Near-Optimal Computation of Runs over General Alphabet via Non-Crossing LCE Queries

Maxime Crochemore, Costas S. Iliopoulos,
Tomasz Kociumaka, Ritu Kundu, Solon P. Pissis,
Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń

King's College London, UK
University of Warsaw, Poland

SPIRE 2016
Beppu, Japan
October 18, 2016

LCE Queries

For positions i, j in a word w, $\mathrm{LCE}(i, j)$ is length of the longest common prefix of $w[i..]$ and $w[j..]$.

LCE Queries

For positions i, j in a word w, $\mathrm{LCE}(i, j)$ is length of the longest common prefix of $w[i..]$ and $w[j..]$.

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

LCE Queries

LCE Queries

For positions i, j in a word w, $\mathrm{LCE}(i, j)$ is length of the longest common prefix of $w[i..]$ and $w[j..]$.

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

$$\mathrm{LCE}(2, 7) =$$

LCE Queries

LCE Queries

For positions i, j in a word w, $\mathrm{LCE}(i, j)$ is length of the longest common prefix of $w[i..]$ and $w[j..]$.

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

$$\mathrm{LCE}(2, 7) = 4$$

LCE Queries

LCE Queries

For positions i, j in a word w, $\mathrm{LCE}(i, j)$ is length of the longest common prefix of $w[i..]$ and $w[j..]$.

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

$$\mathrm{LCE}(2, 7) = 4$$

LCE Problem

For a given word w of length n, answer a sequence of $q = \mathcal{O}(n)$ queries $\mathrm{LCE}(i, j)$ in an on-line manner.

Algorithms for the LCE Problem

Integer alphabet

Letters can be sorted in $\mathcal{O}(n)$ time (e.g., integers $\{1, \ldots, n\}$).

$\mathcal{O}(n)$ Range Minimum Queries on the LCP table

Algorithms for the LCE Problem

Integer alphabet

Letters can be sorted in $\mathcal{O}(n)$ time (e.g., integers $\{1, \ldots, n\}$).

$\mathcal{O}(n)$ Range Minimum Queries on the LCP table

General alphabet

Symbols can be accessed only via comparisons $(<, =, >)$

$\mathcal{O}(n^2)$ Symbol-by-symbol naive check

$\mathcal{O}(n \log n)$ Reduction to integer alphabet

Algorithms for the LCE Problem

Integer alphabet

Letters can be sorted in $\mathcal{O}(n)$ time (e.g., integers $\{1, \ldots, n\}$).

$\mathcal{O}(n)$ Range Minimum Queries on the LCP table

General alphabet

Symbols can be accessed only via comparisons $(<, =, >)$

$\mathcal{O}(n^2)$	Symbol-by-symbol naive check
$\mathcal{O}(n \log n)$	Reduction to integer alphabet
$\mathcal{O}(n \log^{2/3} n)$	Kosolobov; IPL 2016
$\mathcal{O}(n \log \log n)$	Gawrychowski, K., Rytter, Waleń; CPM 2016

Runs (Maximal Repetitions)

Definition

A run is a maximal periodic fragment $w[i..j]$. For $p = \mathrm{per}(w[i..j])$,

- $2p \leq |w[i..j]|$,
- p is not a period of $w[i-1..j]$ and $w[i..j+1]$.

Definition

A run is a maximal periodic fragment $w[i..j]$. For $p = \mathrm{per}(w[i..j])$,

- $2p \leq |w[i..j]|$,
- p is not a period of $w[i-1..j]$ and $w[i..j+1]$.

a a b a b a a b a b b

Runs (Maximal Repetitions)

Definition

A run is a maximal periodic fragment $w[i..j]$. For $p = \mathrm{per}(w[i..j])$,

- $2p \leq |w[i..j]|$,
- p is not a period of $w[i-1..j]$ and $w[i..j+1]$.

a a b a b a a b a b b

period 1: $w[1..2]$, $w[6..7]$, $w[10..11]$

Runs (Maximal Repetitions)

Definition

A run is a maximal periodic fragment $w[i..j]$. For $p = \mathrm{per}(w[i..j])$,

- $2p \leq |w[i..j]|$,
- p is not a period of $w[i-1..j]$ and $w[i..j+1]$.

a a b a b a a b a b b

period 1: $w[1..2]$, $w[6..7]$, $w[10..11]$

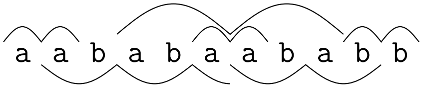
period 2: $w[2..6]$, $w[7..10]$

Runs (Maximal Repetitions)

Definition

A run is a maximal periodic fragment $w[i..j]$. For $p = \mathrm{per}(w[i..j])$,

- $2p \leq |w[i..j]|$,
- p is not a period of $w[i-1..j]$ and $w[i..j+1]$.



period 1: $w[1..2]$, $w[6..7]$, $w[10..11]$

period 2: $w[2..6]$, $w[7..10]$

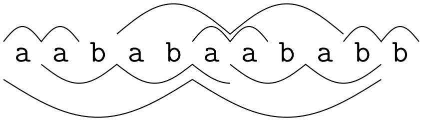
period 3: $w[5..8]$

Runs (Maximal Repetitions)

Definition

A run is a maximal periodic fragment $w[i..j]$. For $p = \mathrm{per}(w[i..j])$,

- $2p \leq |w[i..j]|$,
- p is not a period of $w[i-1..j]$ and $w[i..j+1]$.



period 1: $w[1..2]$, $w[6..7]$, $w[10..11]$

period 2: $w[2..6]$, $w[7..10]$

period 3: $w[5..8]$

period 5: $w[1..10]$

Computing Runs

Integer alphabet:

Kolpakov, Kucherov (1999): $\mathcal{O}(n)$ time using LZ factorization

Bannai et. al (2015): $\mathcal{O}(n)$ time using Lyndon roots & LCE

Computing Runs

Integer alphabet:

Kolpakov, Kucherov (1999): $\mathcal{O}(n)$ time using LZ factorization

Bannai et. al (2015): $\mathcal{O}(n)$ time using Lyndon roots & LCE

General alphabet:

Kosolobov (2015): LZ factorization: $\Omega(n \log n)$ comparisons

Kosolobov (2015): $\mathcal{O}(n^2)$ time, $\mathcal{O}(n)$ comparisons

Computing Runs

Integer alphabet:

Kolpakov, Kucherov (1999): $\mathcal{O}(n)$ time using LZ factorization

Bannai et. al (2015): $\mathcal{O}(n)$ time using Lyndon roots & LCE

General alphabet:

Kosolobov (2015): LZ factorization: $\Omega(n \log n)$ comparisons

Kosolobov (2015): $\mathcal{O}(n^2)$ time, $\mathcal{O}(n)$ comparisons

Improvements via LCE queries:

Kosolobov (2016): $\mathcal{O}(n \log^{2/3} n)$ time, $\mathcal{O}(n)$ comparisons

Gawrychowski et al. (2016): $\mathcal{O}(n \log \log n)$ time, $\mathcal{O}(n)$ comp.

Integer alphabet:

Kolpakov, Kucherov (1999): $\mathcal{O}(n)$ time using LZ factorization

Bannai et. al (2015): $\mathcal{O}(n)$ time using Lyndon roots & LCE

General alphabet:

Kosolobov (2015): LZ factorization: $\Omega(n \log n)$ comparisons

Kosolobov (2015): $\mathcal{O}(n^2)$ time, $\mathcal{O}(n)$ comparisons

Improvements via LCE queries:

Kosolobov (2016): $\mathcal{O}(n \log^{2/3} n)$ time, $\mathcal{O}(n)$ comparisons

Gawrychowski et al. (2016): $\mathcal{O}(n \log \log n)$ time, $\mathcal{O}(n)$ comp.

Improvements via non-crossing LCE queries:

This work: $\mathcal{O}(n\alpha(n))$ time, $\mathcal{O}(n)$ comparisons.

Non-Crossing LCE Queries

Definition

Pairs of integers $\{i, j\}$ and $\{i', j'\}$ are crossing if $i < i' < j < j'$ or $i' < i < j' < j$.

Definition

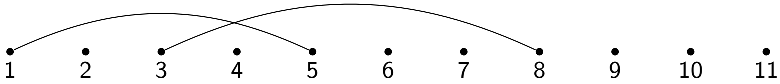
Pairs of integers $\{i, j\}$ and $\{i', j'\}$ are crossing if $i < i' < j < j'$ or $i' < i < j' < j$.

1 2 3 4 5 6 7 8 9 10 11

Non-Crossing LCE Queries

Definition

Pairs of integers $\{i, j\}$ and $\{i', j'\}$ are crossing if $i < i' < j < j'$ or $i' < i < j' < j$.

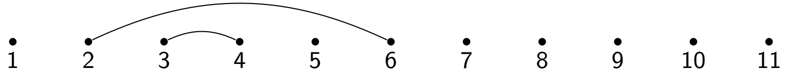


$\{1, 5\}$ and $\{3, 8\}$ crossing

Non-Crossing LCE Queries

Definition

Pairs of integers $\{i, j\}$ and $\{i', j'\}$ are crossing if $i < i' < j < j'$ or $i' < i < j' < j$.



