computing longest (common) Lyndon subsequences

Hideo Bannai[1], Tomohiro I[2], Tomasz Kociumaka[3],
Dominik Köppl[1], Simon J. Puglisi[4]

[1] M&D Data Science Center, Tokyo Medical and Dental University
[2] Department of Artificial Intelligence, Kyushu Institute of Technology
[3] University of California, Berkeley
[4] Department of Computer Science, Helsinki University

Lyndon

- a string is called *Lyndon* if it is lexicographically smaller ($\prec$) than all its proper suffixes

example:

- a, ab, aab
- not Lyndon:
  - aba (a $\prec$ aba)
  - abab (ab $\prec$ abab)

input

- text T of length n
- $T[i]$: character $\in \Sigma$
- Σ: alphabet, $\sigma := |\Sigma|$ alphabet size
- $\sigma = n^{O(1)}$, i.e., Σ is integer alphabet

Lyndon factorization [Chen+ '58]

- factorization $T = T_1 \ldots T_t$ with $T_x \succeq T_{x+1}$ $\forall x$
- T_x is Lyndon; called *Lyndon factor*
- factorization uniquely defined
- linear time [Duval' 88]

Lyndon factorization [Chen+ '58]

- factorization $T = T_1 \ldots T_t$ with $T_x \geq T_{x+1}$ $\forall$ x
- T_x is Lyndon; called *Lyndon factor*
- factorization uniquely defined
- linear time [Duval' 88]

example:
T_1 = bcc
T_2 = adb
T_3 = accbcd

	1	2	3	4	5	6	7	8	9	10	11	12
T =	b	c	c	a	d	b	a	c	c	b	c	d

longest Lyndon substring S

- answer S is longest Lyndon factor

	1	2	3	4	5	6	7	8	9	10	11	12
$T =$	b	c	c	a	d	b	a	c	c	b	c	d

longest Lyndon substring S

- answer S is longest Lyndon factor
- assume $S = T_x[i..|T_x|]\ T_{x+1} \cdots\ T_y T_{y+1}[1..j]$
- then $T_{y+1}[1..j] \preceq T_x \preceq T_x[i..|T_x|]$ by the Lyndon factorization $\Rightarrow$ S not Lyndon

$T_x[i..|T_x|]$ $T_{y+1}[1..j]$

	1	2	3	4	5	6	7	8	9	10	11	12
$T =$	b	c	c	a	d	b	a	c	c	b	c	d

substring → subsequence

- can find longest Lyndon substring in O(n) time

for longest Lyndon *subsequence*:

- usually: to find longest *** subsequence: use dynamic programming (DP)
- can we use DP / greedy approach here?

DP?

compute solution for $T[1..i{+}1]$ from $T[1..i]$?

	1	2	3	4	5	6	7	8	9	10	11	12
T =	b	c	c	a	d	b	a	c	c	b	c	d

longest Lyndon subsequence of
$T[1..9]$: bccdcc
$T[1..11]$: abaccbc
$T[1..12]$: bbcbccbcd

looks difficult!

our results

compute longest Lyndon subsequence in

- O(n^3) time and
- O(n) space

not in this talk but in the paper:

- *online*: O($n^3\sigma$) time, O($n^3\sigma$) space
- longest *common*: O($n^4\sigma$) time, O(n^3) space

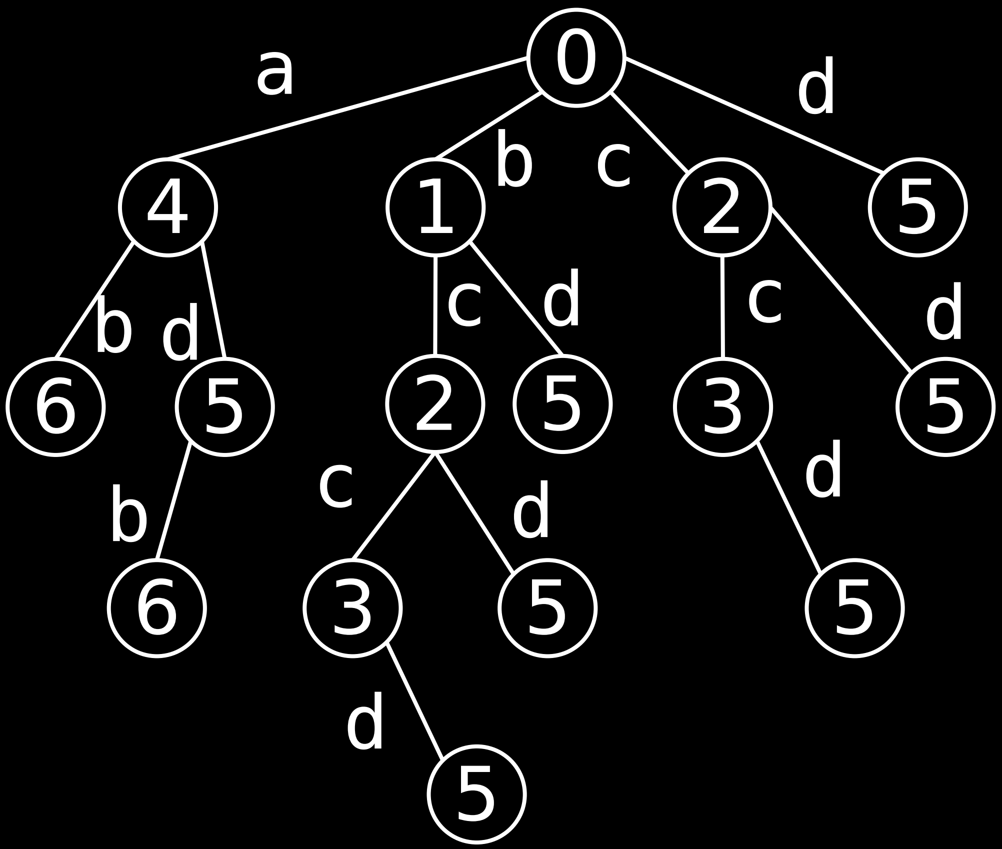
but how? (if not greedy / DP)

