McCreight's suffix tree construction algorithm
Motivation

**Recall:** the suffix tree is an extremely useful data structure with space usage and construction time in $O(n)$.

Now we see algorithms for constructing a suffix tree in time $O(n)$. 
A suffix tree of a sequence, $x$, is a *compressed trie* of all suffixes of the sequence $x$.

$x = \text{abaab}$

1: abaab$
2: baab$
3: aab$
4: ab$
5: b$
6: $
Iteratively, for $i=1,...,n+1$ build tries, $T_i$, where ... is a trie of sequences $x[1..n+1]$, $x[2..n+1]$, ..., $x[i..n+1]$.
McCreight's algorithm

Iteratively, for \( i=1,\ldots,n+1 \) build tries, \( T_i \), where ... is a trie of sequences \( x[1..n+1] \), \( x[2..n+1] \), ..., \( x[i..n+1] \).

For \( i=n+1 \), \( T_i \) is the suffix tree for \( x \).
McCreight's algorithm

Iteratively, for $i=1,...,n+1$ build tries, $T_i$, where ... is a trie of sequences $x[1..n+1], x[2..n+1], ..., x[i..n+1]$.

For $i=n+1$, $T_i$ is the suffix tree for $x$.

The essential trick is being clever in how we insert $x[i..n]$ into $T_i$ so we don't spend $O(n^2)$ all in all.
A common prefix of $x$ and $y$ is a string, $p$, that is a prefix of both:

$x$: $\text{a}$  $\text{b}$
$y$: $\text{a}$  $\text{b}$
$p$: $

The longest common prefix, $p=LCP(x,y)$, is a prefix such that: $x[|p|+1] \neq y[|p|+1]$

$x$: $\text{a}$  $\text{a}$  $\text{a}$
$y$: $\text{b}$  $\text{b}$  $\text{b}$
$p$: $

Terminology
For suffixes of $x$, $x[i..n]$, $x[j..n]$, their longest common prefix is their *lowest common ancestor* in the suffix tree:
Head and tail

- Let $\text{head}(i)$ denote the longest LCP of $x[i..n]$ and $x[j..n]$ for all $j < i$.
- Let $\text{tail}(i)$ be the string such that $x[i..n] = \text{head}(i)\text{tail}(i)$.
- Iteration $i$ in McCreight’s algorithm consist of:
  - finding (or inserting) the node for $\text{head}(i)$,
  - and appending $\text{tail}(i)$.
“The Trick”

The trick in McCreight's algorithm is a clever way of finding $\text{head}(i)$.
Lemma 5.2.1

Let head(i) = x[i..i+h]. Then x[i+1..i+h] is a prefix of head(i+1)
Proof

- Trivial for h=0 (head(i) empty), so assume h>0:
  - Let head(i) = ay: 
  - By def. exists j<i such that LCP(i,j)=ay
  - Thus suffix j+1 and i+1 share prefix y
  - Thus y is a prefix of LCP(i+1,j+1)
  - Thus y is a prefix of head(i+1)
Suffix link

- Define $s(u) = ""$ if $u=""$, and $v$ if $u=av$
- As a pointer from $x[i..k]$ to $x[i+1..k]$:

\[
\begin{align*}
&x[i..k] \\
&i
\end{align*}
\]

\[
\begin{align*}
&x[i+1..k] \\
&i+1
\end{align*}
\]

\[
s
\]

- (ex 5.2.3: if $u$ is a node, so is $s(u)$)
Corollary of Lemma 5.2.1

- $s(head(i))$ is a prefix of $head(i+1)$
- Thus: $s(head(i))$ is an ancestor of $head(i+1)$

- $s(head(i))$ can be used as a shortcut!
Slowscan and fastscan

- **Slowscan**: if we do not know if string $y$ is in $T_i$, we must search character by character.
- **Fastscan**: if we *do know* that $y$ is in $x$, we can jump directly from node to node.
  - At node $u$ at (path-)depth $d$, follow the edge with label starting with $y[d]$.
  - Continue until we reach the end of $y$.
    - On a node (if $y$ is in $T_i$).
    - Or on an edge (if $y$ is a prefix of a string in $T_i$).
Sketch of McCreight's algorithm

