Linear Time Suffix Tree Construction
(Weiner 1973)

Define: $S_i = s_i s_{i+1} \cdots s_n \$$

$T(\geq i)$ = suffix tree of S_i.

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

$T(\geq 1)$ = The tree we seek.

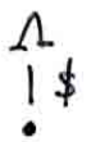
Weiner's Idea:

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

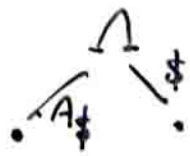
EXAMPLE: ABABBA\$

\$
A\$
BA\$
BBA\$
ABBA\$
BABBA\$
ABABBA\$

$T(\geq 7)$

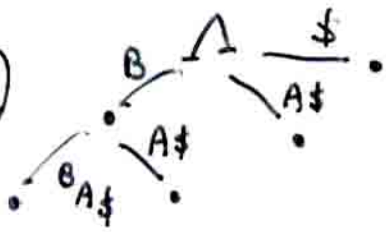


$T(\geq 6)$

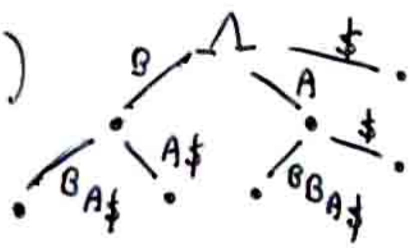


$T(\geq 5)$

$T(\geq 4)$

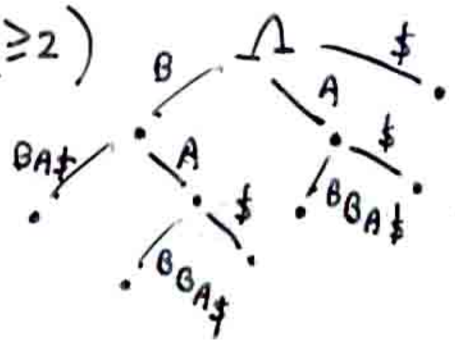


$T(\geq 3)$

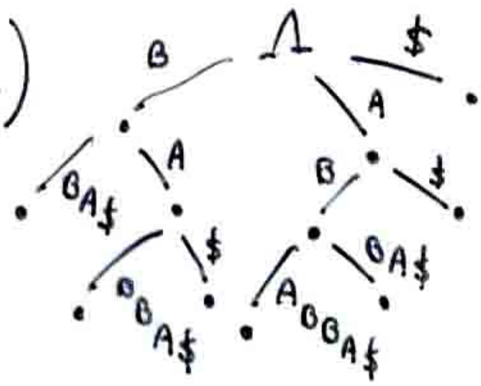


Key Observation:
height of tree grows by at most 1 in each insertion.

$T(\geq 2)$



$T(\geq 1)$



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

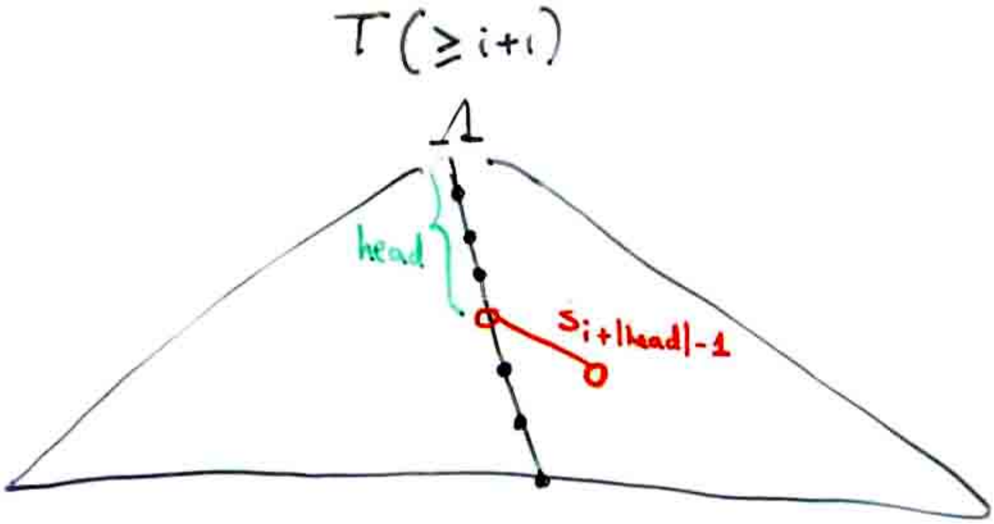
(i) Insert S_i to $T(\geq i+1)$ and create $T(\geq i)$.

end

end Algorithm

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

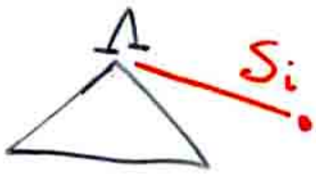
1. Find head.
2. If head not a node then break an edge and make it one.
3. Add $S_{i+|head|-1}$ as additional son of head.