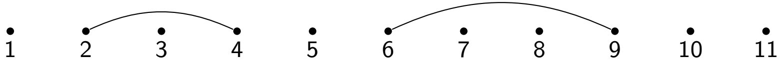
$\{1, 5\}$ and $\{3, 8\}$ crossing

$\{2, 6\}$ and $\{3, 8\}$ non-crossing

Non-Crossing LCE Queries

Definition

Pairs of integers $\{i, j\}$ and $\{i', j'\}$ are crossing if $i < i' < j < j'$ or $i' < i < j' < j$.



$\{1, 5\}$ and $\{3, 8\}$ crossing

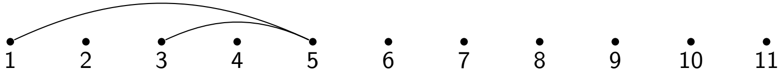
$\{2, 6\}$ and $\{3, 8\}$ non-crossing

$\{2, 4\}$ and $\{6, 9\}$ non-crossing

Non-Crossing LCE Queries

Definition

Pairs of integers $\{i, j\}$ and $\{i', j'\}$ are crossing if $i < i' < j < j'$ or $i' < i < j' < j$.



$\{1, 5\}$ and $\{3, 8\}$ crossing

$\{2, 6\}$ and $\{3, 8\}$ non-crossing

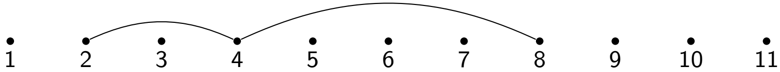
$\{2, 4\}$ and $\{6, 9\}$ non-crossing

$\{1, 5\}$ and $\{3, 5\}$ non-crossing

Non-Crossing LCE Queries

Definition

Pairs of integers $\{i, j\}$ and $\{i', j'\}$ are crossing if $i < i' < j < j'$ or $i' < i < j' < j$.

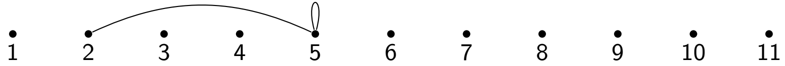


$\{1, 5\}$ and $\{3, 8\}$	crossing
$\{2, 6\}$ and $\{3, 8\}$	non-crossing
$\{2, 4\}$ and $\{6, 9\}$	non-crossing
$\{1, 5\}$ and $\{3, 5\}$	non-crossing
$\{2, 4\}$ and $\{4, 8\}$	non-crossing

Non-Crossing LCE Queries

Definition

Pairs of integers $\{i, j\}$ and $\{i', j'\}$ are crossing if $i < i' < j < j'$ or $i' < i < j' < j$.



$\{1, 5\}$ and $\{3, 8\}$	crossing
$\{2, 6\}$ and $\{3, 8\}$	non-crossing
$\{2, 4\}$ and $\{6, 9\}$	non-crossing
$\{1, 5\}$ and $\{3, 5\}$	non-crossing
$\{2, 4\}$ and $\{4, 8\}$	non-crossing
$\{2, 5\}$ and $\{5, 5\}$	non-crossing

Non-Crossing LCE Queries

Definition

Pairs of integers $\{i, j\}$ and $\{i', j'\}$ are crossing if $i < i' < j < j'$ or $i' < i < j' < j$.

1	2	3	4	5	6	7	8	9	10	11

$\{1, 5\}$ and $\{3, 8\}$ crossing
$\{2, 6\}$ and $\{3, 8\}$ non-crossing
$\{2, 4\}$ and $\{6, 9\}$ non-crossing
$\{1, 5\}$ and $\{3, 5\}$ non-crossing
$\{2, 4\}$ and $\{4, 8\}$ non-crossing
$\{2, 5\}$ and $\{5, 5\}$ non-crossing

Theorem (Our main technical result)

The LCE *problem can be solved in* $\mathcal{O}(n\alpha(n))$ *time in the general alphabet model if args* $\{i, j\}$ *of the* LCE *queries are non-crossing.*

Properties of Non-Crossing Families

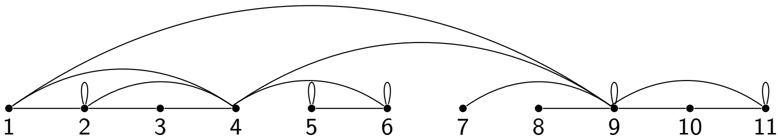
Fact

A family of non-crossing pairs from $\{1, \ldots, n\}$ *contains less than* $3n$ *distinct pairs.*

Properties of Non-Crossing Families

Fact

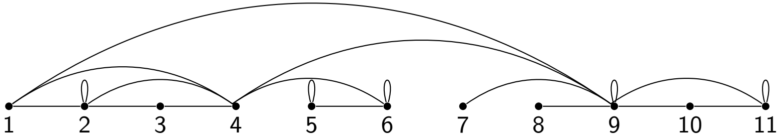
A family of non-crossing pairs from $\{1, \ldots, n\}$ contains less than $3n$ distinct pairs.



Properties of Non-Crossing Families

Fact

A family of non-crossing pairs from $\{1, \ldots, n\}$ contains less than $3n$ distinct pairs.



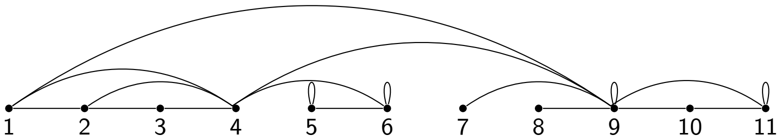
Proof. Pairs $\{i, j\}$ form the edge set of an outerplanar graph:

- at most n loops
- simple outerplanar graph has less than $2n$ edges.

Properties of Non-Crossing Families

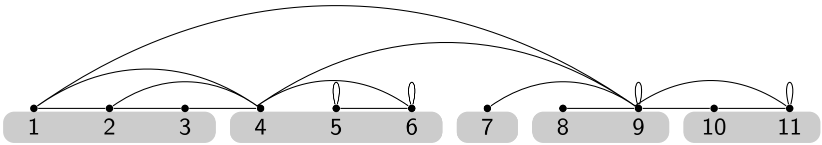
Fact

Let us partition $\{1, \ldots, n\}$ into b contiguous blocks.
A family of non-crossing pairs involves less than $3b$ pairs of blocks (block-pairs).



Fact

Let us partition $\{1, \ldots, n\}$ into b contiguous blocks.
A family of non-crossing pairs involves less than $3b$ pairs of blocks (block-pairs).

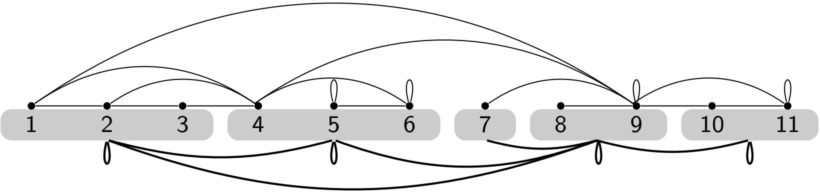


Proof.
Block-pairs form the edge set of an outerplanar graph on b vertices.

Properties of Non-Crossing Families

Fact

Let us partition $\{1, \ldots, n\}$ into b contiguous blocks.
A family of non-crossing pairs involves less than $3b$ pairs of blocks (block-pairs).



Proof.

Block-pairs form the edge set of an outerplanar graph on b vertices.

Limited LCE Queries (CPM'16)

Intuition: An $\mathrm{LCE}(i, j)$ query is easy if the LCE value is small.

Limited LCE Queries (CPM'16)

Intuition: An $\mathrm{LCE}(i, j)$ query is easy if the LCE value is small.

Limited LCE queries:

$$\mathrm{LCE}_{\leq \ell}(i, j) = \min(\ell, \mathrm{LCE}(i, j))$$

Limited LCE Queries (CPM'16)

Intuition: An $\mathrm{LCE}(i,j)$ query is easy if the LCE value is small.

Limited LCE queries:

$$\mathrm{LCE}_{\leq\ell}(i,j) = \min(\ell, \mathrm{LCE}(i,j))$$

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

Limited LCE Queries (CPM'16)

Intuition: An $\mathrm{LCE}(i,j)$ query is easy if the LCE value is small.

Limited LCE queries:

$$\mathrm{LCE}_{\leq \ell}(i,j) = \min(\ell, \mathrm{LCE}(i,j))$$

$$\begin{array}{rccccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ w = & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{b} \\ & & \uparrow & & & & & & & \uparrow & & \end{array}$$

$$\mathrm{LCE}_{\leq 3}(2,9) =$$

Intuition: An $\mathrm{LCE}(i, j)$ query is easy if the LCE value is small.

Limited LCE queries:

$$\mathrm{LCE}_{\leq \ell}(i, j) = \min(\ell, \mathrm{LCE}(i, j))$$

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

$$\mathrm{LCE}_{\leq 3}(2, 9) = 2$$

Limited LCE Queries (CPM'16)

Intuition: An $\mathrm{LCE}(i,j)$ query is easy if the LCE value is small.

Limited LCE queries:

$$\mathrm{LCE}_{\leq \ell}(i,j) = \min(\ell, \mathrm{LCE}(i,j))$$

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

$$\mathrm{LCE}_{\leq 3}(2,9) = 2$$

$$\mathrm{LCE}_{\leq 3}(2,7) =$$

Limited LCE Queries (CPM'16)

Intuition: An $\mathrm{LCE}(i, j)$ query is easy if the LCE value is small.