- Begin with the tree $T_1$:
- For $i=1,\ldots,n$, build tree $T_{i+1}$ satisfying:
  - $T_{i+1}$ is a compressed trie for $x[j..n]$, $j \leq i+1$
  - All non-terminal nodes (with the possible exception of head($i$)) have a suffix link $s(-)$
- Each iteration must:
  - Add node $i+1$
  - Potentially add head($i+1$)
  - Add tail($i+1$)
  - Add suffix link head($i$) $\rightarrow$ s(head($i$))
At the beginning of iteration $i$...

Let $\text{head}(i) = uv$ and $\text{parent}(\text{head}(i)) = u$ and $w = s(u)v = s(\text{head}(i))$

By the invariant, $s(\text{parent}(\text{head}(i)))$ and the suffix link exists; by the lemma, $w$ is an ancestor of $\text{head}(i+1)$
Steps in iteration $i$...

Move quickly to $w$, then search for head($i+1$) starting there
Observe: $w$ is in $T_i$

- head(i) is a prefix of $x[j..n]$ for some $j < i$
- Thus $w$ is a prefix of $x[j+1..n]$ for some $j < i$
  - i.e. $w$ is a prefix of some suffix $j \leq i$
  - i.e. $w$ is in $T_i$

- Consequently: we can search for $w$ from $s(u)$ using `fastscan`!
If \( w \) is a node

- Update \( s(\text{head}(i)) := w \)
- Then search for \( \text{head}(i+1) \) using \texttt{slowscan}
If $w$ is on an edge

- If $w$ is not a node, then all suffix $j<i$ with prefix $w$ agree on the next letter
- By definition of head($i$) there is $j<i$ such that suffix $x[i..n]$ and $x[j..n]$ differs after head($i$)
  - $x[i+1..n]$ must also disagree at that character
  - Thus head($i+1$) must be $w$

- Add node $w$, update head($i$):=$w$ and set head($i+1$):=$w$
McCreight's algorithm

Construct tree for \( x[1..n] \)

for \( i = 1 \) to \( n \) do
  if head\((i)\)="" then
    head\((i+1)\) = slowscan("",s(tail\((i)\)))
    add \( i+1 \) and head\((i+1) \) as node if necessary
    continue
  
  \( u = \text{parent}(\text{head}(i)) \); \( v = \text{label}(u,\text{head}(i)) \)
  if \( u\neq"" \) then \( w = \text{fastscan}(s(u),v) \)
  else \( w = \text{fastscan}("",v[2..|v|]) \)
  if \( w \) is an edge then
    add a node for \( w \)
    head\((i+1)\) = \( w \)
  else if \( w \) is a node then
    head\((i+1)\) = slowscan\((w,\text{tail}(i))\)
    add head\((i+1) \) as node if necessary
  \( s(\text{head}(i)) = w \)
add leaf \( i+1 \) and edge between head\((i+1) \) and \( i+1 \)
Example: $x = abaab$

Construct tree for $x[1..n]$

for $i = 1$ to $n$ do
  if $\text{head}(i) = ""$ then
    $\text{head}(i+1) = \text{slowscan}("","s(\text{tail}(i)))$
    add $i+1$ and $\text{head}(i+1)$ as node if necessary
  continue
  $u = \text{parent}(\text{head}(i))$ ; $v = \text{label}(u,\text{head}(i))$
  if $u \neq ""$ then $w = \text{fastscan}(s(u),v)$
  else $w = \text{fastscan}("",v[2..\mid v \mid])$
  if $w$ is an edge then
    add a node for $w$
    $\text{head}(i+1) = w$
  else if $w$ is a node then
    $\text{head}(i+1) = \text{slowscan}(w,\text{tail}(i))$
    add $\text{head}(i+1)$ as node if necessary
  $s(\text{head}(i)) = w$
add leaf $i+1$ and edge between $\text{head}(i+1)$ and $i+1$
Example: $x = abaab$

