Algorithmic Challenges: From Suffix Array to Suffix Tree

Michael Levin

Higher School of Economics

Algorithms on Strings
Data Structures and Algorithms

Outline

Construct suffix Tree

Input: String S
Output: Suffix tree of S

- You already know how to construct suffix tree
- But $O(|S|^2)$ will only work for short strings
- You will learn to build it in $O(|S| \log |S|)$ which enables very long texts!

General Plan

- Construct suffix array in $O(|S| \log |S|)$
- Compute additional information in $O(|S|)$
- Construct suffix tree from suffix array and additional information in $O(|S|)$

Suffix array and suffix tree

$S = ababaa\$$
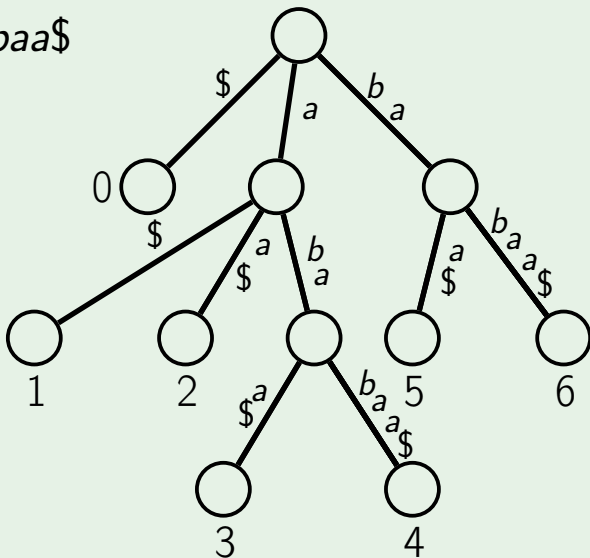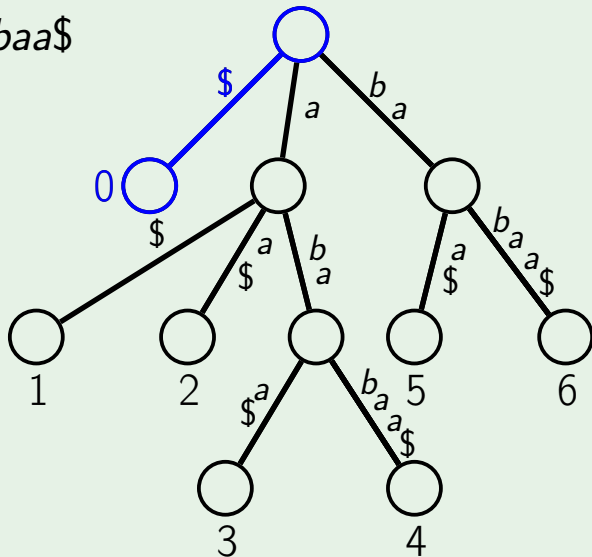
0 $\$$

1 $a\$$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

Suffix array and suffix tree

$S = ababaa\$$

0 $\$$

1 $a\$$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

Suffix array and suffix tree

$S = ababaa\$$

0 $

1 a$

2 aa$

3 abaa$

4 ababaa$

5 baa$

6 babaa$

Suffix array and suffix tree
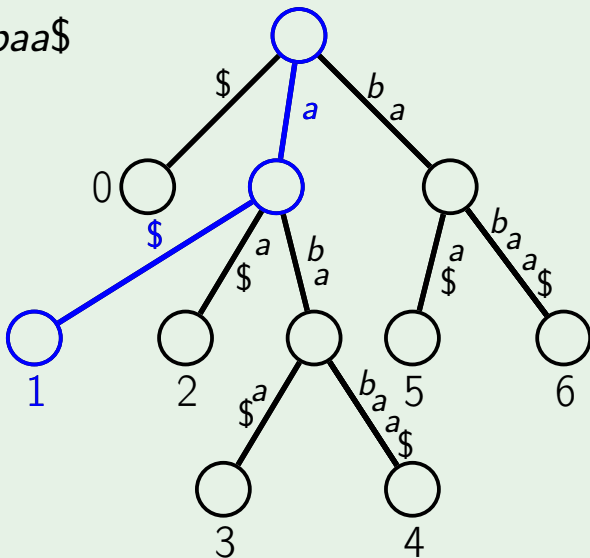
$S = ababaa\$$

