Linear Time Suffix Tree Construction (Weiner 1973)

Define: \( S_i = A_i A_{i+1} \ldots A_n \$ \)

\( T(\geq i) = \text{suffix tree of } S_i. \)

Examples: \( T(\geq n+1) = \)

\[ \text{Tree} \]

\[ S \]

\( T(\geq 1) = \text{The tree we seek.} \)

Weiner's Idea:

- Start with \( T(\geq n+1) \)
- Given \( T(\geq i+1) \), construct \( T(\geq i) \) by adding \( S_i \) to the tree.
EXAMPLE: $ABABBA$

$T(\geq 7)$
\[ \Lambda \xrightarrow{A} \$

$T(\geq 6)$
\[ \Lambda \xrightarrow{B} A$

$T(\geq 5)$
\[ \Lambda \xrightarrow{BA} A$

$T(\geq 4)$
\[ BA \xrightarrow{A} \$

$T(\geq 3)$
\[ BA \xrightarrow{A} \$

$T(\geq 2)$
\[ BA \xrightarrow{A} \$

$T(\geq 1)$
\[ BA \xrightarrow{A} \$

Key Observation:
height of tree grows by at most 1 in each insertion.
Because of Observation: We want a way of inserting a new leaf not by going from root down, rather by going from leaves up.

**Weiner's Algorithm:**

Construct $T(\geq n+1)$

For $i=n$ downto 1 do

(i) Insert $S_i$ to $T(\geq i+1)$ and create $T(\geq i)$.

end

end Algorithm

(i) Let head be the longest prefix of $S_i$ that is in tree $T(\geq i+1)$ (at least implicitly).

1. Find head.
2. If head not a node then break an edge and make it one.
3. Add $S_{i+1}|_{head-1}$ as additional son of head.
Only remaining problem: Find head quickly.

Cases:

1. head = $A$. Means new letter introduced.
   Add node and edge: $A \rightarrow S_i$

2. head ≠ $A$.
   Go from $S_{i+1}$ to head.
   How?

$S_i = S_i : S_{i+1}$

head = $S_i : x$ where $x =$ longest prefix of $S_{i+1}$
   s.t. $A : x$ is implicitly in $T(\geq i+1)$

Let $x' =$ longest prefix of $S_{i+1}$ s.t.
   $S_i : x'$ is a node in $T(\geq i+1)$.
(We will say is a node and mean ends in a node.)
Situation in $T(\geq i+1)$

$S_i = A; S_{i+1}$

It is crucial to our complexity that $A; x'$ is closer to root (has smaller depth) than $x'$.

**Lemma:** If $au$ ends in a node in $T(\geq i)$ then $u$ ends in a node in $T(\geq i+1)$

In particular, $\text{depth}(x') \geq \text{depth}(A; x')$.

**Proof:** If $au$ ends in a leaf then clearly $u$ ends in a leaf.

Otherwise, $au$ ends in an internal node, i.e.

$\exists b, c \text{ s.t. } aub, auc \in T(\geq i) \Rightarrow$

$\exists b, c \text{ s.t. } ub, uc \in T(\geq i+1) \Rightarrow$

$\exists$ node in $T(\geq i+1)$ in which $u$ ends.
All we need now:

For every node \( u \) and every \( \sigma \in \Sigma \):

- flag indicating if \( \sigma u \in T(\geq \sigma i + 1) \) implicitly,
- pointer to node where \( \sigma u \) ends if it appears explicitly.

The Algorithm:

1. Run up path of \( S_{i+1} \) until \( S_i \) flag appears.
2. Keep running up path until \( S_i \) pointer appears, keeping track of difference between flag & pointer.
3. Jump to \( S_i \).
4. Break edge \( i \) and add \( S_i \) appropriately.
Updating Pointers and Flags (head ≠ A).

1. Update flags that change in \( T(≥i+1) \) because head and Si nodes are introduced.

2. Update pointers that change in \( T(≥i+1) \) because of new nodes head and Si.

3. Set flags and pointers at new nodes head and Si.
1. Set all $S_i$ flags from $S_i+1$ to $x$.

2. $S_{i+1}$ points to $S_i$; $S_{i+1} = S_i$.
   $x$ points to $S_i$; $x = \text{head}$.

3. $S_i$ is longest string in $T(\geq i)$ so no flags nor pointers.

If head was already a node in $T(\geq i+1)$ then we are done.

If head is new, every flag and pointer in the node below it sets appropriate flag in head.

No pointers.

(By lemma. If $a\text{head}$ is a node in $T(\geq i)$ then $a\text{head}$ is a node in $T(\geq i+1)$, which was not the case.)
Updating Pointers & Flags (head = A)

1. $S_{i+1}$ points to $S_i$.

2. All nodes on path $S_{i+1}$ set $S_i$ flag.

3. $S_i$ new symbol so no pointers nor flags at $S_i$. 
Time: Intuitively: At every step depth is incremented by at most 1. We may run up a lot, but down only 1. So, charge running up a path to "building" it down. Total is linear.

Formally:

\[ \text{Total time} \leq \left| \left( \text{depth (} S_{n+1} \text{)} - \text{depth (} S_n \text{)} \right) + \right. \]
\[ \left. \left( \text{depth (} S_n \text{)} - \text{depth (} S_{n-1} \text{)} \right) + \cdots + \left( \text{depth (} S_{i+1} \text{)} - \text{depth (} S_i \text{)} \right) + \left( \text{depth (} S_2 \text{)} - \text{depth (} S_1 \text{)} \right) \right| + O(1) \]
\[ = O(n + \max \text{depth (} T \text{)}) = O(n). \]