Construct tree for $x[1..n]$

```
for i = 1 to n do
    if head(i) = "" then
        head(i+1) = slowscan("", s(tail(i)))
        add i+1 and head(i+1) as node if necessary
    continue
    u = parent(head(i)); v = label(u, head(i))
    if u ≠ "" then
        w = fastscan(s(u), v)
    else
        w = fastscan("", v[2..|v|])
    if w is an edge then
        add a node for w
        head(i+1) = w
    else if w is a node then
        head(i+1) = slowscan(w, tail(i))
        add head(i+1) as node if necessary
    s(head(i)) = w
    add leaf i+1 and edge between head(i+1) and i+1
```
Example: $x = abaab$

Construct tree for $x[1..n]$

```plaintext
for i = 1 to n do
  if head(i) = "" then
    head(i+1) = slowscan("", s(tail(i)))
    add $i+1$ and head($i+1$) as node if necessary
    continue
  u = parent(head(i)) ; v = label(u, head(i))
  if $u 
eq ""$ then
    w = fastscan(s(u), v)
  else
    w = fastscan("", v[2..|v|])
  if w is an edge then
    add a node for w
    head($i+1$) = w
  else if w is a node then
    head($i+1$) = slowscan(w, tail(i))
    add head($i+1$) as node if necessary
  s(head(i)) = w
  add leaf $i+1$ and edge between head($i+1$) and $i+1$
```

head(2) = ""
tail(2) = baab$

Diagram:

```
  abaac
  /   \
 /     /
baab /     /$baab$
  |    |
  1    2
```

$head(2) = ""$
tail(2) = baab$
Example: $x = abaab$

Construct tree for $x[1..n]$

```plaintext
for i = 1 to n do
    if head(i) = "" then
        head(i+1) = slowscan("", s(tail(i)))
        add $i+1$ and head($i+1$) as node if necessary
    continue
    u = parent(head(i)) ; v = label(u, head(i))
    if u = "" then w = fastscan(s(u), v)
    else
        w = fastscan("", v[2..|v|])
    if w is an edge then
        add a node for w
        head(i+1) = w
    else if w is a node then
        head(i+1) = slowscan(w, tail(i))
        add head($i+1$) as node if necessary
    s(head(i)) = w
    add leaf $i+1$ and edge between head($i+1$) and $i+1$
```

Example: $x = abaab$

Construct tree for $x[1..n]$

for $i = 1$ to $n$ do
    if head($i$) = "" then
        head($i+1$) = slowscan("", s(tail($i$)))
        add $i+1$ and head($i+1$) as node if necessary
        continue
    
    $u = \text{parent}(\text{head}(i))$ ; $v = \text{label}(u, \text{head}(i))$
    if $u 
eq "$" then $w = \text{fastscan}(s(u), v)$
    else $w = \text{fastscan}(\"", v[2..|v|])$
    if $w$ is an edge then
        add a node for $w$
        head($i+1$) = $w$
    else if $w$ is a node then
        head($i+1$) = slowscan($w$, tail($i$))
        add head($i+1$) as node if necessary
        $s(\text{head}(i)) = w$
    
add leaf $i+1$ and edge between head($i+1$) and $i+1$
Example: $x = abaab$

Construct tree for $x[1..n]$

for $i = 1$ to $n$ do
  if head($i$) = "" then
    head($i+1$) = slowscan("", $s(tail(i))$)
    add $i+1$ and head($i+1$) as node if necessary
  continue
  $u = parent(head(i))$ ; $v = label(u, head(i))$
  if $u 
eq ""$ then
    $w = fastscan(s(u), v)$
  else
    $w = fastscan("", v[2..|v|])$
  if $w$ is an edge then
    add a node for $w$
    head($i+1$) = $w$
  elseif $w$ is a node then
    head($i+1$) = slowscan($w, tail(i)$)
    add head($i+1$) as node if necessary
  $s(head(i)) = w$
  add leaf $i+1$ and edge between head($i+1$) and $i+1$
Example: $x = abaab$

