Ukkonens Alg.

Idea: Build suffix tree for $S[1..i+1]$ from suffix tree for $S[1..i]$.

(Note: this makes it online in the sense that chars of string s can be given incrementally.)

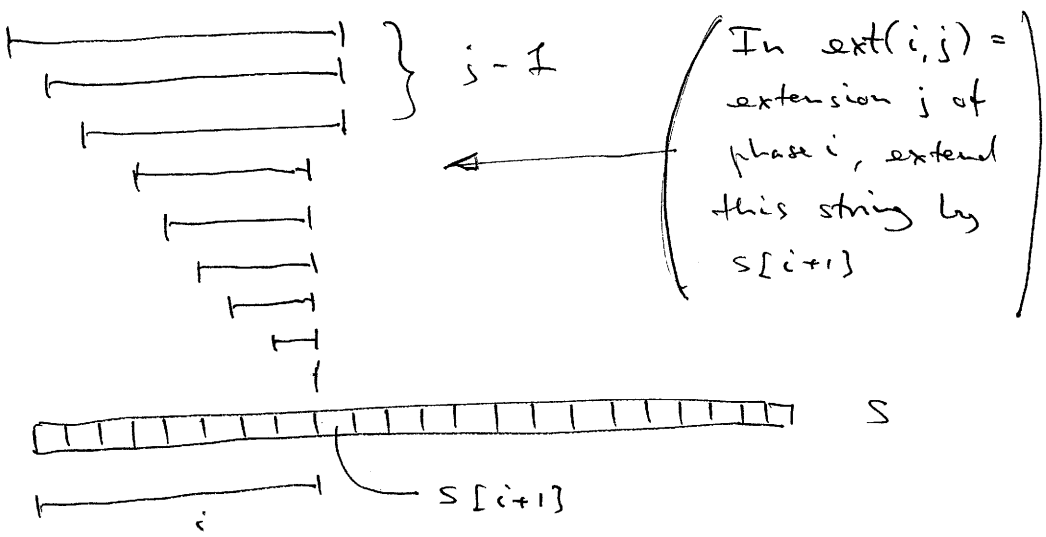
In more detail:

$|S|$ phases of the above type

Start by suffix tree for $S[1..1]$ (trivial to make)

In each phase, extend each suffix in tree by char $S[i+1]$

Thus, invariant of alg. is that ~~in~~ before j'th extension of ith phase, we have a trie containing the following strings:



The alg. works on implicit suffix trees.

If we add char $\$ \notin \Sigma$ as last char of S, this does not matter (no diff. to standard suffix trees in this case).

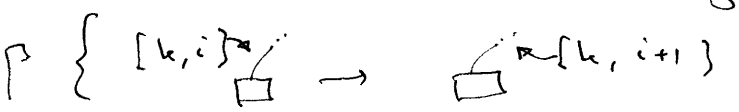
Extension Cases:

$Ext(i,j)$ = extension of string $S[j..i]$ by char $S[i+1]$ to $S[j..i+1]$. (In the tree maintained.)

For $\beta = S[j..i]$, we have the following cases [in tree at start of $ext(i,j)$]:

Case I: β ends at leaf.

Action: change label on edge to leaf:
(no structural change)



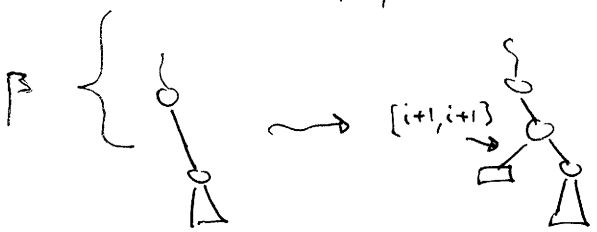
Case II: β ends at internal node u or inside edge AND $\beta \cdot S[i+1]$ is not path in tree

Subcase a: At internal node.

Action: add leaf as child of u:

Subcase b: Inside edge.

Action: Add new internal node ("break edge"), add new leaf.



Case III: β end at internal node or inside edge AND $\beta \cdot S[i+1]$ is path in tree

Action: do nothing (ok, as implicit suffix trees).

Observations:

1) Once created, internal nodes never get deleted, and their path labels do not change.

2) Only case II b creates new internal nodes.

3) Once created, leaf nodes never get deleted, stay leaf nodes, and only can have end point of label increased. (Start points stay fixed).

