Repetitions in strings: a "constant" problem

July 7, 7PM (JST)



CC BY-SA-NC

Hideo Bannai
M&D Data Science Center
Tokyo Medical and Dental University, Japan

Repeating Structures in Strings

- They occur in DNA.

- Has **constant**ly been one of our favorite topics!
  - Infinite square free sequences [Thue, 1906][Thue, 1912]
  - periods, squares, runs (maximal repetitions), gapped repeats, maximal repeats

- Some of us have been obsessed with certain **constant** factors concerning repetitions.

This talk:

I will focus on (exact) repetitions (contiguous repeats) and review the "Runs" theorem and some related results.

Periods

Definition

string w of length n has **period** p $(1 \leq p < n)$
$\Leftrightarrow w[i] = w[i+p]$ for any position $1 \leq i \leq n - p$

Example

	1	2	3	4	5	6	7	8	9	10	11
$w =$	b	b	a	b	b	a	b	b	a	b	b

3, 6, 9, 10 are periods of w. The smallest period is 3.

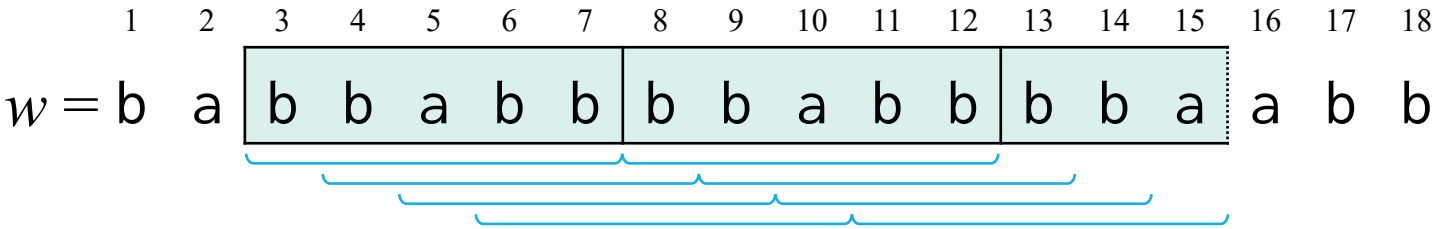
In this talk: "period" will (mostly) mean "smallest period"

Runs (Maximal Repetitions)

Definition

A substring occurrence $r = w[i..j]$ is a **run** of w
$\Leftrightarrow$ r has smallest period $p \leq |r|/2$, and
$w[i–1..j]$ and $w[i..j+1]$ do not have period p

Example

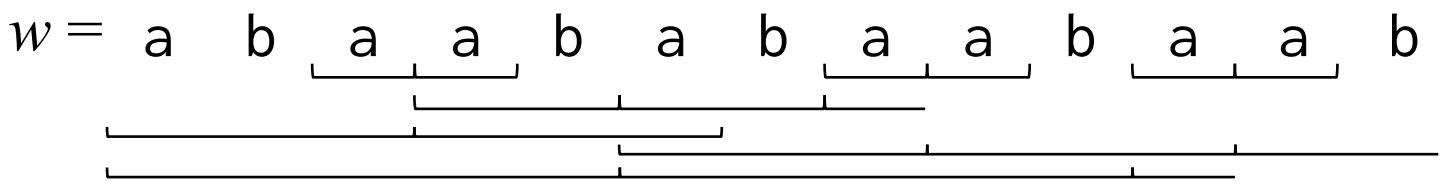


$r = w[3..15]$ is a run with period p=5

Maximum number of runs

Runs can overlap with other runs.

Example



7 runs in length 13 string

Question: How many runs can a string of length n contain?

Theorem [Kolpakov&Kuchreov 1999]

$\rho(n) = O(n)$, where $\rho(n)$ is the maximum number of runs in a string of length n.

…whereas there can be $\Theta(n^2)$ square occurrences, or $\Theta(n \log n)$ primitive square occurrences.

A "constant" problem

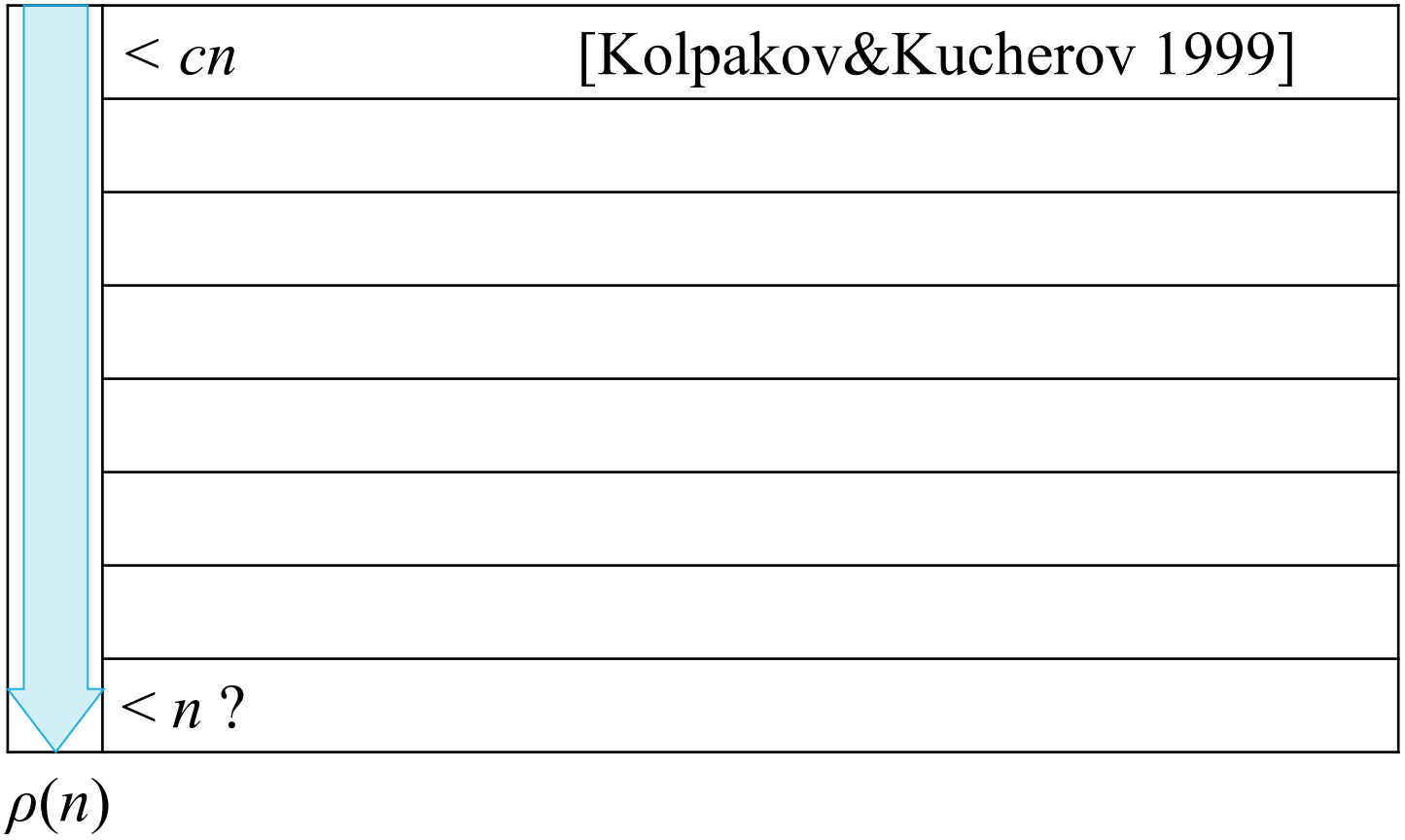
[Kolpakov&Kucherov 1999]

" **5. Concluding remarks**

The main drawback of our proof of Theorem 2 is t[illegible]t **does not** allow to extract a **"reasonable" constant factor** in [illegible]e linear bound. It seems however that this constant factor is [illegible]quite small. **Computer experiments suggest that the number of maximal repetitions is actually smaller than *n*** and the sum of their exponents smaller than $2n$, at least for the binary alphabet.
It would be interesting to **find a simpler proof** of Theorems 2,4 **implying a small multiplicative constant** in the linear bound."