Construct tree for $x[1..n]$

\[
\text{for } i = 1 \text{ to } n \text{ do}
\]

- if $\text{head}(i) = ""$ then
  - $\text{head}(i+1) = \text{slowscan}("", s(\text{tail}(i)))$
  - add $i+1$ and $\text{head}(i+1)$ as node if necessary
- continue

- $u = \text{parent}(\text{head}(i))$ ; $v = \text{label}(u, \text{head}(i))$
- \text{if } $u \neq ""$ then $w = \text{fastscan}(s(u), v)$
- else $w = \text{fastscan}("", v[2..|v|])$

- if $w$ is an edge then
  - add a node for $w$
  - $\text{head}(i+1) = w$
- else if $w$ is a node then
  - $\text{head}(i+1) = \text{slowscan}(w, \text{tail}(i))$
  - add $\text{head}(i+1)$ as node if necessary
  - $s(\text{head}(i)) = w$
- add leaf $i+1$ and edge between $\text{head}(i+1)$ and $i+1$
Example: $x = abaab$

Construct tree for $x[1..n]$

for $i = 1$ to $n$ do
  if head($i$) = ""
    head($i+1$) = slowscan(""; $s(tail(i))$
    add $i+1$ and head($i+1$) as node if necessary
    continue
  $u = parent(head(i))$ ; $v = label(u,head(i))$
  if $u$ ≠ "" then $w = fastscan(s(u),v)$
  else $w = fastscan(""; $v[2..|v|])$
  if $w$ is an edge then
    add a node for $w$
    head($i+1$) = $w$
  else if $w$ is a node then
    head($i+1$) = slowscan($w, tail(i)$)
    add head($i+1$) as node if necessary
  $s(head(i)) = w$
  add leaf $i+1$ and edge between head($i+1$) and $i+1$
Example: $x = abaab$

Construct tree for $x[1..n]$

for $i = 1$ to $n$ do
  if head($i$) = "" then
    head($i+1$) = slowscan("",$s(tail($i$)))
    add $i+1$ and head($i+1$) as node if necessary
  continue
  $u = \text{parent}(\text{head}(i))$; $v = \text{label}(u,\text{head}(i))$
  if $u =""$ then $w = \text{fastscan}(s(u),v)$
  else $w = \text{fastscan}("",v[2..|v|])$
  if $w$ is an edge then
    add a node for $w$
    head($i+1$) = $w$
  else if $w$ is a node then
    head($i+1$) = slowscan($w$,tail($i$))
    add head($i+1$) as node if necessary
  $s(\text{head}(i)) = w$
  add leaf $i+1$ and edge between head($i+1$) and $i+1$
Example: $x = abaab$

Construct tree for $x[1..n]$

for $i = 1$ to $n$ do
  if head($i$) = "" then
    head($i+1$) = slowscan("", $s(tail(i))$)
    add $i+1$ and head($i+1$) as node if necessary
    continue
  $u = parent(head(i))$; $v = label(u, head(i))$
  if $u \neq ""$ then $w = fastscan(s(u), v)$
  else $w = fastscan("", v[2..|v|])$
  if $w$ is an edge then
    add a node for $w$
    head($i+1$) = $w$
  else if $w$ is a node then
    head($i+1$) = slowscan($w, tail(i)$)
    add head($i+1$) as node if necessary
  $s(head(i)) = w$
  add leaf $i+1$ and edge between head($i+1$) and $i+1$
Example: $x = abaab$

Construct tree for $x[1..n]$

for $i = 1$ to $n$ do

  if head($i$) = "" then
    head($i+1$) = slowscan("", $s(tail(i))$)
    add $i+1$ and head($i+1$) as node if necessary
    continue