4) Actually, as all leaves represents strings stored (and all strings stored gets extended

in each phase), we see that after phase i, all leaves have $i+1$ as endpoint of label.

Claim 1: If case III happens at $ext(i,j)$, then $ext(i,k)$ for $k > j$ [ie., rest of phase] is also case III instances.

Proof: Later.

Claim 2: If $ext(i,j)$ is of type I or II, then $ext(k,j)$ is of type I for $k > i$.

Proof: Later

Note that by observation 4), the action of all Type I extensions in a phase can be covered by having the edge label [startindex, endindex] of leaves be changed into [startindex, $*$], where $*$ is a special value [-1 or ∞ [$\sim$ MAX_INT]], which is always interpreted as the phase number i.

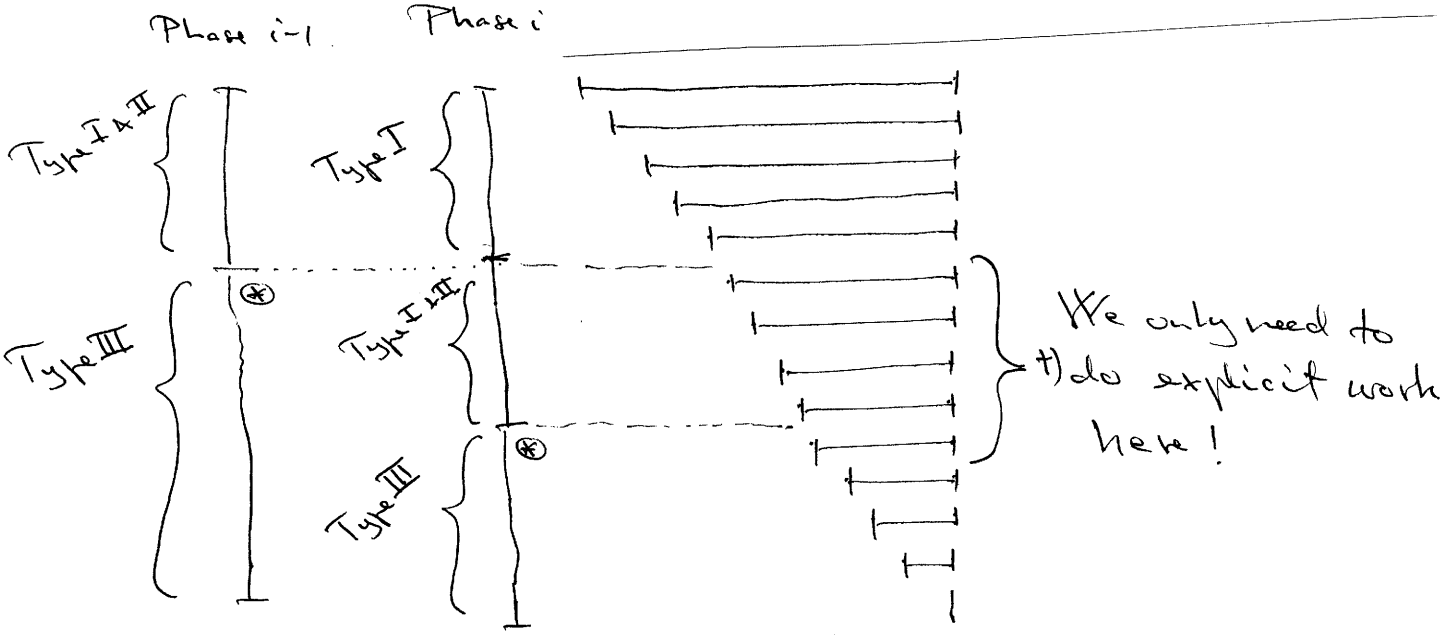
Note that type III extensions require (by def. of action) no work.

By Claim 1 and 2 we have that:

- any phase consists of Type I & II extensions followed by some III extensions
- the j's of the first part (before first Type III) will later only generate Type I extensions

---

So in phase $i-1$ and i we have:



⊛ = first type III of phase.

Note: ⊛ cannot move backwards (only forwards) from a phase to next.

Still need to navigate smarter than simple scan to end of strings (in tree) extended in t).