trie of Lyndon subsequences

T = bccadba

properties

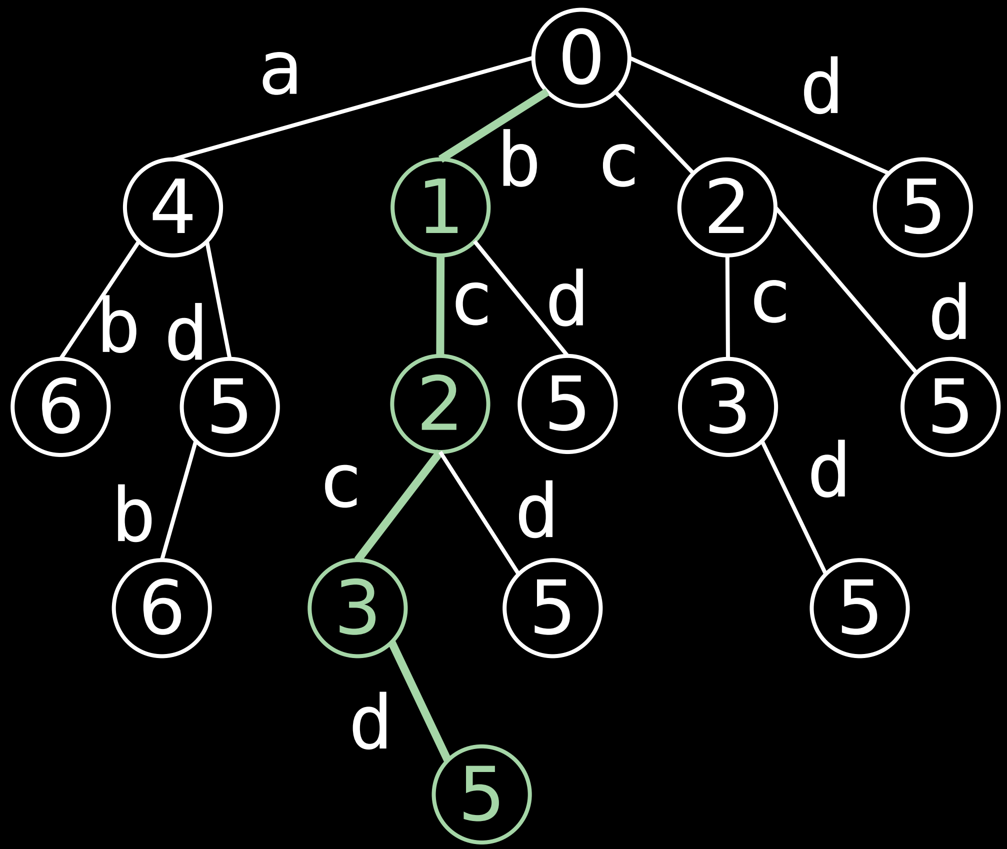
- label $c(v)$ of node v = end of leftmost occurrence of string read from root to v
- $c(\text{parent}(v)) < c(v)$



$$\begin{array}{ccccccccc} & & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ T & = & \mathrm{b} & \mathrm{c} & \mathrm{c} & \mathrm{a} & \mathrm{d} & \mathrm{b} & \mathrm{a} \end{array}$$

trie of Lyndon subsequences

- leaves are Lyndon subsequences
- deepest leaf is longest Lyndon subsequence!



1	2	3	4	5	6	7

$T = $ b c c a d b a

enumerate all?

- idea: build trie on all Lyndon subsequences and take deepest leaf
- # distinct Lyndon subsequences = $O(2^n)$
  e.g., for $T = 1 \cdots n$ this number is $\Theta(2^n)$
- exact number still unknown!
  (only expected number [Hirakawa+ '21])

pre-Lyndon / immature

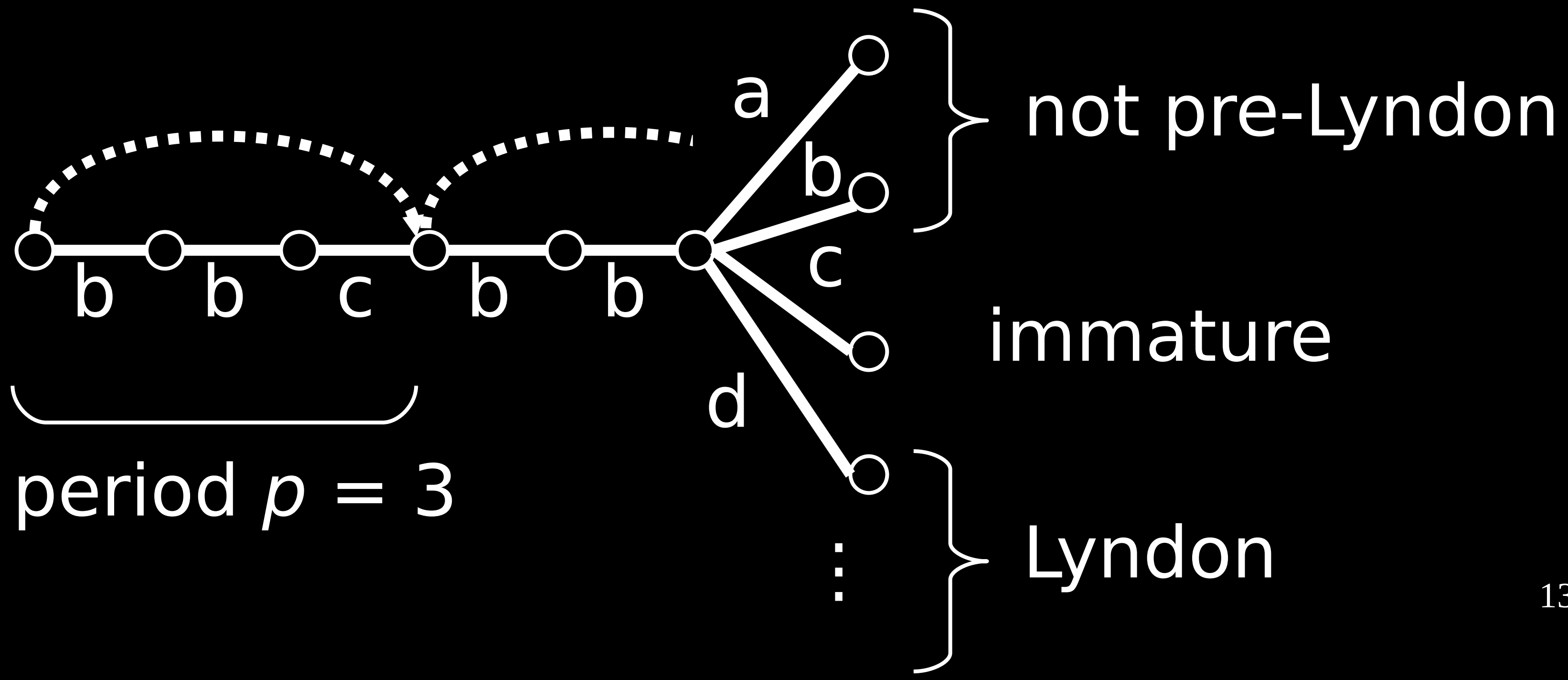
- so: build trie on-the-fly
- but which nodes lead to Lyndon subsequences?

for that, we need some definitions:

- a string is called *pre-Lyndon* if it is a prefix of a Lyndon string
- a pre-Lyndon string is called *immature* if it is not Lyndon (e.g., it is a proper prefix of a Lyndon string)

