Longest Palindromic Substring in Sublinear Time

Panagiotis Charalampopoulos[1] Solon P. Pissis[2,3]
Jakub Radoszewski[4]

[1]Reichman University, Herzliya, Israel → Birkbeck, University of London, UK
[2]CWI, Amsterdam, The Netherlands
[3]Vrije Universiteit, Amsterdam, The Netherlands
[4]University of Warsaw, Poland

CPM 2022

The Longest Palindromic Substring problem

Longest Palindromic Substring (LPS)

Input: String S of length n over alphabet $[0, \sigma) \subseteq [0, n)$.

Output: $[i, j]$ such that $S[i \mathinner{.\,.} j]$ is a longest palindromic substring of S.

$$S = \texttt{abcaabcbaabda}$$

Time	Space (words)	Paper
$\mathcal{O}(n)$	$\mathcal{O}(n)$	Manacher, JACM 1975
$\mathcal{O}(n)$	$\mathcal{O}(n)$	Gusfield, Textbook 1997
$\mathcal{O}(n \log \sigma / \log n)$	$\mathcal{O}(n \log \sigma / \log n)$	**This paper**

Our algorithm works in sublinear time if $\sigma = 2^{o(\log n)}$.

The suffix tree

The compacted trie of all the suffixes of the string.

$S = \overset{0\,1\,2\,3\,4\,5\,6}{\texttt{CAGAGA\#}}$

0 : CAGAGA#

1 : AGAGA#

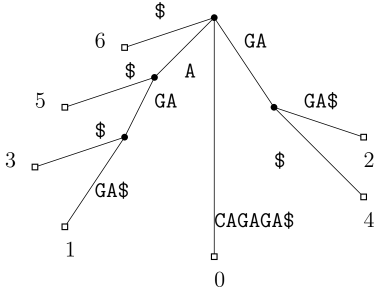
2 : GAGA#

3 : AGA#

4 : GA#

5 : A#

6 : #



The suffix tree of S

We assume that the terminating symbol # is lex-smallest.

Theorem (Farach, FOCS 1997)

The suffix tree of S can be constructed in $\mathcal{O}(n)$ time using $\mathcal{O}(n)$ space.

The suffix tree

The compacted trie of all the suffixes of the string.

$$S = \overset{0\,1\,2\,3\,4\,5\,6}{\texttt{CAGAGA\#}}$$

6 : #

5 : A#

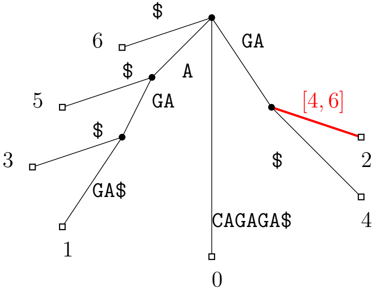
3 : AGA#

1 : AGAGA#

0 : CAGAGA#

2 : GA#

4 : GAGA#



The suffix tree of S

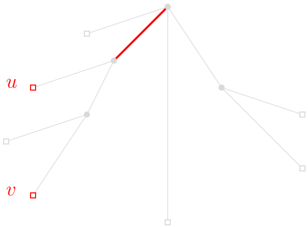
We assume that the terminating symbol # is lex-smallest.

Theorem (Farach, FOCS 1997)

The suffix tree of S can be constructed in $\mathcal{O}(n)$ time using $\mathcal{O}(n)$ space.

Longest common prefix (LCP) queries

PREPROCESS: a string S of length n over alphabet $[0, \sigma) \subseteq [0, n)$
QUERY: a pair (i, j); return the length of the LCP of $(S[i\,..], S[j\,..])$
The lowest common ancestor (LCA) of two nodes u and v is the deepest node that is an ancestor of both u and v.

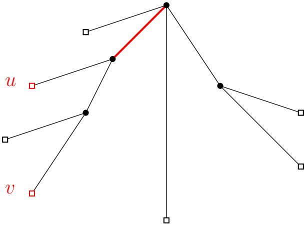


Theorem (Bender and Farach-Colton, LATIN 2000)

Any tree of size $\mathcal{O}(N)$ can be preprocessed in $\mathcal{O}(N)$ time and space so that the LCA of any two nodes can be computed in $\mathcal{O}(1)$ time.

Longest common prefix (LCP) queries

PREPROCESS: a string S of length n over alphabet $[0, \sigma) \subseteq [0, n)$
QUERY: a pair (i, j); return the length of the LCP of $(S[i\,..], S[j\,..])$
The lowest common ancestor (LCA) of two nodes u and v is the deepest node that is an ancestor of both u and v.



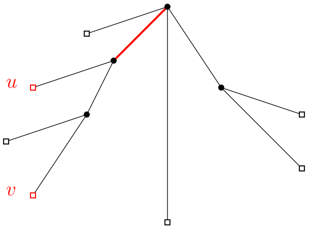
Theorem (Bender and Farach-Colton, LATIN 2000)

Any tree of size $\mathcal{O}(N)$ can be preprocessed in $\mathcal{O}(N)$ time and space so that the LCA of any two nodes can be computed in $\mathcal{O}(1)$ time.

Longest common prefix (LCP) queries

PREPROCESS: a string S of length n over alphabet $[0, \sigma) \subseteq [0, n)$
QUERY: a pair (i, j); return the length of the LCP of $(S[i\,..], S[j\,..])$

The lowest common ancestor (LCA) of two nodes u and v is the deepest node that is an ancestor of both u and v.



Theorem (Bender and Farach-Colton, LATIN 2000)

Any tree of size $\mathcal{O}(N)$ can be preprocessed in $\mathcal{O}(N)$ time and space so that the LCA of any two nodes can be computed in $\mathcal{O}(1)$ time.

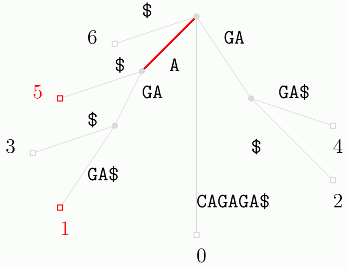
Longest common prefix (LCP) queries

PREPROCESS: a string S of length n over alphabet $[0, \sigma) \subseteq [0, n)$

QUERY: a pair (i, j); return the length of the LCP of $(S[i\,..], S[j\,..])$

Example

Let $S =$ `CAGAGA$`. Let $(1, 5)$ be the query. The answer is $1 = |\texttt{A}|$.



Theorem (Landau and Vishkin, TCS 1986)

LCP queries in S can be answered in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$ time preprocessing.

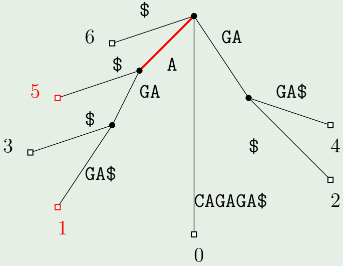
Longest common prefix (LCP) queries

PREPROCESS: a string S of length n over alphabet $[0, \sigma) \subseteq [0, n)$

QUERY: a pair (i, j); return the length of the LCP of $(S[i \mathinner{.\,.}], S[j \mathinner{.\,.}])$

Example

Let $S = \texttt{CAGAGA\$}$. Let $(1, 5)$ be the query. The answer is $1 = |\texttt{A}|$.



Theorem (Landau and Vishkin, TCS 1986)

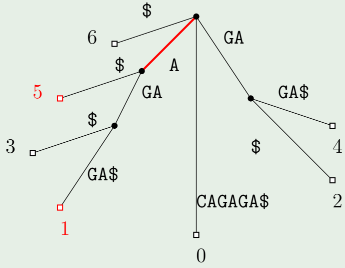
LCP queries in S can be answered in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$ time preprocessing.

Longest common prefix (LCP) queries

PREPROCESS: a string S of length n over alphabet $[0,\sigma) \subseteq [0,n)$
QUERY: a pair (i,j); return the length of the LCP of $(S[i\,..], S[j\,..])$

Example

Let $S = \texttt{CAGAGA\$}$. Let $(1,5)$ be the query. The answer is $1 = |\texttt{A}|$.

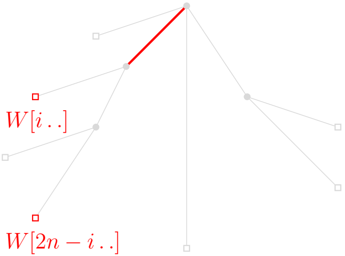


Theorem (Landau and Vishkin, TCS 1986)

LCP queries in S can be answered in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$ time preprocessing.

Gusfield's algorithm for LPS

- Construct the suffix tree of $W = S\#S^R\$$.
- Preprocess the suffix tree for LCA queries.
- Say we are interested in odd-length palindromes.
- Answer LCP queries for $W[i\,..]$ and $W[2n-i\,..]$, for all i.



