**Suffix Trees**

**Text (T):** SHESELLSSSEASHELLSBYTHESEASHORE

**Pattern (P):** ASH

**Goal:** see if P is in T

\[ \Omega(P) \rightarrow \text{multiple searches: } \Omega(\Sigma(P_i)) \]

1) as fast as possible, after pre-processing T
2) but also minimize pre-processing time/space

\[ \Omega(T) \]
Text : XABXAC

Suffixes: 1 XABXAC
2 ABXAC
3 BXAC
4 XAC
5 AC

|T| = "T" : 6

Search for pattern P: it starts at some index i, so is in suffix starting at i.
Text: XABXAC

using start/end indices,

\[ \text{size(tree)} = O(T) \]

assuming \(|\text{alphabet}| = \text{const.}\)

search time: \(O(P)\) (to find if \(\exists\) match)

(augment: subtree sizes for \# matches)

\[ \text{every suffix is in tree} \]

\[ \# \text{leaves} = T \Rightarrow \text{size(tree)} = O(T) \]

\(\#\) nodes (every internal node has \(\geq 2\) children)
The suffixes are still there, but we can't enumerate matches. Also we can't answer other queries e.g. "find words starting with AB..."
solution

alphabet: A, B, $, X

if we follow previous rules we get
Ukkonen's Algorithm to build a suffix tree in $O(T)$ time

- Iteratively build suffix tree: iteration $i$ builds tree on $T[1...i]$

$XABXA$

At iteration $i$, we make $\sum_{i=1}^{\lfloor\frac{|T|}{i}\rfloor} i$ extensions. We must reduce this.
At iteration $i$, we make $i$ "extensions"

Each extension handles one of the existing suffixes (including empty suffix)

3 cases depending on how current suffix "ends"

1) at a leaf

2) not at a leaf, but new character is already there

3) not at a leaf, new char. not there
Another example: $AXABXB$

<table>
<thead>
<tr>
<th>suffix</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$AXABXB$</td>
<td>$XABXB$</td>
<td>$AXBXB$</td>
<td>$BXB$</td>
<td>$XB$</td>
<td>$B$</td>
</tr>
</tbody>
</table>

Let $T[6] = B$
(iteration 6)

extension 1 2 3 4 5 6

case 1 1 1 1 1 3 2

1) at a leaf → extend current label
2) not at a leaf, but new character is already there → do nothing
3) not at a leaf, new char. not there → make new edge w/new character
At iteration $i$, we make $i$ "extensions".

For each extension: 3 cases depending on how current suffix "ends"

- Each looks easy & quick
- How do we determine this?

* Scanning whole suffix $\to O(i^2)$ per iteration
* Using indexing doesn't help (yet)

$$\sum_{j=1}^{i} i-j$$

$A\times A\times B\times B$

1) at a leaf $\to$ extend current label
2) not at a leaf, but new character is already there $\to$ do nothing
3) not at a leaf, new char. not there $\to$ make new edge w/new character
How do we determine where a current suffix "ends"? (quickly)

Suffix Links

Say we just found the end of a suffix, XABCDE
(maybe while extending it w/F or something else)

Then ABCDE is also a suffix, so it is also in the tree
and it is the next extension
so it would help to have a link from every xα to α ...
except maybe there was no node at xα

T[1...i-1] = XABCDEFGXABCDE
$T[1...i-1] = XABCDEFGXABCDE$

Actual suffix tree

(now enjoy this unused slide)

Lots of repeated structure

For illustration
Instead move up until a node is found, use a suffix link from node to node, then "move down"

next issue:

how to make suffix links
When do we create a new node?

- Case where we process $x\alpha$ to extend $w$ and $\exists X\alpha Z \ (Z \neq w)$
  
  (end of $X\alpha$ is in mid-edge)

When do we create a suffix link from $x\alpha$ to $\alpha$?

- First we need to know that a node exists at $\alpha$. 
When do we create a new node?

- Case where we process $x\alpha$ to extend $w$ and $\exists x\alpha z \ (z \neq w)$
  (end of $x\alpha$ is in mid-edge)

When do we create a suffix link from $x\alpha$ to $\alpha$?