Only remaining problem: Find head quickly.

Cases: 1. head = Λ. Means new letter introduced.

Add node and edge:



2. head ≠ Λ.

Go from S_{i+1} to head.

HOW?

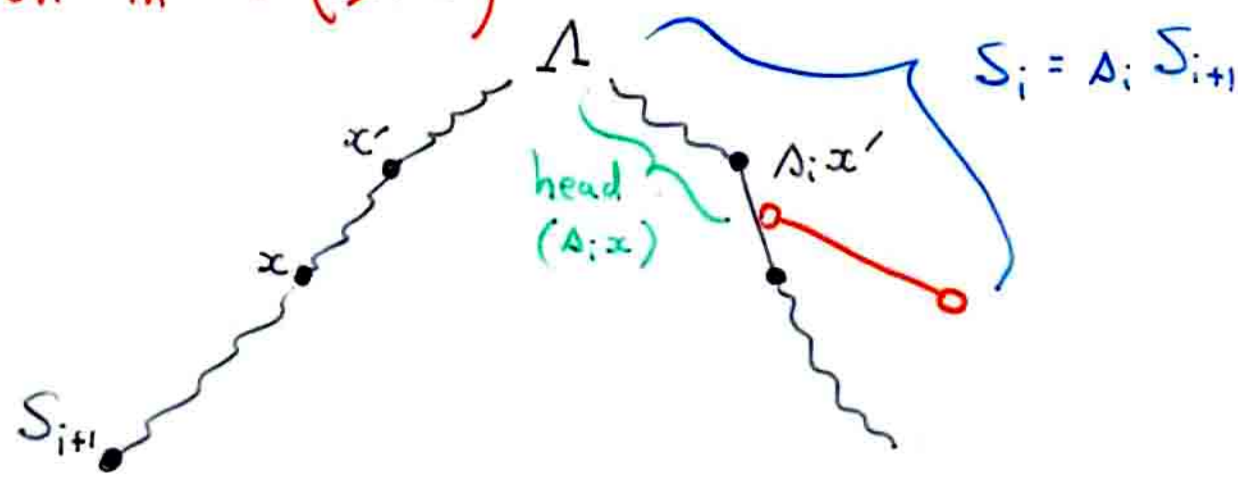
$S_i = s_i S_{i+1}$

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

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 $T(\geq i+1)$



It is crucial to our complexity that $s_i x'$ 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, $\text{depth}(x') \geq \text{depth}(s_i x')$.

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

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

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

$\exists\, b,c$ 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 $\sigma u \in T(\geq i+1)$ implicitly.

pointer to node where σu ends if it appears explicitly.

The Algorithm:

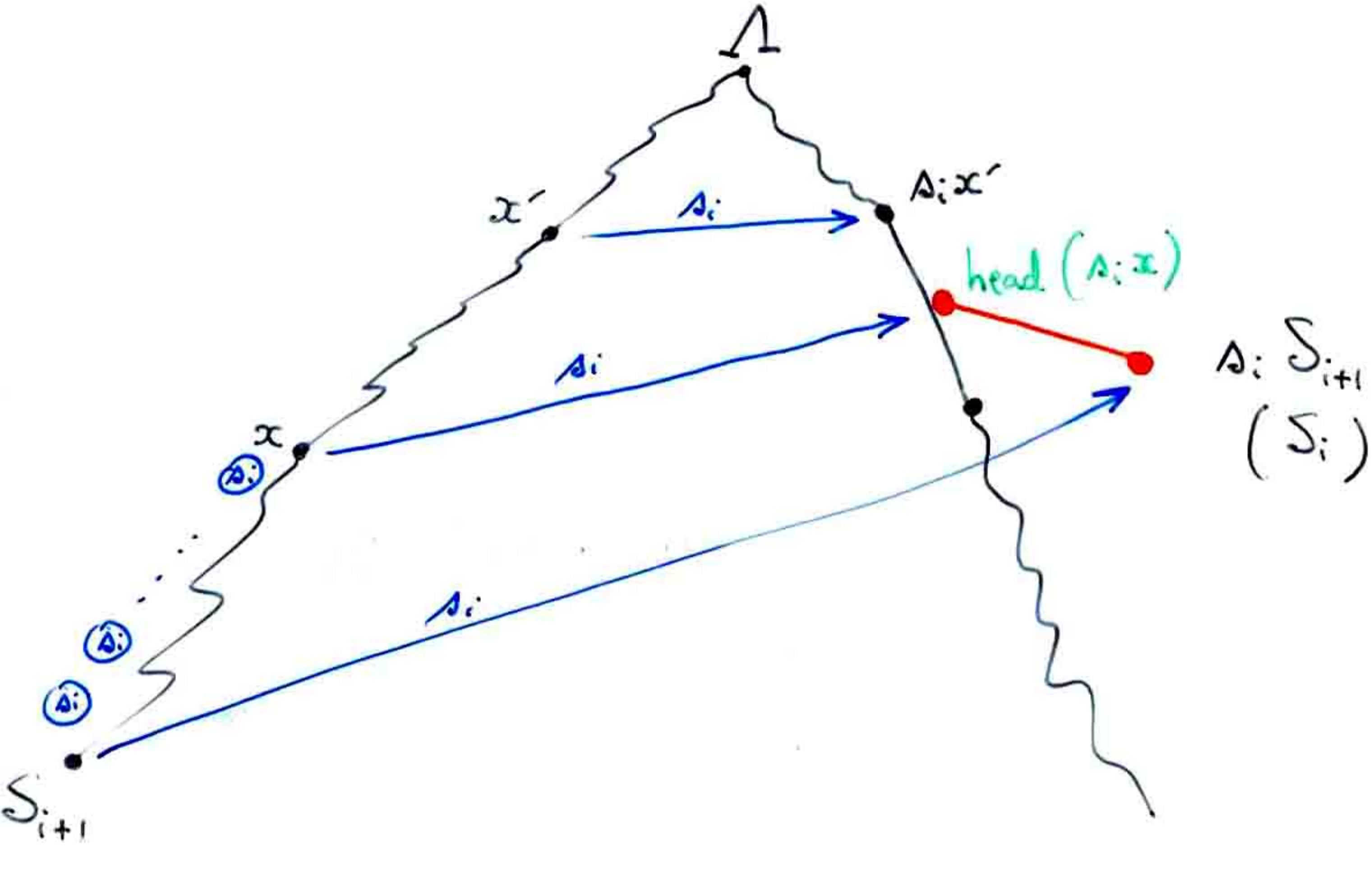
- Run up path of S_{i+1} until s_i flag appears
- Keep running up path until s_i pointer appears, keeping track of difference between flag & pointer.
- Jump to $s_i x'$.
- Break edge & add $s_i x$ appropriately.