Def: For an internal node v (≠ root), let its path label be $x \cdot \alpha$ (x char, α string, $|\alpha| \geq 0$). If another internal node has path label α, we define $sl(v) = u$ ["sl" for "suffix-link"].

Claim 3: For any internal node v created at some extension in alg., $sl(v)$ will exist already or be created by next extension.

Proof: Later

Note: by OBS 1, neither v nor $sl(v)$ will change later.

Since new internal nodes are only created by Type II (b) extensions, it follows that if we after creation of internal node v in ext(i,j) access the node $sl(v)$ [known to exist by Claim 3] ~~if~~ ext$(i,j+1)$ during then we can during the work in t) maintain $sl(v)$ as explicit links from node v to node $sl(v)$, except for the node (if any) created during the last

Hint →

extension.

[Or Type II a (at root) if no Type III takes place before j reaches i+1]

As t) ends by a Type III extension (no new nodes), all internal nodes (≠ root) has a link s(v) set by end of phase. (end of t)).

(I.e. pointer)

We now give the algorithm for the work t) of a phase.

By last extension point in tree, we mean for a just finished extension ext(i, j) (in t)) the first internal node above or at position in the tree of the end of this last ("inserted" string s[j..i+1]. We keep pointer to this internal node.

(Can be at leaf, at int. node or inside edge, depending on ext. type)

As we start at j given by ⊛ of last phase, the last extention point at beginning of a phase i is at most one node above ext. point of first inserted string s[j..i+1]. (check ext. cases)

⊛

So first extension can be done in O(1) time.

ext(i, j+1)

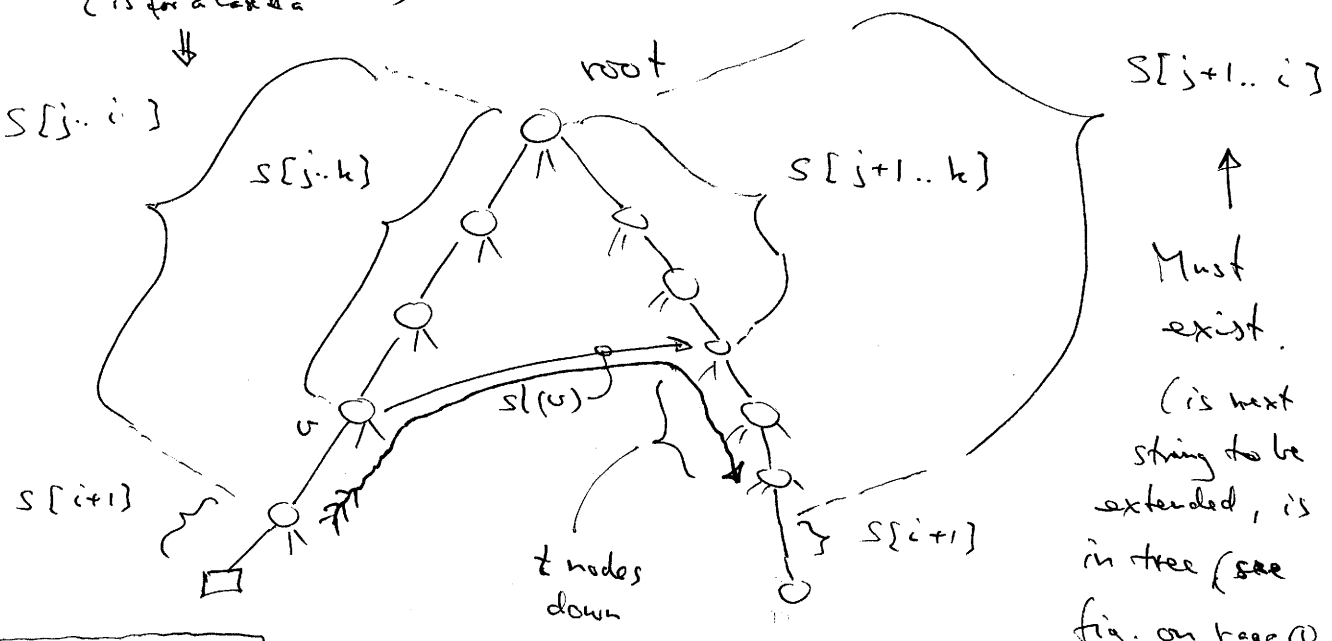
At each subsequent ext. (in t) part) of phase, we do:

- go up to first internal node v (strictly) above last ext. point. (I.e., go to parent of insertion point.)
- follow sl(v) pointer (must exist, as invariant that only last insertion point has (possibly) sl-pointer not set).
- from sl(v), seek downwards for next ext. point for string s[j+1 .. i+1] of the next extention ext(i, j+1).