key observation

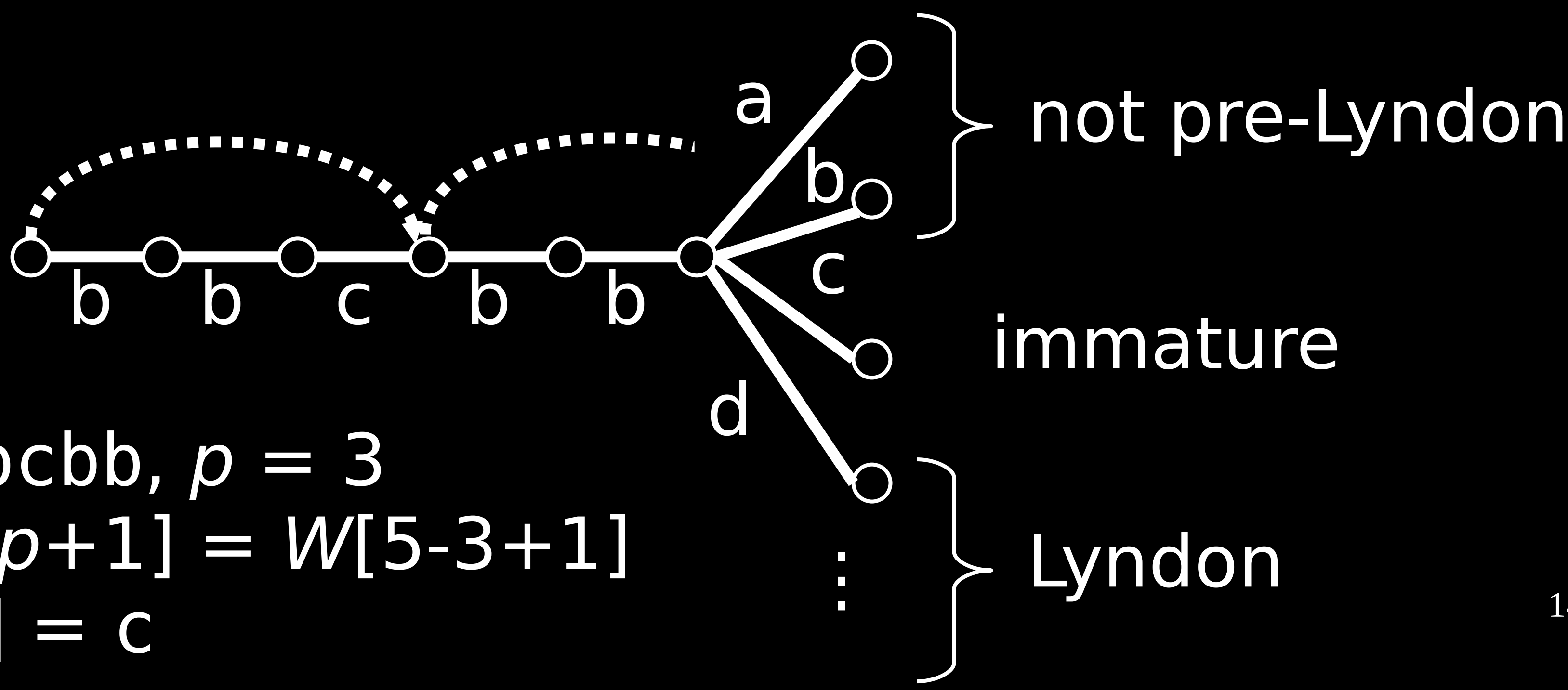
- a node has at most one child that is immature
- determined by its (minimal) period *p*



key observation

W pre-Lyndon, c character, p period of W

- $c = W[|W|\text{-}p+1] \Leftrightarrow Wc$ immature
- $c > W[|W|\text{-}p+1] \Leftrightarrow Wc$ Lyndon



W = bbcbb, $p = 3$
$W[|W|\text{-}p+1] = W[5\text{-}3+1]$
$= W[3] =$ c

key observation

W pre-Lyndon, c character, p period of W

- $c = W[|W|-p+1] \Leftrightarrow Wc$ immature
- $c > W[|W|-p+1] \Leftrightarrow Wc$ Lyndon

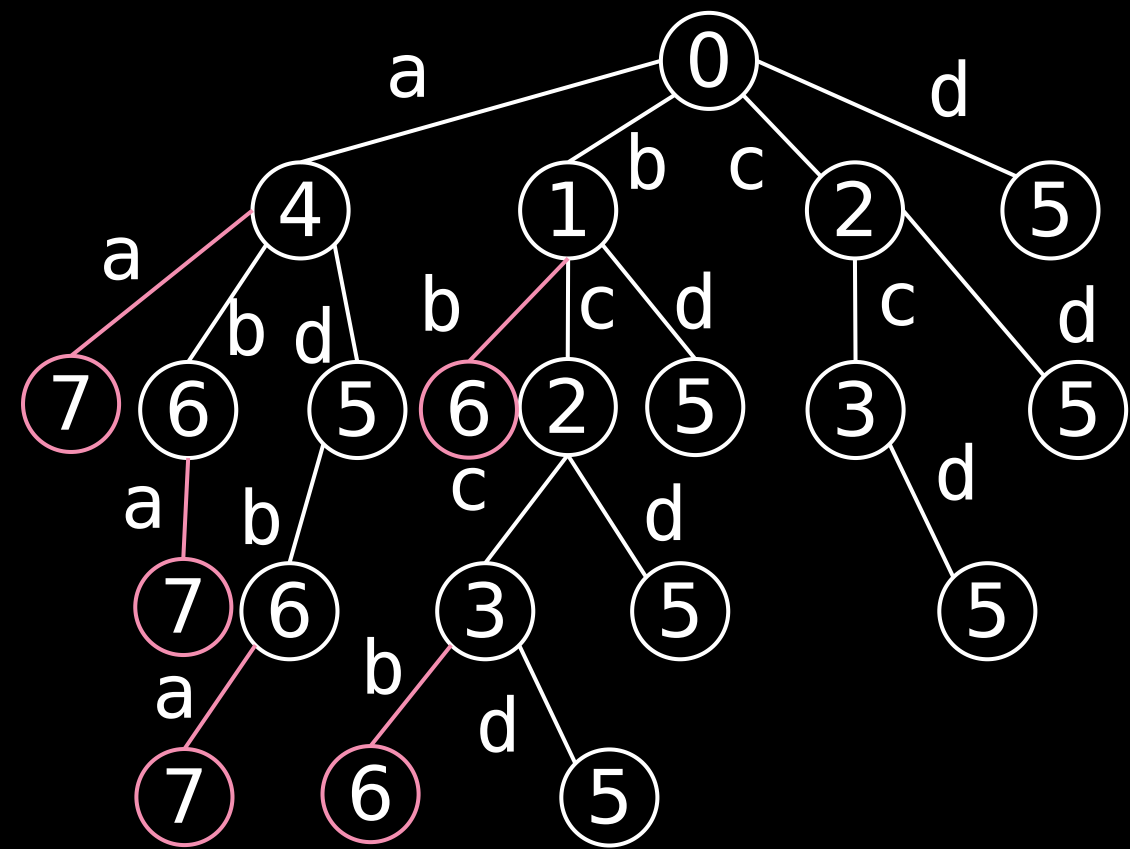
p is also
the period of Wc

more properties

- W Lyndon $\Rightarrow p = |W|$
  ($\Rightarrow W[|W|-p+1] = W[1]$)
- $W[1..p]$ is Lyndon
  (otherwise W is not pre-Lyndon)

trie of pre-Lyndon subsequences

- leaves no longer necessarily Lyndon (pink)
- how can we represent this trie more efficiently?