0 $\$$

1 $a\$$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

Suffix array and suffix tree

$S = ababaa\$$

0 $\$$

1 $a\$$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

Suffix array and suffix tree
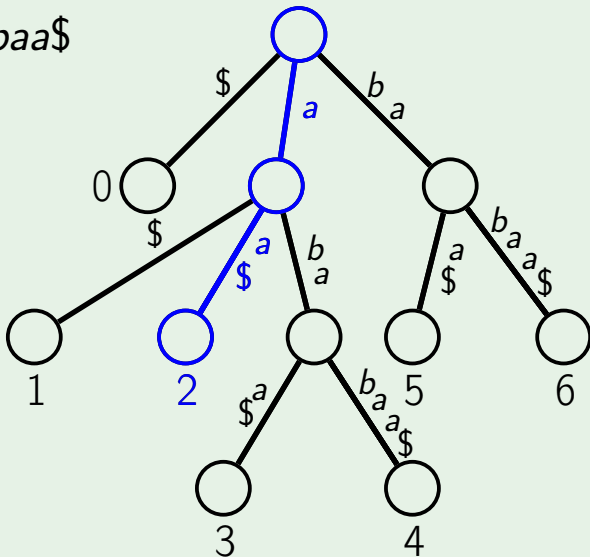
$S = ababaa\$$

0 $\$$

1 $a\$$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

Suffix array and suffix tree
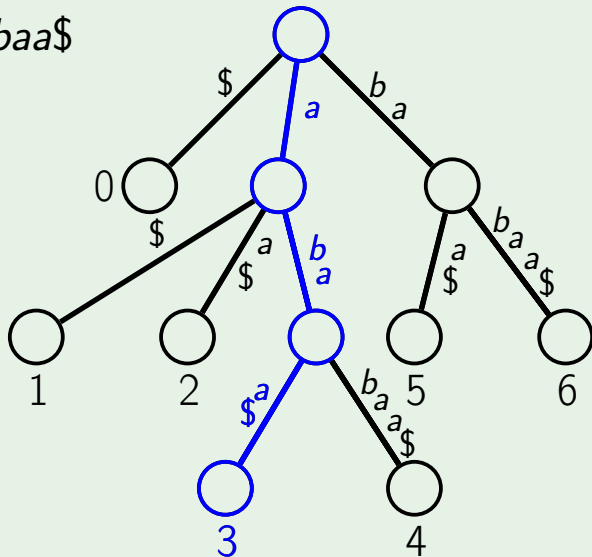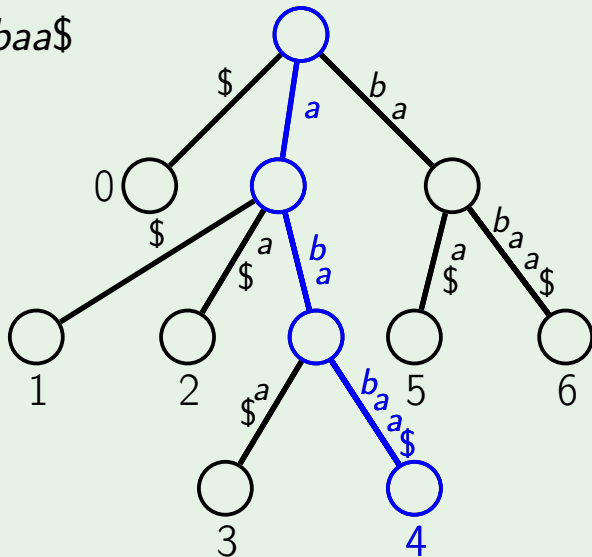
$S = ababaa\$$

0 $\$$

1 $a\$$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

Suffix array and suffix tree

$S = ababaa\$$

0 $\$$

1 $a\$$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

Definition

The longest common prefix (or just "lcp") of two strings S and T is the longest such string u that u is both a prefix of S and T. We denote by $LCP(S, T)$ the length of the "lcp" of S and T.