Updating Pointers and Flags (head $\neq \Lambda$)

1. Update flags that change in $T(\geq i+1)$ because head and S_i nodes are introduced.

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

3. Set flags and pointers at new nodes head and S_i.



1. Set all Δ_i flags from S_{i+1} to x.

2. S_{i+1} points to Δ_i $S_{i+1} = S_i$

   x points to Δ_i x = head.

3. S_i is longest string in $T(\geq i)$

   so no flags nor pointers.

   If head was already a node in $T(\geq i+1$

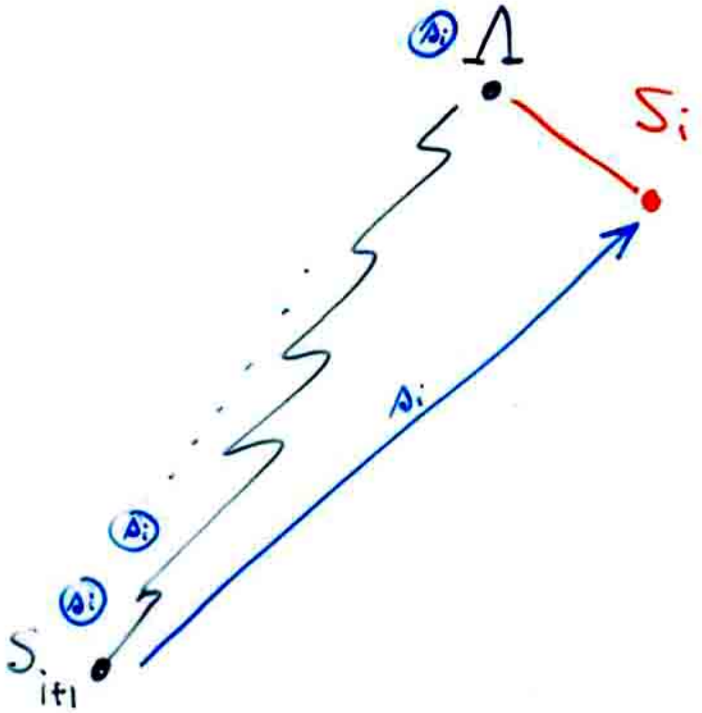
   then we are done.

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

   No pointers.

   (By lemma. If a head is a node in $T(\geq i)$ then head is a node in $T(\geq i+1)$, which was not the case.)

Updating Pointers & Flags (head = Λ)



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:

$$\text{Total time} \leq \Big| \big(\text{depth}(S_{n+1}) - \text{depth}(S_n)\big) + \big(\text{depth}(S_n) - \text{depth}(S_{n-1})\big) + \cdots + \big(\text{depth}(S_{i+1}) - \text{depth}(S_i)\big) + \cdots + \big(\text{depth}(S_2) - \text{depth}(S_1)\big) \Big| + O(n$$

$$= O(n + \text{maxdepth}(T)) = O(n).$$