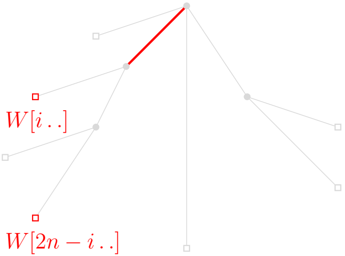
$S = \texttt{C}\underline{\texttt{ATGTA}}\texttt{TT}$

$S^R = \texttt{TTATGTAC}$

$W = S\#S^R\$ = \texttt{CAT}\underline{\overset{3}{\texttt{G}}\texttt{TA}}\texttt{TT\#TTAT}\underline{\overset{13}{\texttt{G}}\texttt{TA}}\texttt{C\$}$

Gusfield's algorithm for LPS

- Construct the suffix tree of $W = S\#S^R\$$.
- Preprocess the suffix tree for LCA queries.
- Say we are interested in odd-length palindromes.
- Answer LCP queries for $W[i\,..]$ and $W[2n-i\,..]$, for all i.



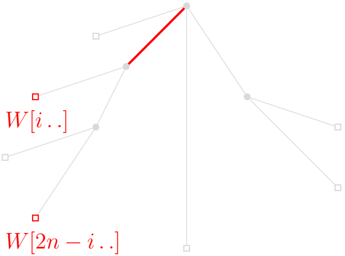
$S = \texttt{C}\underline{\texttt{ATGTA}}\texttt{TT}$

$S^R = \texttt{TTATGTAC}$

$W = S\#S^R\$ = \texttt{CAT}\underline{\overset{3}{\texttt{G}}\texttt{TA}}\texttt{TT\#TTAT}\underline{\overset{13}{\texttt{G}}\texttt{TA}}\texttt{C\$}$

Gusfield's algorithm for LPS

- Construct the suffix tree of $W = S\#S^R\$$.
- Preprocess the suffix tree for LCA queries.
- Say we are interested in odd-length palindromes.
- Answer LCP queries for $W[i\,..]$ and $W[2n-i\,..]$, for all i.



$S =$ CATGTATT

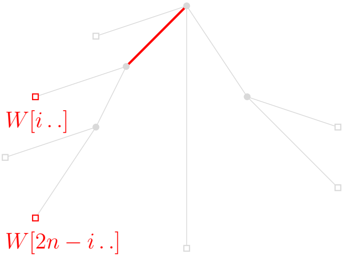
$S^R =$ TTATGTAC

3 13

$W = S\#S^R\$ =$ CATGTATT#TTATGTAC$

Gusfield's algorithm for LPS

- Construct the suffix tree of $W = S\#S^R\$$.
- Preprocess the suffix tree for LCA queries.
- Say we are interested in odd-length palindromes.
- Answer LCP queries for $W[i\,..]$ and $W[2n-i\,..]$, for all i.



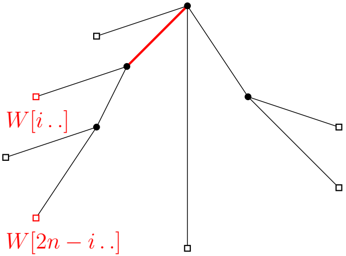
$S = \texttt{C}\underline{\texttt{AT}\texttt{GTA}}\texttt{TT}$

$S^R = \texttt{TTATGTAC}$

$W = S\#S^R\$ = \texttt{CAT}\overset{3}{\underline{\texttt{GTA}}}\texttt{TT\#TTAT}\overset{13}{\underline{\texttt{GTA}}}\texttt{C\$}$

Gusfield's algorithm for LPS

- Construct the suffix tree of $W = S\#S^R\$$.
- Preprocess the suffix tree for LCA queries.
- Say we are interested in odd-length palindromes.
- Answer LCP queries for $W[i\,..]$ and $W[2n-i\,..]$, for all i.

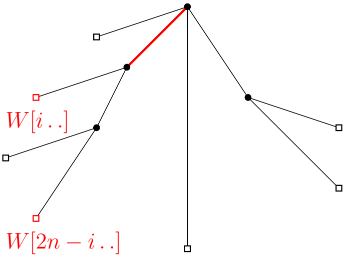


$S = \texttt{CATGTATT}$

$S^R = \texttt{TTATGTAC}$

$W = S\#S^R\$ = \texttt{CAT}\overset{3}{\texttt{G}}\texttt{TATT\#TTAT}\overset{13}{\texttt{G}}\texttt{TAC\$}$

Gusfield's algorithm for LPS



$S = \texttt{C}\underline{\texttt{AT}\underset{\bullet}{\texttt{G}}\texttt{TA}}\texttt{TT}$

$S^R = \texttt{TTATGTAC}$

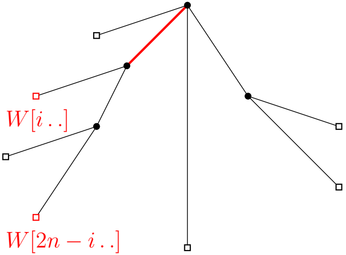
$W = S\#S^R\$ = \texttt{CAT}\overset{3}{\underline{\texttt{GTA}}}\underset{\times}{\texttt{T}}\texttt{T\#TTAT}\overset{13}{\underline{\texttt{GTA}}}\underset{\times}{\texttt{C}}\texttt{\$}$

- A longest LCP represents a longest odd-length palindrome.
- Even-length palindromes are handled analogously.
- Take the longer of the two as the globally longest.

Theorem (Gusfield, Textbook 1997)

A longest palindromic substring in S can be computed in $\mathcal{O}(n)$ time.

Gusfield's algorithm for LPS



$S = \texttt{C}\underline{\texttt{ATGTA}}\texttt{TT}$

$S^R = \texttt{TTATGTAC}$

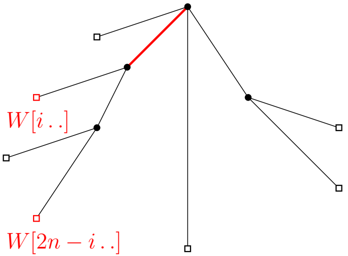
$W = S\#S^R\$ = \texttt{CAT}\overset{3}{\underline{\texttt{GTA}}}\underset{\times}{\texttt{T}}\texttt{T\#TTAT}\overset{13}{\underline{\texttt{GTA}}}\underset{\times}{\texttt{C}}\texttt{\$}$

- A longest LCP represents a longest odd-length palindrome.
- Even-length palindromes are handled analogously.
- Take the longer of the two as the globally longest.

Theorem (Gusfield, Textbook 1997)

A longest palindromic substring in S can be computed in $\mathcal{O}(n)$ time.

Gusfield's algorithm for LPS



$S = \texttt{C}\underline{\texttt{ATGTA}}\texttt{TT}$

$S^R = \texttt{TTATGTAC}$

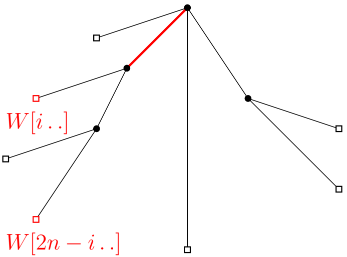
$W = S\#S^R\$ = \texttt{CAT}\underline{\overset{3}{\texttt{G}}\texttt{TA}}\underset{\times}{\texttt{T}}\texttt{T\#TTAT}\underline{\overset{13}{\texttt{G}}\texttt{TA}}\underset{\times}{\texttt{C}}\texttt{\$}$

- A longest LCP represents a longest odd-length palindrome.
- Even-length palindromes are handled analogously.
- Take the longer of the two as the globally longest.

Theorem (Gusfield, Textbook 1997)

A longest palindromic substring in S can be computed in $\mathcal{O}(n)$ time.

Gusfield's algorithm for LPS



$S = \texttt{C}\underline{\texttt{ATGTA}}\texttt{TT}$

$S^R = \texttt{TTATGTAC}$

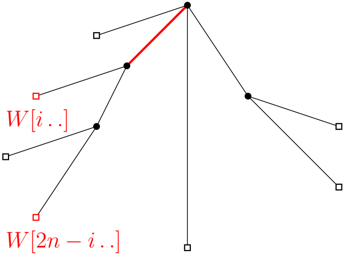
$W = S\#S^R\$ = \texttt{CAT}\underline{\overset{3}{\texttt{G}}\texttt{TA}}\texttt{TT\#TTAT}\underline{\overset{13}{\texttt{G}}\texttt{TA}}\texttt{C\$}$

- A longest LCP represents a longest odd-length palindrome.
- Even-length palindromes are handled analogously.
- Take the longer of the two as the globally longest.

Theorem (Gusfield, Textbook 1997)

A longest palindromic substring in S can be computed in $\mathcal{O}(n)$ time.

Gusfield's algorithm for LPS



$S = \texttt{C}\underline{\texttt{AT}\underset{\bullet}{\texttt{G}}\texttt{TA}}\texttt{TT}$

$S^R = \texttt{TTATGTAC}$

$W = S\#S^R\$ = \texttt{CAT}\underline{\overset{3}{\texttt{G}}\texttt{TA}}\underset{\times}{\texttt{T}}\texttt{T\#TTAT}\underline{\overset{13}{\texttt{G}}\texttt{TA}}\underset{\times}{\texttt{C}}\texttt{\$}$

- A longest LCP represents a longest odd-length palindrome.
- Even-length palindromes are handled analogously.
- Take the longer of the two as the globally longest.

