computing longest (common) Lyndon subsequences

Hideo Bannai\textsuperscript{1}, Tomohiro I\textsuperscript{2}, Tomasz Kociumaka\textsuperscript{3}, Dominik Köppl\textsuperscript{1}, Simon J. Puglisi\textsuperscript{4}

\textsuperscript{1} M&D Data Science Center, Tokyo Medical and Dental University
\textsuperscript{2} Department of Artificial Intelligence, Kyushu Institute of Technology
\textsuperscript{3} University of California, Berkeley
\textsuperscript{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 $\in \Sigma$
- $\Sigma$: alphabet, $\sigma := |\Sigma|$ alphabet size
- $\sigma = n^{O(1)}$, i.e., $\Sigma$ is integer alphabet
Lyndon factorization
[Chen+ '58]

- factorization $T = T_1...T_t$ with $T_x \geq T_{x+1}$ $\forall x$
- $T_x$ is Lyndon; called Lyndon factor
- factorization uniquely defined
- linear time [Duval' 88]
Lyndon factorization
[Chen+ '58]

• factorization $T = T_1...T_t$ with $T_x \geq T_{x+1} \forall x$

• $T_x$ is Lyndon; called Lyndon factor

• factorization uniquely defined

• linear time [Duval' 88]

example:
$T_1 = bcc$
$T_2 = adb$
$T_3 = accbcd$

$T = b \ c \ c \ | \ a \ d \ b \ | \ a \ c \ c \ c \ b \ c \ d$
longest Lyndon substring $S$

- answer $S$ is longest Lyndon factor

$$T = b\, c\, c\, a\, d\, b\, a\, c\, c\, c\, b\, c\, d$$
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 $T_{y+1}[1..j] \leq T_x \leq T_x[i..|T_x|]$ by the Lyndon factorization $\Rightarrow S$ not Lyndon
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 $T[1..i]$?

$T = \text{bcccadbadbacccbcd}$

longest Lyndon subsequence of
$T[1..9]$ : $\text{bccdccc}$
$T[1..11]$ : $\text{abaccbc}$
$T[1..12]$ : $\text{bbcbccbcbcd}$

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

\[ T = \text{bcccadba} \]

properties

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

• leaves are Lyndon subsequences
• deepest leaf is longest Lyndon subsequence!
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 $p$

period $p = 3$
key observation

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

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

$W = bbcbb$, $p = 3$
$W[|W|-p+1] = W[5-3+1] = W[3] = c$

not pre-Lyndon

immature

Lyndon
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

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?

\[ T = \text{b c c c a d b a} \]
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 $\rho$ on stack is 2

compare $T[S[|S|-\rho+1]] = a$ 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

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

$\Rightarrow VW$ Lyndon and $VW < UW$
lemma

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

\[
VW \text{ Lyndon and } VW \prec UW
\]

$\Rightarrow VW$ Lyndon and $VW < UW$

proof.

1. take suffix $S$ of $VW$
   - $|S| \leq |W|$

   Lyndon
   - $U$ $W$
   - $V$ $S$

   Lyndon
   - $U$ $W$
   - $V$ $S$
lemma

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

$\Rightarrow VW$ Lyndon and $VW < UW$

proof.

take suffix $S$ of $VW$

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

such that

\begin{align*}
\text{Lyndon} & \quad \text{Lyndon} \\
V & \leq W & U & \leq W \\
V \not\triangleleft U & \iff V < U \text{ and } V \text{ not prefix of } U \\
& \quad \text{such that} \\
V \triangleleft U & \Rightarrow VW < UW \\
UW & \text{ Lyndon} \\
V & < U \\
|V| & \geq |U| \\
S & > UW > VW
\end{align*}
lemma

\[ V \triangle U :\Leftrightarrow V < U \text{ and } V \text{ not prefix of } U \]

- \( V \text{ Lyndon} \)
- \( U, W \in \Sigma^* \)

\[ \Rightarrow VW \text{ Lyndon and } VW < UW \]

proof.

take suffix \( S \) of \( VW \)

2. \(|S| > |W|\)

\[ V' \]

\[ S \]

\[ W \]

\[ V \]

\[ U \text{ Lyndon} \]

\[ V < U \]

\[ |V| \geq |U| \]
lemma

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

\[ \Rightarrow V W \text{ Lyndon and } V W \prec U W \]

proof.

take suffix $S$ of $V W$

2. $|S| > |W|$

- $U W$ Lyndon
- $V < U$
- $|V| \geq |U|$

$V \triangleleft U :\Leftrightarrow V < U$ and $V$ not prefix of $U$

\[ V \triangleleft V' \text{ since } V \text{ is Lyndon} \]
\[ \Rightarrow V W \prec V' \]

\[ \Rightarrow S > V' > V W \]
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 $\ell$, we prune $u$ if $L[\ell] < c(u)$ (the label of $u$)
algorithmic execution

initial state: $S = \emptyset$

$L = \infty \infty \infty \infty \infty \infty \infty \infty$
algorithmic execution

$S = (4,1) (6,2)$

Lyndon subsequences:

- $a$
- $ab$

$L = 4 \ 6 \ \infty \ \infty \ \infty \ \infty \ \infty \ \infty$
algorithmic execution

\[ S = (4,1) \ (5,2) \ (6,3) \]

Lyndon subsequences:
- ad
- adb

\[ L = \begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
4 & 5 & 6 & \infty & \infty & \infty & \infty
\end{array} \]

since ad ends earlier than ab, we can update \( L[2] \)
algorithmic execution

\[ S = (1,1) \ (2,2) \ (3,3) \ (5,4) \]

find subsequences

- \( b \)
- \( bc \)
- \( bcc \)
- \( bccd \)

\[ L = 1 \ 2 \ 3 \ 5 \ \infty \ \infty \ \infty \]
skip bcd

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

$L = 1 2 3 5 \infty \infty \infty$
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

→ 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 $\sigma$ 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 $j < i = L[\ell]$

$\Rightarrow$ 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)
  - $\min \{ T[j] \mid j \in J \land T[j] > c \}$ in $O(\lg n)$ time (range successor query)
suppose node has lowered \( L[\ell] \leftarrow i \) with a subsequence \( Wc \), \( \rho \) : period of \( W \)

\[
\text{RMQ } [k+1..L[\ell]-1] > W[|W|-% \rho+1] ?
\]

\[
\text{RMQ } [k+1..L[\ell]-1] = W[|W|-% \rho+1] ?
\]

- YES: can lower \( L[\ell] \Rightarrow \) use wavelet tree to find next node
- NO: found the immature child \( \Rightarrow \) descend
- YES: 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