Example

$LCP(\text{"ababc"}, \text{"abc"}) = 2$
$LCP(\text{"a"}, \text{"b"}) = 0$

Suffix array, suffix tree and lcp

$S = ababaa\$$
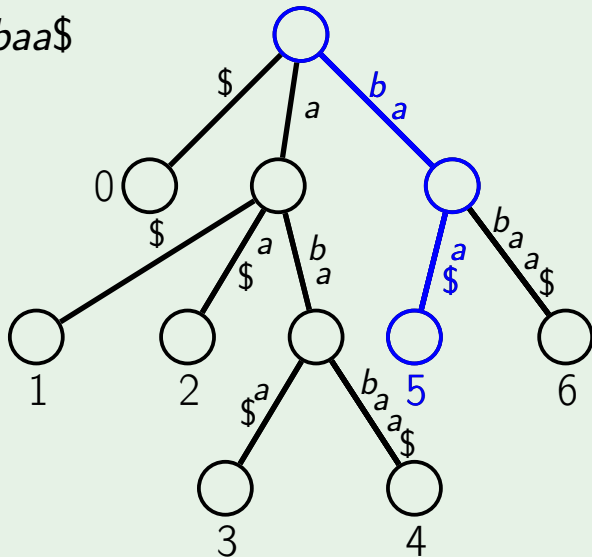
0 $\$$

1 $a\$$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

Suffix array, suffix tree and lcp

$S = ababaa\$$

0 $\$$

1 $a\$$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

$S = ababaa\$$

0 $\$$

1 $a\$$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

Suffix array, suffix tree and lcp

Suffix array, suffix tree and lcp

$S = ababaa\$$

0 $\$$

1 $a\$$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

Suffix array, suffix tree and lcp
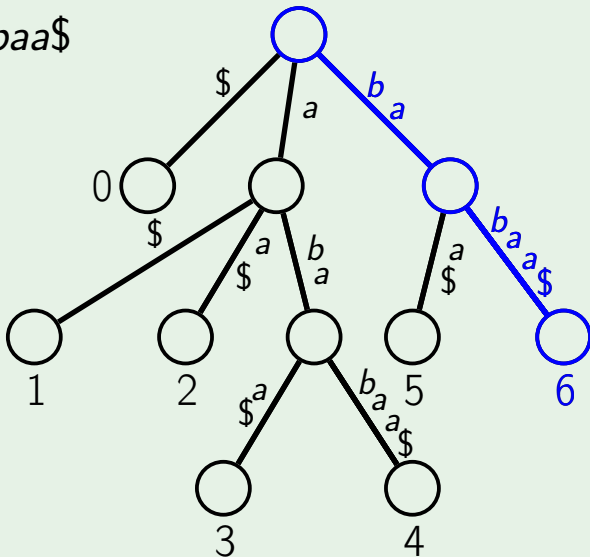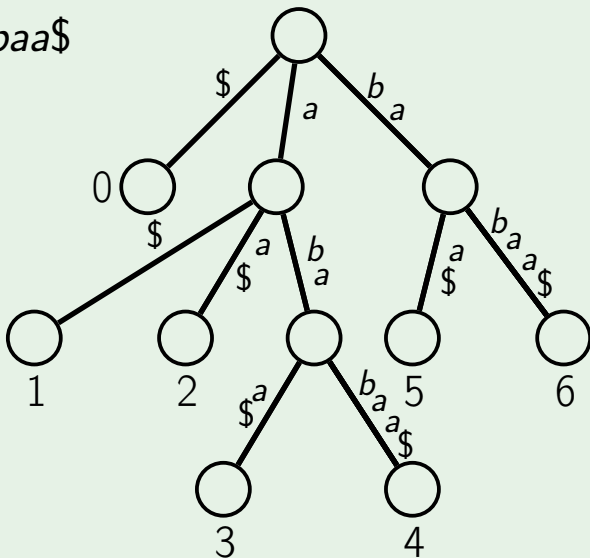
$S = ababaa\$$