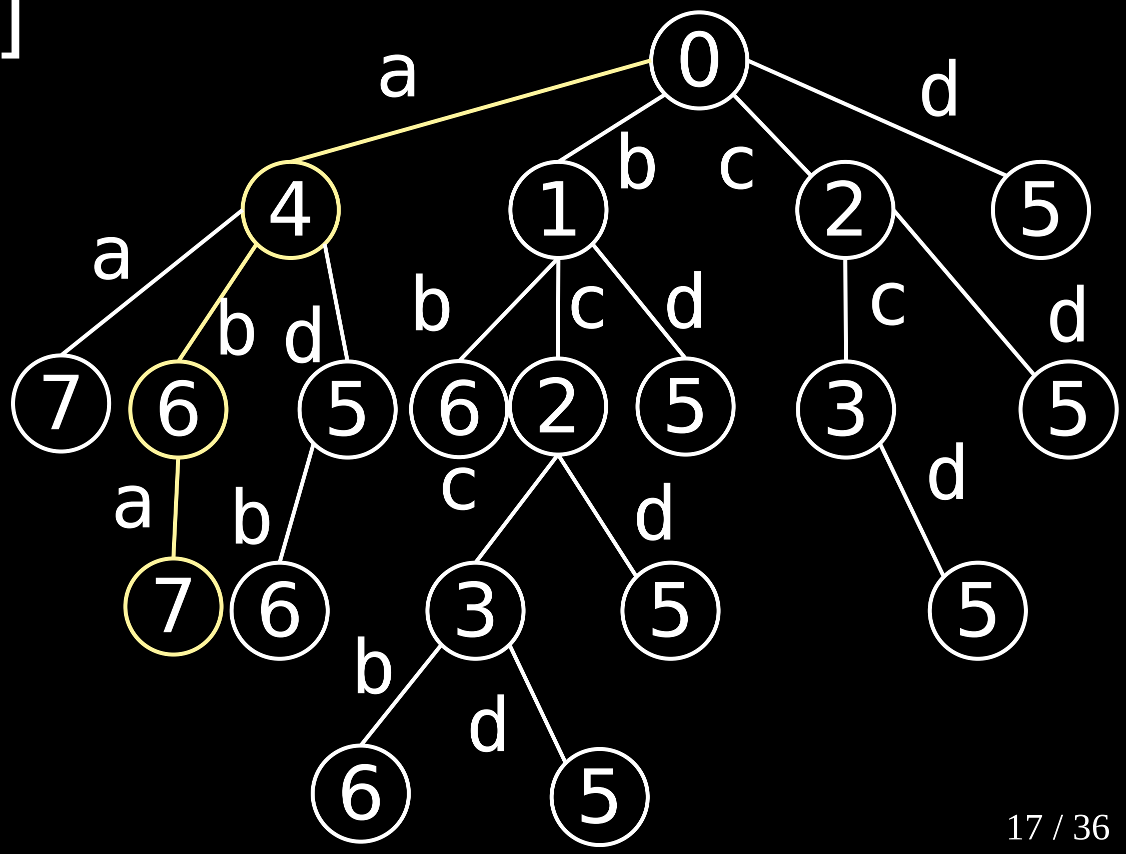


1	2	3	4	5	6	7

$$T = \mathrm{b\ c\ c\ a\ d\ b\ a}$$

stack

- implicitly represent trie by a stack S simulating a depth-first search (DFS)
- $T[S[1]] \cdots T[S[|S|]]$ is pre-Lyndon subsequence
- $S = [4,6,7]$

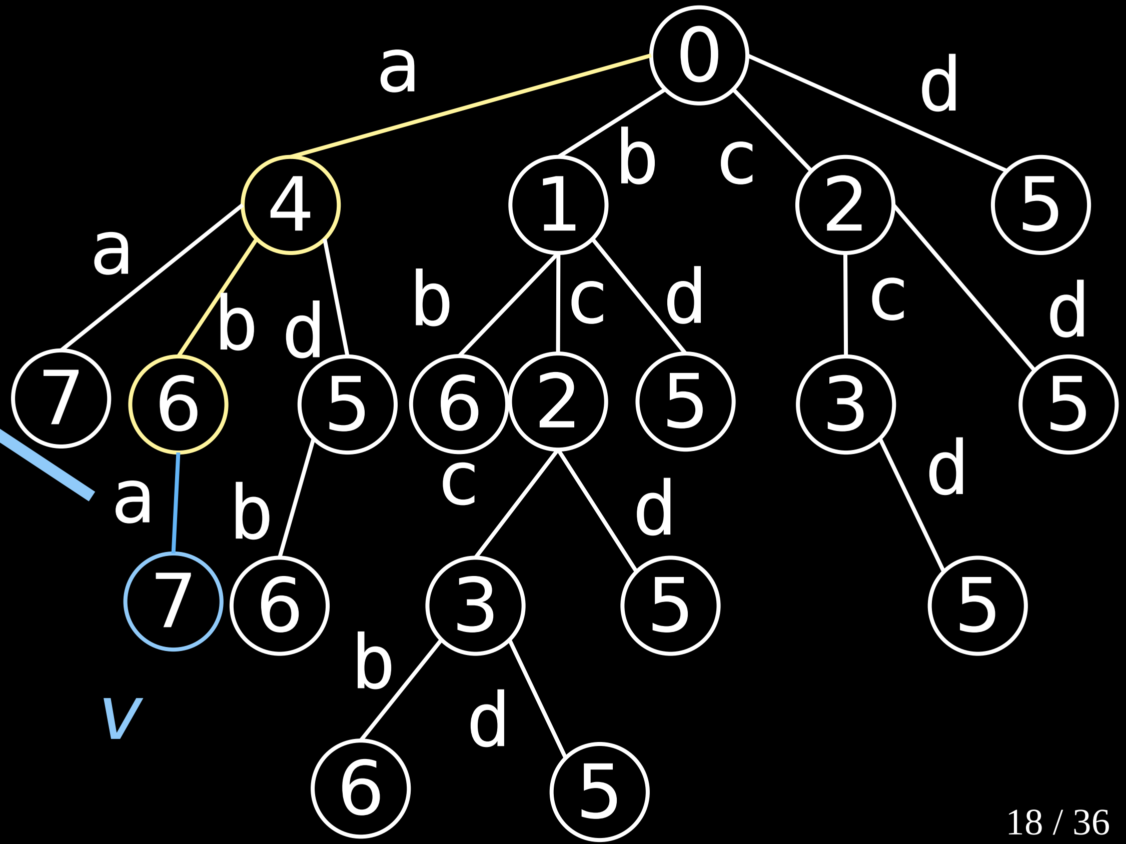


stack + periods

- augment S with periods: S = [(4,1),(6,2)]
- when visiting v:
- last period p on stack is 2

(node, period)

compare
$T[S[|S|\text{-}p+1]] = \text{a}$
with in-going
edge of v
⇒ immature



time?

⇒ space is O(n), but time can still be O(2^n)

- idea: explore in lexicographic order, but "prune" a node if its subtree cannot lead to a longer Lyndon subsequence
- if we visit a node u whose subsequence is lexicographically larger than a longer subsequence of an already visited node v, we prune u
- why can we do that?

lemma

- V Lyndon
- $U, W \in \Sigma^*$

such that

$$\begin{cases} UW \text{ Lyndon} \\ V \prec U \\ |V| \geq |U| \end{cases}$$

$\Rightarrow VW$ Lyndon and $VW \prec UW$

lemma

- V Lyndon
- $U, W \in \Sigma^*$

such that

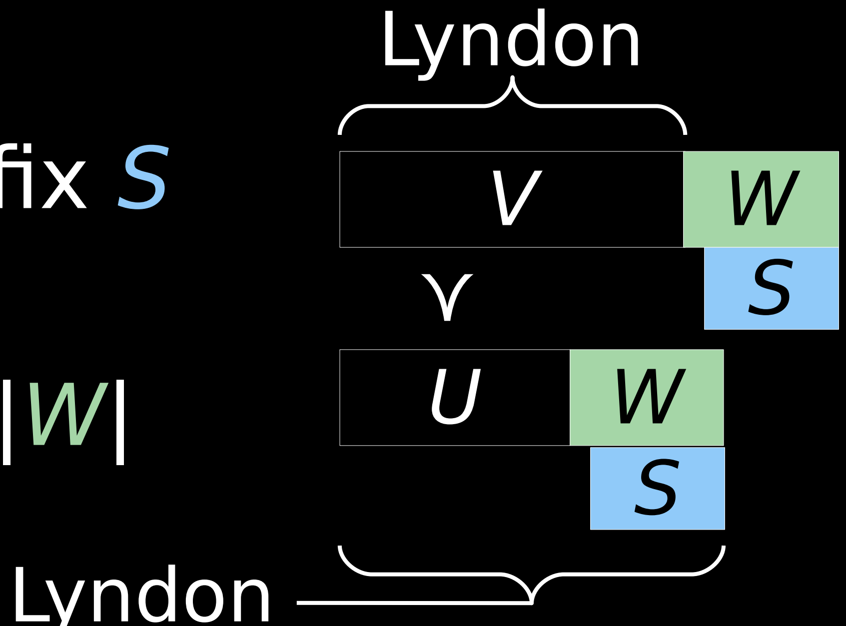
- UW Lyndon
- $V \prec U$
- $|V| \geq |U|$