Limited LCE queries:

$$\mathrm{LCE}_{\leq \ell}(i, j) = \min(\ell, \mathrm{LCE}(i, j))$$

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

$$\mathrm{LCE}_{\leq 3}(2, 9) = 2$$

$$\mathrm{LCE}_{\leq 3}(2, 7) = 3$$

Intuition: An $\mathrm{LCE}(i,j)$ query is easy if the LCE value is small.

Limited LCE queries:

$$\mathrm{LCE}_{\leq \ell}(i,j) = \min(\ell, \mathrm{LCE}(i,j))$$

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

$$\mathrm{LCE}_{\leq 3}(2,9) = 2$$

$$\mathrm{LCE}_{\leq 3}(2,7) = 3$$

Lemma (Gawrychowski et al., CPM 2016)

A sequence of $\mathrm{LCE}_{\leq \ell_q}(i_q, j_q)$ queries can be answered on-line in $\mathcal{O}((n + \sum \log \ell_q) \cdot \alpha(n))$ time in the general alphabet model.

Answering Non-Crossing LCE Queries: Idea

Non-crossing LCE queries in $\mathcal{O}(n \log \log n \cdot \alpha(n))$ time:

Answering Non-Crossing LCE Queries: Idea

Non-crossing LCE queries in $\mathcal{O}(n \log\log n \cdot \alpha(n))$ time:

1. Introduce a partition into blocks of size 2^k for $k = \lfloor \log\log n \rfloor$.

Answering Non-Crossing LCE Queries: Idea

Non-crossing LCE queries in $\mathcal{O}(n \log\log n \cdot \alpha(n))$ time:

1. Introduce a partition into blocks of size 2^k for $k = \lfloor \log\log n \rfloor$.
2. For short queries ($\mathrm{LCE}(i,j) \leq 3 \cdot 2^k$), apply limited LCE:
   - $\mathcal{O}(\log\log n \cdot \alpha(n))$ amortized time per query.

Answering Non-Crossing LCE Queries: Idea

Non-crossing LCE queries in $\mathcal{O}(n \log\log n \cdot \alpha(n))$ time:

1. Introduce a partition into blocks of size 2^k for $k = \lfloor \log\log n \rfloor$.
2. For short queries $(\mathrm{LCE}(i,j) \le 3 \cdot 2^k)$, apply limited LCE:
   - $\mathcal{O}(\log\log n \cdot \alpha(n))$ amortized time per query.
3. For long queries, exploit the fact that many such queries involving the same block-pair yield certain structure.

Answering Non-Crossing LCE Queries: Idea

Non-crossing LCE queries in $\mathcal{O}(n \log\log n \cdot \alpha(n))$ time:

1. Introduce a partition into blocks of size 2^k for $k = \lfloor \log\log n \rfloor$.
2. For short queries $(\mathrm{LCE}(i,j) \le 3 \cdot 2^k)$, apply limited LCE:
   - $\mathcal{O}(\log\log n \cdot \alpha(n))$ amortized time per query.
3. For long queries, exploit the fact that many such queries involving the same block-pair yield certain structure.
4. For each block-pair (out of $\mathcal{O}(n/\log n)$ involved):
   - **learn** the structure using $\mathcal{O}(1)$ unlimited LCE queries $\mathcal{O}(\log n \cdot \alpha(n))$ amortize time per query
   - **exploit** the structure answering the remaining queries $\mathcal{O}(1)$ time per query.

Answering Non-Crossing LCE Queries: Idea

Non-crossing LCE queries in $\mathcal{O}(n \log\log n \cdot \alpha(n))$ time:

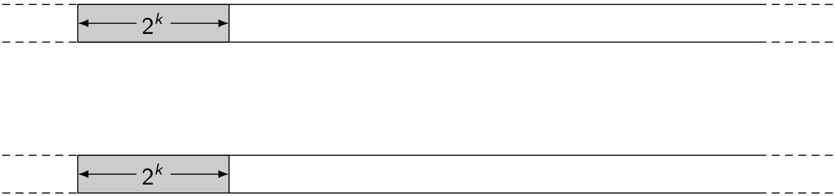
1. Introduce a partition into blocks of size 2^k for $k = \lfloor \log\log n \rfloor$.
2. For short queries ($\mathrm{LCE}(i,j) \leq 3 \cdot 2^k$), apply limited LCE:
   - $\mathcal{O}(\log\log n \cdot \alpha(n))$ amortized time per query.
3. For long queries, exploit the fact that many such queries involving the same block-pair yield certain structure.
4. For each block-pair (out of $\mathcal{O}(n/\log n)$ involved):
   - **learn** the structure using $\mathcal{O}(1)$ unlimited LCE queries $\mathcal{O}(\log n \cdot \alpha(n))$ amortize time per query
   - **exploit** the structure answering the remaining queries $\mathcal{O}(1)$ time per query.

Non-crossing LCE queries in $\mathcal{O}(n\alpha(n))$ time:

- apply the idea above to $\mathcal{O}(\log n)$ levels,
- blocks of length 2^k in level k ($k = 0, \ldots, \log n$).

Intuition:

- **learn** the structure using $\mathcal{O}(1)$ (unlimited) LCE queries,
- **exploit** the structure answering the remaining queries.

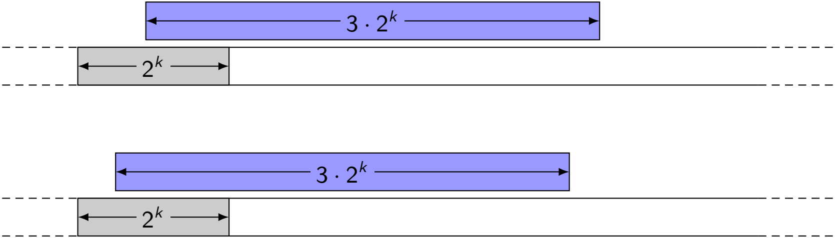


Intuition:

- **learn** the structure using $\mathcal{O}(1)$ (unlimited) LCE queries,
- **exploit** the structure answering the remaining queries.

Algorithm:

- Initially, no structure known.



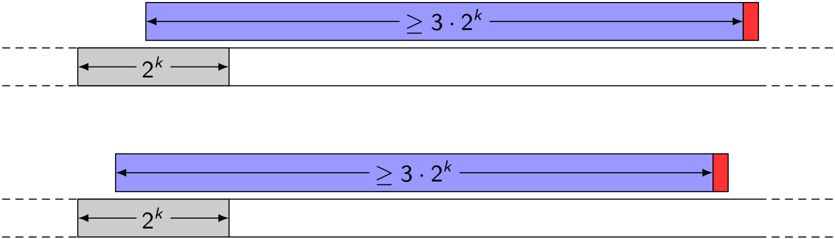
Answering Long Queries Involving a Block-Pair

Intuition:

- **learn** the structure using $\mathcal{O}(1)$ (unlimited) LCE queries,
- **exploit** the structure answering the remaining queries.

Algorithm:

- Initially, no structure known.
- We have to **learn** the answer to the first query.



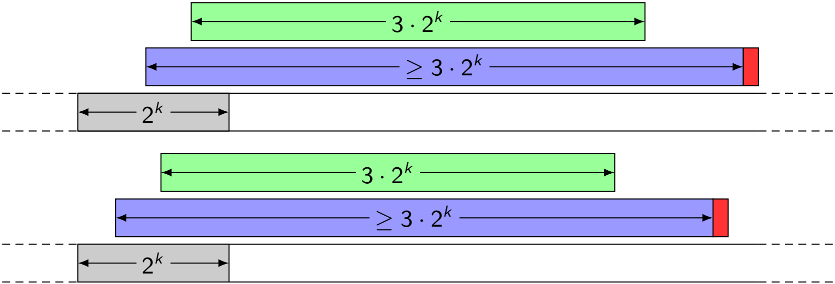
Answering Long Queries Involving a Block-Pair

Intuition:

- **learn** the structure using $\mathcal{O}(1)$ (unlimited) LCE queries,
- **exploit** the structure answering the remaining queries.

Algorithm:

- Initially, no structure known.
- We have to **learn** the answer to the first query.
- The result can be exploited for queries with the same shift.



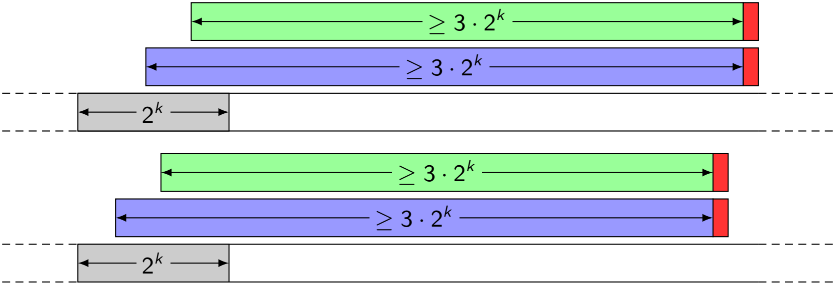
Answering Long Queries Involving a Block-Pair

Intuition:

- **learn** the structure using $\mathcal{O}(1)$ (unlimited) LCE queries,
- **exploit** the structure answering the remaining queries.