0 $\$$

1 $a\$$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

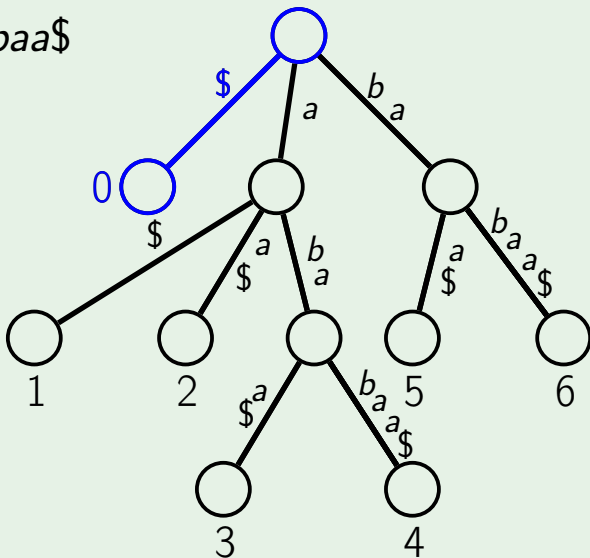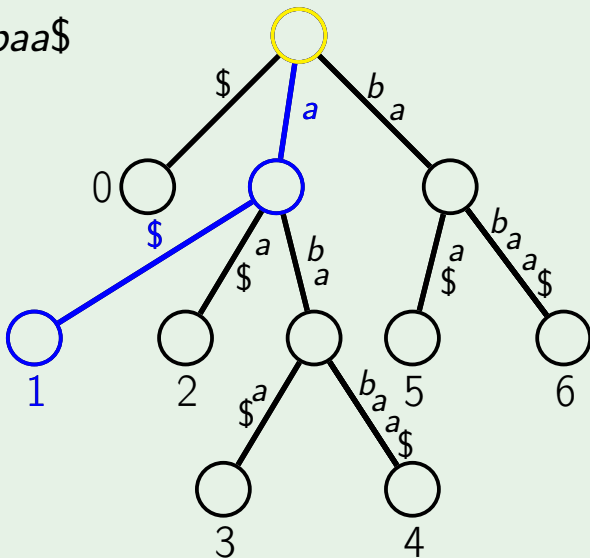# Suffix array, suffix tree and lcp

$S = ababaa\$$

0 \$

1 a\$

2 aa\$

3 abaa\$

4 ababaa\$

5 baa\$

6 babaa\$

Suffix array, suffix tree and lcp

$S = ababaa\$$

0 $\$$

1 $a\$$

2 $aa\$$
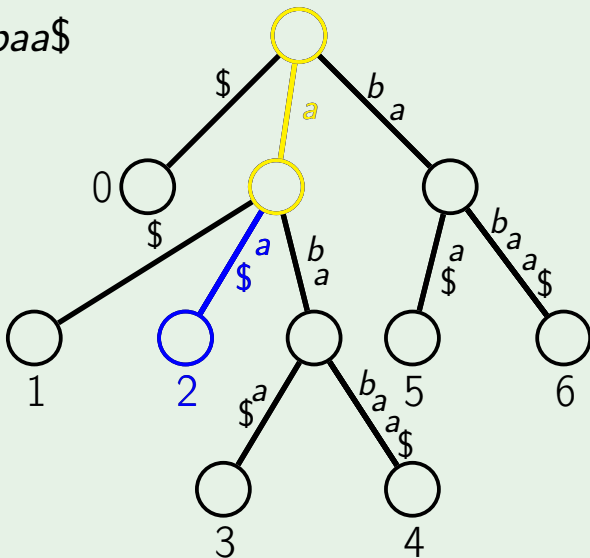
3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