$\Rightarrow VW$ Lyndon and $VW \prec UW$

proof.

take suffix S of VW

1. $|S| \leq |W|$



$V \lhd U :\Leftrightarrow V \prec U$ and V not prefix of U

lemma

- V Lyndon
- $U, W \in \Sigma^*$

such that

- UW Lyndon
- $V \prec U$
- $|V| \geq |U|$

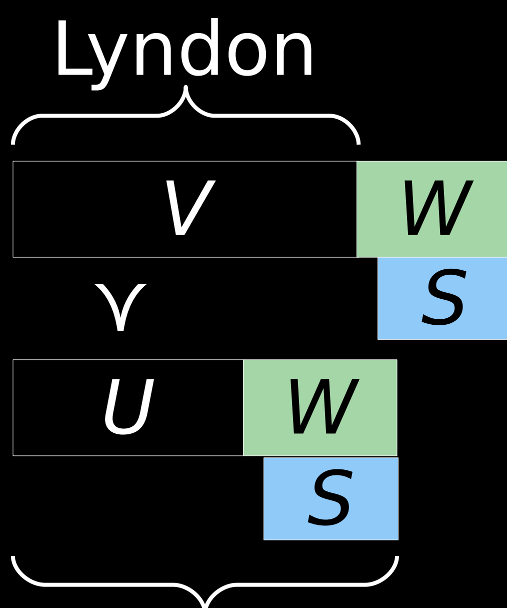
$\Rightarrow VW$ Lyndon and $VW \prec UW$

$V \lhd U \Rightarrow VW \prec UW$

proof.

take suffix S of VW

1. $|S| \leq |W|$



Lyndon

UW Lyndon $\Rightarrow$
$S \succ UW \succ VW$

$V \lhd U :\Leftrightarrow V \prec U$ and V not prefix of U

lemma

- V Lyndon
- $U, W \in \Sigma^*$

such that

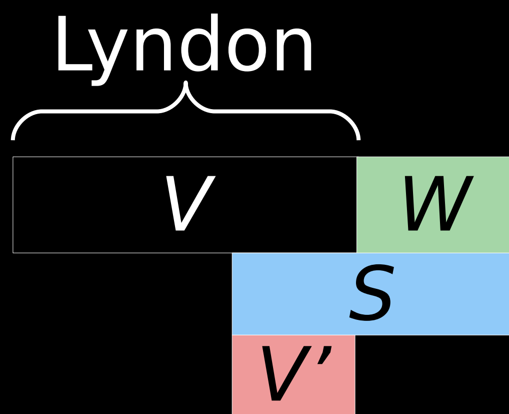
- UW Lyndon
- $V \prec U$
- $|V| \geq |U|$

$\Rightarrow VW$ Lyndon and $VW \prec UW$

proof.

take suffix S of VW

2. $|S| > |W|$



$V \lhd U :\Leftrightarrow V \prec U$ and V not prefix of U

lemma

- V Lyndon
- $U, W \in \Sigma^*$

such that

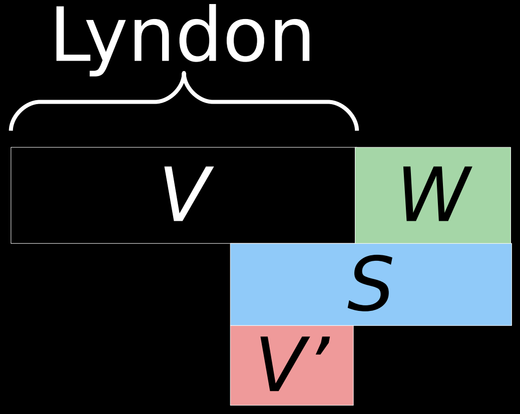
- UW Lyndon
- $V \prec U$
- $|V| \geq |U|$

$\Rightarrow VW$ Lyndon and $VW \prec UW$

proof.

take suffix S of VW

2. $|S| > |W|$



$V \lhd V'$ since V is Lyndon $\Rightarrow VW \prec V'$

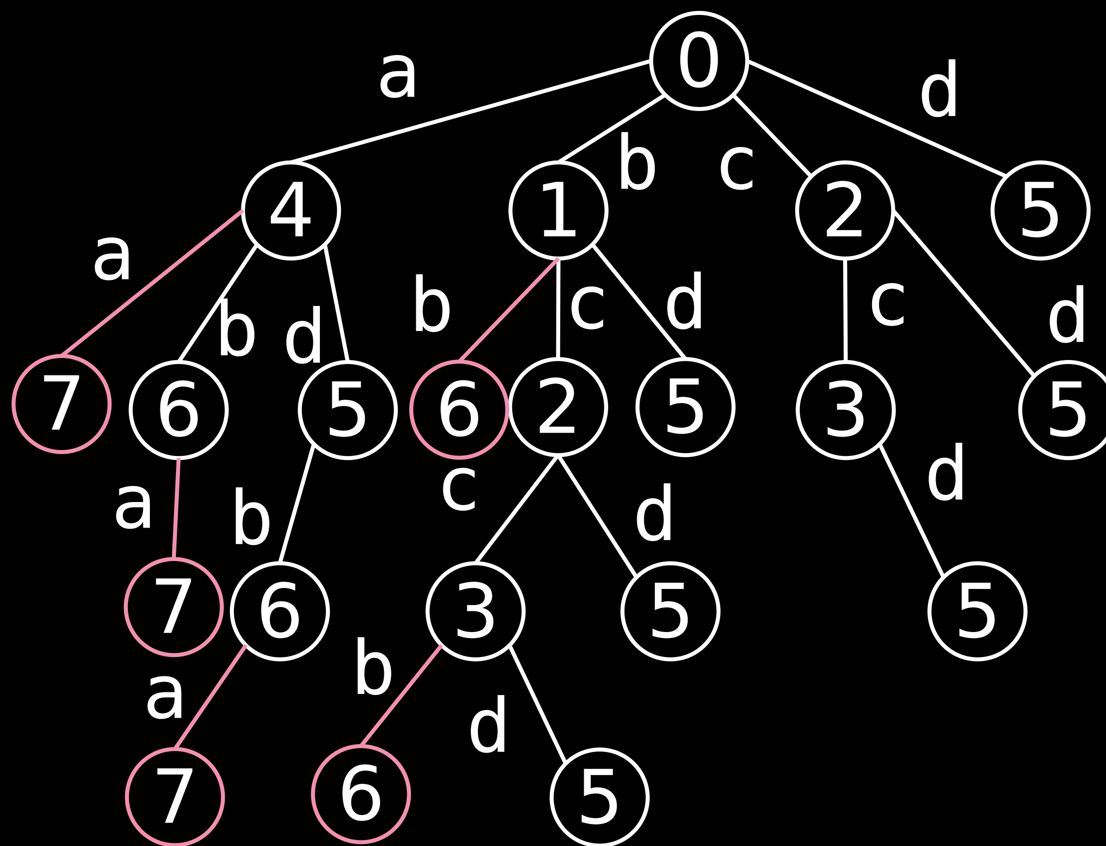
$\Rightarrow S \succ V' \succ VW$

leverage lemma

- maintain dynamic array $L[1..n]$:
  $L[\ell]$: smallest text position i for which we know that $\exists$ Lyndon subsequence W in $T[1..i]$ with $|W| = \ell$
- $L[\ell] = \infty \ \forall\ \ell$ at start
- since our DFS is in lexicographic order, when visiting a node u at depth ℓ, we prune u if $L[\ell] < c(u)$ (the label of u)