Theorem (Gusfield, Textbook 1997)

A longest palindromic substring in S can be computed in $\mathcal{O}(n)$ time.

Question

Theorem (Kempa and Kociumaka, STOC 2019)

LCP queries in S can be answered in $\mathcal{O}(1)$ time after $\mathcal{O}(n/\log_\sigma n)$ time preprocessing.

Question: Can we improve on Gusfield's textbook algorithm?

Answer: Yes

Question

Theorem (Kempa and Kociumaka, STOC 2019)

LCP queries in S can be answered in $\mathcal{O}(1)$ time after $\mathcal{O}(n/\log_\sigma n)$ time preprocessing.

Question: Can we improve on Gusfield's textbook algorithm?

Answer: Yes

Question

Theorem (Kempa and Kociumaka, STOC 2019)

LCP queries in S can be answered in $\mathcal{O}(1)$ time after $\mathcal{O}(n/\log_\sigma n)$ time preprocessing.

Question: Can we improve on Gusfield's textbook algorithm?

Answer: Yes

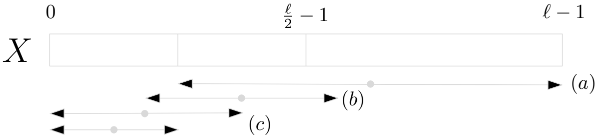
Overview of our new solution

Chunks of length $\ell' = \frac{1}{8} \log_{\sigma} n$ and extended chunks of length $\ell = 4\ell'$
Our algorithm proceeds with processing every chunk separately

Preprocessing stage: tabulation technique

Preprocessing stage

- Consider every distinct length-ℓ string over $[0, \sigma)$
- $\sigma^{\ell} = \sigma^{4\ell'} = \sigma^{\frac{\log_\sigma n}{2}} = \mathcal{O}(\sqrt{n})$ distinct strings
- Each string X is stored in one machine word
- Compute palindromes in X in $\mathcal{O}(\ell)$ time[1]
- For each length-ℓ string X store:
  - (a) a longest palindrome in X
  - (b) a longest palindrome in X that has its center in the 2nd chunk
  - (c) the two longest prefix palindromes of X, if they exist

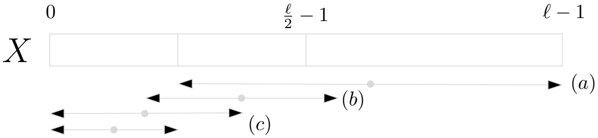


Time: $\mathcal{O}(\sqrt{n} \log_\sigma n)$ and Space: $\mathcal{O}(\sqrt{n})$

[1] Manacher: A New Linear-Time "On-Line" Algorithm for Finding the Smallest Initial Palindrome of a String. J. ACM (1975)

Preprocessing stage

- Consider every distinct length-ℓ string over $[0, \sigma)$
- $\sigma^{\ell} = \sigma^{4\ell'} = \sigma^{\frac{\log_\sigma n}{2}} = \mathcal{O}(\sqrt{n})$ distinct strings
- Each string X is stored in one machine word
- Compute palindromes in X in $\mathcal{O}(\ell)$ time[1]
- For each length-ℓ string X store:
  - (a) a longest palindrome in X
  - (b) a longest palindrome in X that has its center in the 2nd chunk
  - (c) the two longest prefix palindromes of X, if they exist

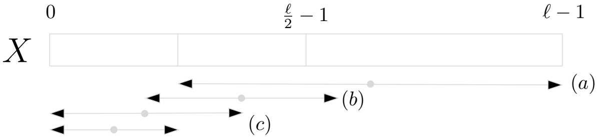


Time: $\mathcal{O}(\sqrt{n} \log_\sigma n)$ and Space: $\mathcal{O}(\sqrt{n})$

[1] Manacher: A New Linear-Time "On-Line" Algorithm for Finding the Smallest Initial Palindrome of a String. J. ACM (1975)

Preprocessing stage

- Consider every distinct length-ℓ string over $[0, \sigma)$
- $\sigma^{\ell} = \sigma^{4\ell'} = \sigma^{\frac{\log_{\sigma} n}{2}} = \mathcal{O}(\sqrt{n})$ distinct strings
- Each string X is stored in one machine word
- Compute palindromes in X in $\mathcal{O}(\ell)$ time[1]
- For each length-ℓ string X store:
  - (a) a longest palindrome in X
  - (b) a longest palindrome in X that has its center in the 2nd chunk
  - (c) the two longest prefix palindromes of X, if they exist

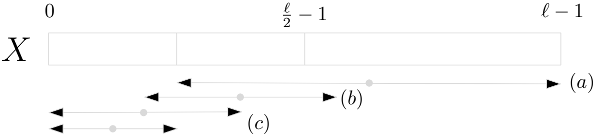


Time: $\mathcal{O}(\sqrt{n} \log_{\sigma} n)$ and Space: $\mathcal{O}(\sqrt{n})$

[1] Manacher: A New Linear-Time "On-Line" Algorithm for Finding the Smallest Initial Palindrome of a String. J. ACM (1975)

Preprocessing stage

- Consider every distinct length-ℓ string over $[0, \sigma)$
- $\sigma^{\ell} = \sigma^{4\ell'} = \sigma^{\frac{\log_\sigma n}{2}} = \mathcal{O}(\sqrt{n})$ distinct strings
- Each string X is stored in one machine word
- Compute palindromes in X in $\mathcal{O}(\ell)$ time[1]
- For each length-ℓ string X store:
  - (a) a longest palindrome in X
  - (b) a longest palindrome in X that has its center in the 2nd chunk
  - (c) the two longest prefix palindromes of X, if they exist

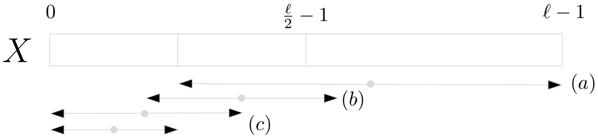


Time: $\mathcal{O}(\sqrt{n}\log_\sigma n)$ and Space: $\mathcal{O}(\sqrt{n})$

[1] Manacher: A New Linear-Time "On-Line" Algorithm for Finding the Smallest Initial Palindrome of a String. J. ACM (1975)

Preprocessing stage

- Consider every distinct length-ℓ string over $[0, \sigma)$
- $\sigma^{\ell} = \sigma^{4\ell'} = \sigma^{\frac{\log_\sigma n}{2}} = \mathcal{O}(\sqrt{n})$ distinct strings
- Each string X is stored in one machine word
- Compute palindromes in X in $\mathcal{O}(\ell)$ time[1]
- For each length-ℓ string X store:
  - (a) a longest palindrome in X
  - (b) a longest palindrome in X that has its center in the 2nd chunk
  - (c) the two longest prefix palindromes of X, if they exist

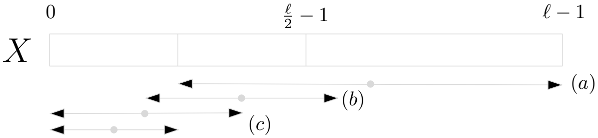


Time: $\mathcal{O}(\sqrt{n} \log_\sigma n)$ and Space: $\mathcal{O}(\sqrt{n})$

[1] Manacher: A New Linear-Time "On-Line" Algorithm for Finding the Smallest Initial Palindrome of a String. J. ACM (1975)

Preprocessing stage

- Consider every distinct length-ℓ string over $[0, \sigma)$
- $\sigma^{\ell} = \sigma^{4\ell'} = \sigma^{\frac{\log_\sigma n}{2}} = \mathcal{O}(\sqrt{n})$ distinct strings
- Each string X is stored in one machine word
- Compute palindromes in X in $\mathcal{O}(\ell)$ time[1]
- For each length-ℓ string X store:
  1. (a) a longest palindrome in X
  2. (b) a longest palindrome in X that has its center in the 2nd chunk
  3. (c) the two longest prefix palindromes of X, if they exist

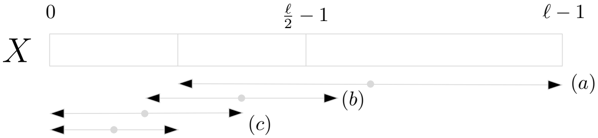


Time: $\mathcal{O}(\sqrt{n}\log_\sigma n)$ and Space: $\mathcal{O}(\sqrt{n})$

[1] Manacher: A New Linear-Time "On-Line" Algorithm for Finding the Smallest Initial Palindrome of a String. J. ACM (1975)

Preprocessing stage

- Consider every distinct length-ℓ string over $[0, \sigma)$
- $\sigma^{\ell} = \sigma^{4\ell'} = \sigma^{\frac{\log_\sigma n}{2}} = \mathcal{O}(\sqrt{n})$ distinct strings
- Each string X is stored in one machine word
- Compute palindromes in X in $\mathcal{O}(\ell)$ time[1]
- For each length-ℓ string X store:
  - (a) a longest palindrome in X
  - (b) a longest palindrome in X that has its center in the 2nd chunk
  - (c) the two longest prefix palindromes of X, if they exist