LCP array

Definition

Consider suffix array A of string S in the raw form, that is
$A[0] < A[1] < A[2] < \cdots < A[|S| - 1]$ are all the suffixes of S in lexicographic order.
LCP array of string S is the array lcp of size $|S| - 1$ such that for each i such that $0 \leq i \leq |S| - 2$,

$$lcp[i] = LCP(A[i], A[i + 1])$$

LCP array

$S = ababaa\$$

0 $\$$

1 $a\$$ $\qquad lcp = [\ ,\ ,\ ,\ ,\ ,\ ]$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

LCP array

$S = ababaa\$$

0 $\$$

1 $a\$$ $lcp = [\ , \ , \ , \ , \ , \ ]$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

LCP array

$S = ababaa\$$

0 $\$$

1 $a\$$ $lcp = [0, \ , \ , \ , \ , \ ]$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

LCP array

$S = ababaa\$$

0 $\$$

1 *a*$\$$ $lcp = [0, 1, \ , \ , \ , \ ]$

2 *aa*$\$$

3 *abaa*$\$$

4 *ababaa*$\$$

5 *baa*$\$$

6 *babaa*$\$$

LCP array

$S = ababaa\$$

0 $\$$

1 $a\$$ $lcp = [0, 1, 1, \ , \ , \ ]$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

LCP array

$S = ababaa\$$

0 $\$$

1 $a\$$ $lcp = [0, 1, 1, 3, \ , \ ]$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

LCP array

$S = ababaa\$$

0 $\$$

1 $a\$$ $lcp = [0, 1, 1, 3, 0, \ ]$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

LCP array

$S = ababaa\$$

0 $\$$

1 $a\$$ $lcp = [0, 1, 1, 3, 0, 2]$

2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$

LCP array property

Lemma

For any $i < j$, $LCP(A[i], A[j]) \leq lcp[i]$ and $LCP(A[i], A[j]) \leq lcp[j-1]$.

Proof

$\dots$

i	*abababab*
$i + 1$	*abababc*
	$\dots$
j	*abbcabab*

Proof

     ...

i *abababaaba*

$i + 1$ *abababc*

     ...

j *abbcabab*

Proof

$\ldots$

| i | *ab*ababab*a* |
| $i + 1$ | *xxxxxxxxx* |

$\ldots$

| j | *ab*bcabab |

If $LCP(A[i], A[j]) > LCP(A[i], A[i+1])$

Proof

$\cdots$

i ab*abababa*

$i + 1$ xx*xxxxxxx* $k = 1$

$\cdots$

j ab*bcabab*

If $LCP(A[i], A[j]) > LCP(A[i], A[i + 1])$
Consider $k = LCP(A[i], A[i + 1])$

Proof

. . .

i ab*ababab*a

$i + 1$ a_ $k = 1$

. . .

j ab*bcabab*

If $k = |A[i + 1]|$, then $A[i + 1] < A[i]$ – contradiction

Proof

$\dots$

i ab*ababa*

$i+1$ a*xxxxxxx* $k=1$

$\dots$

j ab*bcabab*

Otherwise $A[j][k] = A[i][k] \neq A[i+1][k]$

Proof

$\cdots$

i ab*ababa*

$i+1$ ac*xxxxxxx* $k = 1$

$\cdots$ $\bigvee$

j ab*bcabab*

If $A[j][k] = A[i][k] < A[i+1][k]$, then
$A[j] < A[i+1]$ — contradiction

Proof

. . .