algorithmic execution

initial state: $S = \varnothing$

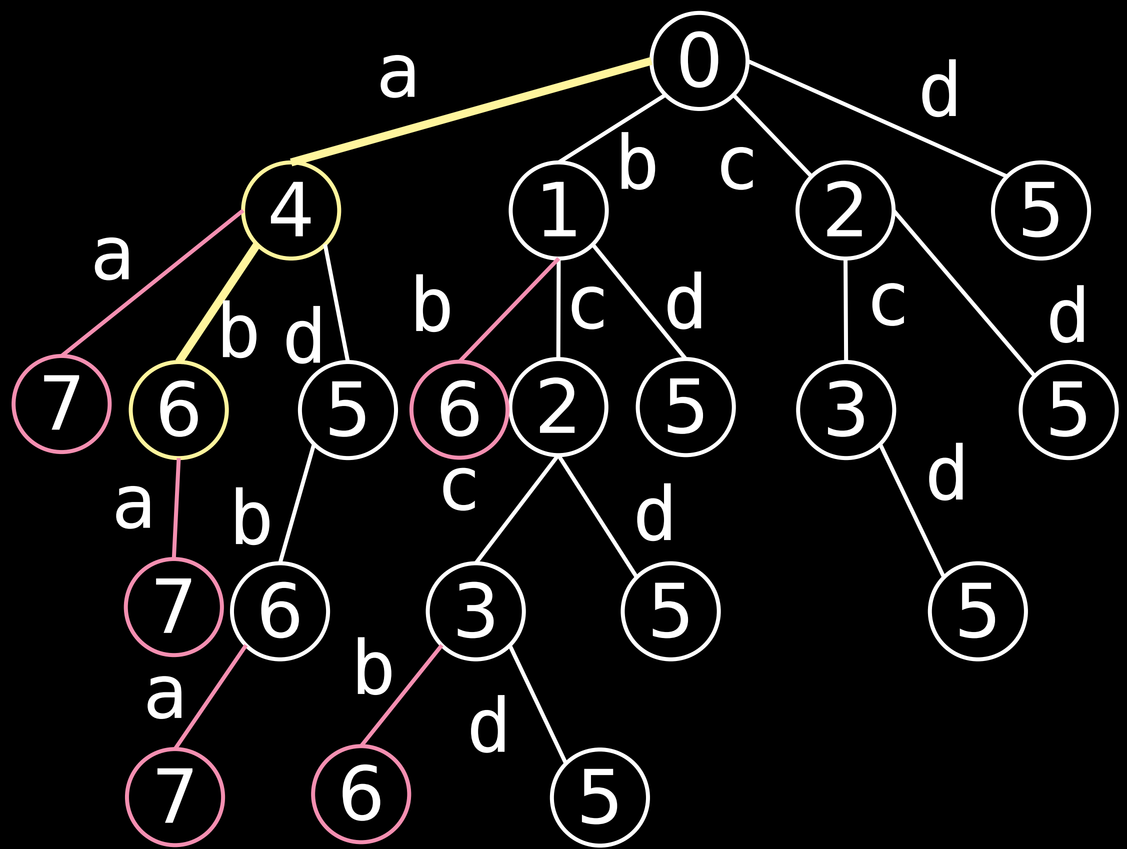


	1	2	3	4	5	6	7
$L =$	∞	∞	∞	∞	∞	∞	∞

algorithmic execution

S = (4,1) (6,2)
Lyndon subsequences:

- a
- ab



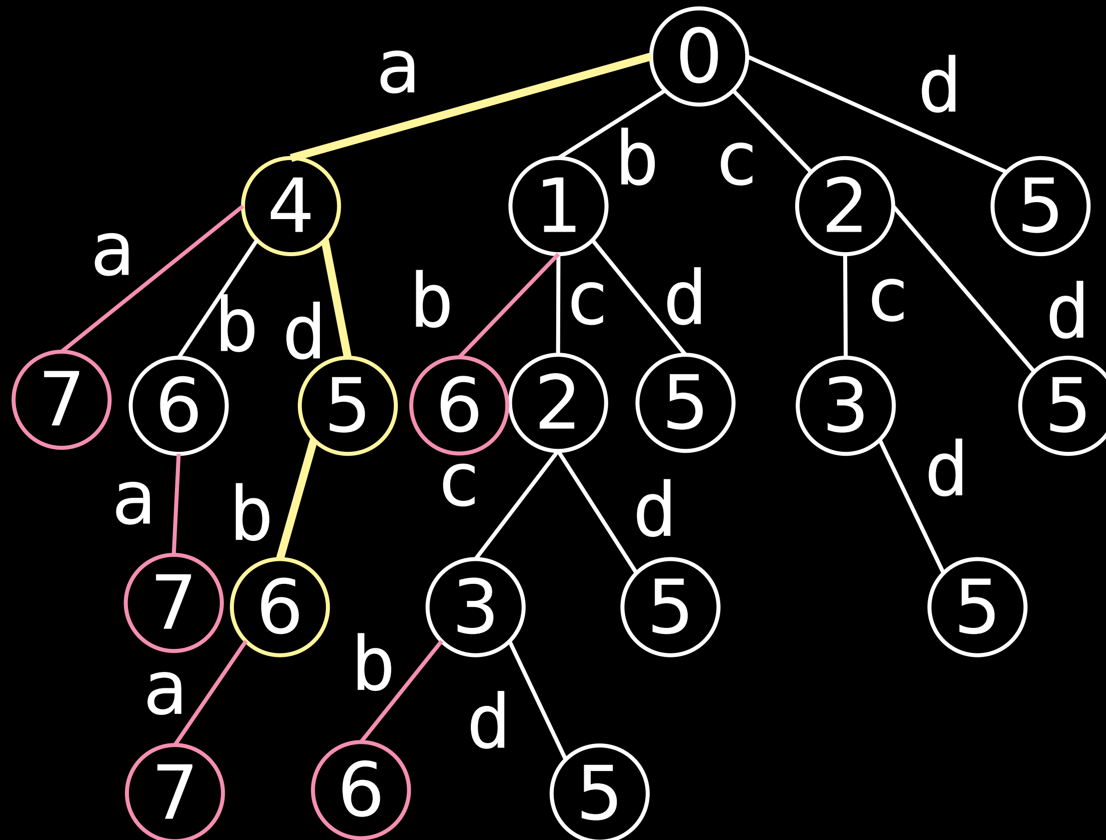
	1	2	3	4	5	6	7
L =	4	6	∞	∞	∞	∞	∞

algorithmic execution

S = (4,1) (5,2) (6,3)

Lyndon subsequences:

- ad
- adb



	1	2	3	4	5	6	7
L =	4	5	6	∞	∞	∞	∞

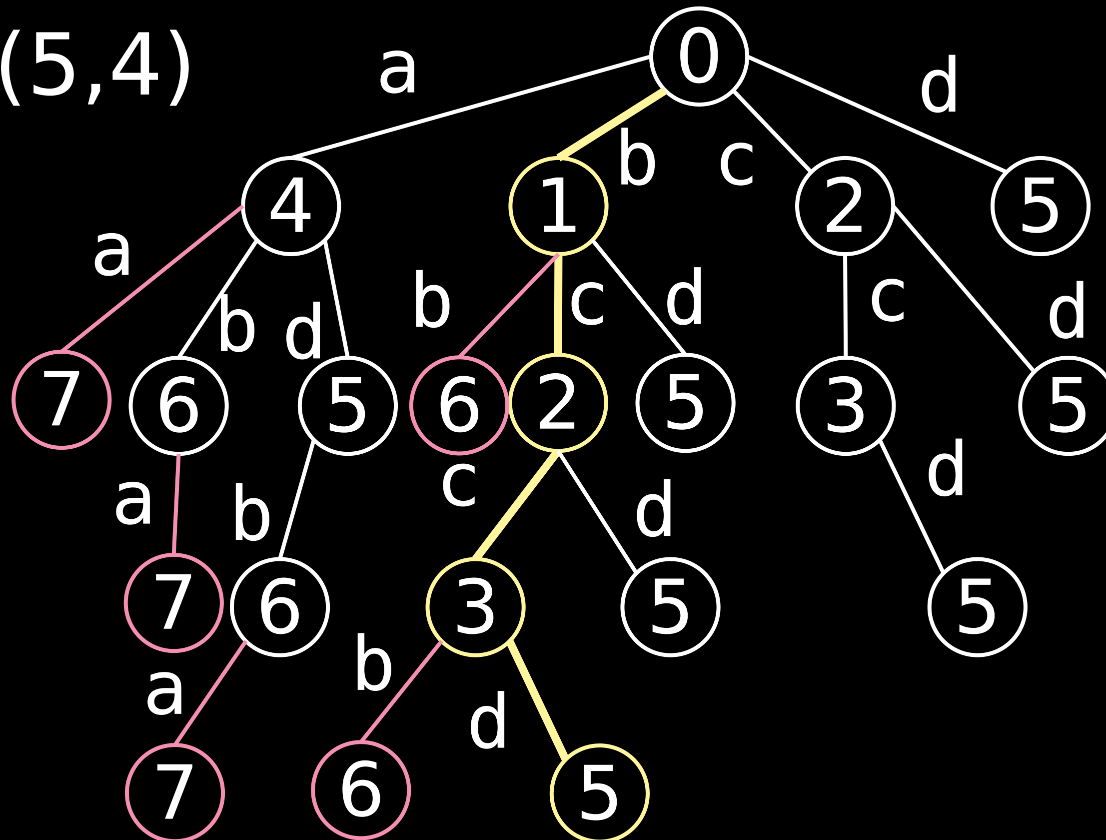
since ad ends earlier than ab, we can update $L[2]$

algorithmic execution

S = (1,1) (2,2) (3,3) (5,4)

find subsequences

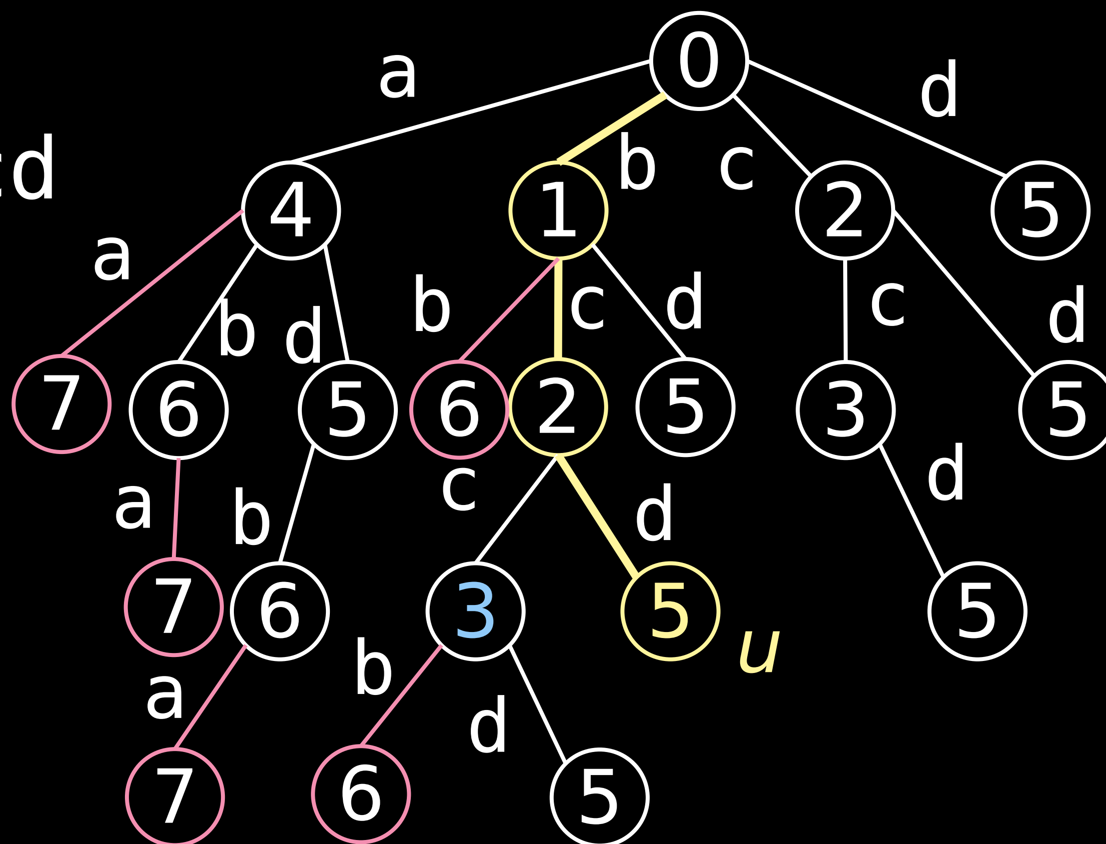
- b
- bc
- bcc
- bccd



	1	2	3	4	5	6	7
L =	1	2	3	5	∞	∞	∞

skip bcd

S = (1,1) (2,2) (5,3)
find subsequence bcd
at node u but
$L[|\text{bcd}|] = 3 < 5$
⇒ prune u



