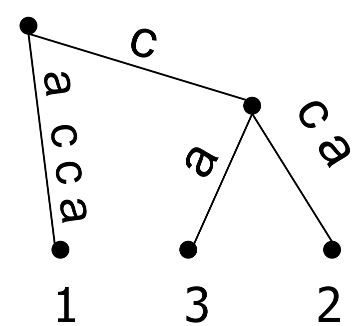
t[3..4] = ca

extend suffix 3
rule 2



[4..4] = a

a is already in
the tree
rule 3



T_4

Ukkonen's algorithm

During phase i+1 (when T_{i+1} is constructed from T_i) the following holds:

(1) If rule 3 applies in extension j, then the path labeled $t\,[j...i]$ in T_i must continue with character $t\,[i+1]$. So, any path labeled $t\,[j' ... i]$ for $j' \geq j$ also continues with character $t\,[i+1]$.

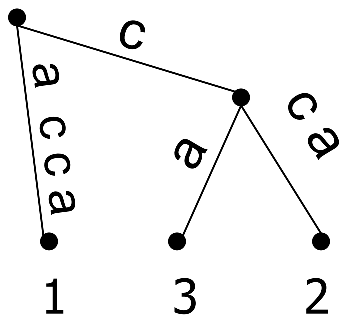
Therefore, rule 3 again applies in extensions $j' = j+1,..., i+1$.

Once rule 3 applies in an extension of phase i+1, this phase may be ended.

Ukkonen's algorithm

(2) If a leaf is created in T_i, then it will remain a leaf in all successive trees $T_{i'}$ for $i' > i$ (once a leaf, always a leaf!).

Reason: A leaf edge is never extended.



T_4

Ukkonen's algorithm

Implication:

- Leaf 1 is created in phase 1. In each phase i+1 there is an initial sequence of successive extensions (starting with extension 1) where rule 1 or rule 2 applies.

- Let j_i denote the last extension in this sequence of phase i.
  Then: $j_i \leq j_{i+1}$

Phase i, extension j	Phase i+1, extension j
Rule 1	Rule 1
Rule 2	Rule 1

Ukkonen's algorithm

Extensions according to rule 1 may be performed implicitly!

Ukkonen's algorithm

Improving the algorithm:

In phase i+1, rule 1 applies in all extensions j for $j \in [1, j_i]$.
Only constant time is required to do those extensions implicitly.

If $j \in [j_i +1, i+1]$, then find the end of the path labeled $t[j \ldots i]$ and extend it by character $t[i+1]$ according to rules 2 or 3.
If rule 3 applies, set $j_{i+1} = j$ -1 and terminate phase i+1.

Ukkonen's algorithm

Example:

phase 1:	compute extensions	$[1 \ldots j_1]$
phase 2:	compute extensions	$(j_1 \ldots j_2]$
phase 3:	compute extensions	$(j_2 \ldots j_3]$
....		
phase i-1:	compute extensions	$(j_{i-2} \ldots j_{i-1}]$
phase i:	compute extensions	$(j_{i-1} \ldots j_i]$

Ukkonen's algorithm

- As long as explicit extensions are performed, keep track of the index j^* of the current explicit extension.

- During the execution of the algorithm, j^* increases.

- As there are only n phases (where $n = |t|$) and j^* is bounded by n, the algorithm performs only n explicit extensions.

Ukkonen's algorithm

Extended pseudo code for Ukk:

Construct tree T_1; $j_1 = 1$;
for $i = 1$ **to** $n - 1$ **do**
begin {phase i+1}
  Do all implicit extensions.
  for $j = j_i + 1$ **to** $i + 1$ **do**
    begin {extension j}
      In the current tree find the end of the path from the root labeled $t[j \dots i]$. If necessary, extend that path by adding character $t[i+1]$, thus ensuring that string $t[j \dots i+1]$ is in the tree.
      $j_{i+1} := j$;
      if rule 3 was applied **then** $j_{i+1} := j - 1$ and phase i+1 ends;
    end;
end;

Ukkonen's algorithm

t = pucupcupu

i:	0	1	2	3	4	5	6	7	8	9
	ε	*p	pu	puc	pucu	pucup	pucupc	pucupcu	pucupcup	pucupcupu
			*u	uc	ucu	ucup	ucupc	ucupcu	ucupcup	ucupcupu
				*c	cu	cup	cupc	cupcu	cupcup	cupcupu
					u	*up	upc	upcu	upcup	upcupu
						p	*pc	pcu	pcup	pcupu
							c	cu	cup	*cupu
								u	up	*upu
									p	pu
										u

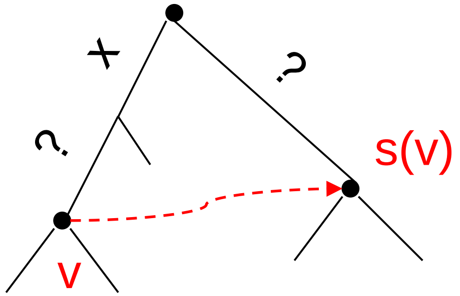
- Suffixes that cause an extension according to rule 2 are marked with *.
- Underlined suffixes indicate the last extension where rules 1 or 2 apply.
- Suffixes that end a phase (the first time rule 3 applies) are colored blue.

Ukkonen's algorithm

The running time may be improved using suffix links.

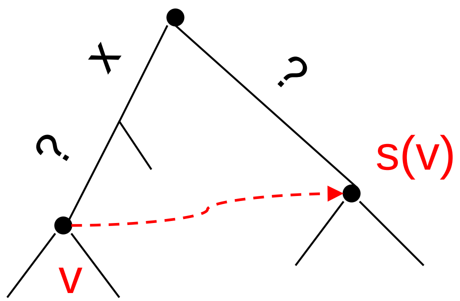
Definition: Let *x*? be an arbitrary string where *x* is a single character and ? some (possibly empty) substring.
For an internal node *v* with edge labels *x*? the following holds:

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



Ukkonen's algorithm

Idea:
By following the suffix links, we do not have to start each search for a split 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.



Ukkonen's algorithm

- Using suffix links, extension rules 2 and 3 can be applied more efficiently.
- Any explicit extension takes amortized O(1) time (not shown here).
- Since there are only n explicit extensions, the total running time of Ukkonen's algorithm is $O(n)$ (where $n = |t|$).

Ukkonen's algorithm

The true suffix tree:

The final implicit suffix tree $T_{\underline{n}}$ can be converted to a true suffix tree in O(n) time.

(1) Add a terminal symbol \$ to the end of t.

(2) Let Ukkonen's algorithm continue with this character.

The resulting tree is the true suffix tree where no suffix is prefix of another suffix. Thus each suffix ends at a leaf.