i ab*ababab*

$\vee$

$i + 1$ aa*xxxxxx* $k = 1$

. . .

j ab*bcabab*

If $A[i][k] > A[i+1][k]$, then $A[i] > A[i+1]$ — contradiction $\square$

Computing LCP array

- For each i, compute $LCP(A[i], A[i + 1])$ via comparing $A[i]$ and $A[i + 1]$ character-by-character
- $O(|S|)$ for each i, $O(|S|)$ different i — total time $O(|S|^2)$
- How to do this faster?

Outline

Idea

Lemma

Let h be the longest common prefix between S_{i-1} and its adjacent (next) suffix in the suffix array of string S. Then the longest common prefix between S_i and its adjacent (next) suffix in the suffix array is at least $h - 1$.

$$S = abracadabra\$$$

index	sorted suffix	LCP
...	...	...
$i = 10$	a\$	
7	abra\$	
...	...	...
$j = 3$	acadabra\$	
...	...	...
$i - 1 = 9$	ra\$	
$j - 1 = 2$	racadabra\$	

$$S = abracadabra\$$$

index	sorted suffix	LCP
...	...	...
$i = 10$	a$	
7	abra$	
...	...	...
$j = 3$	acadabra$	
...	...	...
$i - 1 = 9$	ra$	$h = 2$
$j - 1 = 2$	racadabra$	

$$S = abracadabra\$$$

index	sorted suffix	LCP
...	...	...
$i = 10$	a\$	
7	abra\$	
...	...	...
$j = 3$	acadabra\$	
...	...	...
$i - 1 = 9$	ra\$	$h = 2$
$j - 1 = 2$	racadabra\$	

$S = abracadabra\$$

index	sorted suffix	LCP
...	...	...
$i = 10$	a$	$1 \geq h - 1$
7	abra$	
...	...	...
$j = 3$	acadabra$	
...	...	...
$i - 1 = 9$	ra$	$h = 2$
$j - 1 = 2$	racadabra$	

Idea

- Start by computing $LCP(A[0], A[1])$ directly
- Instead of computing to $LCP(A[1], A[2])$, move $A[0]$ one position to the right **in the string**, get some $A[k]$ and compute $LCP(A[k], A[k+1])$
- Repeat this until LCP array is fully computed
- Length of the LCP never decreases by more than one each iteration

Notation

- Let $A_{n(i)}$ be the suffix starting in the next position **in the string** after $A[i]$

Example

a	b	a	b	d	a	b	c

- $A[0] = \text{``ababdabc''}, A[1] = \text{``abc''}$
- Compute $LCP(A[0], A[1]) = 2$ directly
- $LCP(A_{n(0)}, A_{n(1)}) \geq LCP(A[0], A[1]) - 1$
- $A[0] < A[1] \Rightarrow A_{n(0)} < A_{n(1)}$
- LCP of $A_{n(0)}$ with the next **in order** $A[j]$ is also at least $LCP(A[0], A[1]) - 1$

Example

a	b	a	b	d	a	b	c

- $A[0] = \text{"ababdabc"}$, $A[1] = \text{"abc"}$
- Compute $LCP(A[0], A[1]) = 2$ directly
- $LCP(A_{n(0)}, A_{n(1)}) \geq LCP(A[0], A[1]) - 1$
- $A[0] < A[1] \Rightarrow A_{n(0)} < A_{n(1)}$
- LCP of $A_{n(0)}$ with the next **in order** $A[j]$ is also at least $LCP(A[0], A[1]) - 1$