The "Runs" conjecture

Towards Solving the “Runs” conjecture

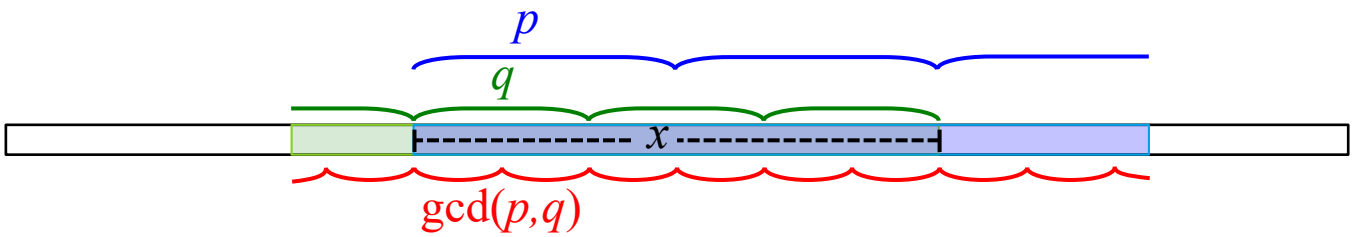


Key tool used

Periodicity Lemma [Fine&Wilf 1965]

If p, q are periods of string x and $|x| \geq p + q - \gcd(p,q)$, then, $\gcd(p,q)$ is also a period of x.

i.e., runs cannot overlap with another too much,
or else, their periods will "synchronize".



Towards Solving the "Runs" conjecture

$\rho(n)$	
$< cn$	[Kolpakov&Kucherov 1999]
$< 5n$	[Rytter 2006]
$< 3.48n$	[Puglisi et al. 2006]
$< 1.6n$	[Crochemore&Ilie 2008]
$< 1.52n$ (binary)	[Giraud 2008]
$< 1.29n$ (binary)	[Giraud 2009]
$< 1.029n$	[Crochemore et al. 2011]
$< n$?	

at least 60 cases

proof by computer verification

basic idea: exhaustive case analysis on how runs with short periods can overlap

➔ 3 CPU years of computation to get bound of $1.029n$

Towards Solving the “Runs” conjecture

$< cn$	[Kolpakov&Kucherov 1999]
$< 5n$	[Rytter 2006]
$< 3.48n$	[Puglisi et al. 2006]
$< 1.6n$	[Crochemore&Ilie 2008]
$< 1.52n$ (binary)	[Giraud 2008]
$< 1.29n$ (binary)	[Giraud 2009]
$< 1.029n$	[Crochemore et al. 2011]
$< n$?	

$\rho(n)$

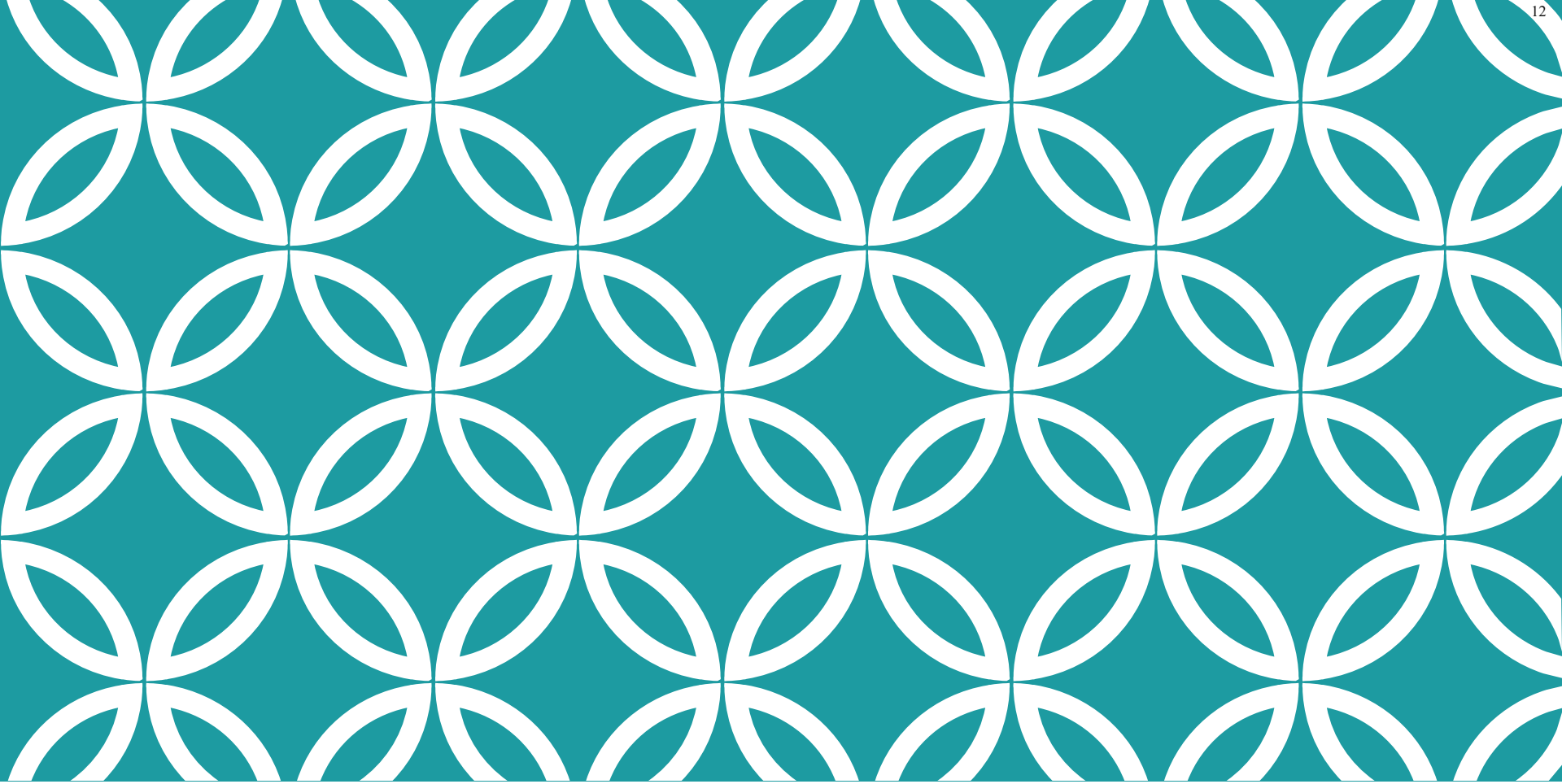
$0.927n <$	[Franek et al. 2008]