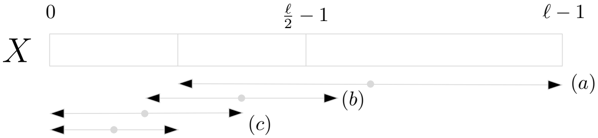


Time: $\mathcal{O}(\sqrt{n} \log_\sigma n)$ and Space: $\mathcal{O}(\sqrt{n})$

[1] Manacher: A New Linear-Time “On-Line” Algorithm for Finding the Smallest Initial Palindrome of a String. J. ACM (1975)

Preprocessing stage

- Consider every distinct length-ℓ string over $[0, \sigma)$
- $\sigma^{\ell} = \sigma^{4\ell'} = \sigma^{\frac{\log_\sigma n}{2}} = \mathcal{O}(\sqrt{n})$ distinct strings
- Each string X is stored in one machine word
- Compute palindromes in X in $\mathcal{O}(\ell)$ time[1]
- For each length-ℓ string X store:
  - (a) a longest palindrome in X
  - (b) a longest palindrome in X that has its center in the 2nd chunk
  - (c) the two longest prefix palindromes of X, if they exist

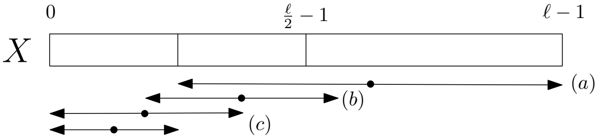


Time: $\mathcal{O}(\sqrt{n} \log_\sigma n)$ and Space: $\mathcal{O}(\sqrt{n})$

[1] Manacher: A New Linear-Time "On-Line" Algorithm for Finding the Smallest Initial Palindrome of a String. J. ACM (1975)

Preprocessing stage

- Consider every distinct length-ℓ string over $[0, \sigma)$
- $\sigma^{\ell} = \sigma^{4\ell'} = \sigma^{\frac{\log_\sigma n}{2}} = \mathcal{O}(\sqrt{n})$ distinct strings
- Each string X is stored in one machine word
- Compute palindromes in X in $\mathcal{O}(\ell)$ time[1]
- For each length-ℓ string X store:
  - (a) a longest palindrome in X
  - (b) a longest palindrome in X that has its center in the 2nd chunk
  - (c) the two longest prefix palindromes of X, if they exist

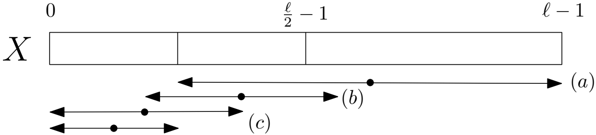


Time: $\mathcal{O}(\sqrt{n} \log_\sigma n)$ and Space: $\mathcal{O}(\sqrt{n})$

[1] Manacher: A New Linear-Time "On-Line" Algorithm for Finding the Smallest Initial Palindrome of a String. J. ACM (1975)

Preprocessing stage

- Consider every distinct length-ℓ string over $[0, \sigma)$
- $\sigma^{\ell} = \sigma^{4\ell'} = \sigma^{\frac{\log_{\sigma} n}{2}} = \mathcal{O}(\sqrt{n})$ distinct strings
- Each string X is stored in one machine word
- Compute palindromes in X in $\mathcal{O}(\ell)$ time[1]
- For each length-ℓ string X store:
  - a. a longest palindrome in X
  - b. a longest palindrome in X that has its center in the 2nd chunk
  - c. the two longest prefix palindromes of X, if they exist



Time: $\mathcal{O}(\sqrt{n} \log_{\sigma} n)$ and Space: $\mathcal{O}(\sqrt{n})$

[1] Manacher: A New Linear-Time "On-Line" Algorithm for Finding the Smallest Initial Palindrome of a String. J. ACM (1975)

Overview of our new solution

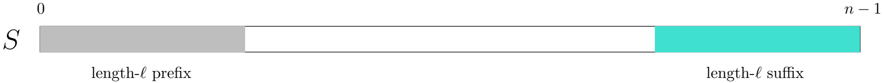
Chunks of length $\ell' = \frac{1}{8} \log_\sigma n$ and extended chunks of length $\ell = 4\ell'$
Our algorithm proceeds with processing every chunk separately

Preprocessing stage: tabulation technique

Main algorithm: for each chunk C in an extended chunk X of S, compute a longest palindrome in S with a center in C using the precomputed information for X, periodicity, and LCP queries

Main algorithm: Basic structure

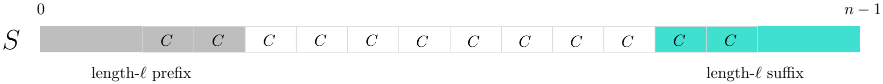
For the corner cases:



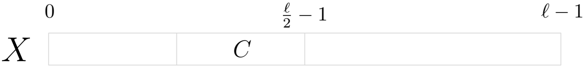
we use the precomputed data (a) to compute a longest palindrome

(any palindrome centered in the first or last $\frac{\ell}{2}$ positions is of length $\leq \ell$)

For the middle part, we decompose:

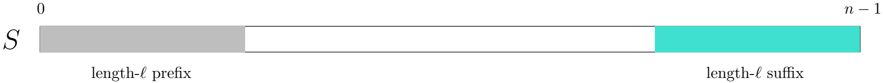


and process chunk C as the second quarter of an extended chunk X:



Main algorithm: Basic structure

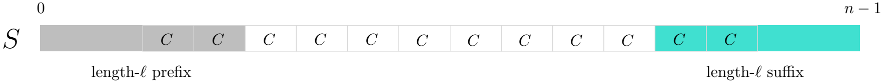
For the corner cases:



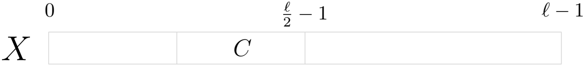
we use the precomputed data (a) to compute a longest palindrome

(any palindrome centered in the first or last $\frac{\ell}{2}$ positions is of length $\leq \ell$)

For the middle part, we decompose:

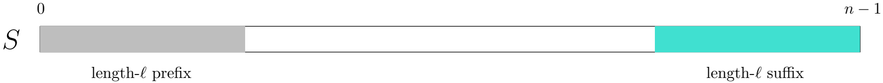


and process chunk C as the second quarter of an extended chunk X:



Main algorithm: Basic structure

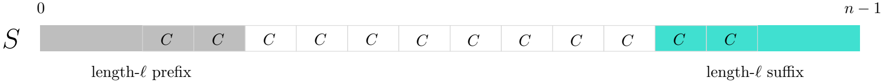
For the corner cases:



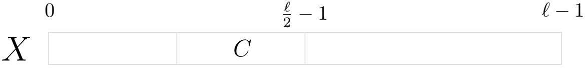
we use the precomputed data (a) to compute a longest palindrome

(any palindrome centered in the first or last $\frac{\ell}{2}$ positions is of length $\leq \ell$)

For the middle part, we decompose:

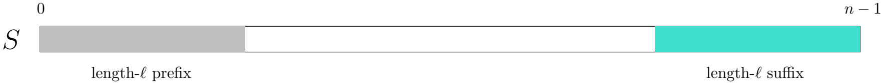


and process chunk C as the second quarter of an extended chunk X:



Main algorithm: Basic structure

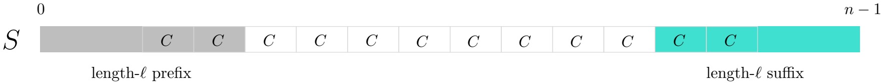
For the corner cases:



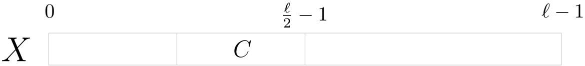
we use the precomputed data (a) to compute a longest palindrome

(any palindrome centered in the first or last $\frac{\ell}{2}$ positions is of length $\leq \ell$)

For the middle part, we decompose:

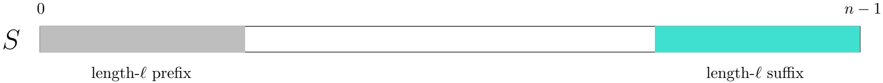


and process chunk C as the second quarter of an extended chunk X:



Main algorithm: Basic structure

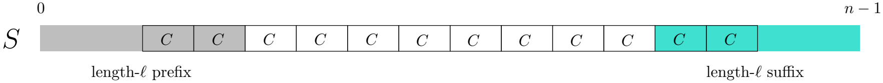
For the corner cases:



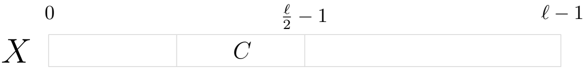
we use the precomputed data (a) to compute a longest palindrome

(any palindrome centered in the first or last $\frac{\ell}{2}$ positions is of length $\leq \ell$)

For the middle part, we decompose:

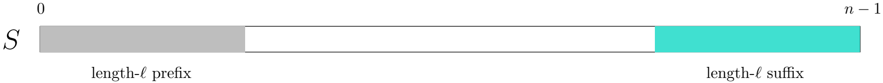


and process chunk C as the second quarter of an extended chunk X:



Main algorithm: Basic structure

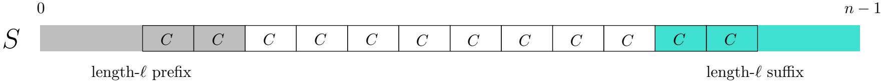
For the corner cases:



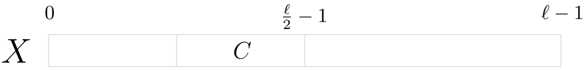
we use the precomputed data (a) to compute a longest palindrome

(any palindrome centered in the first or last $\frac{\ell}{2}$ positions is of length $\leq \ell$)

For the middle part, we decompose:

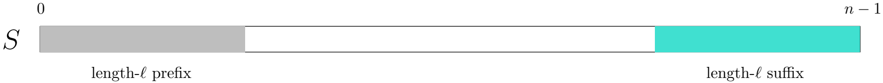


and process chunk C as the second quarter of an extended chunk X:



Main algorithm: Basic structure

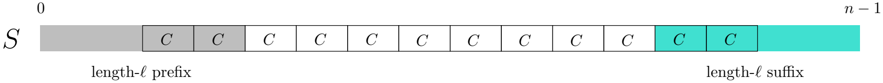
For the corner cases:



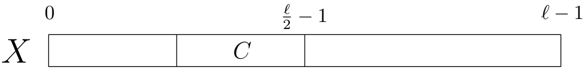
we use the precomputed data (a) to compute a longest palindrome

(any palindrome centered in the first or last $\frac{\ell}{2}$ positions is of length $\leq \ell$)

For the middle part, we decompose:



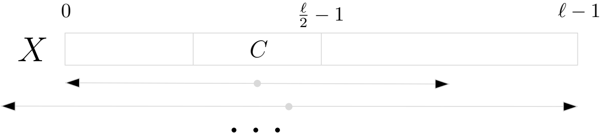
and process chunk C as the second quarter of an extended chunk X:



Maximal palindromes of S with centers in C

$\mathcal{P}_X$: the set of maximal palindromes in S with centers in C that:

- either exceed X
- or are prefixes of X



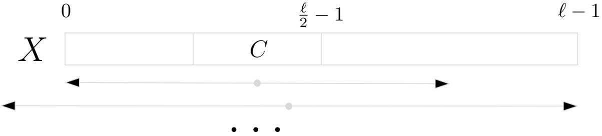
Set $\mathcal{P}_X$ of palindromes with centers in chunck C

Our goal: show that the longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $\mathcal{O}(1)$ LCP queries on substrings of $S\#S^R\$$

Maximal palindromes of S with centers in C

$\mathcal{P}_X$: the set of maximal palindromes in S with centers in C that:

- either exceed X
- or are prefixes of X



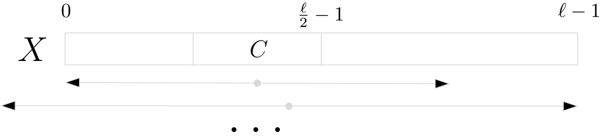
Set $\mathcal{P}_X$ of palindromes with centers in chunck C

Our goal: show that the longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $\mathcal{O}(1)$ LCP queries on substrings of $S\#S^R\$$

Maximal palindromes of S with centers in C

$\mathcal{P}_X$: the set of maximal palindromes in S with centers in C that:

- either exceed X
- or are prefixes of X



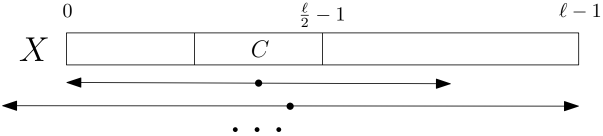
Set $\mathcal{P}_X$ of palindromes with centers in chunck C

Our goal: show that the longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $\mathcal{O}(1)$ LCP queries on substrings of $S\#S^R\$$

Maximal palindromes of S with centers in C

$\mathcal{P}_X$: the set of maximal palindromes in S with centers in C that:

- either exceed X
- or are prefixes of X



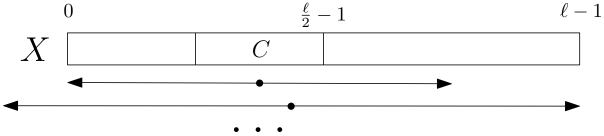
Set $\mathcal{P}_X$ of palindromes with centers in chunck C

Our goal: show that the longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $\mathcal{O}(1)$ LCP queries on substrings of $S\#S^R\$$

Maximal palindromes of S with centers in C

$\mathcal{P}_X$: the set of maximal palindromes in S with centers in C that:

- either exceed X
- or are prefixes of X

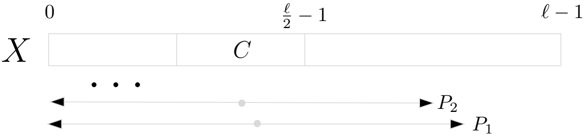


Set $\mathcal{P}_X$ of palindromes with centers in chunck C

Our goal: show that the longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $\mathcal{O}(1)$ LCP queries on substrings of $S\#S^R\$$

Prefix palindromes of X with centers in C

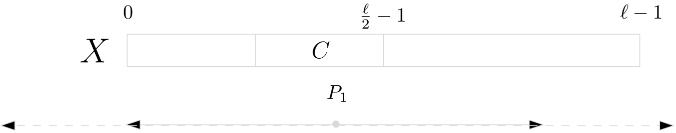
$\mathcal{Q}_X$: the set of palindromes which are prefixes of X with a center in C



Set $\mathcal{Q}_X$ of palindromes with centers in chunck C

Observations:

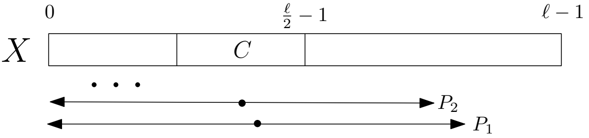
- Each $P \in \mathcal{P}_X$ has a subpalindrome $P' \in \mathcal{Q}_X$ with the same center
- If P_1 does not exist, then $\mathcal{P}_X = \emptyset$
- If P_1 exists but P_2 does not, then $|\mathcal{P}_X| = 1$
    - Compute the only palindrome in $\mathcal{P}_X$ from P_1 using one LCP query



If $|\mathcal{P}_X| = 1$ then ask one LCE query to extend P_1

Prefix palindromes of X with centers in C

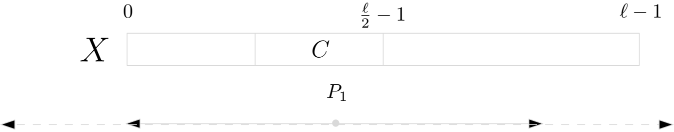
$\mathcal{Q}_X$: the set of palindromes which are prefixes of X with a center in C



Set $\mathcal{Q}_X$ of palindromes with centers in chunck C

Observations:

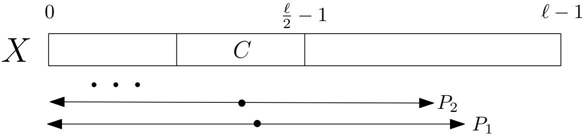
- Each $P \in \mathcal{P}_X$ has a subpalindrome $P' \in \mathcal{Q}_X$ with the same center
- If P_1 does not exist, then $\mathcal{P}_X = \emptyset$
- If P_1 exists but P_2 does not, then $|\mathcal{P}_X| = 1$
  - Compute the only palindrome in $\mathcal{P}_X$ from P_1 using one LCP query



If $|\mathcal{P}_X| = 1$ then ask one LCE query to extend P_1

Prefix palindromes of X with centers in C

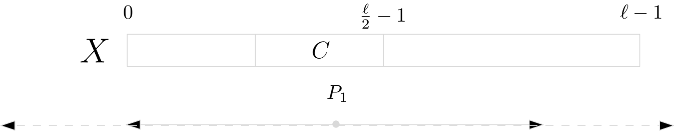
$\mathcal{Q}_X$: the set of palindromes which are prefixes of X with a center in C



Set $\mathcal{Q}_X$ of palindromes with centers in chunck C

Observations:

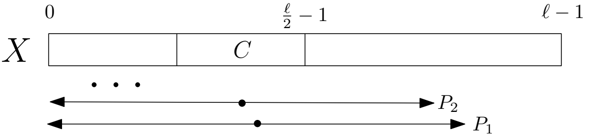
- Each $P \in \mathcal{P}_X$ has a subpalindrome $P' \in \mathcal{Q}_X$ with the same center
- If P_1 does not exist, then $\mathcal{P}_X = \emptyset$
- If P_1 exists but P_2 does not, then $|\mathcal{P}_X| = 1$
  - Compute the only palindrome in $\mathcal{P}_X$ from P_1 using one LCP query