	1	2	3	4	5	6	7
L =	1	2	3	5	∞	∞	∞

1	2	3	4	5	6	7
b	c	c	a	d	b	a

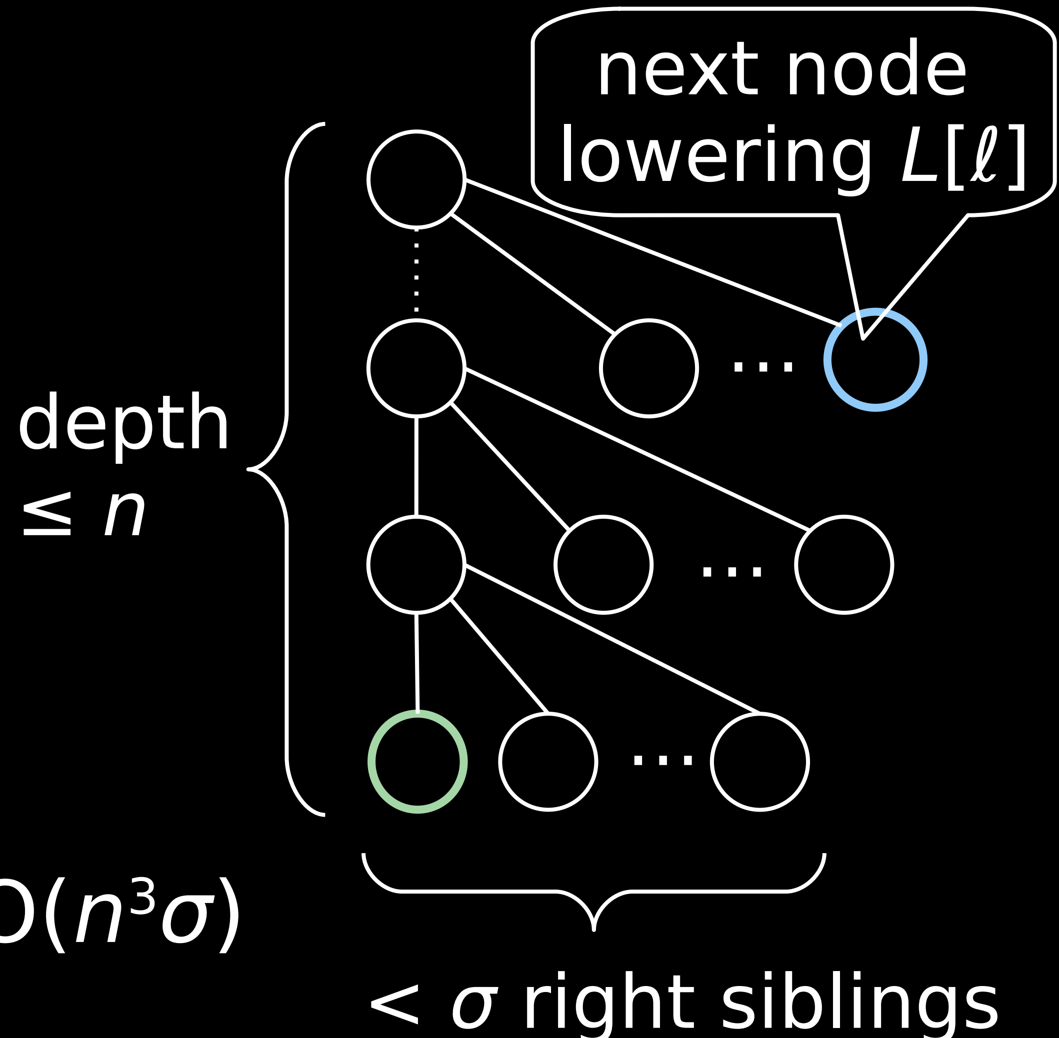
observation

- either
  - decrement $L[\ell]$,
  - prune subtree, or
  - visit immature child (immature is not Lyndon and thus cannot lower L)
- $L[\ell] \in [1..n] \Rightarrow$ all values of L can be decremented $O(n^2)$ times

node visits

node visits between two $L[\ell]$ decrements is $O(n\sigma)$

- $\ell \leq n$
- $\leq \sigma$ siblings that are pre-Lyndon



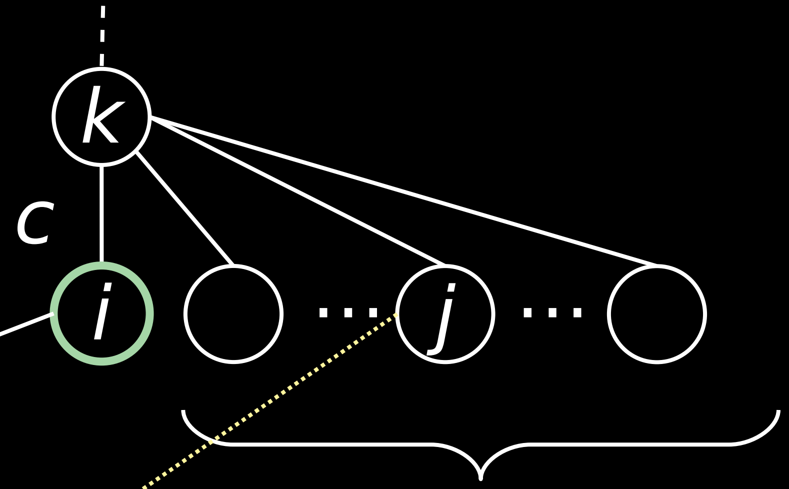
$\Rightarrow$ total node visits: $O(n^3\sigma)$

total time

parent → child traversal in O(1) time:

- let $T[i]$ store an array $F_i[1..\sigma]$ such that $F_i[c]$ is the next occurrence of c in T, i.e., $F_i[c] = \min \{ j \in [i..n] \mid T[j] = c \}$
- total time: $O(n^3 \sigma)$, space: $O(n\sigma)$
- can shave off σ in time&space by using RMQ + wavelet tree instead of $F_i[c]$

skip nodes



suppose node has lowered $L[\ell]$

can lower $L[\ell]$ only if $j < i = L[\ell]$

⇒ consider only those with a label in $[k{+}1..L[\ell]{-}1]$

data structures

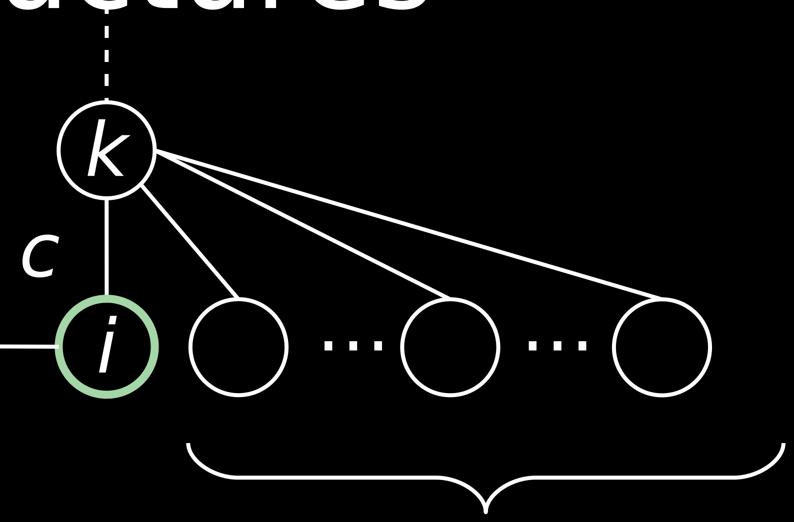
- build range maximum query (RMQ) data structure and wavelet tree on T

given an interval J and a character c, query:

- max $\{ T[j] \mid j \in J \}$ in O(1) time (RMQ)
- min $\{ T[j] \mid j \in J \wedge T[j] > c \}$ in O($\lg n$) time (range successor query)

leverage data structures

suppose node has lowered $L[\ell] \leftarrow i$ with a subsequence Wc, p : period of W



RMQ $[k+1..L[\ell]-1] > W[|W|-p+1]$?

YES

can lower $L[\ell] \Rightarrow$ use wavelet tree to find next node

NO

RMQ $[k+1..L[\ell]-1] = W[|W|-p+1]$?

YES

found the immature child $\Rightarrow$ descend

NO

skip all siblings

time complexity

- $O(n^2)$ nodes lower L
- each such node
  - found by wavelet tree query: $O(\lg n)$ time
  - can have $O(n)$ immature ancestors

$\Rightarrow O(n^3)$ immature nodes,
each found in $O(1)$ time by RMQ

- total time: $O(n^2 \lg \sigma + n^3) = O(n^3)$ time
- data structures use $O(n)$ space

summary

computing longest Lyndon subsequence

- $O(n^3)$ time and $O(n)$ space
- simulate DFS on trie of all pre-Lyndon subsequences with a stack
- use wavelet tree + RMQ data structure to
  - speed up trie navigation
  - skip nodes that cannot lower L
- output is actually the lexicographically smallest among all longest ones