From Conjecture to Theorem

$< cn$	[Kolpakov&Kucherov 1999]
$< 5n$	[Rytter 2006]
$< 3.48n$	[Puglisi et al. 2006]
$< 1.6n$	[Crochemore&Ilie 2008]
$< 1.52n$ (binary)	[Giraud 2008]
$< 1.29n$ (binary)	[Giraud 2009]
$< 1.029n$	[Crochemore et al. 2011]
$< n$	

$\rho(n)$

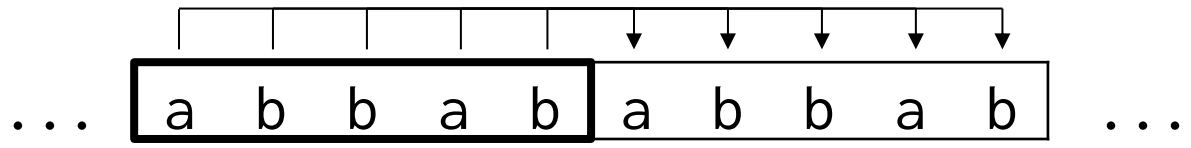
$0.944575712n <$	[Simpson 2010]
$0.944565n <$	[Matsubara et al. 2008]
$0.927n <$	[Franek et al. 2008]



The "Runs" Theorem

A property of runs

A run contains a square, and thus all cyclic rotations of its root



One of them (the lexicographically smallest one)
must be a **Lyndon** word!

Key tool: Lex orders and Lyndon words

Definition [Lyndon 1954]

A string is a **Lyndon word**, if it is lex smaller than any of its proper cyclic rotations

or equivalently

A string is a **Lyndon word**, if it is lex smaller than any of its non-empty proper suffixes

Example (a<b<c)

acacb ≺	cacba
	acbac
	cbaca
	bacac

acacb ≺	cacb
	acb
	cb
	b

Key tool: Lex orders and Lyndon words

Definition [Lyndon 1954]

A string is a **Lyndon word**, if it is lex smaller than any of its proper cyclic rotations

or equivalently

A string is a **Lyndon word**, if it is lex smaller than any of its non-empty proper suffixes

Key property of Lyndon words

Lyndon words cannot have borders

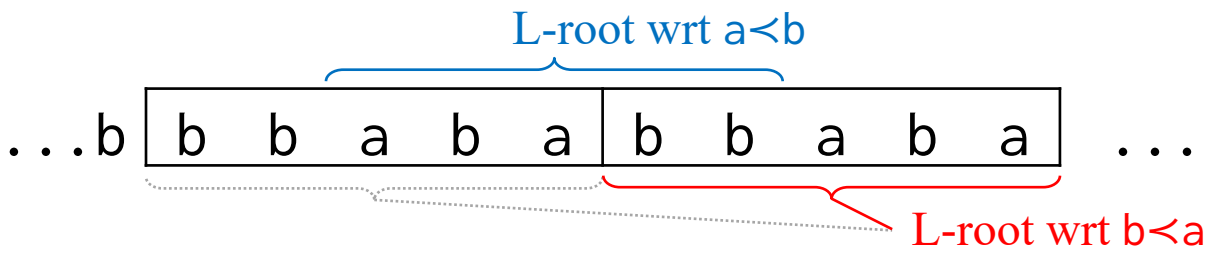
a non-empty proper prefix that is also a suffix

Proof: If for some x,y,z: $w = xy = zx$, then $x \prec xy = zx \prec x$.

L-roots

For any lex order, a run with period p contains at least one occurrence of a substring of length p that is a Lyndon word (L-root).

[Crochemore et al. 2012], [Crochemore et al. 2014]



For technical reasons, we will **not** consider L-root occurrences at the beginning of the run – but there will still be at least one occurrence.

The Proof

Consider two opposite lex orders $\prec_0$ and $\prec_1$ ($a \prec_0 b \Leftrightarrow b \prec_1 a$)

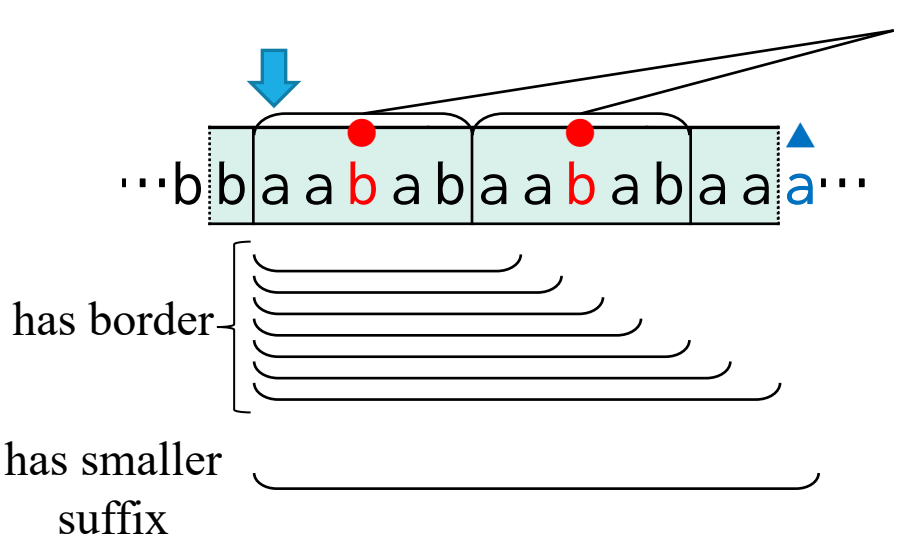
1. For any run $r = w[i..j]$, choose $\prec_{r_l} \in \{\prec_0, \prec_1\}$ so that $w[j+1] \prec_{r_l} w[j+1-p]$ (i.e., the character following the run is smaller than the corresponding character in the run).
   Then, any occurrence of an L-root wrt $\prec_{r_l}$ in the run is the longest Lyndon word wrt $\prec_{r_l}$ that starts at the same position.

2. For each position, only one of the longest Lyndon words starting at that position wrt $\prec_0$ and $\prec_1$ can be an L-root (which is not at the beginning) of a run.

1. L-roots and Longest Lyndon words

Consider two opposite lex orders $\prec_0$ and $\prec_1$ ($a \prec_0 b \Leftrightarrow b \prec_1 a$)

1. For any run $r = w[i..j]$, choose $\prec_{r_l} \in \{\prec_0, \prec_1\}$ so that $w[j+1] \prec_{r_l} w[j+1-p]$ (i.e., the character following the run is smaller than the corresponding character in the run).
   Then, any occurrence of an L-root wrt $\prec_{r_l}$ in the run is the longest Lyndon word wrt $\prec_{r_l}$ that starts at the same position.



