Building suffix tree

S = ababaa$

6 $

5 a$

4 aa$

2 abaa$

0 ababaa$

3 baa$

1 babaa$

Building suffix tree

S = ababaa$

6 $

5 a$

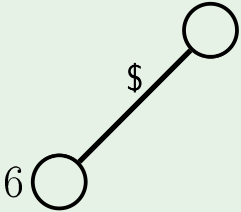
4 aa$

2 abaa$

0 ababaa$

3 baa$

1 babaa$



Building suffix tree

S = ababaa$

6 $

5 a$

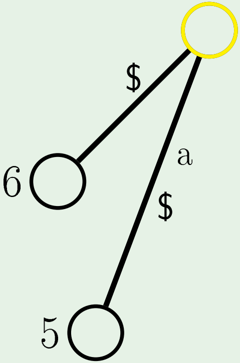
4 aa$

2 abaa$

0 ababaa$

3 baa$

1 babaa$



Building suffix tree

S = ababaa$

6 $

5 a$

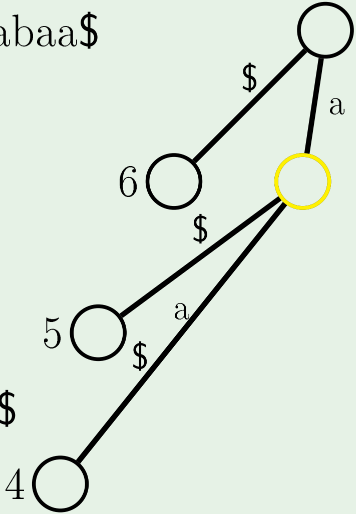
4 aa$

2 abaa$

0 ababaa$

3 baa$

1 babaa$



Building suffix tree

$S = \text{ababaa\$}$

6 \$

5 a\$

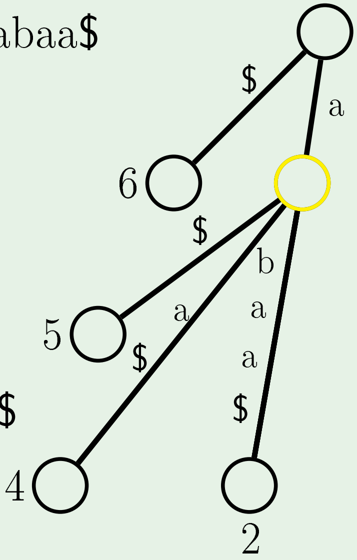
4 aa\$

2 abaa\$

0 ababaa\$

3 baa\$

1 babaa\$



Building suffix tree

S = ababaa$

6 $

5 a$

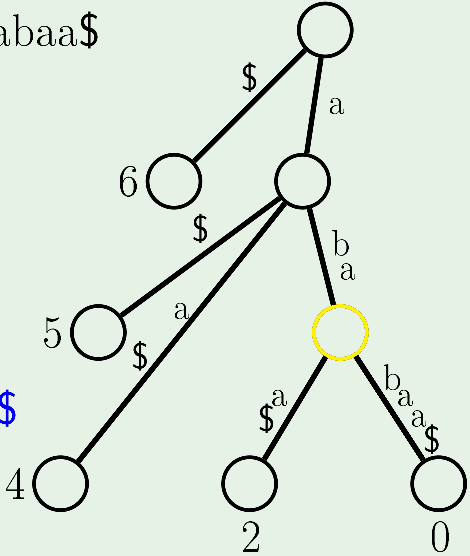
4 aa$

2 abaa$

0 ababaa$

3 baa$

1 babaa$



Building suffix tree

$S = ababaa\$$

6 \$

5 a\$

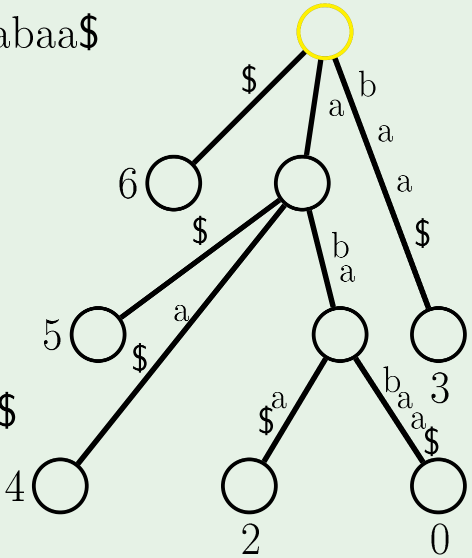
4 aa\$

2 abaa\$

0 ababaa\$

3 baa\$

1 babaa\$



Building suffix tree

S = ababaa$

6 $

5 a$

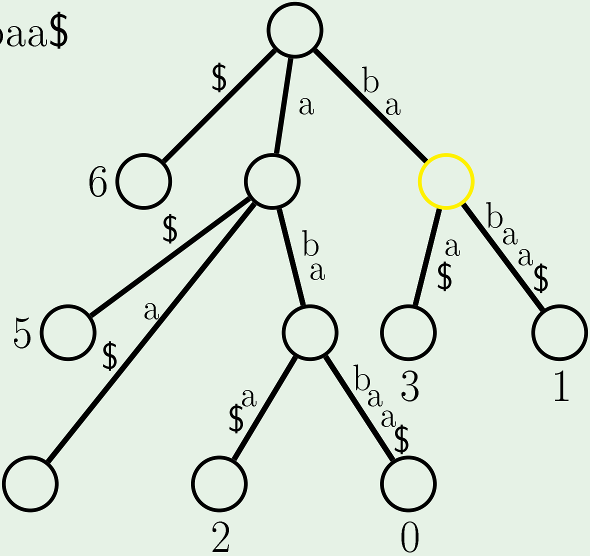
4 aa$

2 abaa$

0 ababaa$

3 baa$

1 babaa$



Algorithm

- Build suffix array and LCP array
- Start from only root vertex
- Grow first edge for the first suffix
- For each next suffix, go up from the leaf until LCP with previous is below
- Build a new edge for the new suffix

class SuffixTreeNode:

SuffixTreeNode parent
Map<char, SuffixTreeNode> children
integer stringDepth
integer edgeStart
integer edgeEnd

STFromSA(S, order, lcpArray)

```
root ← new SuffixTreeNode(
   children = {}, parent = nil, stringDepth = 0,
   edgeStart = −1, edgeEnd = −1)
lcpPrev ← 0
curNode ← root
for i from 0 to |S| − 1:
   suffix ← order[i]
   while curNode.stringDepth > lcpPrev:
      curNode ← curNode.parent
   if curNode.stringDepth == lcpPrev:
      curNode ← CreateNewLeaf(curNode, S, suffix)
   else:
      edgeStart ← order[i − 1] + curNode.stringDepth
      offset ← lcpPrev − curNode.stringDepth
      midNode ← BreakEdge(curNode, S, edgeStart, offset)
      curNode ← CreateNewLeaf(midNode, S, suffix)
   if i < |S| − 1:
      lcpPrev ← lcpArray[i]
return root
```

CreateNewLeaf(node, S, suffix)

$$
\begin{aligned}
&\mathrm{leaf} \leftarrow \mathrm{new\ SuffixTreeNode}(\\
&\quad \mathrm{children} = \{\},\\
&\quad \mathrm{parent} = \mathrm{node},\\
&\quad \mathrm{stringDepth} = |S| - \mathrm{suffix},\\
&\quad \mathrm{edgeStart} = \mathrm{suffix} + \mathrm{node.stringDepth},\\
&\quad \mathrm{edgeEnd} = |S| - 1)\\
&\mathrm{node.children}[S[\mathrm{leaf.edgeStart}]] \leftarrow \mathrm{leaf}\\
&\mathrm{return\ leaf}
\end{aligned}
$$

BreakEdge(node, S, start, offset)

```
startChar ← S[start]
midChar ← S[start + offset]
midNode ← new SuffixTreeNode(
    children = {},
    parent = node,
    stringDepth = node.stringDepth + offset,
    edgeStart = start,
    edgeEnd = start + offset − 1)
midNode.children[midChar] ← node.children[startChar]
node.children[startChar].parent ← midNode
node.children[startChar].edgeStart+ = offset
node.children[startChar] ← midNode
return midNode
```

Analysis

Lemma

This algorithm runs in $O(|S|)$

Proof

- Total number of edges in suffix tree is $O(|S|)$
- For each edge, we go at most once down and at most once up
- Constant time to create a new edge and possibly a new node