Example

a	b	a	b	d	a	b	c

- $LCP(A_{n(0)}, A_{n(1)}) \geq LCP(A[0], A[1]) - 1$
- $A[0] < A[1] \Rightarrow A_{n(0)} < A_{n(1)}$
- LCP of $A_{n(0)}$ with the next **in order** $A[j]$ is also at least $LCP(A[0], A[1]) - 1$
- Compute $LCP(A_{n(0)}, A[j])$ directly, but don't compare first $LCP(A[0], A[1]) - 1$ characters: they are equal

Algorithm

- Compute $LCP(A[0], A[1])$ directly, save as lcp
- First suffix goes to the next **in the string**
- Second suffix is the next **in the order**
- Compute LCP knowing that first $lcp - 1$ characters are equal, save lcp
- Repeat

LCPOfSuffixes($S, i, j, equal$)

$lcp \leftarrow \max(0, equal)$
`while` $i + lcp < |S|$ `and` $j + lcp < |S|$:
   `if` $S[i + lcp] == S[j + lcp]$:
      $lcp \leftarrow lcp + 1$
   `else`:
      `break`
`return` lcp

InvertSuffixArray(*order*)

pos ← array of size $|order|$
for i from 0 to $|pos| - 1$:
   pos[*order*[i]] ← i
return *pos*

ComputeLCPArray(S, $order$)

```
lcpArray ← array of size |S| − 1
lcp ← 0
posInOrder ← InvertSuffixArray(order)
suffix ← order[0]
for i from 0 to |S| − 1:
  orderIndex ← posInOrder[suffix]
  if orderIndex == |S| − 1:
    lcp ← 0
    suffix ← (suffix + 1) mod |S|
    continue
  nextSuffix ← order[orderIndex + 1]
  lcp ← LCPOfSuffixes(S, suffix, nextSuffix, lcp − 1)
  lcpArray[orderIndex] ← lcp
  suffix ← (suffix + 1) mod |S|
return lcpArray
```

Analysis

Lemma

This algorithm computes LCP array in $O(|S|)$

Proof

- Each comparison increases lcp
- $lcp \leq |S|$
- Each iteration lcp decreases by at most 1
- Number of comparisons is $O(|S|)$ $\square$

Outline

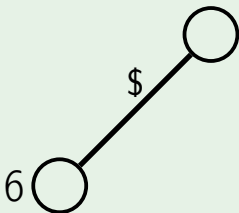## Building suffix tree

$S = ababaa\$$

○

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$

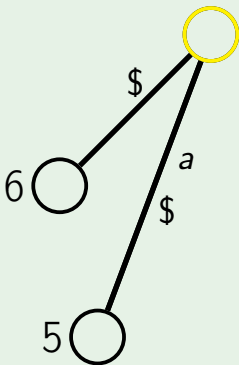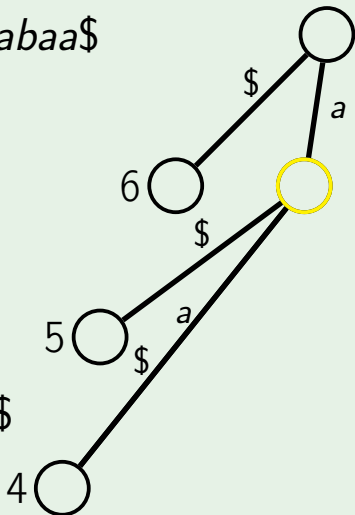# Building suffix tree
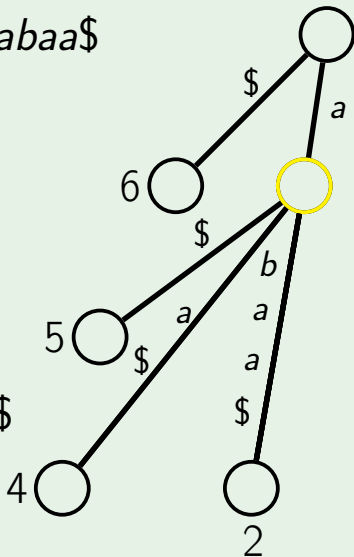
$S = ababaa\$$