Algorithm:

- Initially, no structure known.
- We have to **learn** the answer to the first query.
- The result can be exploited for queries with the same shift.



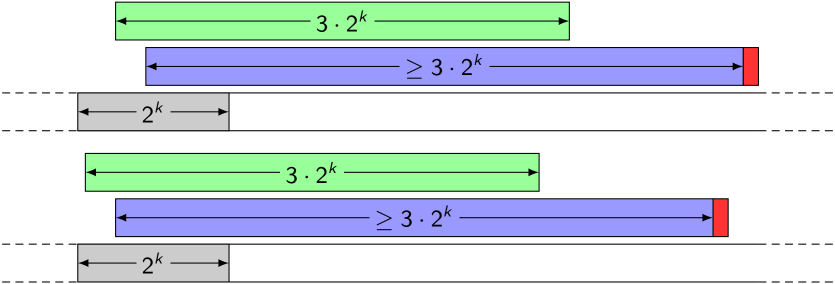
Answering Long Queries Involving a Block-Pair

Intuition:

- **learn** the structure using $\mathcal{O}(1)$ (unlimited) LCE queries,
- **exploit** the structure answering the remaining queries.

Algorithm:

- Initially, no structure known.
- We have to **learn** the answer to the first query.
- The result can be exploited for queries with the same shift.

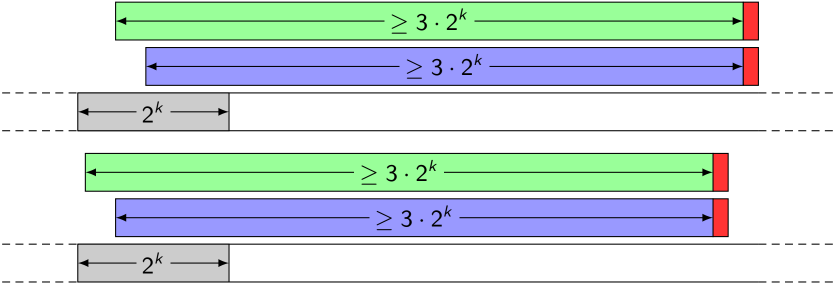


Intuition:

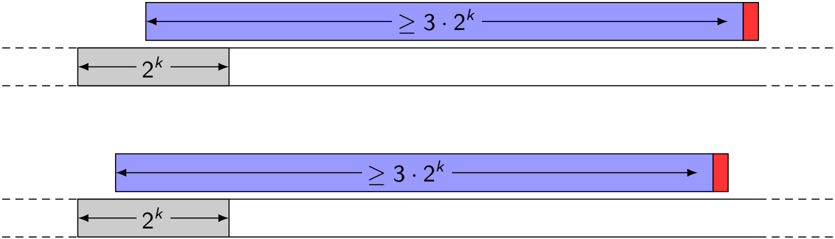
- **learn** the structure using $\mathcal{O}(1)$ (unlimited) LCE queries,
- **exploit** the structure answering the remaining queries.

Algorithm:

- Initially, no structure known.
- We have to **learn** the answer to the first query.
- The result can be exploited for queries with the same shift.



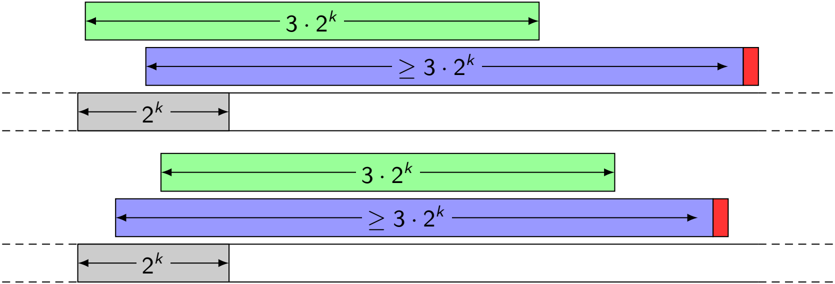
Algorithm (continued):



Answering Long Queries Involving a Block-Pair

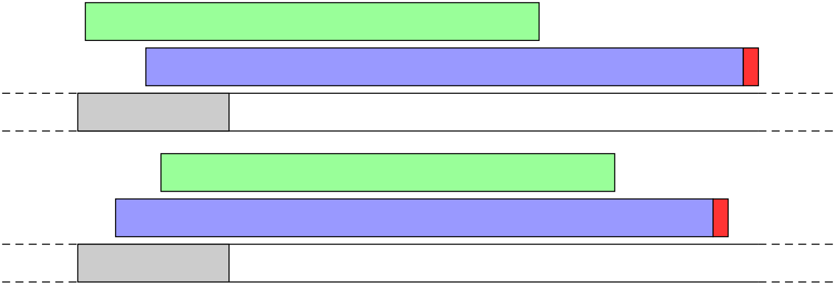
Algorithm (continued):

- Long query with a different shift yields periodic structure.



Algorithm (continued):

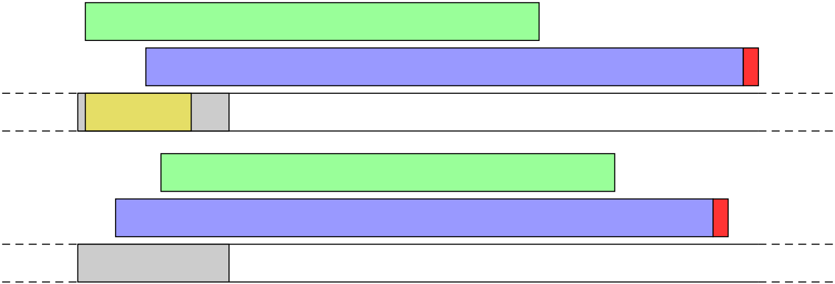
- Long query with a different shift yields periodic structure.



Answering Long Queries Involving a Block-Pair

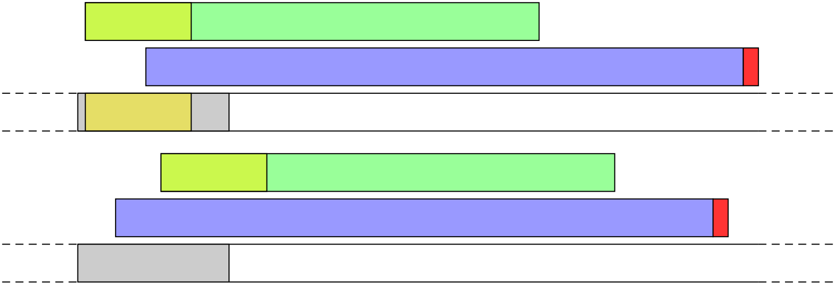
Algorithm (continued):

- Long query with a different shift yields periodic structure.



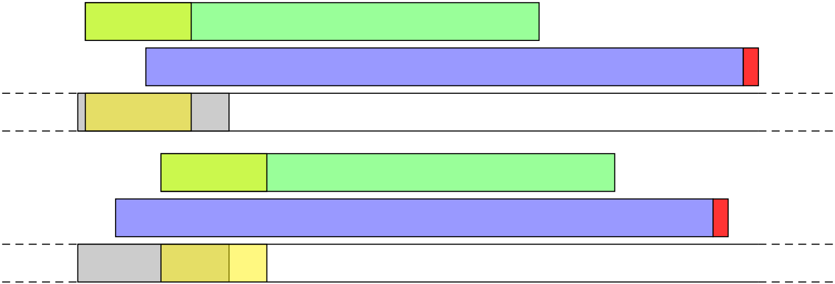
Algorithm (continued):

- Long query with a different shift yields periodic structure.



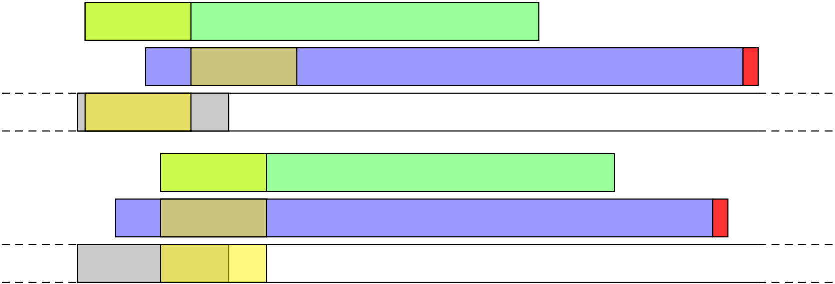
Algorithm (continued):

- Long query with a different shift yields periodic structure.



Algorithm (continued):

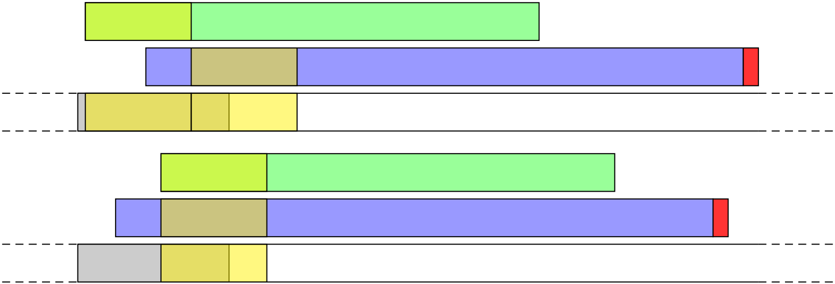
- Long query with a different shift yields periodic structure.