We claim they are the longest Lyndon words that start there

The Proof

Consider two opposite lex orders $\prec_0$ and $\prec_1$ ($a \prec_0 b \Leftrightarrow b \prec_1 a$)

1. For any run $r = w[i..j]$, choose $\prec_{r_l} \in \{\prec_0, \prec_1\}$ so that $w[j+1] \prec_{r_l} w[j+1-p]$ (i.e., the character following the run is smaller than the corresponding character in the run).
   Then, any occurrence of an L-root wrt $\prec_{r_l}$ in the run is the longest Lyndon word wrt $\prec_{r_l}$ that starts at the same position.

   This already gives $2n$ bound since
   - different runs cannot share the same L-root.
   - at most $2n$ longest Lyndon words (2 for each position)

2. For each position, only one of the longest Lyndon words starting at that position wrt $\prec_0$ and $\prec_1$ can be an L-root (which is not at the beginning) of a run.

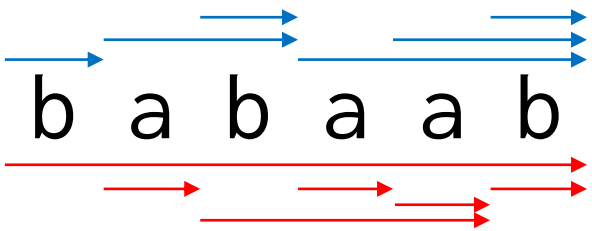
2. L-roots at the same position

Consider two opposite lex orders $\prec_0$ and $\prec_1$ ($a \prec_0 b \Leftrightarrow b \prec_1 a$)

2. For each position, only one of the longest Lyndon words starting at that position wrt $\prec_0$ and $\prec_1$ can be an L-root (which is not at the beginning) of a run.

Observation:

The longest Lyndon word starting at any position is a single character, for at least one of $\prec_0$ or $\prec_1$.



longest Lyndon wrt a $\prec_0$ b

Choose order s.t. next different char is smaller.

longest Lyndon wrt b $\prec_1$ a

2. L-roots at the same position

Consider two opposite lex orders $\prec_0$ and $\prec_1$ ($a \prec_0 b \Leftrightarrow b \prec_1 a$)

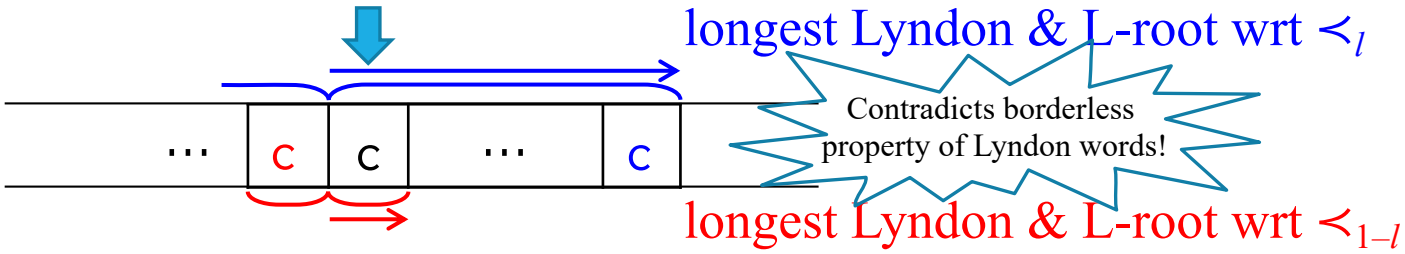
2. For each position, only one of the longest Lyndon words starting at that position wrt $\prec_0$ and $\prec_1$ can be an L-root (which is not at the beginning) of a run.

Observation:

The longest Lyndon word starting at any position is a single character, for at least one of $\prec_0$ or $\prec_1$.

Proof of 2.

Suppose the longest Lyndon word for both lex orders are L-roots which are not at the beginning of the run.



The "Runs" Theorem

Theorem [Bannai et al. 2017]

$$\rho(n) < n$$

Current best upperbound combining these ideas with computations: $0.9482n$ (for binary strings) [Holub 2017]

Corollary

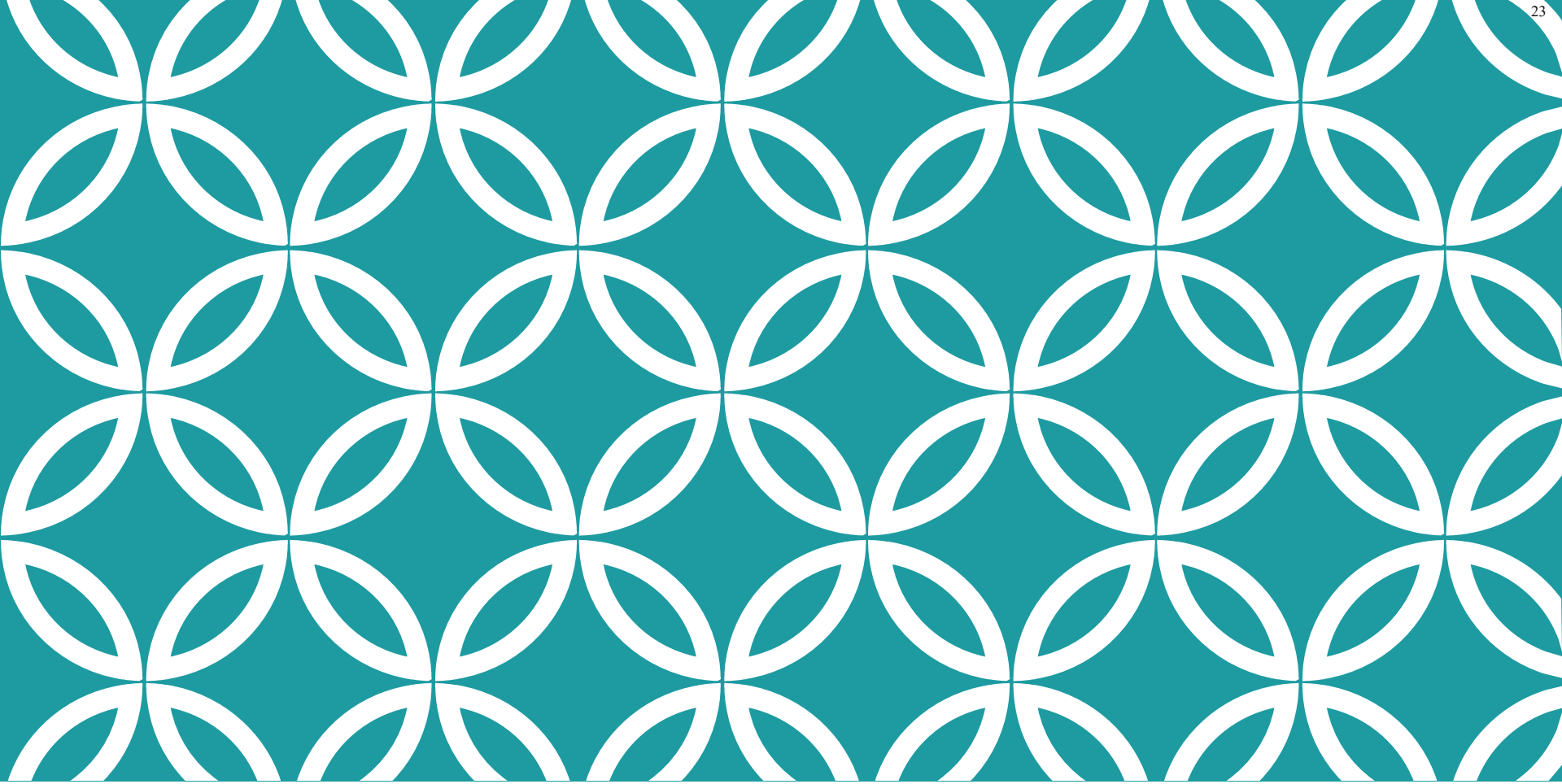
Sum of exponents of all runs $< 3n$

Proof

For each run r with exponent e_r, there are $\lfloor e_r - 1 \rfloor$ L-root occurrences corresponding to longest Lyndon words which are not at the beginning of the run.

