Algorithmic Challenges: From Suffix Array to Suffix Tree

Michael Levin

Higher School of Economics

Algorithms on Strings
Algorithms and Data Structures at edX

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

$S = \text{ababaa\$}$

0 \$

1 a\$

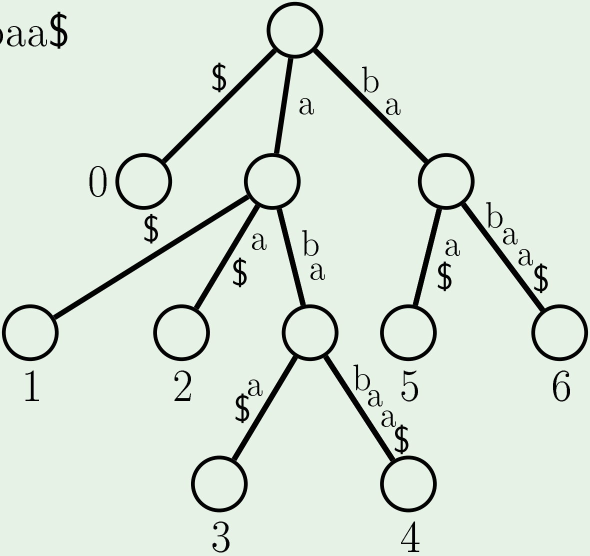
2 aa\$

3 abaa\$

4 ababaa\$

5 baa\$

6 babaa\$



Suffix array and suffix tree

S = ababaa\$

0 \$

1 a\$

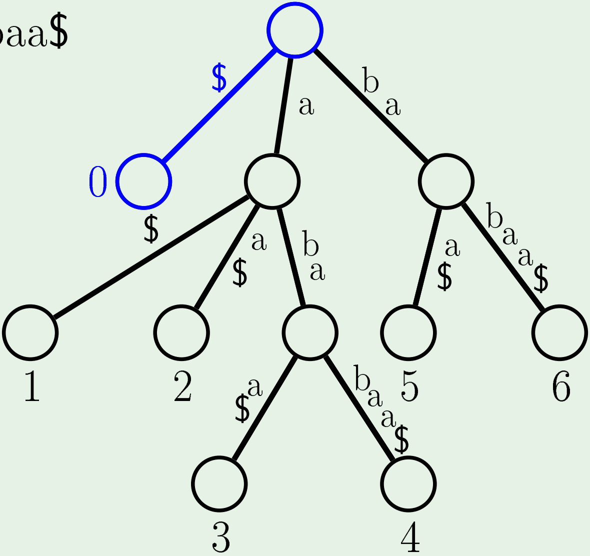
2 aa\$

3 abaa\$

4 ababaa\$

5 baa\$

6 babaa\$



Suffix array and suffix tree

$S = \text{ababaa\$}$

0 \$

1 a\$

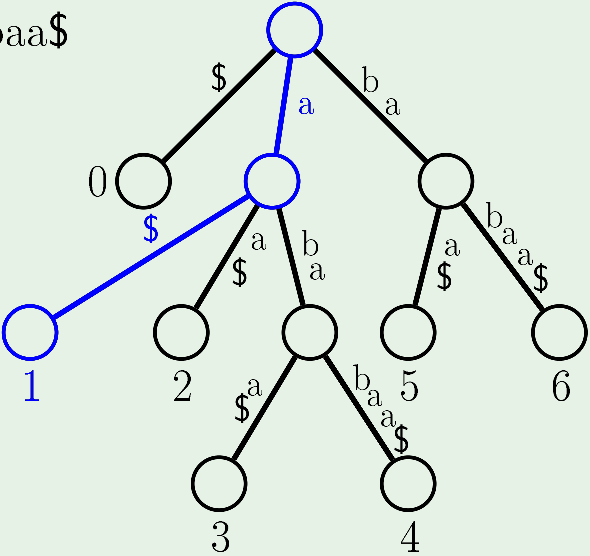
2 aa\$

3 abaa\$

4 ababaa\$

5 baa\$

6 babaa\$



Suffix array and suffix tree

S = ababaa$

0 $

1 a$

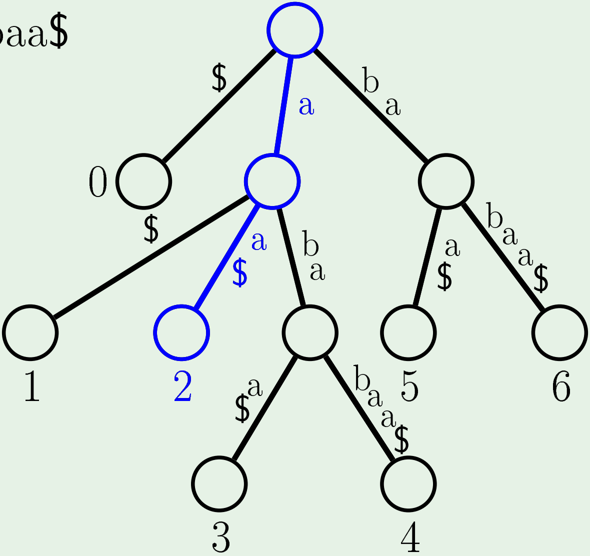
2 aa$

3 abaa$

4 ababaa$

5 baa$

6 babaa$



Suffix array and suffix tree

S = ababaa$

0 $

1 a$

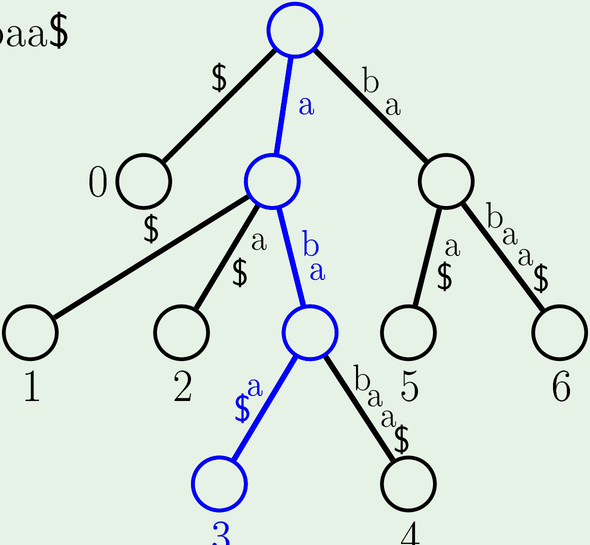
2 aa$

3 abaa$

4 ababaa$

5 baa$

6 babaa$



Suffix array and suffix tree

S = ababaa$

0 $