Answering Long Queries Involving a Block-Pair

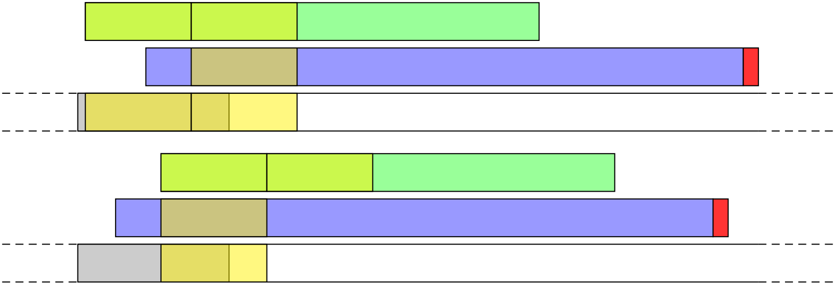
Algorithm (continued):

- Long query with a different shift yields periodic structure.



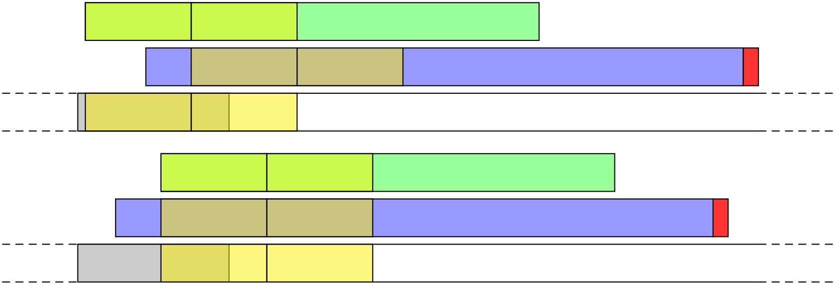
Algorithm (continued):

- Long query with a different shift yields periodic structure.



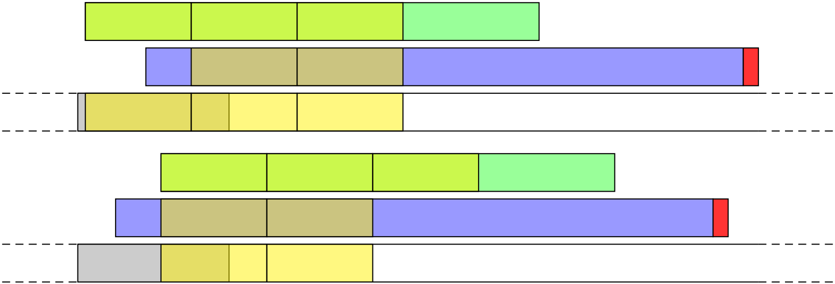
Algorithm (continued):

- Long query with a different shift yields periodic structure.



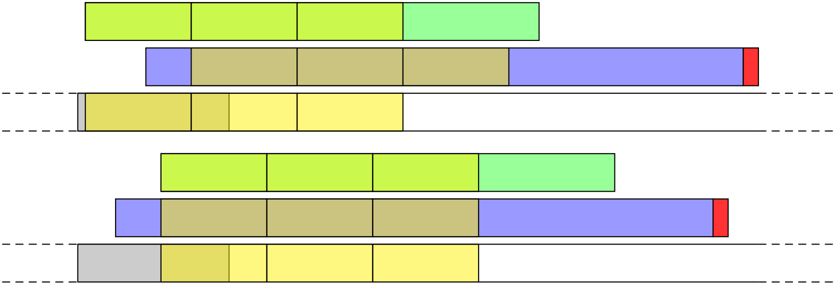
Algorithm (continued)**:**

- Long query with a different shift yields periodic structure.



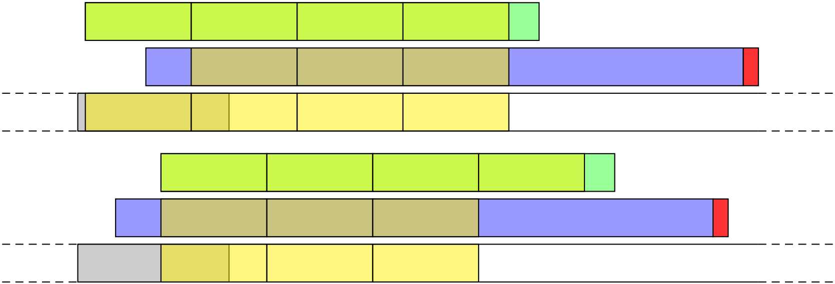
Algorithm (continued):

- Long query with a different shift yields periodic structure.



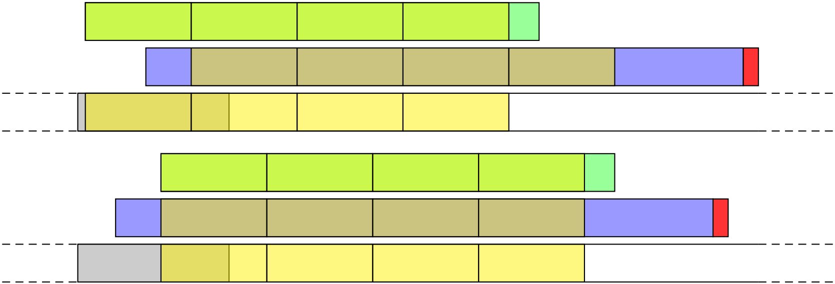
Algorithm (continued):

- Long query with a different shift yields periodic structure.



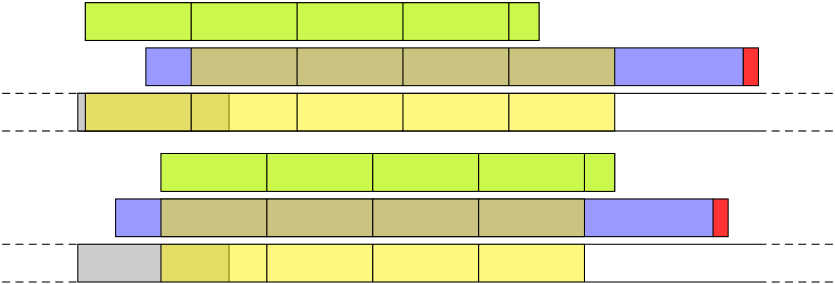
Algorithm (continued):

- Long query with a different shift yields periodic structure.



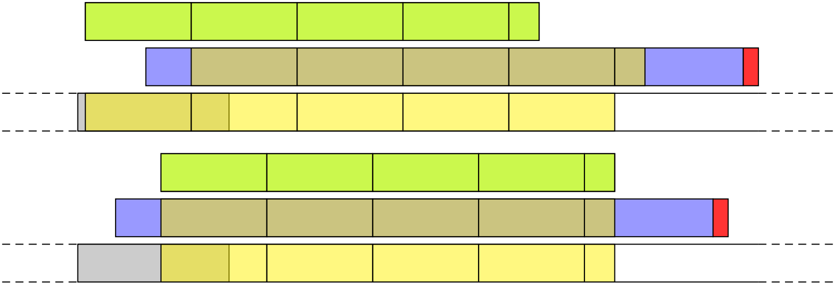
Algorithm (continued):

- Long query with a different shift yields periodic structure.



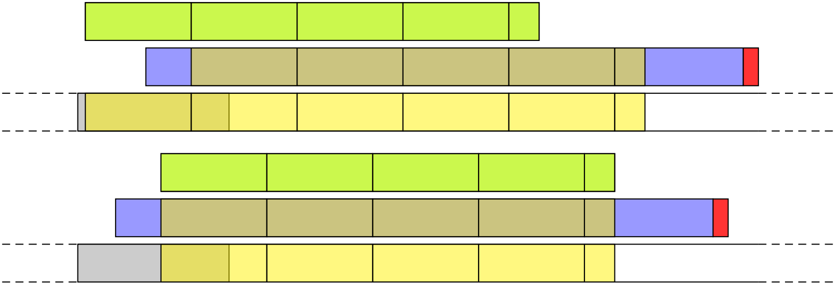
Algorithm (continued):

- Long query with a different shift yields periodic structure.



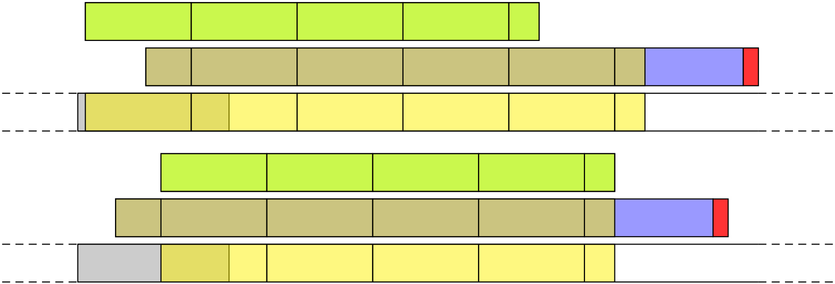
Algorithm (continued):

- Long query with a different shift yields periodic structure.