From previous arguments, $\sum_r \lfloor e_r - 1 \rfloor < n$. Then,

$$\sum_r e_r < \sum_r (e_r - 2) + 2n \leq \sum_r \lfloor e_r - 1 \rfloor + 2n < 3n.$$



Efficient computation of runs

Computing runs in linear time [Kolpakov & Kucherov 1999]

(assuming constant or linearly sortable alphabet)

Basically, the same as [Main 1989] that finds leftmost occurrences of runs in linear time

1. Compute LZ77 factorization of string (each factor is longest substring that occurs previously)
2. Compute all runs that cross LZ77 factor boundaries ➔ linear in the length of adjacent factors ➔ linear in total
3. Runs that are contained in an LZ77 factor can be copied from the previous occurrence of the factor

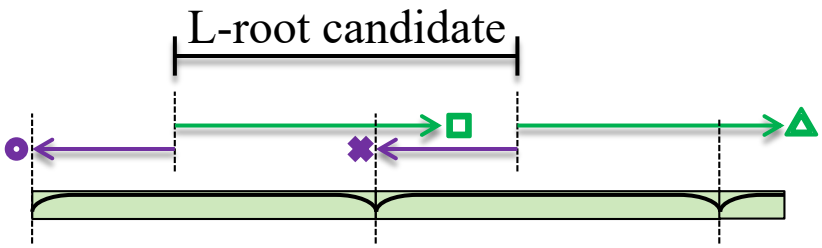
Linearity of computation follows from linearity of # of runs.

Depends on LZ77 factorization

A new approach for computing runs [Bannai et al. 2017]

For lex orders $\prec_0$ and $\prec_1$ ($a \prec_0 b \Leftrightarrow b \prec_1 a$)

1. Compute the longest Lyndon word starting at each position
   - $w[i..j]$ is longest Lyndon word starting at i for smallest j s.t. $w[j+1..n] \prec w[i..n]$
   - each of the $2n$ longest Lyndon words is a candidate for an L-root of a run
2. Compute Longest Common Extensions (forward & backward) on each L-root candidate to see how far the periodicity extends
   - O(n) LCEs
   - a run is found iff the sum of LCEs are long enough





Assuming constant or linearly sortable alphabet, there are well known standard methods for computing 1.(NSV on ISA) and 2. (LCE) in total O(n) time.
(1. can also be computed by O(n) LCE queries)

Significance of new approach

For unordered alphabets (only equality comparisons), determining square-freeness (thus computing runs) requires $\Omega(n \log n)$ time in the worst case [Main & Lorentz 1984]

Can we do better for general ordered alphabets of size σ? [Breslauer 1992]

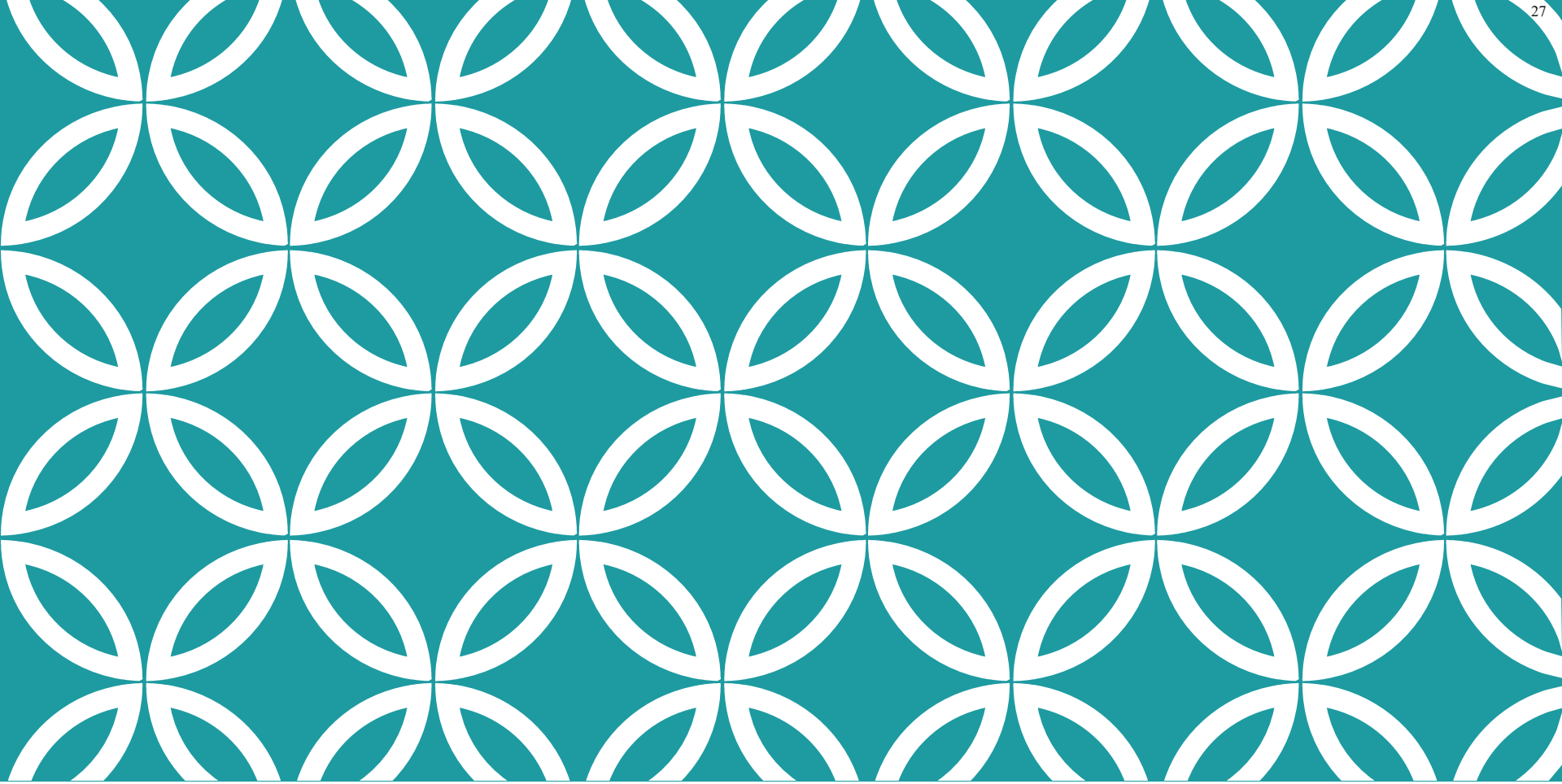
- No hope if we use LZ77: Computing LZ77 factorization requires $\Omega(n \log \sigma)$ symbol comparisons [Kosolobov 2015]