  $u = parent(head(i))$ ; $v = label(u, head(i))$

  if $u 
eq ""$ then
    $w = fastscan(s(u), v)$
  else
    $w = fastscan("", v[2..|v|])$

  if $w$ is an edge then
    add a node for $w$
    head($i+1$) = $w$
  else if $w$ is a node then
    head($i+1$) = slowscan($w$, tail($i$))
    add head($i+1$) as node if necessary
    $s(head(i)) = w$

add leaf $i+1$ and edge between head($i+1$) and $i+1$
Example: $x = abaab$

Construct tree for $x[1..n]$

```plaintext
for i = 1 to n do
    if head(i) = "" then
        head(i+1) = slowscan("", s(tail(i)))
        add i+1 and head(i+1) as node if necessary
        continue
    u = parent(head(i)) ; v = label(u, head(i))
    if u ≠ "" then
        w = fastscan(s(u), v)
    else
        w = fastscan("", v[2..|v|])
    if w is an edge then
        add a node for w
        head(i+1) = w
    else if w is a node then
        head(i+1) = slowscan(w, tail(i))
        add head(i+1) as node if necessary
        s(head(i)) = w
    add leaf i+1 and edge between head(i+1) and i+1
```
Example: $x = abaab$

Construct tree for $x[1..n]$

for $i = 1$ to $n$ do

  if $\text{head}(i)=\"\"$ then
    $\text{head}(i+1) = \text{slowscan}(\"\", s(\text{tail}(i)))$
    add $i+1$ and $\text{head}(i+1)$ as node if necessary
  continue

  $u = \text{parent}(\text{head}(i))$ ; $v = \text{label}(u, \text{head}(i))$

  if $u\neq\"\"$ then $w = \text{fastscan}(s(u), v)$
  else $w = \text{fastscan}(\"\", v[2..|v|])$

  if $w$ is an edge then
    add a node for $w$
    $\text{head}(i+1) = w$
  else if $w$ is a node then
    $\text{head}(i+1) = \text{slowscan}(w, \text{tail}(i))$
    add $\text{head}(i+1)$ as node if necessary
  $s(\text{head}(i)) = w$

add leaf $i+1$ and edge between $\text{head}(i+1)$ and $i+1$
Example: \(x = abaab\)

Construct tree for \(x[1..n]\)

for \(i = 1\) to \(n\) do
  if head\((i)\)="" then
    head\((i+1)\) = slowscan("",s(tail\((i)\)))
    add \(i+1\) and head\((i+1)\) as node if necessary
  continue

  \(u =\) parent(head\((i)\)) ; \(v =\) label\((u,\)head\((i)\))
  if \(u\neq""\) then \(w =\) fastscan\((s(u),v)\)
  else \(w =\) fastscan("",\(v[2..|v|]\))

  if \(w\) is an edge then
    add a node for \(w\)
    head\((i+1)\) = \(w\)
  else if \(w\) is a node then
    head\((i+1)\) = slowscan(w,tail\((i)\))
    add head\((i+1)\) as node if necessary
  \(s(\)head\((i)\)) = \(w\)

add leaf \(i+1\) and edge between head\((i+1)\) and \(i+1\)
Example: \( x = abaab \)

Construct tree for \( x[1..n] \)

for \( i = 1 \) to \( n \) do

if head\((i)\) = "" then
  head\((i+1)\) = slowscan("", s(tail\((i)\)))
  add \( i+1 \) and head\((i+1)\) as node if necessary
  continue

\( u = \text{parent}(\text{head}(i)) \) ; \( v = \text{label}(u, \text{head}(i)) \)

if \( u \neq "" \) then \( w = \text{fastscan}(s(u), v) \)
else \( w = \text{fastscan}("", v[2..|v|]) \)

if \( w \) is an edge then
  add a node for \( w \)
  head\((i+1)\) = \( w \)
else if \( w \) is a node then
  head\((i+1)\) = slowscan\((w, \text{tail}(i))\)
  add head\((i+1)\) as node if necessary

\( s(\text{head}(i)) = w \)

add leaf \( i+1 \) and edge between head\((i+1)\) and \( i+1 \)
Example: $x = abaab$