If $|\mathcal{P}_X| = 1$ then ask one LCE query to extend P_1

Prefix palindromes of X with centers in C

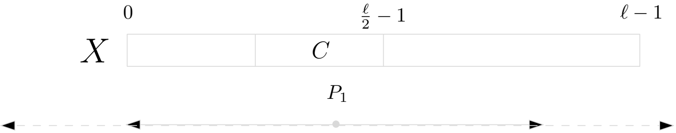
$\mathcal{Q}_X$: the set of palindromes which are prefixes of X with a center in C



Set $\mathcal{Q}_X$ of palindromes with centers in chunck C

Observations:

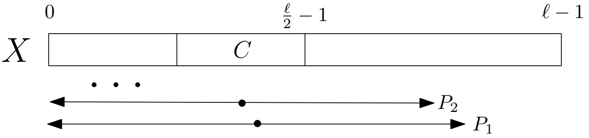
- Each $P \in \mathcal{P}_X$ has a subpalindrome $P' \in \mathcal{Q}_X$ with the same center
- If P_1 does not exist, then $\mathcal{P}_X = \emptyset$
- If P_1 exists but P_2 does not, then $|\mathcal{P}_X| = 1$
  - Compute the only palindrome in $\mathcal{P}_X$ from P_1 using one LCP query



If $|\mathcal{P}_X| = 1$ then ask one LCE query to extend P_1

Prefix palindromes of X with centers in C

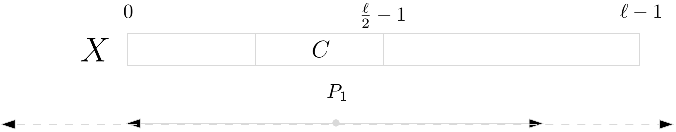
$\mathcal{Q}_X$: the set of palindromes which are prefixes of X with a center in C



Set $\mathcal{Q}_X$ of palindromes with centers in chunck C

Observations:

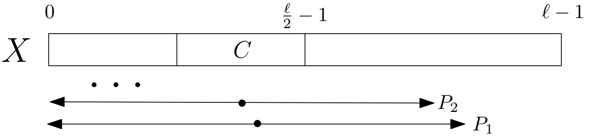
- Each $P \in \mathcal{P}_X$ has a subpalindrome $P' \in \mathcal{Q}_X$ with the same center
- If P_1 does not exist, then $\mathcal{P}_X = \emptyset$
- If P_1 exists but P_2 does not, then $|\mathcal{P}_X| = 1$
  - Compute the only palindrome in $\mathcal{P}_X$ from P_1 using one LCP query



If $|\mathcal{P}_X| = 1$ then ask one LCE query to extend P_1

Prefix palindromes of X with centers in C

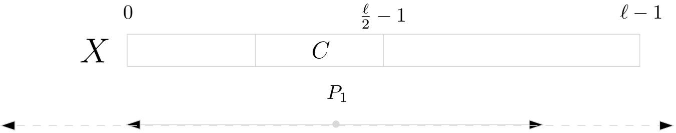
$\mathcal{Q}_X$: the set of palindromes which are prefixes of X with a center in C



Set $\mathcal{Q}_X$ of palindromes with centers in chunck C

Observations:

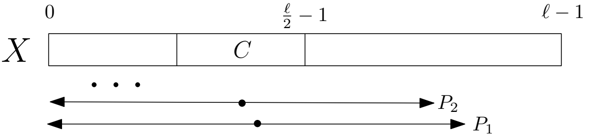
- Each $P \in \mathcal{P}_X$ has a subpalindrome $P' \in \mathcal{Q}_X$ with the same center
- If P_1 does not exist, then $\mathcal{P}_X = \emptyset$
- If P_1 exists but P_2 does not, then $|\mathcal{P}_X| = 1$
    - Compute the only palindrome in $\mathcal{P}_X$ from P_1 using one LCP query



If $|\mathcal{P}_X| = 1$ then ask one LCE query to extend P_1

Prefix palindromes of X with centers in C

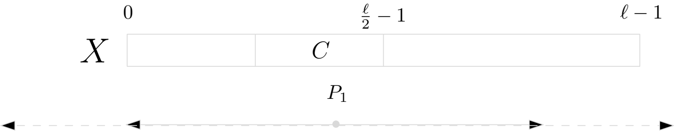
$\mathcal{Q}_X$: the set of palindromes which are prefixes of X with a center in C



Set $\mathcal{Q}_X$ of palindromes with centers in chunck C

Observations:

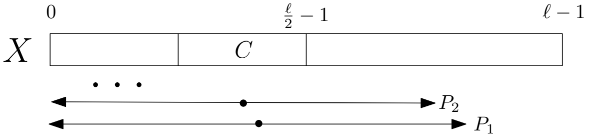
- Each $P \in \mathcal{P}_X$ has a subpalindrome $P' \in \mathcal{Q}_X$ with the same center
- If P_1 does not exist, then $\mathcal{P}_X = \emptyset$
- If P_1 exists but P_2 does not, then $|\mathcal{P}_X| = 1$
  - Compute the only palindrome in $\mathcal{P}_X$ from P_1 using one LCP query



If $|\mathcal{P}_X| = 1$ then ask one LCE query to extend P_1

Prefix palindromes of X with centers in C

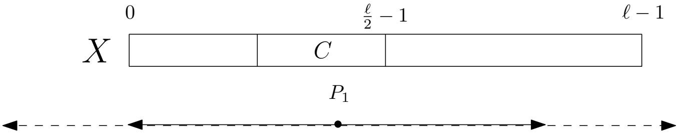
$\mathcal{Q}_X$: the set of palindromes which are prefixes of X with a center in C



Set $\mathcal{Q}_X$ of palindromes with centers in chunck C

Observations:

- Each $P \in \mathcal{P}_X$ has a subpalindrome $P' \in \mathcal{Q}_X$ with the same center
- If P_1 does not exist, then $\mathcal{P}_X = \emptyset$
- If P_1 exists but P_2 does not, then $|\mathcal{P}_X| = 1$
  - Compute the only palindrome in $\mathcal{P}_X$ from P_1 using one LCP query



If $|\mathcal{P}_X| = 1$ then ask one LCE query to extend P_1

Both P_1 and P_2 exist in $\mathcal{Q}_X \iff$ Periodicity

By $\mathrm{per}(V)$ we denote the *smallest* period of a string V.

Example

$V = \mathrm{abbaabba} = \mathrm{abba} \cdot \mathrm{abba}$ and $\mathrm{per}(V) = 4$.

Lemma (Fici et al., JDA 2014)

Let U be a proper prefix of a palindrome V. Then $|V| - |U|$ is a period of V if and only if U is a palindrome. In particular, $\mathrm{per}(V) = |V| - |U|$ if and only if U is the longest palindromic proper prefix of V.

Example

$V = \mathrm{abbaabba}$, $U = \mathrm{abba}$, and $\mathrm{per}(V) = 4$.

As a consequence, we obtain the following, where $p = \mathrm{per}(P_1)$ and $k \geq 0$.

Lemma

The lengths of prefix palindromes in $\mathcal{Q}_X$ are structured: $|P_1| - kp$.

Both P_1 and P_2 exist in $\mathcal{Q}_X \iff$ Periodicity

By per(V) we denote the *smallest* period of a string V.

Example

$V = \texttt{abbaabba} = \texttt{abba} \cdot \texttt{abba}$ and per$(V) = 4$.

Lemma (Fici et al., JDA 2014)

Let U be a proper prefix of a palindrome V. Then $|V| - |U|$ is a period of V if and only if U is a palindrome. In particular, per$(V) = |V| - |U|$ if and only if U is the longest palindromic proper prefix of V.

Example

$V = \texttt{abbaabba}$, $U = \texttt{abba}$, and per$(V) = 4$.

As a consequence, we obtain the following, where $p = \text{per}(P_1)$ and $k \geq 0$.

Lemma

The lengths of prefix palindromes in $\mathcal{Q}_X$ are structured: $|P_1| - kp$.

Both P_1 and P_2 exist in $\mathcal{Q}_X \iff$ Periodicity

By $\text{per}(V)$ we denote the *smallest* period of a string V.

Example

$V = \texttt{abbaabba} = \texttt{abba} \cdot \texttt{abba}$ and $\text{per}(V) = 4$.

Lemma (Fici et al., JDA 2014)

Let U be a proper prefix of a palindrome V. Then $|V| - |U|$ is a period of V if and only if U is a palindrome. In particular, $\text{per}(V) = |V| - |U|$ if and only if U is the longest palindromic proper prefix of V.

Example

$V = \text{abbaabba}$, $U = \text{abba}$, and $\text{per}(V) = 4$.

As a consequence, we obtain the following, where $p = \text{per}(P_1)$ and $k \geq 0$.

Lemma

The lengths of prefix palindromes in $\mathcal{Q}_X$ are structured: $|P_1| - kp$.

Both P_1 and P_2 exist in $\mathcal{Q}_X \iff$ Periodicity

By per(V) we denote the *smallest* period of a string V.

Example

$V = \mathtt{abbaabba} = \mathtt{abba} \cdot \mathtt{abba}$ and per$(V) = 4$.