<u>Computing runs for general ordered alphabets based on new approach</u>

- $O(n \log^{2/3} n)$ time [Kosolobov 2016]
- $O(n \log\log n)$ time [Gawrychowski et al. 2016]

} faster LCE queries

- $O(n\alpha(n))$ time [Crochemore et al. 2016] faster "non-crossing" LCE queries
- $O(n)$ time [Ellert&Fischer 2021] new!
  - $O(n)$ time computation of longest Lyndon words [Bille et al. 2020]
  - $O(n)$ time for all required LCEs [Ellert&Fischer 2021]



Variations on a theme

of Runs in Run-Length Encoded (RLE) Strings

Theorem [Fujishige et al. 2017]

The number of runs in a string w is less than $m + k$ where
m = # of RLE factors of w, k = number of runs with period 1

For any r with period $p \geq 2$, any L-root occurrence (which is not at the beginning) wrt $\prec_{rl}$

- starts at the beginning of an RLE factor

  If it is not at the beginning of an RLE factor,
  there is an L-root occurrence of a run with period 1
  at the same position, contradicting the proof of the runs theorem.

- and ends at the end of an RLE factor.

  If xc is a Lyndon word (for non-empty x), xcc is a Lyndon word.

Computing Runs in RLE Strings

Theorem [Fujishige et al. 2017]

Given the RLE of size m of string w, all runs can be computed in $O(m\ \alpha(m))$ time and $O(m)$ space.

Proof outline:

Computing runs with period 1 are straightforward.

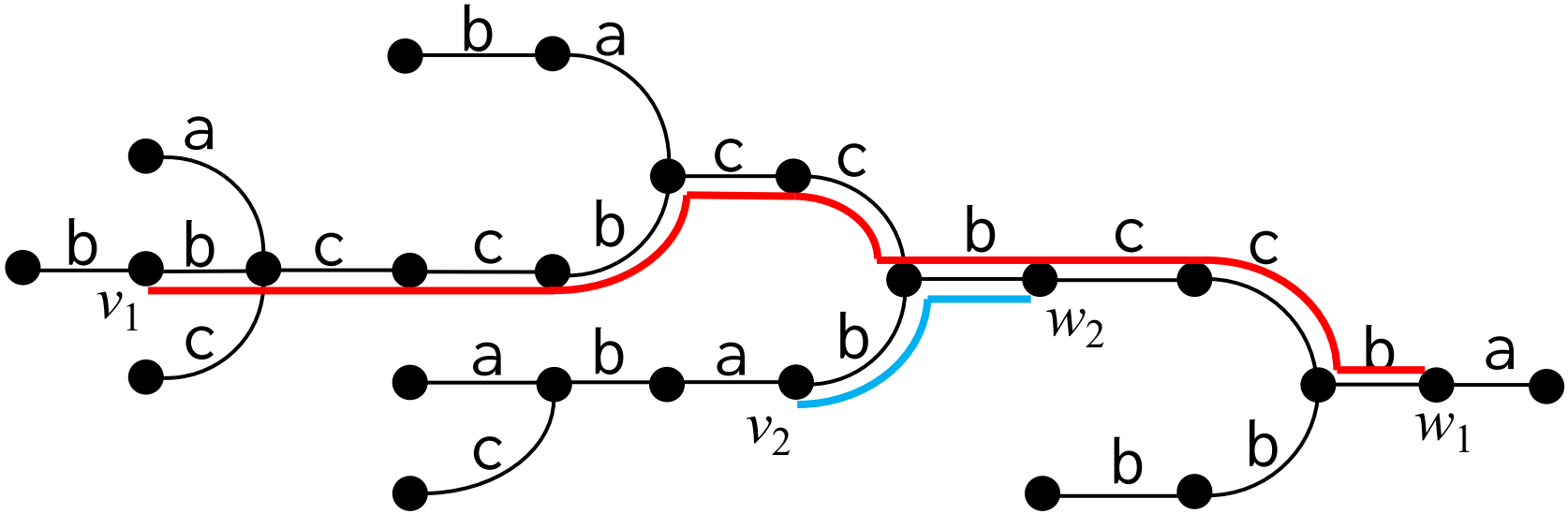
For runs with period ≥ 2:

- L-root candidates start and end at RLE boundaries.
- Runs can be computed by $O(m)$ non-crossing LCE queries on RLE using [Crochemore et al. 2016].

Runs in Tries [Sugahara et al. 2019]

Trie: Rooted tree with edges labeled with single character, and labels to children are distinct.

- Representation of a set of strings.
  For a trie with n edges, total length of strings can be $\Theta(n^2)$.
  The trie of the set of reversed strings can also be $\Theta(n^2)$.



A path $v \ldots w$ is a *run* if w is ancestor of v and

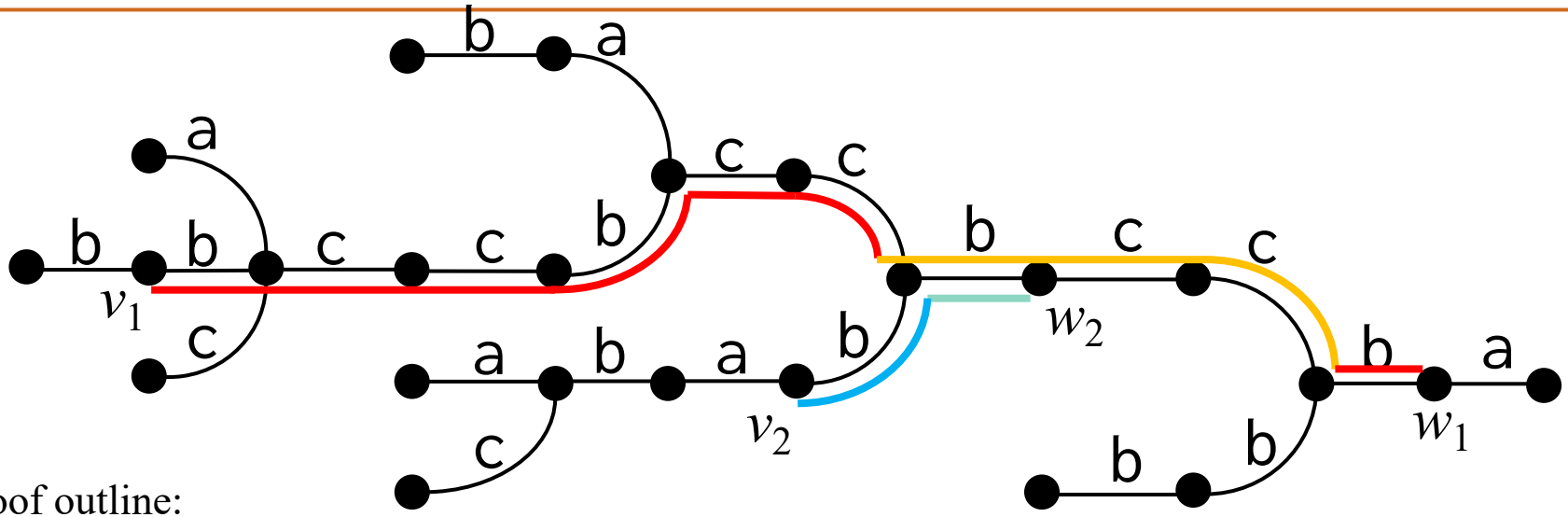
- period(pathlabel($v..w$)) $= p \leq$ |pathlabel($v..w$)|/2 and
- period(pathlabel($v'..w$)) $\neq p$ for **any** child v' of v
- period(pathlabel($v..w'$)) $\neq p$ for parent w' of w

Runs in Tries – Upper bound

Theorem [Sugahara et al. 2019]

$$\rho_T(n) < n$$

where $\rho_T(n)$ = max # of runs in a trie with n edges.