Construct tree for $x[1..n]$

for $i = 1$ to $n$ do
  if head($i$) = "" then
    head($i+1$) = slowscan("", $s(tail(i))$)
    add $i+1$ and head($i+1$) as node if necessary
  continue

  $u = parent(head(i))$ ; $v = label(u, head(i))$
  if $u<>""$ then $w = fastscan(s(u), v)$
  else $w = fastscan("", v[2..|v|])$

  if $w$ is an edge then
    add a node for $w$
    head($i+1$) = $w$
  else if $w$ is a node then
    head($i+1$) = slowscan($w$, tail($i$))
    add head($i+1$) as node if necessary
  $s(head(i)) = w$

add leaf $i+1$ and edge between head($i+1$) and $i+1$
Done

• Nothing new for i=5, inserting $...
Correctness

• Correctness follows from the invariant:
  • At iteration $i$ we have a trie of all suffixes $j < i$.
  • After the final iteration we have a trie of all suffixes of $x\$, i.e. we have the suffix tree of $x$. 

Everything but searching is constant time per suffix, so the running time is $O(n + \text{"slowscan" + "fastscan")}$. We are not using more space than time, so the space usage is the same.
Slowscan time usage

- We use slowscan to find head(i+1) from w=s(head(i)), i.e. time |head(i+1)|-|head(i)|+1 for iteration i

- A telescoping sum
  - Sum = |head(n+1)|-|head(1)|+n
  - Equal to n since head(n+1)=head(1)=""
Fastscan time usage

- Fastscan uses time proportional to the number of nodes it process
- Define $d(v)$ as the (node-)depth of node $v$
  - Fastscan increases the node depth
  - Following parent and suffix pointers decreases the node depth
- Time usage of fastscan is bounded by the total depth-increase (amortized analysis)
Proposition

- $d(v) \leq d(s(v)) + 1$
- Proof:
  - For any ancestor $u$ of $v$, $s(u)$ is an ancestor of $s(v)$
  - Except for the empty prefix and the single letter prefix of $v$, the $s(u)$'s are different
Corollary

- In each step, before calling fastscan, we decrease the depth by at most 2:

\[ d(u) = d(\text{head}(i))-1; \]
\[ d(w) \geq d(u)-1 \]

- The total decrease is thus 2n
Time usage of fastscan

- The time usage of fastscan is bounded by $n$ plus the total decrease of depth,
  - i.e. the time usage of fastscan is $O(n)$
We iteratively build tries of suffixes of $x$.

Using *suffix links* and *fastscan* we can quickly find where to insert the next suffix in our current trie.

By amortized analysis, the total running time becomes linear.
Ukkonen's suffix tree construction algorithm
Motivation

Yet another suffix tree construction algorithm...

Why?
Motivation

Yet another suffix tree construction algorithm... Why?

An online algorithm, i.e. one we can update with new sequences
Sketch of algorithm

- Recall McCreight’s approach:
  - For $i = 1 .. n+1$, build compressed trie of \( \{ x[j..n]$ | $j \leq i \} \)

- Ukkonen’s approach:
  - For $i = 1 .. n+1$, build compressed trie of \( \{ x[j..i] \ | \ j \leq i \} \)
  - Compressed trie of all suffixes of prefix $x[1..i]$ of $x$
  - A suffix tree except for “leaf” property
McCreight's algorithm – $x=aba$

$T_1 : \{x[j..n] | j \leq 1 \}$

$T_2 : \{x[j..n] | j \leq 2 \}$

$T_3 : \{x[j..n] | j \leq 3 \}$

$T_4 : \{x[j..n] | j \leq 4 \}$
Ukkonen's algorithm – \( x = \text{aba} \)

- \( T_1 : \{x[j..1]\} \)
- \( T_2 : \{x[j..2]\} \)
- \( T_3 : \{x[j..3]\} \)
- \( T_4 : \{x[j..n]\} \)

Note: no node for \( x[3..3] = \text{“a”} \)
Tasks in iteration $i$

In iteration $i$ we must

- Update each $x[j..i]$ to $x[j..i+1]$
- Add string $x[i+1]$ (special case of above)
First attempt...