1 a$

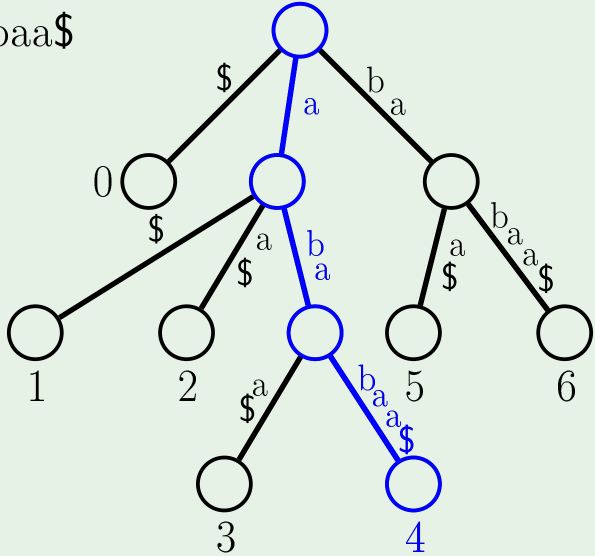
2 aa$

3 abaa$

4 ababaa$

5 baa$

6 babaa$



Suffix array and suffix tree

S = ababaa$

0 $

1 a$

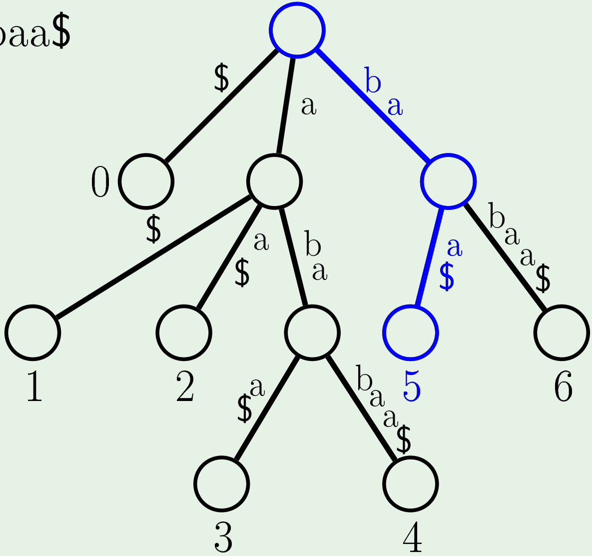
2 aa$

3 abaa$

4 ababaa$

5 baa$

6 babaa$



Suffix array and suffix tree

S = ababaa$

0 $

1 a$

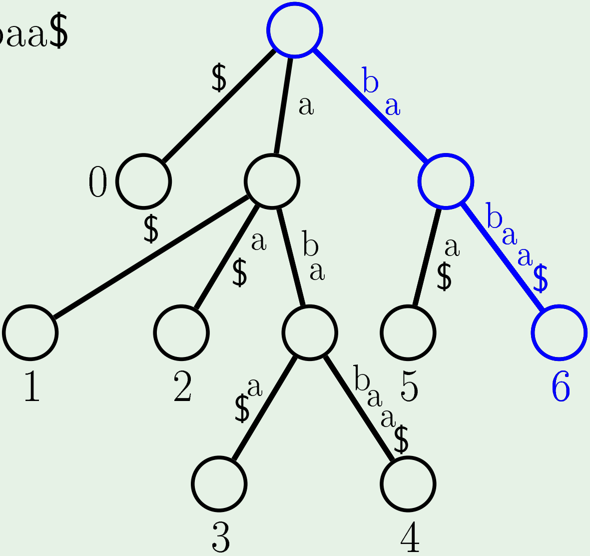
2 aa$

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("ababc", "abc") = 2

LCP("a", "b") = 0

Suffix array, suffix tree and lcp

$S = ababaa\$$

0 $\$$

1 $a\$$

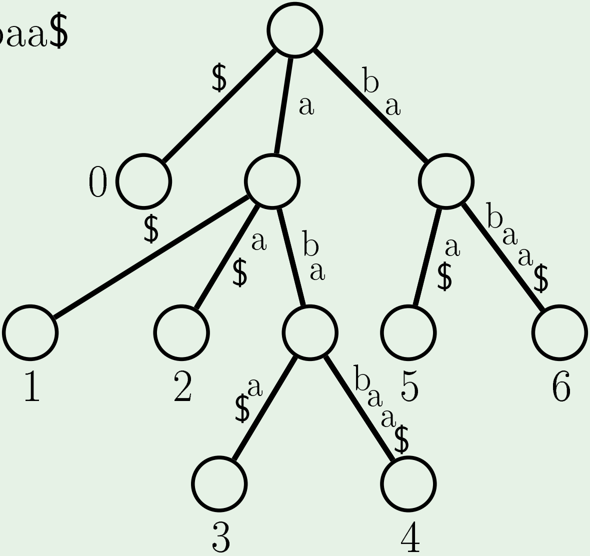
2 $aa\$$

3 $abaa\$$

4 $ababaa\$$

5 $baa\$$

6 $babaa\$$



Suffix array, suffix tree and lcp

S = ababaa$

0 $

1 a$

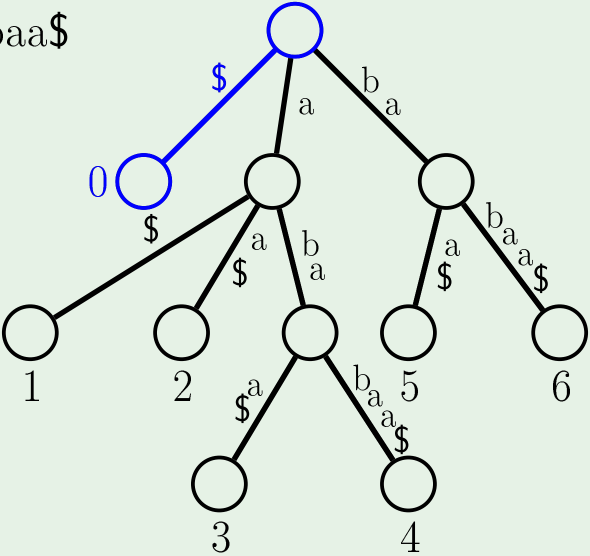
2 aa$

3 abaa$

4 ababaa$

5 baa$

6 babaa$



Suffix array, suffix tree and lcp

S = ababaa$

0 $

1 a$

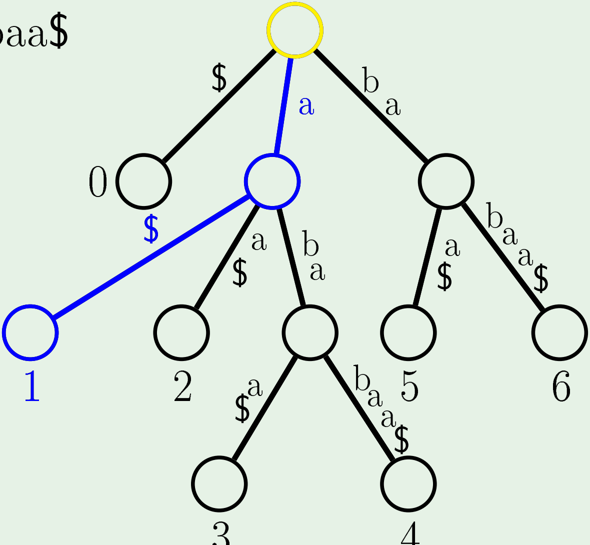
2 aa$

3 abaa$

4 ababaa$

5 baa$

6 babaa$



Suffix array, suffix tree and lcp

S = ababaa$

0 $

1 a$

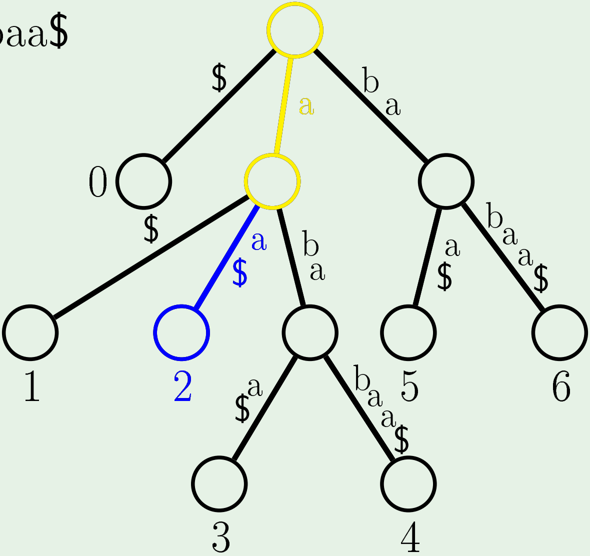