Proof outline:

- $\prec_{r_l}$/L-roots/longest Lyndon words can be defined analogously to strings
- Unlike strings, there can be two L-roots starting at same node. However,
  - The node must be branching
  - A leaf is never a starting point of a L-root (which is not at a beginning of a run)

Runs in Tries – Lower bounds

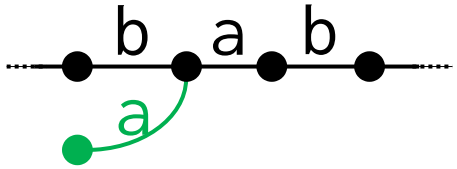
A string is a trie. Therefore, for sufficiently large n,

$$0.944575712n < \rho(n) \leq \rho_T(n)$$

Theorem [Sugahara et al. submitted]

For sufficiently large n, $0.9932348n < \rho_T(n)$.

(Computational) Proof outline:

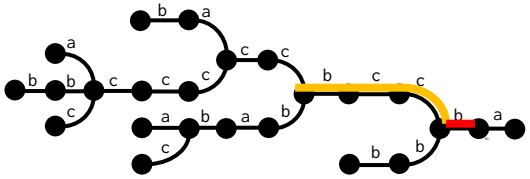


Starting from a run rich string, we can add:

- a branching edge "a" at any occurrence of "bab" to create run "aa"
- a branching edge "b" at any occurrence of "aba" to create run "bb"

This increases the density of runs, since at least one run is generated for each edge added.

Computing Runs in Tries
[Sugahara et al. 2019]



(assuming constant or linearly sortable alphabet)

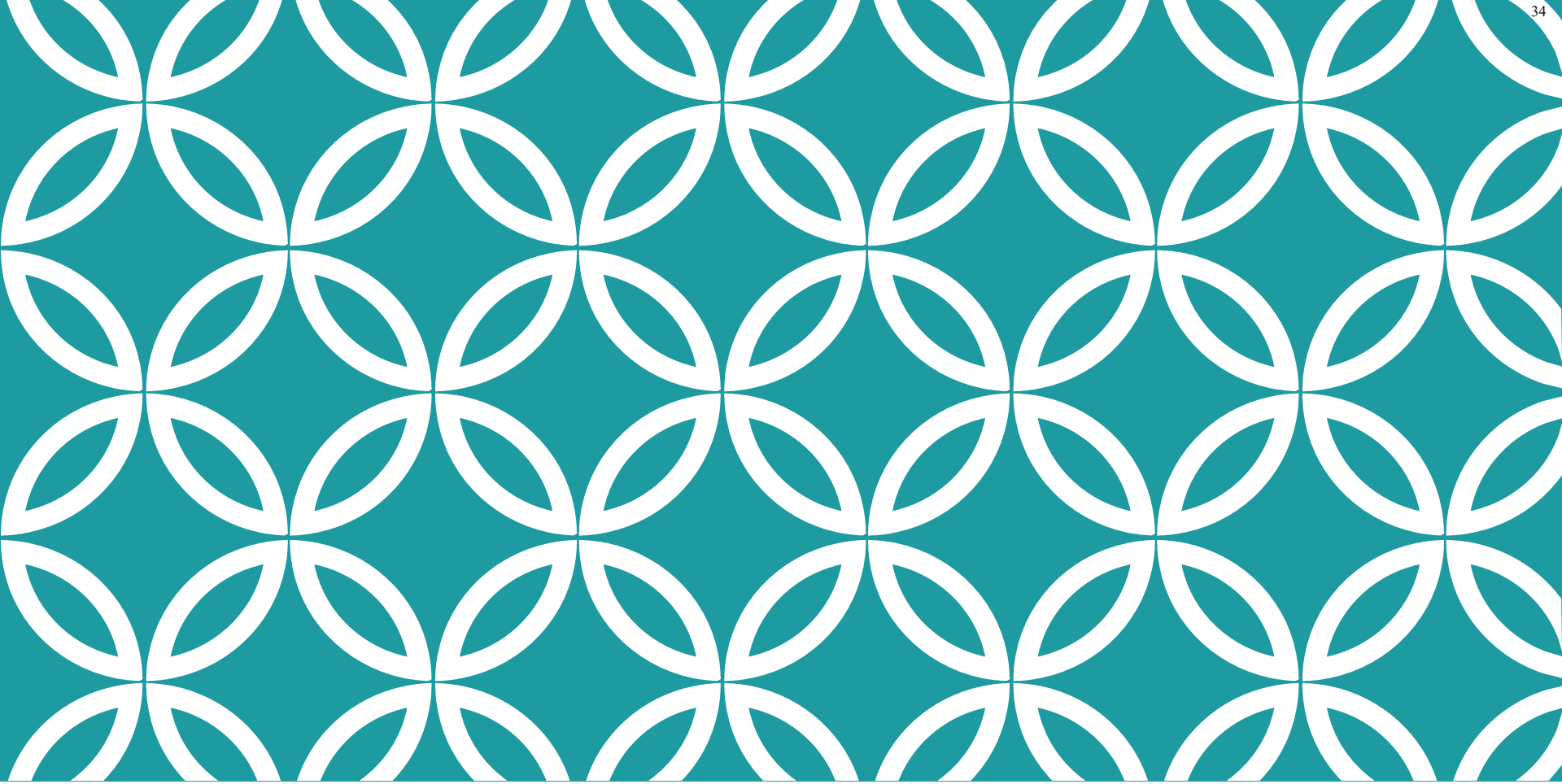
For lex orders $\prec_0$ and $\prec_1$ ($a \prec_0 b \Leftrightarrow b \prec_1 a$)

1. Compute longest Lyndon word starting at each node.
   - NSV on ISA ⇨ Lexicographically smaller lowest ancestor
   - O(n) time using inverse suffix array of trie [Shibuya 1999]
     and decremental nearest marked ancestor queries [Gabow&Tarjan 1985]

2. Compute O(n) LCE queries in
   forward (to root) & backward (to leaf) directions
   - Forward: O(n) total time using standard techniques
   - Backward: Difficult since the path can be branching.
     O(n (loglog n)^2) total time using **path-tree LCE queries** [Bille et al. 2016]

Backward (path-tree) LCE queries can be answered in
O(n log log n) total time by processing them in a batch.