- perform extension action [if Type II].
- on the way downwards, sl(last ext. point) was either met or created [by claim 3]. So set sl-pointer for last ext. point.

See figure:

{This particular fig. is for a Case II a}



$k \leq i$ always

So search down takes $O(1)$ ft. node [only first char of edge label needs to be checked].

Claim 4 (where sl(v) pointer is traversed in algorithm)

For all int nodes v: $\text{nodedepth}(v) \leq \text{node depth}(sl(v)) + 1$

Def.: [Node depth(u) = # nodes above u].

Proof: later.

Complexity: By Claim 4 (and first step on page ⑧), we go ≤ 2 $\underline{\text{up}}$ in node depth in each extension. (More precisely, the ext. point).

We go $t \geq 0$ down (right part of figure above)

i) Time spent on extension is $O(1 + t)$.

1 increasing j (⇐ Those where SI(U) links are followed, i.e. all but first in phase)

ii) As j never decreases, we have m ($= |S|$) such ext. at most.

The decrease in node depth of ext. point during these is $\leq 2 \cdot m$. (≤ 2 for each)

The increase is $\sum_k t_k$ (sum of all t's)

Node depth $\leq m$ always.

$$\Rightarrow \quad \sum_k t_k - 2m \leq m$$

iii) $$\Leftrightarrow \quad \sum_k t_k \leq 3m.$$

iv) Each phase costs $O(1)$ + time spent on extensions increasing j.

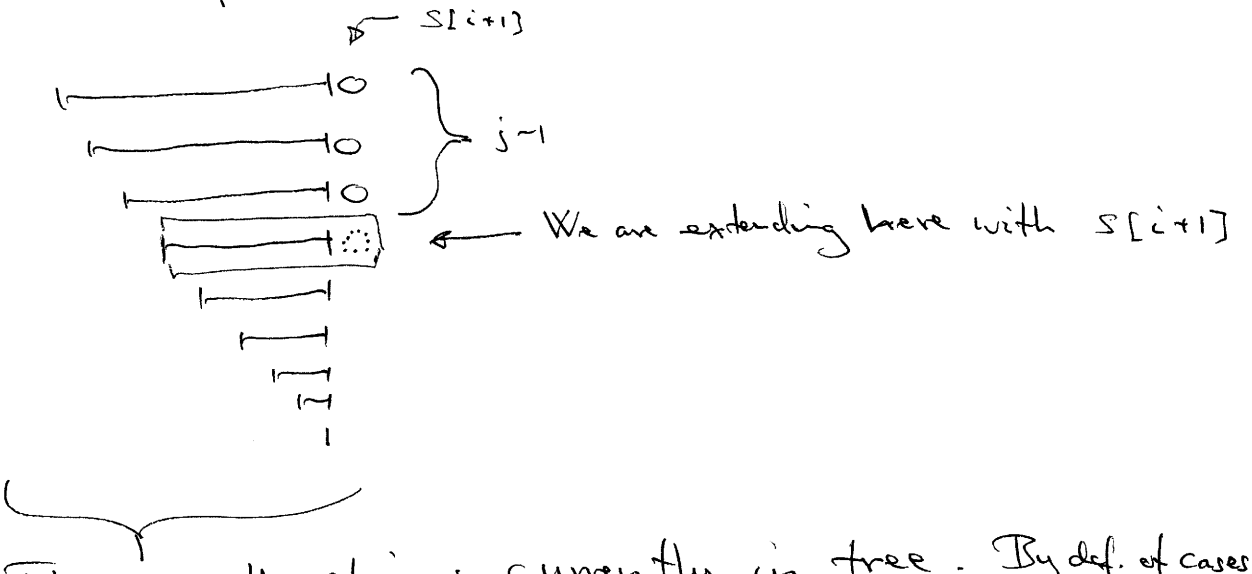
v) There are m phases.