“Obvious” algorithm:

\[
\text{For } i=1,\ldots,n+1: \\
\text{for } j=1,\ldots,i: \\
\quad \text{find } x[j..i] \\
\quad \text{append } x[i+1]
\]
First attempt...

“Obvious” algorithm:

For $i=1,...,n+1$:
  for $j=1,...,i$:
    find $x[j..i]$
    append $x[i+1]$

- Running time $O(n^3)$
- Need lots of tricks to get $O(n)$!
Updating leaves for free

If we label leaves with \((k, \infty)\) – denoting “k to the current i”, updating a leaf is automatic.
Updating existing strings is free

If $x[j..i+1]$ is already in the tree, the update is automatic
“Real” operations

If we can recognize the free operations, we need only deal with the remaining
Lemma 5.2.4

Let $j$ denote suffix $x[j..i]$ of $x[1..i]$

a) If $j>1$ is a leaf node in $T_i$, then so is $j-1$

b) If, from $j<i$, there is a path in $T_i$ that begins with “a”, then there is a path in $T_i$ from $j+1$ beginning with “a”
Lemma 5.2.4

Let j denote suffix $x[j..i]$ of $x[1..i]$

a) **If $j>1$ is a leaf node in $T_i$, then so is $j-1$**

b) **If, from $j<i$, there is a path in $T_i$ that begins with “a”, then there is a path in $T_i$ from $j+1$ beginning with “a”**
Proof of lemma 5.2.4 a)

If \( j>1 \) is a leaf node in \( T_i \), then so is \( j-1 \)

Assume \( j-1 \) is not a leaf. Then there exists \( k<j-1 \) such that:

Then \( j-1 \) thus:

\[
x[k..i] \quad x[j-1..i] \quad x[j..i] \quad j-1 \quad k+1
\]

\[
x[k+1..i] \quad x[j..i] \quad x[k+1..i] \quad x[j..i]
\]
Lemma 5.2.4

Let $j$ denote suffix $x[j..i]$ of $x[1..i]$

a) If $j>1$ is a leaf node in $T_i$, then so is $j-1$

b) If, from $j<i$, there is a path in $T_i$ that begins with “a”, then there is a path in $T_i$ from $j+1$ beginning with “a”
Proof of lemma 5.2.4 b)

If, from \( j < i \), there is a path in \( T_i \) that begins with “a”, then there is a path in \( T_i \) from \( j+1 \) beginning with “a”.

Assume \( j \) is followed by “a”, then there exists \( k < j \) such that:

\[
\begin{array}{c}
  j \quad \text{“a”} \\
  k \\
\end{array}
\]

thus:

\[
\begin{array}{c}
  j+1 \quad \text{“a”} \\
  k+1 \\
\end{array}
\]

Hence \( j+1 \) is followed by “a”.
Corollary

In iteration $i$, there exist indices $j_L$ and $j_R$ such that:

- All suffixes $j \leq j_L$ are leaves
- All suffixes $j \geq j_R$ are already in the trie
Consequence of corollary

I and III are free operations
Updated algorithm

Implicitly handling “free” operations:

For $i=1,\ldots,n+1$:
  for $j=j_L,\ldots,j_R$:
    find $x[j..i]$
    append $x[i+1]$
Updated algorithm

Implicitly handling “free” operations:

For $i=1,...,n+1$:
  for $j=j_L,...,j_R$:
    find $x[j..i]$
    append $x[i+1]$

How do we find $j_L$ and $j_R$?
Updated algorithm

Implicitly handling "free" operations:

For $i=1,...,n+1$:
   for $j=j_L,...,j_R$:
      find $x[j..i]$
      append $x[i+1]$

- $j_L$ in iteration $i$ is the last leaf inserted in iteration $i-1$
  (all smaller indices are already leaves)
- $j_R$ in iteration $i$ is the first index where $x[j..i+1]$ is
  already in the trie (all larger indices are already in the trie)
Suffixes in II are made into leaves