- First we need to know that a node exists at $\alpha$.
- Either it does already, or it will be created at next extension (we will process $\alpha$)

So we can assume that all "old" nodes have outgoing suffix links
Example of suffix link target node existing before source node

\[ s(v) \]

"to be created at extension 9"

1 2 3 4 5 6 7 8 9

\[ \alpha Z \alpha Y \alpha Y \alpha Y \alpha \omega \]

currently processing extension 8
Technical details

For each edge we store the start/end indices. While processing a suffix that is the prefix of a larger suffix we would be using indices implicitly (to find actual chars from T). Besides that, we just care about \(|\text{start-end}| = \text{length}\)

WHAT MATTERS:

using indices we can move between nodes in \(O(1)\) time
When following a suffix link, node depth (from root) can
- stay same
- increase (down to leaf)
- decrease (i.e. move up)

(example to follow)

\[ \alpha = xy... \]

2 examples \[ \rightarrow \] new1 -> new2 -> new3 -> s(v)
example of $v$ with suffix link to $s(v)$
where $\text{depth}(v) \ll \text{depth}(s(v))$
When following a suffix link, node depth (from root) can

- stay same
- increase (down to ~leaf)
- decrease (only by 1)

(i.e. move up)

We've seen: if $xy$ leads to $v$ then $y$ is in tree, leading to $s(v)$
So every node on path from $x$ to $v$ has a "mirror" node on path from root to $s(v)$

(reverse not necessarily true)
When following a suffix link, node depth (from root) can
- stay same
- increase (down to leaf, i.e., move up)
- decrease (only by 1)

amortize: total node hops per iteration = $O(i)$

[total upward = $O(i)$ & max depth = $O(i)$]

**CONCLUSION**

During iteration $i$, we visit $O(i)$ nodes
...spending $O(i)$ time to find each, excluding the first one: total work $O(i)$
Story so far: find "end" of next extension using suffix link from end of current extension (plus a bit of up/down)

What if current extension makes no new node?

extend $\alpha w + w$ but $\alpha w$ already in tree ($\alpha w$ also in tree)

All future extensions in current iteration will also follow same case

Table of cases applied in each extension of each iteration

1) at a leaf $\rightarrow$ extend current label
2) not at a leaf, but new character is already there $\rightarrow$ do nothing
3) not at a leaf, new char. not there $\rightarrow$ make new edge $w$/new character
Story so far: find "end" of next extension using suffix link from end of current extension (plus a bit of up/down)

What if current extension makes no new node?

First extension of every iteration: case 1
(longest suffix ends at leaf)

1) at a leaf → extend current label
2) not at a leaf, but new character is already there → do nothing
3) not at a leaf, new char. not there → make new edge w/new character
Story so far: find "end" of next extension using suffix link from end of current extension (plus a bit of up/down)

What if current extension makes no new node?

All suffixes considered in the future will divert elsewhere in the tree or stop short on this path, or extend this path.

"once a leaf, always a leaf"

What if arbitrary extension j of iteration i uses case 1?

1) at a leaf → extend current label
2) not at a leaf, but new character is already there → do nothing
3) not at a leaf, new char. not there → make new edge w/new character
Any future extension $j$ has same prefix, so will follow same path & use same rule.

1) at a leaf $\rightarrow$ extend current label
2) not at a leaf, but new character is already there $\rightarrow$ do nothing
3) not at a leaf, new char. not there $\rightarrow$ make new edge w/ new character

For future iterations:
- If $k < j$, the extension diverges;
- If $k > j$, the extension is too short.

All suffixes considered in the future will divert elsewhere in the tree, or stop short on this path, or extend this path.

"Once a leaf, always a leaf."
what if extension j uses case 3? → creates a leaf
... same logic, must have 1's below

We've determined that below any 1 we can have only 1's

1) at a leaf → extend current label
2) not at a leaf, but new character is already there → do nothing
3) not at a leaf, new char. not there → make new edge & new character
How to handle the 1's: use a global variable = iteration #

- recall, the 1's involve leaf (edge) extensions
- edges are represented by indices in $T$.
- the "end" index just gets incremented: $i-1 \rightarrow i$
Last step: add $\$: makes all implicit suffixes proper

Conclusion: suffix tree construction $\rightarrow O(T)$ time & space