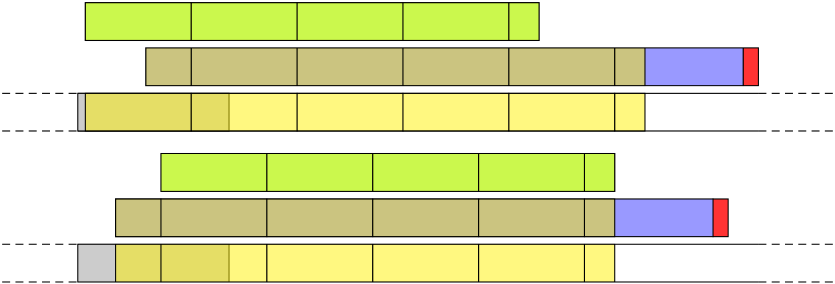
Algorithm (continued):

- Long query with a different shift yields periodic structure.



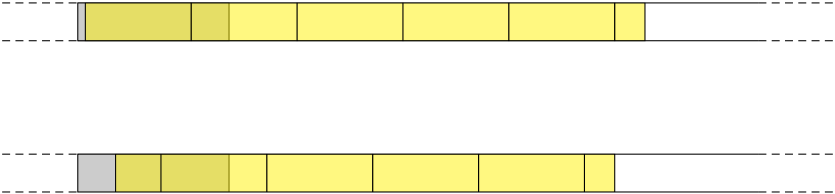
Algorithm (continued):

- Long query with a different shift yields periodic structure.



Algorithm (continued):

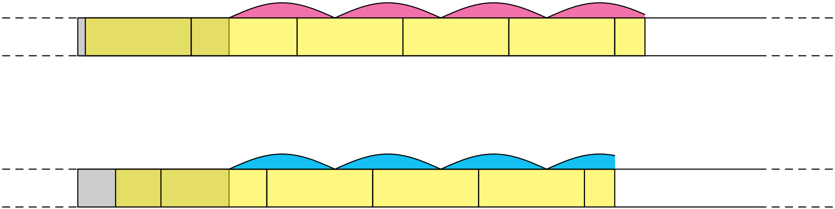
- Long query with a different shift yields periodic structure.
- Fragments following the blocks have a common period p.



Answering Long Queries Involving a Block-Pair

Algorithm (continued)**:**

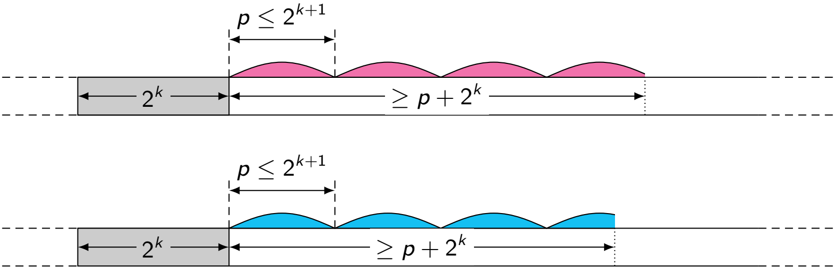
- Long query with a different shift yields periodic structure.
- Fragments following the blocks have a common period p.



Answering Long Queries Involving a Block-Pair

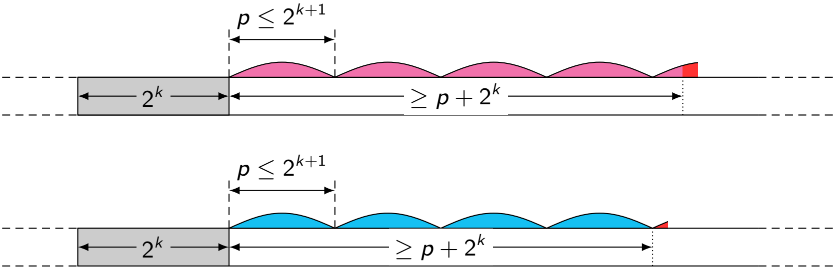
Algorithm (continued)**:**

- Long query with a different shift yields periodic structure.
- Fragments following the blocks have a common period p.



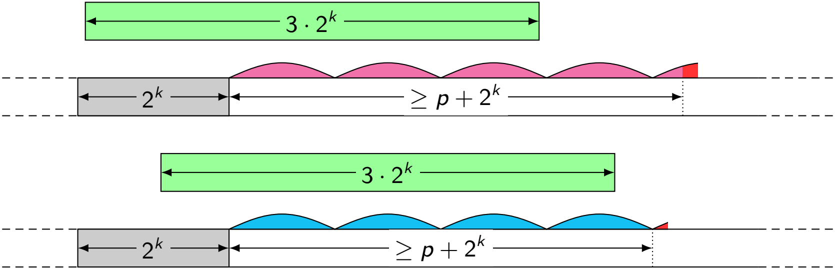
Algorithm (continued)**:**

- Long query with a different shift yields periodic structure.
- Fragments following the blocks have a common period p.
- We use LCE queries to **learn** how far the period continues.



Algorithm (continued)**:**

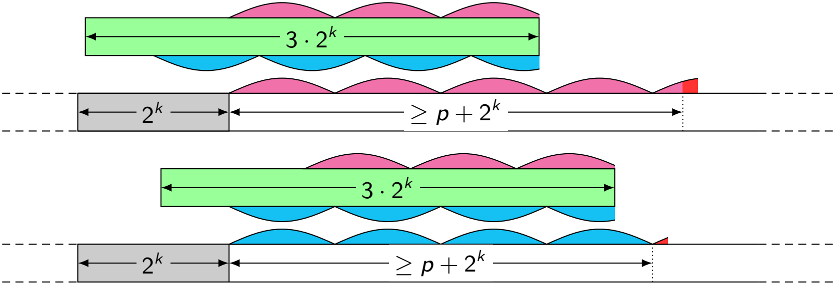
- Long query with a different shift yields periodic structure.
- Fragments following the blocks have a common period p.
- We use LCE queries to **learn** how far the period continues.
- This structure suffices to answer almost all long queries.



Answering Long Queries Involving a Block-Pair

Algorithm (continued)**:**

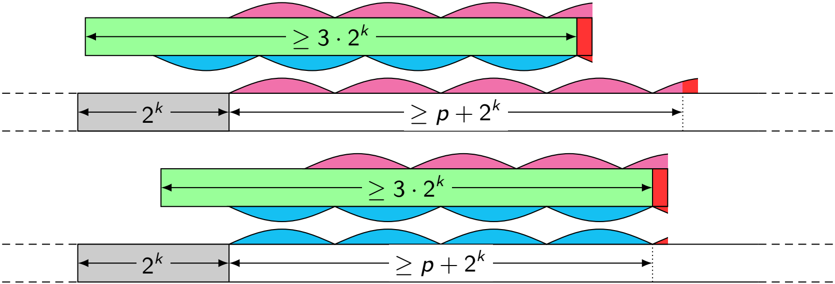
- Long query with a different shift yields periodic structure.
- Fragments following the blocks have a common period p.
- We use LCE queries to **learn** how far the period continues.
- This structure suffices to answer almost all long queries.



Answering Long Queries Involving a Block-Pair

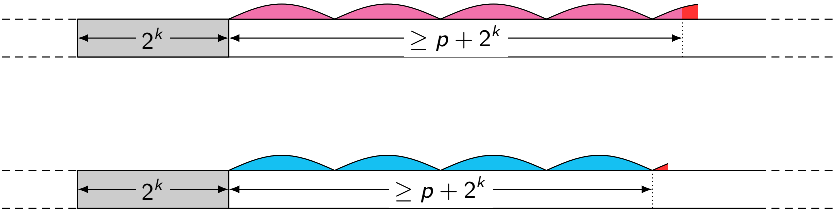
Algorithm (continued):

- Long query with a different shift yields periodic structure.
- Fragments following the blocks have a common period p.
- We use LCE queries to **learn** how far the period continues.
- This structure suffices to answer almost all long queries.



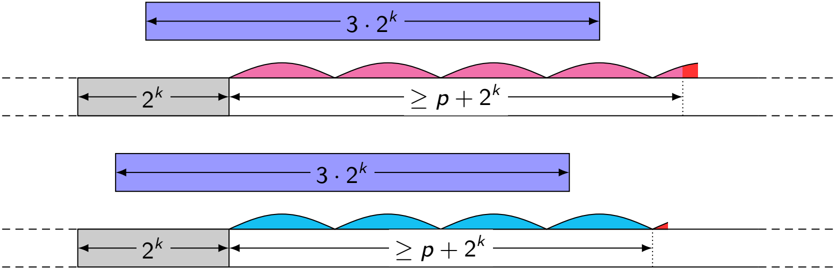
Algorithm (continued)**:**

- Long query with a different shift yields periodic structure.
- Fragments following the blocks have a common period p.
- We use LCE queries to **learn** how far the period continues.
- This structure suffices to answer almost all long queries.



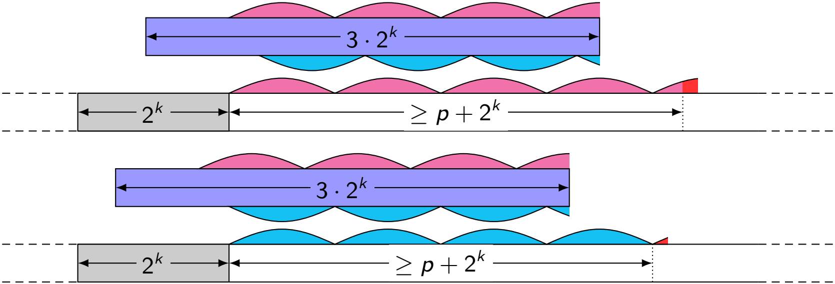
Algorithm (continued)**:**

- Long query with a different shift yields periodic structure.
- Fragments following the blocks have a common period p.
- We use LCE queries to **learn** how far the period continues.
- This structure suffices to answer almost all long queries.