Whenever $j_L < j < j_R$, $j$ is made a leaf:

![Diagram showing suffixes and indices]

$x[j..i]$ $x[j..i]$ $x[i+1]$
Suffixes in II are made into leaves

Whenever $j_L < j < j_R$, $j$ is made a leaf:

Once $j$ is a leaf, it will be in I and never in II again
Time to go from II to I

We handle j in II or implicitly in III time 2n:

"Path" length is 2n
Updated running time

<table>
<thead>
<tr>
<th>For i=1,...,n+1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>for j=j_L,...,j_R:</td>
</tr>
<tr>
<td>find x[j..i]</td>
</tr>
<tr>
<td>append x[i+1]</td>
</tr>
</tbody>
</table>

Running time is $2n \cdot T(\text{find } x[j..i])$

Only 2n of these:
Updated running time

For $i=1,...,n+1$:
   for $j=j_L,...,j_R$:
      find $x[j..i]$
      append $x[i+1]$

Running time is $2n \cdot T(\text{find } x[j..i])$

- We just have to deal with $T(\text{find } x[j..i])$ in $O(1)$
What do we know about $x[j..i]$?
Is it already in the tree?
Will that help us?
Using *fastscan* and *s*(-)

- When searching for $x[j..i]$, it is already in the trie
  - We can use *fastscan* for the search
  - $T(\text{find } x[j..i])$ in $O(d)$ where $d$ is the (node-)depth of $x[j..i]$
- If we keep suffix links, *s*(-), in the tree we can use these as shortcuts
Suffix links

Invariant: All inner nodes have suffix links
**Suffix links**

**Invariant:** All inner nodes have suffix links

Ensuring the invariant?

For $i=1,\ldots,n+1$:

for $j=j_L,\ldots,j_R$:

- find $x[j..i]$,
- append $x[i+1]$
**Suffix links**

**Invariant:** All inner nodes have suffix links

Ensuring the invariant:

- We only insert inner nodes $x[j..i]$ when adding leaves $j$
- Whenever we insert a new node, $x[j..i]$ for some $j<i$, we also find or insert $x[j+1..i]$, and can update $s(x[j..i]) := x[j+1..i]$
- If we insert $x[i..i]$, then $s(x[i..i]) := \varepsilon$
Finding $x[j+1..i]$ from $x[j..i]$

Starting from here (initial $j$ is $j_L$ and we can keep a pointer to that node between iterations)

Using `fastscan` here
Bound on \texttt{fastscan}

Time usage by \texttt{fastscan} is bounded by \( n \) – for the maximal (node-)depth in the trie – plus total decrease of (node-)depth

- Decrease in depth:
  - Moving to parent\((j)\): 1
  - Moving to \( s(\text{parent}(j)) \): max 1
  - “Restarting” at \( j_L \): ?
Bound on fastscan

Time usage by fastscan is bounded by $n$ – for the maximal (node-)depth in the trie – plus total decrease of (node-)depth

- Decrease in depth:
  - Moving to parent($j$): 1
  - Moving to $s$ (parent($j$)): max 1
  - “Restarting” at $j_L$: ?

Using fastscan here
When searching for $x[j_L+1..i]$, update $s(x[j_L..i])$ to point to the nearest ancestor of $x[j_L+1..i]$.
Hacking the suffix link

When searching for $x[j_L+1..i]$, update $s(x[j_L..i])$ to point to the nearest ancestor of $x[j_L+1..i]$

"Restart" in constant time
Searching time

- Vertical steps are paid for by the previous horizontal step (free restarting)
- Horizontal steps are total `fastscan` bounded by $O(n)$
- Runtime $O(n)$
Summary

• We have seen a new suffix tree construction algorithm

• Ukkonen’s algorithm is an “online” algorithm:
  - As long as no suffix is a prefix of another, the intermediate trees are suffix trees
  - Generalized suffix trees can be built one string at a time