[Sugahara et al. submitted]



Constants concerning Squares

Primitively rooted square occurrences

Max. # of primitively rooted square occurrences:

- Upper Bound [Fraenkel&Simpson 1998]

Three Squares Lemma [Crochemore&Rytter 1995]

Let u^2, v^2, w^2 be three prefixes of some string s.t. w is primitive and $|u|>|v|>|w|$. Then, $|u| \geq |v|+|w|$.

At most $\log_\phi n$ ($\approx 1.441 \log_2 n$) primitive squares start at any position.

Total of at most $n\log_\phi n$ ($\approx \mathbf{1.441} n\log_2 n$) primitive squares

- Lower bound [Fraenkel&Simpson 1999]

$0.796\, n\log_2 n + O(n)$ primitive squares in Fibonacci words

We improve the upperbound to $\boldsymbol{n\log_2 n}$ using Lyndon arguments.

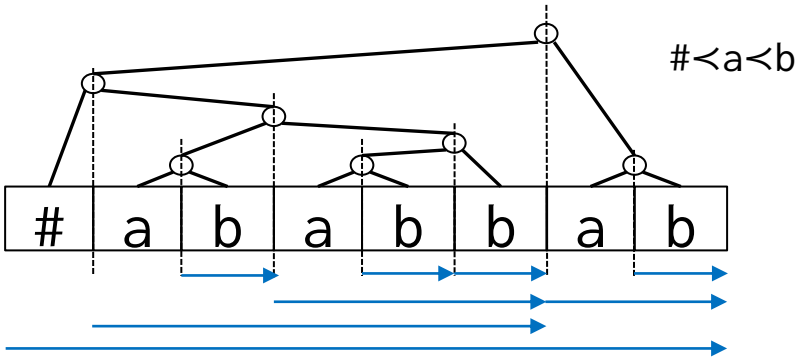
Counting Primitive Square Occurrences via Lyndon Tree
[Bannai et al. 2020]

Lyndon tree (Alternate representation of longest Lyndon words)

For any Lyndon word *w*, there exists a unique full binary tree s.t.:

- The root node is *w*
- each node is a Lyndon substring of *w*, and is a concatenation of its children
- the longest Lyndon word starting at each position is a right child of its parent (root also considered a right child).

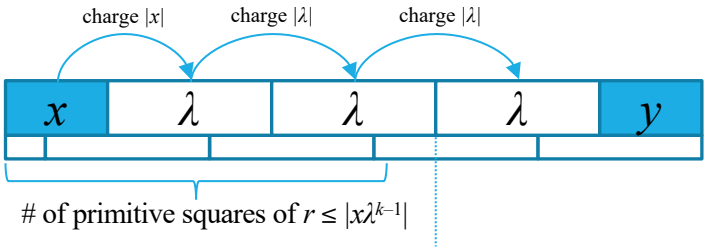
can add a small character at beginning to make any word Lyndon



Counting Primitive Square Occurrences via Lyndon Tree
[Bannai et al. 2020]

Observation

For run r, let $r = x\lambda^k y$, where λ is L-root, $1 \leq |x| \leq |\lambda|$, and $|y| < |\lambda|$.



- # of primitive squares of $r \leq |x\lambda^{k-1}|$
- these can be counted by charging at most $|\lambda|$ to the L-root to the right

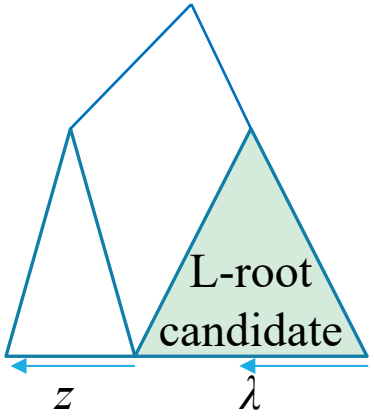
An L-root wrt $\prec_{r_l}$ is a right node in Lyndon tree wrt $\prec_{r_l}$

- Each right node is charged at most $\min\{ |z|, |\lambda| \}$, since $z\lambda$ (being Lyndon) cannot have a border.
- Total charges to all right nodes in tree is bounded by:

$$S(w) = \max_{w=z\lambda}\{ S(z)+S(\lambda) + \min\{|z|,|\lambda|\} \}$$

Solving this gives $S(w) = (n/2)\log_2 n$.

Thus, we get a bound of $\boldsymbol{n\log_2 n}$.



Distinct Squares Conjecture
[Fraenkel&Simpson 1998]

Distinct square substrings (not occurrences)

A string of length n contains less than $\mathbf{2n}$ distinct squares (easily proved with three squares lemma) [Fraenkel&Simpson 1998]

Distinct squares conjecture [Fraenkel&Simpson 1998]

A string of length n contains less than $\boldsymbol{n}$ distinct squares.

There is some progress but...

…perhaps we aren't using the right tools...

A Longer List of Related Open Problems

1. ~~Computing runs in linear time for general ordered alphabets~~
2. Better bounds for $\rho(n)$?
   - Upper bound: $0.9482n$ (for binary strings) [Holub 2017]
   - Lower bound: $0.94457564n$ [Simpson 2010]
3. Better bounds for the maximum sum of exponents of runs?
   - Upper bound: $3n$ [Bannai et al. 2017]
   - Lower bound: $2.035n$ [Crochemore et al. 2012]
4. Can $\rho(n)$ always be achieved with binary strings?
5. O(n) time computation of runs in tries?
   - For constant or linearly sortable alphabets?
   - For general ordered alphabets?
6. The Distinct Squares Conjecture
   - Upper bound: $11/6n$ [Deza et al. 2015] $1.5n$? [Thierry 2021 (arxiv)]
   - Lower bound: $n - o(n)$ [Fraenkel&Simpson 1998]
7. Number of Runs in 2D strings
   - Upper bound: $O(n^2 \log^2 n)$ [Charalampopoulos et al. 2020]
   - Lower bound: $\Omega(n^2 \log n)$ [Gawrychowski et al. 2021]
8. Reverse engineering the Lyndon tree/array
   - Easy for binary / unbounded alphabet

Acknowledgments/Bibliography

- [Matsubara et al. 2008]
  Wataru Matsubara, Kazuhiko Kusano, Akira Ishino, Hideo Bannai, Ayumi Shinohara,
  New Lower Bounds for the Maximum Number of Runs in a String, PSC 2008.
- [Bannai et al. 2015]
  Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima Kazuya Tsuruta, Masayuki Takeda,
  A new characterization of maximal repetitions by Lyndon trees, SODA 2015.
- [Bannai et al. 2017]
  Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima Kazuya Tsuruta, Masayuki Takeda,
  The "Runs" Theorem, SIAM J. Comput, 2017.
- [Fujishige et al. 2017]
  Yuta Fujishige, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda,
  Almost Linear Time Computation of Maximal Repetitions in Run Length Encoded Strings, ISAAC 2017.
- [Sugahara et al. 2019]
  Ryo Sugahara, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda,
  Computing Runs on a Trie, CPM 2019.
- [Bannai et al. 2020]
  Hideo Bannai, Takuya Mieno, Yuto Nakashima,
  Lyndon Words, the Three Squares Lemma, and Primitive Squares. SPIRE 2020.