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

$S = ababaa\$$

6 $\$$

5 $a\$$

4 $aa\$$

2 *abaa$\$$*

0 $ababaa\$$

3 $baa\$$

1 $babaa\$$

Building suffix tree
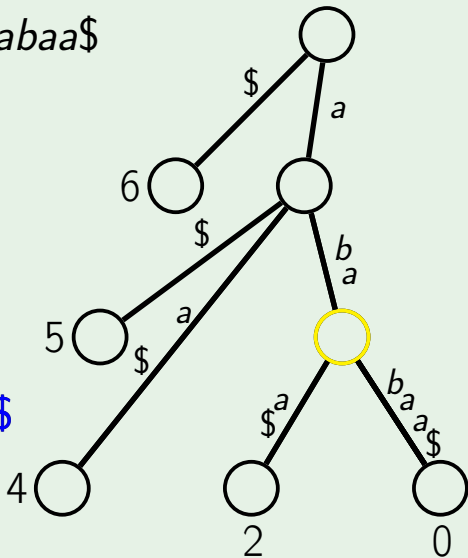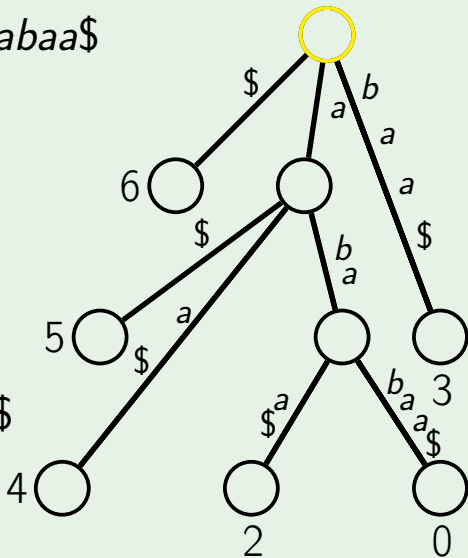
$S = ababaa\$$

6 $\$$

5 $a\$$

4 $aa\$$

2 $abaa\$$

0 $ababaa\$$

3 $baa\$$

1 $babaa\$$

Building suffix tree
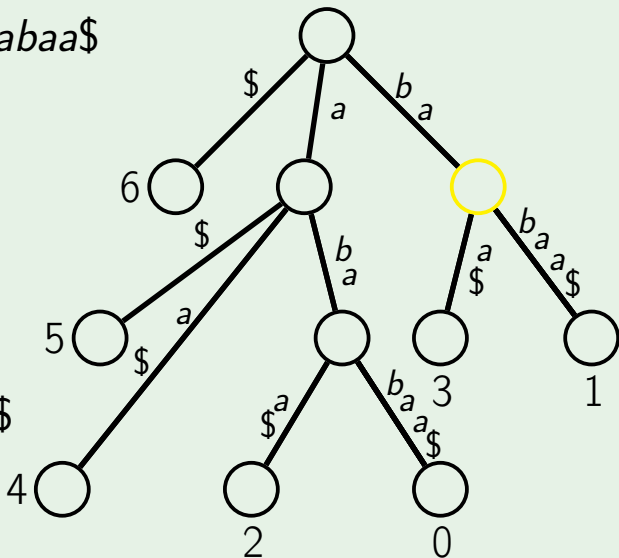
$S = ababaa\$$

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*)

leaf ← new *SuffixTreeNode*(
  children = {},
  parent = *node*,
  stringDepth = $|S| - suffix$,
  edgeStart = *suffix* + *node.stringDepth*,
  edgeEnd = $|S| - 1$)
node.children[*S*[*node.edgeStart*]] ← *leaf*
return *leaf*

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* ← *start* + *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

Conclusion

- Can build suffix tree from suffix array in linear time
- Can build suffix tree from scratch in time $O(|S| \log |S|)$