2 aa$

3 abaa$

4 ababaa$

5 baa$

6 babaa$



Suffix array, suffix tree and lcp

S = ababaa$

0 $

1 a$

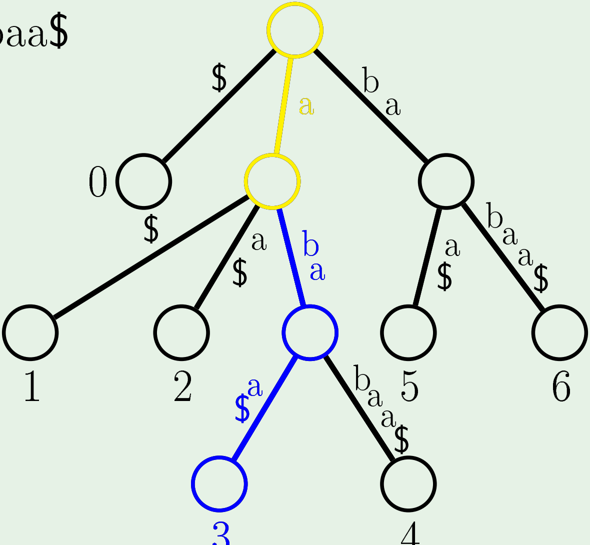
2 aa$

3 abaa$

4 ababaa$

5 baa$

6 babaa$



Suffix array, suffix tree and lcp

S = ababaa$

0 $

1 a$

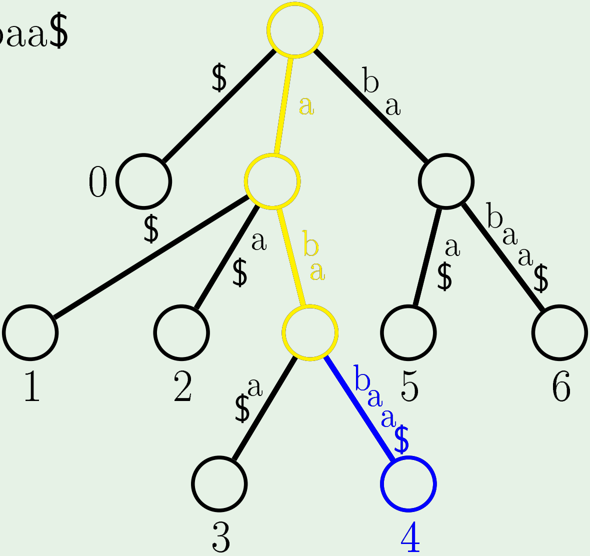
2 aa$

3 abaa$

4 ababaa$

5 baa$

6 babaa$



Suffix array, suffix tree and lcp

S = ababaa\$

0 \$

1 a\$

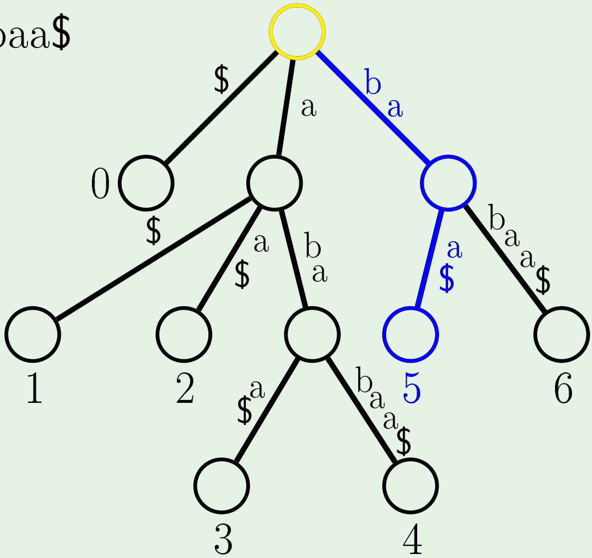
2 aa\$

3 abaa\$

4 ababaa\$

5 baa\$

6 babaa\$



Suffix array, suffix tree and lcp

S = ababaa$

0 $

1 a$

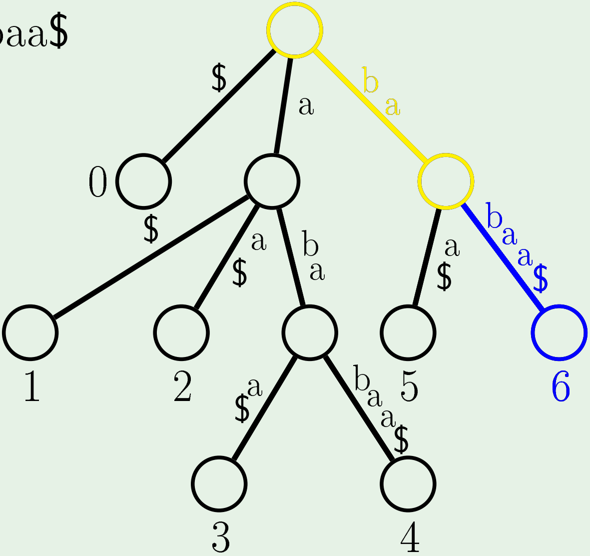
2 aa$

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

$S = \text{ababaa\$}$

0 $

1 a$

2 aa$

3 abaa$

4 ababaa$

5 baa$

6 babaa$

$\text{lcp} = [\ ,\ ,\ ,\ ,\ ,\ ]$

LCP array

$S = \text{ababaa\$}$

0 \$

1 a\$

2 aa\$

3 abaa\$

4 ababaa\$

5 baa\$

6 babaa\$

$\text{lcp} = [\ ,\ ,\ ,\ ,\ ,\ ]$

LCP array

$S = ababaa\$$

0 \$

1 a\$

2 aa\$

3 abaa\$

4 ababaa\$

5 baa\$