In total: $O(m) + O(m) + O(\sum_k t_k)$

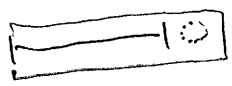
$$= O(m)$$

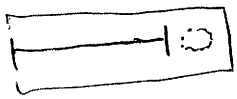
Proof of Claim 1

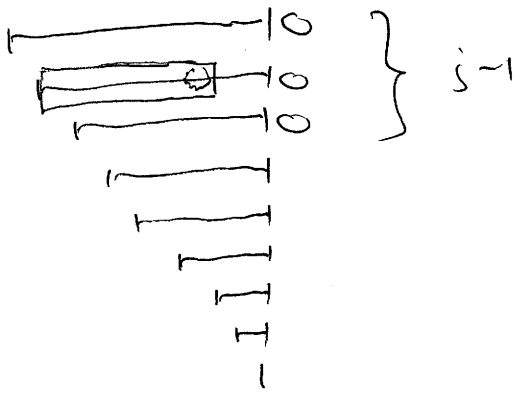
The figure for ext(i,j) [cf. page ①] again:

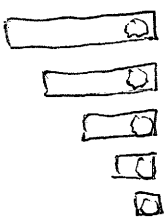


These are the strings currently in tree. By def. of cases:

> Ext(i,j) of Type III ⟺  is a prefix of a string in tree currently.

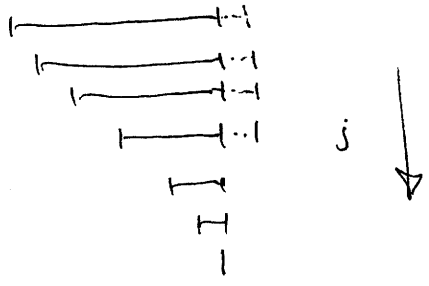
By length of  , it can only be a prefix of the j-1 first (longest strings). Eg.:



Clearly, all the strings  are prefixes of strings currently in tree. As we are always extending strings, this will remain true in the future, in particular rest of this phase. Hence, rest of extensions in phase will be Type III (by def. of ext. cases).

Proof of Claim 2 :

After ext(i,j), leaf extended (type I) or created (type III) has pathlabel s[j.. i+1].
In next phase (k = i+1), precisely ext(i+1, j) will reach this leaf (and will extend its pathlabel), as the rest (ext(i+1, t) for t ≠ j) cannot due to lengths of strings extended [consider the evolving set of strings stored during the phase in the tree :]



So ext(i+1, j) will be of type I.

Repeat argument for ext(k, j). (k > i+1) □

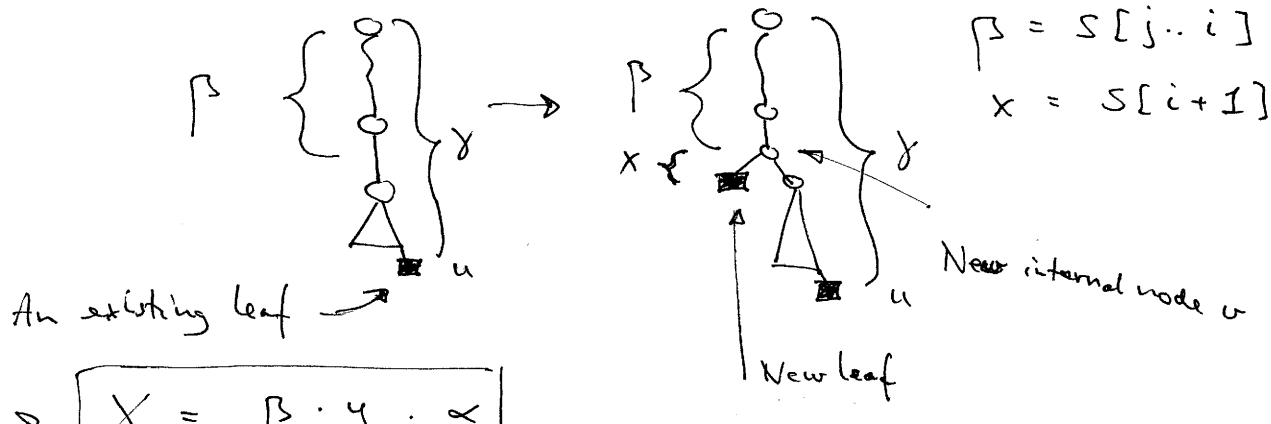
BTW : [Also note that once created, a leaf represents same suffix j (and may be annotated by j at creation)].