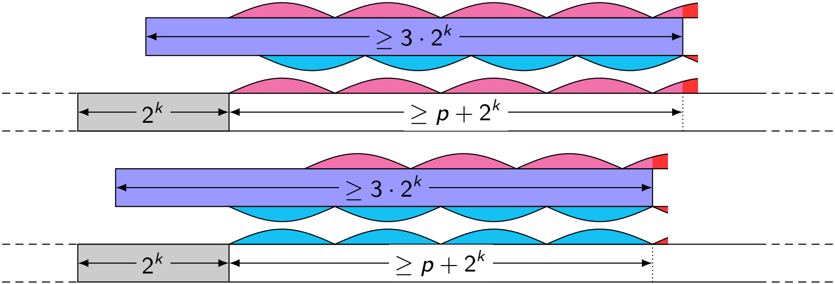
Algorithm (continued)**:**

- Long query with a different shift yields periodic structure.
- Fragments following the blocks have a common period p.
- We use LCE queries to **learn** how far the period continues.
- This structure suffices to answer almost all long queries.



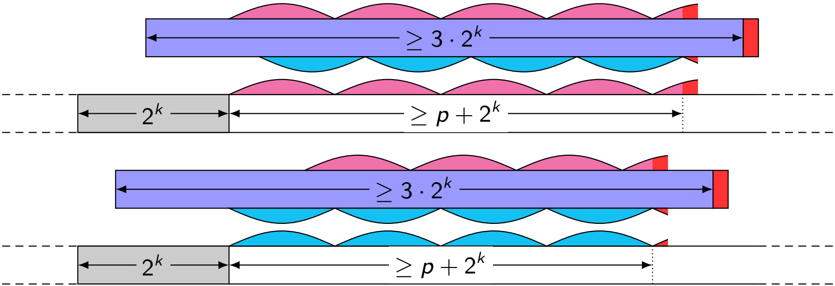
Algorithm (continued)**:**

- Long query with a different shift yields periodic structure.
- Fragments following the blocks have a common period p.
- We use LCE queries to **learn** how far the period continues.
- This structure suffices to answer almost all long queries.
- If period-break positions meet, we need to **learn** the answer.



Algorithm (continued)**:**

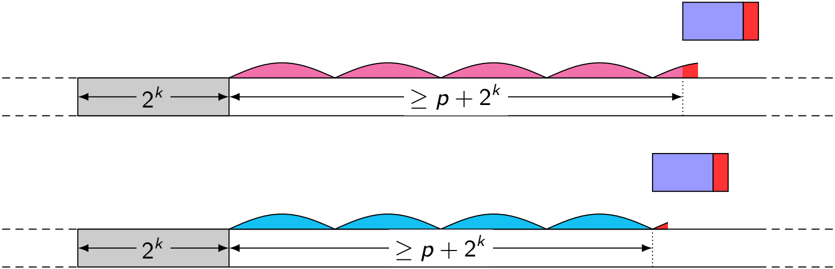
- Long query with a different shift yields periodic structure.
- Fragments following the blocks have a common period p.
- We use LCE queries to **learn** how far the period continues.
- This structure suffices to answer almost all long queries.
- If period-break positions meet, we need to **learn** the answer.



Answering Long Queries Involving a Block-Pair

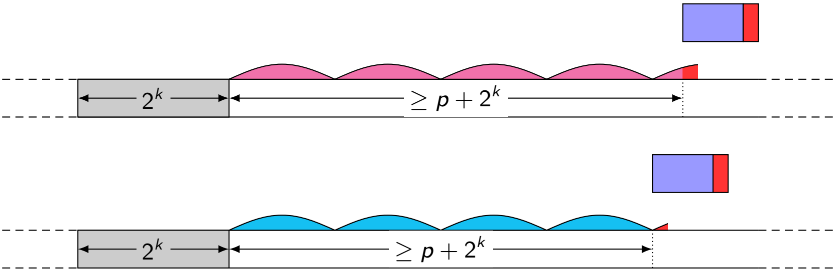
Algorithm (continued)**:**

- Long query with a different shift yields periodic structure.
- Fragments following the blocks have a common period p.
- We use LCE queries to **learn** how far the period continues.
- This structure suffices to answer almost all long queries.
- If period-break positions meet, we need to **learn** the answer.
- We have enough structure to answer all long queries.

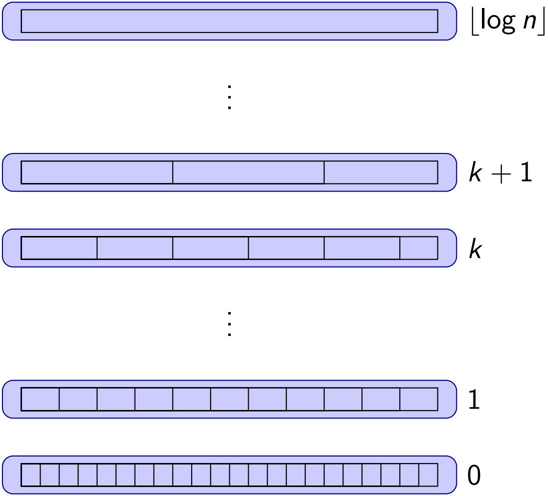


Algorithm (continued)**:**

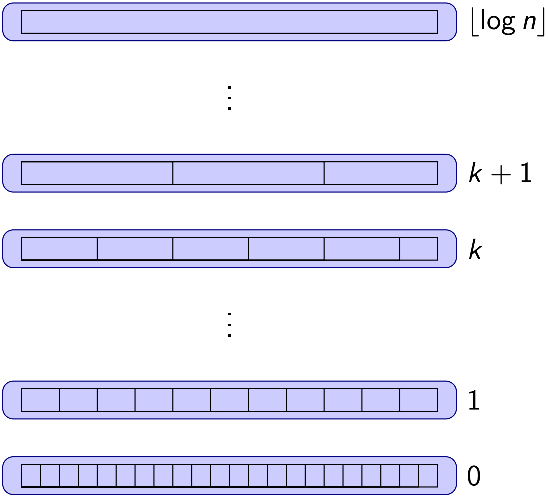
- Long query with a different shift yields periodic structure.
- Fragments following the blocks have a common period p.
- We use LCE queries to **learn** how far the period continues.
- This structure suffices to answer almost all long queries.
- If period-break positions meet, we need to **learn** the answer.
- We have enough structure to answer all long queries.



Answering Non-Crossing LCE Queries: Overview

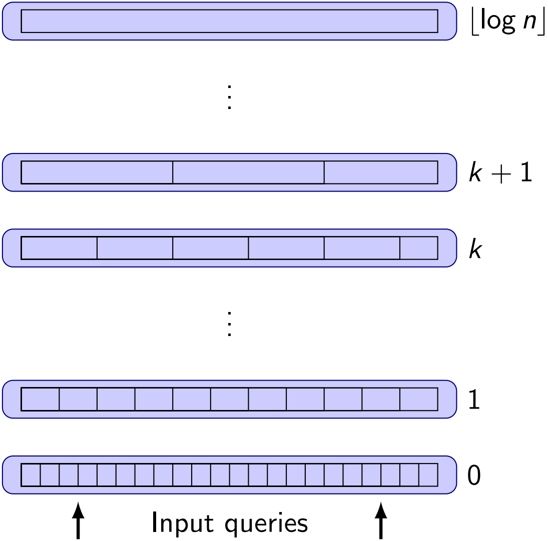


- Blocks of size 2^k at level k.

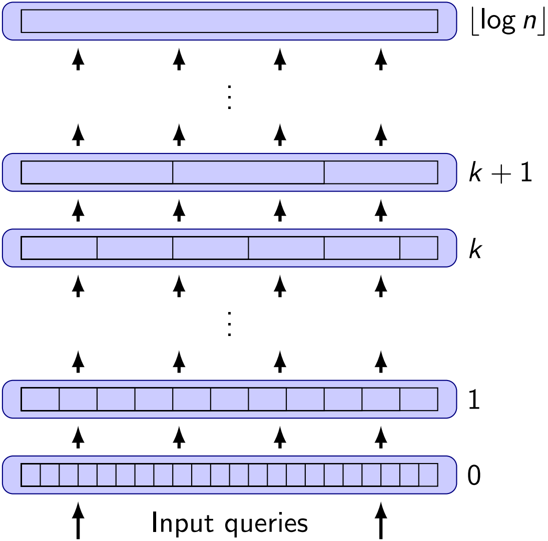


Answering Non-Crossing LCE Queries: Overview

- Blocks of size 2^k at level k.
- Input queries passed to level 0.