6 babaa\$

$\text{lcp} = [0, \ , \ , \ , \ , \ ]$

LCP array

$S = \text{ababaa\$}$

0 \$

1 a\$ $\qquad \text{lcp} = [0, 1, \ , \ , \ , \ ]$

2 aa\$

3 abaa\$

4 ababaa\$

5 baa\$

6 babaa\$

LCP array

$S = \text{ababaa\$}$

0 \$

1 a\$

2 aa\$

3 abaa\$

4 ababaa\$

5 baa\$

6 babaa\$

$\text{lcp} = [0, 1, 1, \ , \ , \ ]$

LCP array

$S = \text{ababaa\$}$

0 $

1 a$

2 aa$

3 abaa$

4 ababaa$

5 baa$

6 babaa$

$\text{lcp} = [0, 1, 1, 3, \ , \ ]$

LCP array

$S = \text{ababaa\$}$

0 \$

1 a\$

2 aa\$

3 abaa\$

4 ababaa\$

5 baa\$

6 babaa\$

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

LCP array

$S = \text{ababaa\$}$

0 $

1 a$

2 aa$

3 abaa$

4 ababaa$

5 baa$

6 babaa$

$$\text{lcp} = [0, 1, 1, 3, 0, 2]$$

LCP array property

Lemma

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

Proof

	$\dots$
i	ababababa
$i+1$	abababc
	$\dots$
j	abbcabab

Proof

	…
i	abababaaba
i + 1	abababc
	…
j	abbcabab

Proof

	...
i	ababababa
i + 1	xxxxxxxxx
	...
j	abbcabab