Lemma (Fici et al., JDA 2014)

Let U be a proper prefix of a palindrome V. Then $|V| - |U|$ is a period of V if and only if U is a palindrome. In particular, per$(V) = |V| - |U|$ if and only if U is the longest palindromic proper prefix of V.

Example

$V = \mathtt{abbaabba}$, $U = \mathtt{abba}$, and per$(V) = 4$.

As a consequence, we obtain the following, where $p = \text{per}(P_1)$ and $k \geq 0$.

Lemma

The lengths of prefix palindromes in $\mathcal{Q}_X$ are structured: $|P_1| - kp$.

Both P_1 and P_2 exist in $\mathcal{Q}_X \iff$ Periodicity

By per(V) we denote the *smallest* period of a string V.

Example

$V = \texttt{abbaabba} = \texttt{abba} \cdot \texttt{abba}$ and $\text{per}(V) = 4$.

Lemma (Fici et al., JDA 2014)

Let U be a proper prefix of a palindrome V. Then $|V| - |U|$ is a period of V if and only if U is a palindrome. In particular, per$(V) = |V| - |U|$ if and only if U is the longest palindromic proper prefix of V.

Example

$V = \texttt{abbaabba}$, $U = \texttt{abba}$, and $\text{per}(V) = 4$.

As a consequence, we obtain the following, where $p = \text{per}(P_1)$ and $k \geq 0$.

Lemma

The lengths of prefix palindromes in $\mathcal{Q}_X$ are structured: $|P_1| - kp$.

Both P_1 and P_2 exist in $\mathcal{Q}_X \iff$ Periodicity

By $\text{per}(V)$ we denote the *smallest* period of a string V.

Example

$V = \texttt{abbaabba} = \texttt{abba} \cdot \texttt{abba}$ and $\text{per}(V) = 4$.

Lemma (Fici et al., JDA 2014)

Let U be a proper prefix of a palindrome V. Then $|V| - |U|$ is a period of V if and only if U is a palindrome. In particular, $\text{per}(V) = |V| - |U|$ if and only if U is the longest palindromic proper prefix of V.

Example

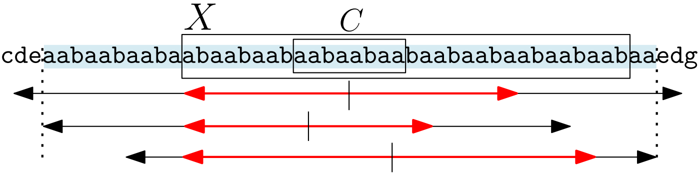
$V = \texttt{abbaabba}$, $U = \texttt{abba}$, and $\text{per}(V) = 4$.

As a consequence, we obtain the following, where $p = \text{per}(P_1)$ and $k \geq 0$.

Lemma

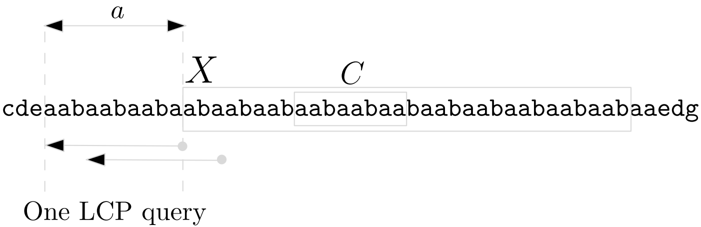
The lengths of prefix palindromes in $\mathcal{Q}_X$ are structured: $|P_1| - kp$.

Main idea: Three prefix palindromes in $\mathcal{Q}_X$ are enough!

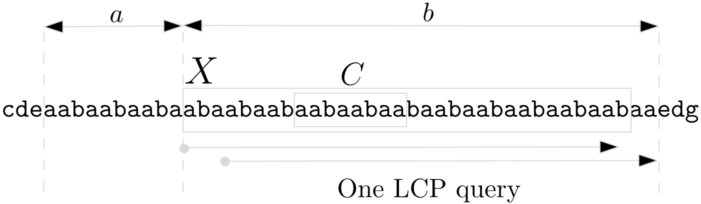


First step: Extending periodicity on both sides

We know $p = |P_1| - |P_2| = \mathrm{per}(P_1)$, so two LCP queries suffice:

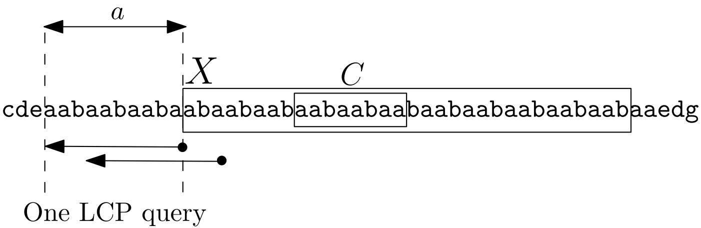


Here $\mathrm{per}(P_1) = 3$ generated by string aba.

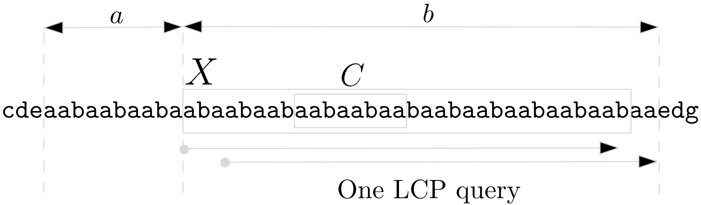


First step: Extending periodicity on both sides

We know $p = |P_1| - |P_2| = \mathrm{per}(P_1)$, so two LCP queries suffice:

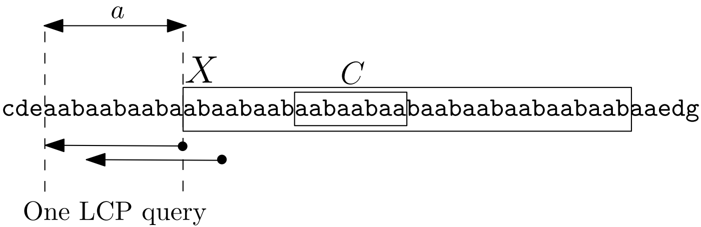


Here $\mathrm{per}(P_1) = 3$ generated by string aba.



First step: Extending periodicity on both sides

We know $p = |P_1| - |P_2| = \text{per}(P_1)$, so two LCP queries suffice:



a

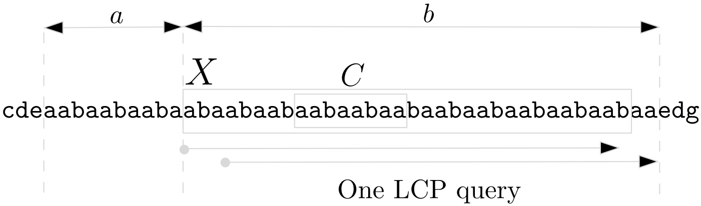
X

C

cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg

One LCP query

Here $\text{per}(P_1) = 3$ generated by string aba.



a

b

X

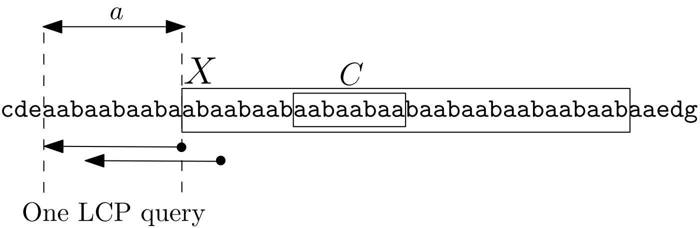
C

cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg

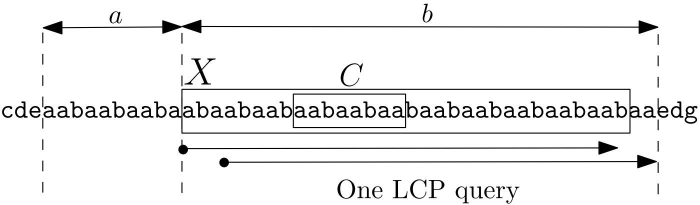
One LCP query

First step: Extending periodicity on both sides

We know $p = |P_1| - |P_2| = \mathrm{per}(P_1)$, so two LCP queries suffice:

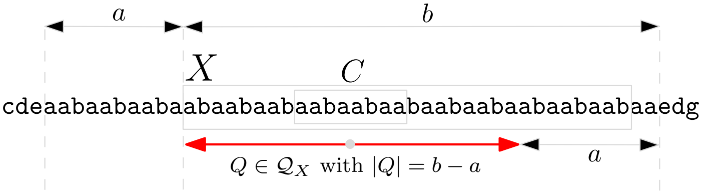


Here $\mathrm{per}(P_1) = 3$ generated by string aba.