Proof of Claim 3

Note that at the last extension ext(i, i+1) of a phase i, we are extending the empty string ε (represented by the root) to the string s[i+1.. i+1]. This is either a Case II a) or Case III, hence does not create a new internal node.

So we need only consider ext(i,j) for j ≤ i. In particular, the next extension is still in same phase i.

By obs. 2), we need only consider a Case II b) extension:



$\boxed{\gamma = \beta \cdot y \cdot \alpha}$ is pathlabel of u (γ is a currently stored string), where $y \in \Sigma$, $x \neq y$ (since we are in Case II b)), and α is some string ($|\alpha| \geq 0$).

By the length of γ ($|\gamma| \geq |\beta| + 1$), we (like in proof of Claim 1) from the figure on page ① see that γ must be one of the first (longest) $j-1$ strings stored, ⊛ say string k ($k \leq j-1$).

Let $\beta = z \cdot \beta'$ ($z \in \Sigma$, β' string, $|\beta'| \geq 0$) – recall $j \leq i$, so $|\beta| \geq 1$, so this is OK. to assume.

By ⊛ and figure page ①, we see that one of the currently stored strings [namely string $k+1$] has $\beta' \cdot y$ as a prefix. After next extension, $ext(i, j+1)$, the string $S[j+1..i+1] = \beta' \cdot x$ is also in the tree. As $x \neq y$, there must be then

an internal node in tree with path label β'.

□

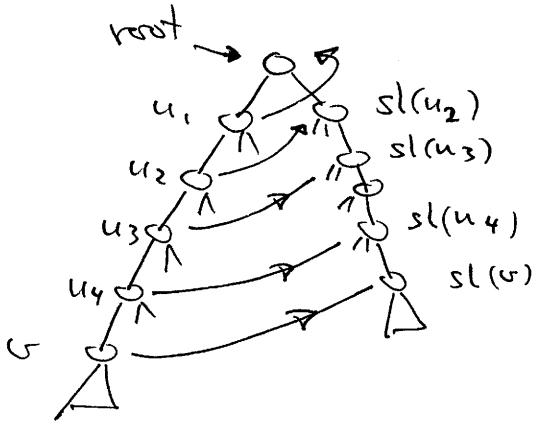
Proof of Claim 4

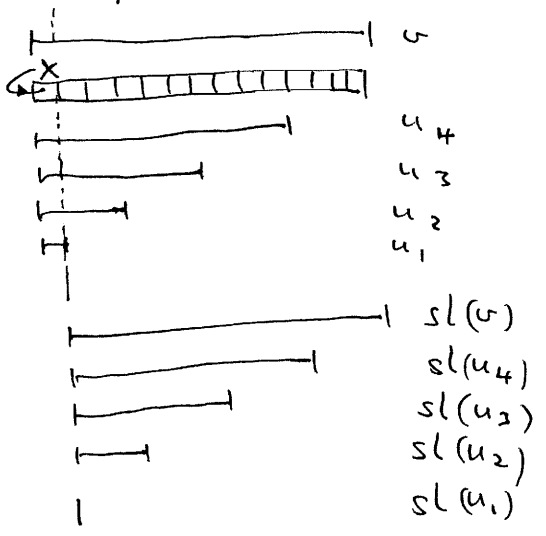
When $v \to sl(v)$ pointer is traversed, all ancestors of v ($\neq$ root) have their sl-link set (the single internal node which possibly has not an sl-link set (the int. node created by last ext., if any) is the child of v, by first point of page (8)).

Such an ancestor ($\neq$ root) u has path label $x \cdot \beta_u$, and $sl(u)$ has path label β_u (by def. of suffix links).

All ancestors have different depths, hence different $|x \cdot \beta_u|$, hence different $|\beta_u|$ ($= |x \cdot \beta_u| - 1$)
So all the $sl(u)$ nodes are different.

The path label $x \cdot \beta_u$ of any ancestor is a prefix of the path label $x \cdot \beta_v$ of v.





So the path label of any $sl(u)$ is a prefix of $sl(v)$ [see figure above]. I.e. they are all on the path to $sl(v)$ in tree (and are different, as seen above).

So # ancestors of $v \neq$ root

$\leq$ # ancestors of $sl(v)$.

□