If $\mathrm{LCP}(A[i], A[j]) > \mathrm{LCP}(A[i], A[i+1])$

Proof

$$\begin{array}{ll} & \ldots \\ i & \text{abababab a} \\ i+1 & \text{xxxxxxxxx} \quad k=1 \\ & \ldots \\ j & \text{abbcabab} \end{array}$$

If $\mathrm{LCP}(A[i], A[j]) > \mathrm{LCP}(A[i], A[i+1])$

Consider $k = \mathrm{LCP}(A[i], A[i+1])$

Proof

	...
i	ababababa
i + 1	a_ $\quad\quad k = 1$
	...
j	abbcabab

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

Proof

$\ldots$

$i \quad ababababa$

$i+1 \quad axxxxxxxx \quad k = 1$

$\ldots$

$j \quad abbcabab$

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

Proof

$\dots$

i ababababa

i + 1 acxxxxxxx $k = 1$

$\dots \bigvee$

j abbcabab

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

Proof

$$\begin{array}{ll} & \ldots \\ i & \text{abababab}a \\ & \quad\;\; \vee \\ i+1 & \text{aaxxxxxx} \quad k=1 \\ & \ldots \\ j & \text{abbcabab} \end{array}$$

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

Computing LCP array

- For each i, compute $\mathrm{LCP}(A[i], A[i+1])$ 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 $h - 1$.

$$S = \text{abracadabra\$}$$

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

$$S = \text{abracadabra\$}$$

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

$$S = \text{abracadabra\$}$$

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

$$S = \text{abracadabra\$}$$

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

Idea

- Start by computing $\mathrm{LCP}(A[0], A[1])$ directly
- Instead of computing to $\mathrm{LCP}(A[1], A[2])$, move $A[0]$ one position to the right in the string, get some $A[k]$ and compute $\mathrm{LCP}(A[k], A[k+1])$
- 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] = $ "ababdabc", $A[1] = $ "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] = $ "ababdabc", $A[1] = $ "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

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

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