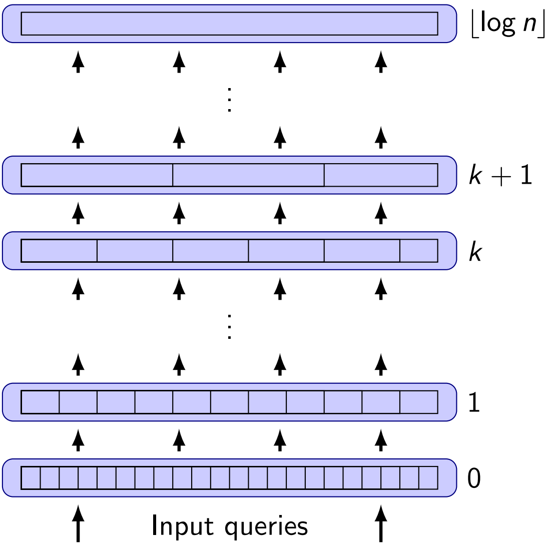


- Blocks of size 2^k at level k.
- Input queries passed to level 0.
- Level k **learns** by asking level $k+1$.



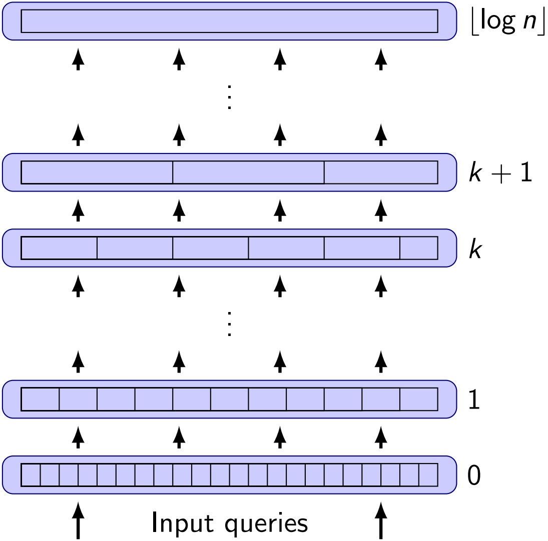
Answering Non-Crossing LCE Queries: Overview

- Blocks of size 2^k at level k.
- Input queries passed to level 0.
- Level k **learns** by asking level $k+1$.
- Queries asked to level k:
  - input queries,
  - $\mathrm{LCE}(i,j)$ for $|i-j| \le 2^k$.



Answering Non-Crossing LCE Queries: Overview

- Blocks of size 2^k at level k.
- Input queries passed to level 0.
- Level k **learns** by asking level $k+1$.
- Queries asked to level k:
  - input queries,
  - $\mathrm{LCE}(i,j)$ for $|i-j| \le 2^k$.
- Block-pairs involved at level k form a non-crossing family.



Running Time Analysis

1. $\mathcal{O}(n\alpha(n))$ amortized preprocessing time (for $\mathrm{LCE}_{\leq \ell}$ queries).

Running Time Analysis

1. $\mathcal{O}(n\alpha(n))$ amortized preprocessing time (for $\mathrm{LCE}_{\leq \ell}$ queries).
2. Each level k answers at most $24n/2^k$ queries:
   - queries in level $k-1$ involve $3n/2^{k-1}$ block-pairs;
   - each block-pairs triggers at most 4 queries to level k.

Running Time Analysis

1. $\mathcal{O}(n\alpha(n))$ amortized preprocessing time (for $\mathrm{LCE}_{\leq \ell}$ queries).
2. Each level k answers at most $24n/2^k$ queries:
   - queries in level $k-1$ involve $3n/2^{k-1}$ block-pairs;
   - each block-pairs triggers at most 4 queries to level k.
3. Each query to level k takes $\mathcal{O}(k\alpha(n))$ amortized time:
   - a $\mathrm{LCE}_{\leq 3 \cdot 2^k}$ query: $\mathcal{O}(k\alpha(n))$ amortized time
   - answering long queries (using block-pair state): $\mathcal{O}(1)$ time excluding queries triggered on level $k+1$.

Running Time Analysis

1. $\mathcal{O}(n\alpha(n))$ amortized preprocessing time (for $\mathrm{LCE}_{\leq \ell}$ queries).
2. Each level k answers at most $24n/2^k$ queries:
   - queries in level $k-1$ involve $3n/2^{k-1}$ block-pairs;
   - each block-pairs triggers at most 4 queries to level k.
3. Each query to level k takes $\mathcal{O}(k\alpha(n))$ amortized time:
   - a $\mathrm{LCE}_{\leq 3 \cdot 2^k}$ query: $\mathcal{O}(k\alpha(n))$ amortized time
   - answering long queries (using block-pair state): $\mathcal{O}(1)$ time excluding queries triggered on level $k+1$.
4. Total running time:

$$\mathcal{O}\left(n\alpha(n)\right) + \sum_{k=1}^{\log n} \frac{24n}{2^k} \cdot \mathcal{O}(k\alpha(n)) = \mathcal{O}(n\alpha(n)).$$

Running Time Analysis

1. $\mathcal{O}(n\alpha(n))$ amortized preprocessing time (for $\mathrm{LCE}_{\leq \ell}$ queries).
2. Each level k answers at most $24n/2^k$ queries:
   - queries in level $k-1$ involve $3n/2^{k-1}$ block-pairs;
   - each block-pairs triggers at most 4 queries to level k.
3. Each query to level k takes $\mathcal{O}(k\alpha(n))$ amortized time:
   - a $\mathrm{LCE}_{\leq 3\cdot 2^k}$ query: $\mathcal{O}(k\alpha(n))$ amortized time
   - answering long queries (using block-pair state): $\mathcal{O}(1)$ time excluding queries triggered on level $k+1$.
4. Total running time:

$$\mathcal{O}\left(n\alpha(n)\right) + \sum_{k=1}^{\log n} \frac{24n}{2^k} \cdot \mathcal{O}(k\alpha(n)) = \mathcal{O}(n\alpha(n)).$$

Theorem (Our main technical result)

The LCE problem can be solved in $\mathcal{O}(n\alpha(n))$ time in the general alphabet model if the $\mathrm{LCE}(i,j)$ queries are non-crossing.

Runs Computation

The algorithm of Bannai et al. (SODA 2015):

The algorithm of Bannai et al. (SODA 2015):

1. Construct the Lyndon tree of w with respect to the lexicographic order.

Runs Computation

The algorithm of Bannai et al. (SODA 2015):

1. Construct the Lyndon tree of w with respect to the lexicographic order.
2. Check which nodes correspond to Lyndon roots of runs.

The algorithm of Bannai et al. (SODA 2015):

1. Construct the Lyndon tree of w with respect to the lexicographic order.
2. Check which nodes correspond to Lyndon roots of runs.
3. Construct the Lyndon tree of w with respect to the reverse lexicographic order.
4. Check which nodes correspond to Lyndon roots of runs.

The algorithm of Bannai et al. (SODA 2015):

1. Construct the Lyndon tree of w with respect to the lexicographic order.
   - $\mathcal{O}(n)$ non-crossing LCE queries in w.
2. Check which nodes correspond to Lyndon roots of runs.
3. Construct the Lyndon tree of w with respect to the reverse lexicographic order.
4. Check which nodes correspond to Lyndon roots of runs.

Runs Computation

The algorithm of Bannai et al. (SODA 2015):

1. Construct the Lyndon tree of w with respect to the lexicographic order.
   - $\mathcal{O}(n)$ non-crossing LCE queries in w.
2. Check which nodes correspond to Lyndon roots of runs.
   - $\mathcal{O}(n)$ non-crossing LCE queries in w (extension to the right),
   - $\mathcal{O}(n)$ non-crossing LCE queries in w^R (extension to the left).
3. Construct the Lyndon tree of w with respect to the reverse lexicographic order.
4. Check which nodes correspond to Lyndon roots of runs.

Runs Computation

The algorithm of Bannai et al. (SODA 2015):

1. Construct the Lyndon tree of w with respect to the lexicographic order.
   - $\mathcal{O}(n)$ non-crossing LCE queries in w.
2. Check which nodes correspond to Lyndon roots of runs.
   - $\mathcal{O}(n)$ non-crossing LCE queries in w (extension to the right),
   - $\mathcal{O}(n)$ non-crossing LCE queries in w^R (extension to the left).
3. Construct the Lyndon tree of w with respect to the reverse lexicographic order.
   - $\mathcal{O}(n)$ non-crossing LCE queries in w.
4. Check which nodes correspond to Lyndon roots of runs.
   - $\mathcal{O}(n)$ non-crossing LCE queries in w (extension to the right),
   - $\mathcal{O}(n)$ non-crossing LCE queries in w^R (extension to the left).

Runs Computation

The algorithm of Bannai et al. (SODA 2015):

1. Construct the Lyndon tree of w with respect to the lexicographic order.
   - $\mathcal{O}(n)$ non-crossing LCE queries in w.
2. Check which nodes correspond to Lyndon roots of runs.
   - $\mathcal{O}(n)$ non-crossing LCE queries in w (extension to the right),
   - $\mathcal{O}(n)$ non-crossing LCE queries in w^R (extension to the left).
3. Construct the Lyndon tree of w with respect to the reverse lexicographic order.
   - $\mathcal{O}(n)$ non-crossing LCE queries in w.
4. Check which nodes correspond to Lyndon roots of runs.
   - $\mathcal{O}(n)$ non-crossing LCE queries in w (extension to the right),
   - $\mathcal{O}(n)$ non-crossing LCE queries in w^R (extension to the left).

Theorem

Runs in a word of length n over a general ordered alphabet can be computed in $\mathcal{O}(n\alpha(n))$ time.

Questions?

Thank you for your attention!