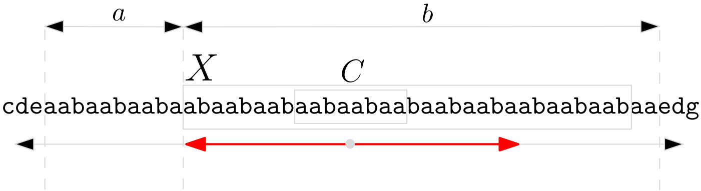


Case 1: Reach the periodicity endpoints simultaneously

Consider $Q \in \mathcal{Q}_X$ with $|Q| = b - a$:


a
b
X
C
cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg
$Q \in \mathcal{Q}_X$ with $|Q| = b - a$
a

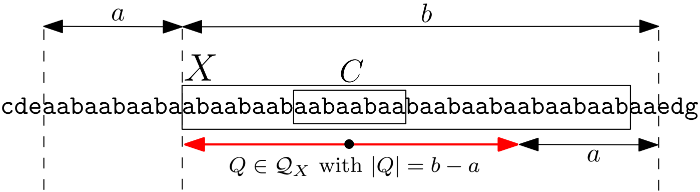
We use one LCP query to extend it.



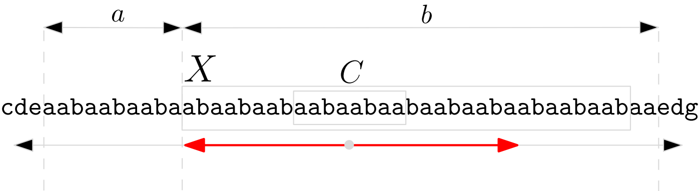
Note that indeed the palindrome may extend beyond periodicity!

Case 1: Reach the periodicity endpoints simultaneously

Consider $Q \in \mathcal{Q}_X$ with $|Q| = b - a$:


a
b
X
C
cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg
$Q \in \mathcal{Q}_X$ with $|Q| = b - a$
a

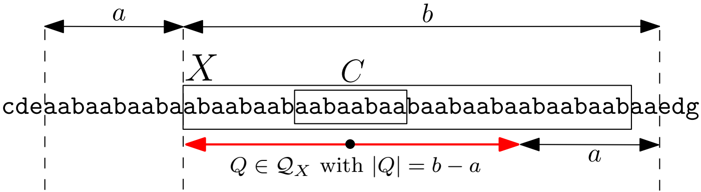
We use one LCP query to extend it.



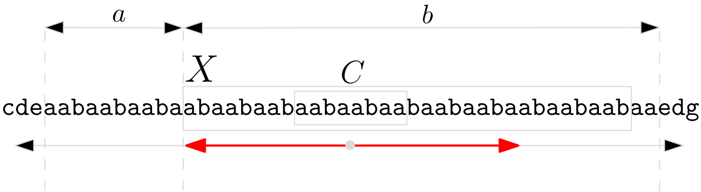
Note that indeed the palindrome may extend beyond periodicity!

Case 1: Reach the periodicity endpoints simultaneously

Consider $Q \in \mathcal{Q}_X$ with $|Q| = b - a$:


a
b
X
C
cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg
a
$Q \in \mathcal{Q}_X$ with $|Q| = b - a$

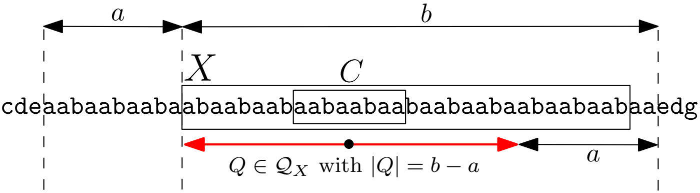
We use one LCP query to extend it.



Note that indeed the palindrome may extend beyond periodicity!

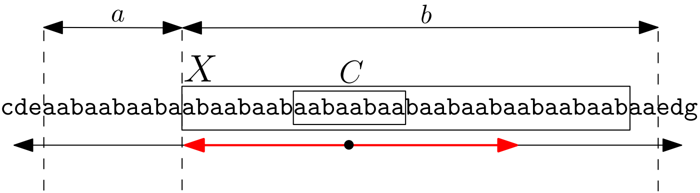
Case 1: Reach the periodicity endpoints simultaneously

Consider $Q \in \mathcal{Q}_X$ with $|Q| = b - a$:



a
b
X
C
cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg
$Q \in \mathcal{Q}_X$ with $|Q| = b - a$
a

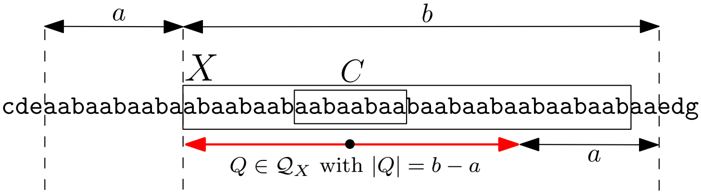
We use one LCP query to extend it.



Note that indeed the palindrome may extend beyond periodicity!

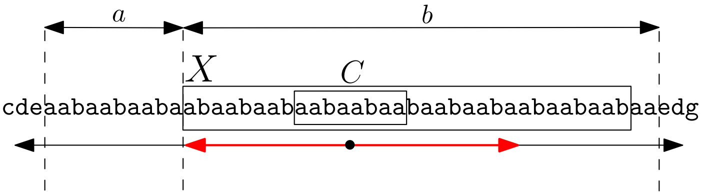
Case 1: Reach the periodicity endpoints simultaneously

Consider $Q \in \mathcal{Q}_X$ with $|Q| = b - a$:



a
b
X
C
cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg
$Q \in \mathcal{Q}_X$ with $|Q| = b - a$
a

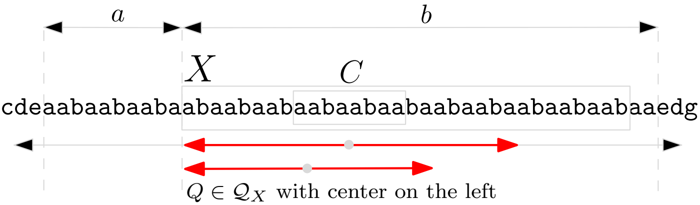
We use one LCP query to extend it.



Note that indeed the palindrome may extend beyond periodicity!

Case 2: Palindrome breaks when periodicity breaks (left)

Consider $Q \in \mathcal{Q}_X$ with its center on the left of the Case 1:



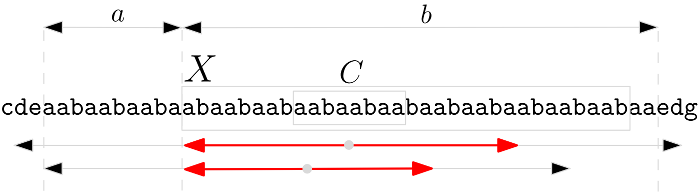
a b

X C

cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg

$Q \in \mathcal{Q}_X$ with center on the left

No LCP query is required to extend it...



a b

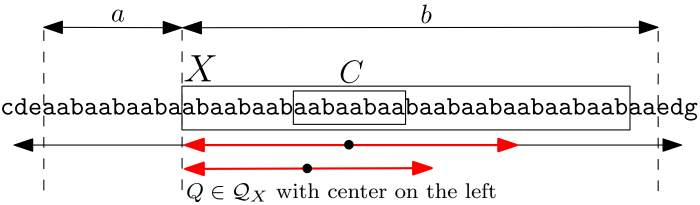
X C

cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg

...because the periodicity breaks first on the left.

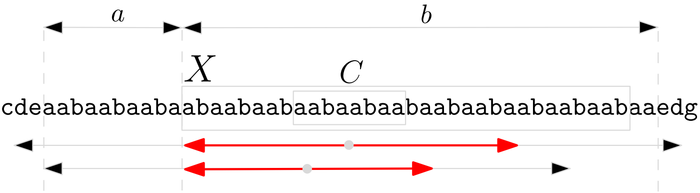
Case 2: Palindrome breaks when periodicity breaks (left)

Consider $Q \in \mathcal{Q}_X$ with its center on the left of the Case 1:



a
b
X
C
cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg
$Q \in \mathcal{Q}_X$ with center on the left

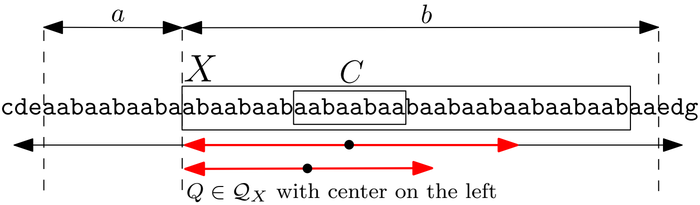
No LCP query is required to extend it...



...because the periodicity breaks first on the left.

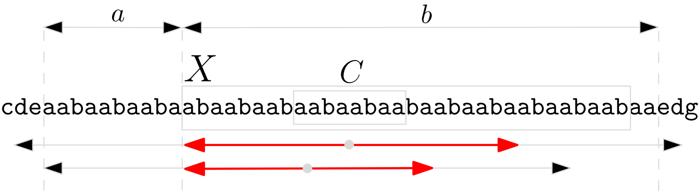
Case 2: Palindrome breaks when periodicity breaks (left)

Consider $Q \in \mathcal{Q}_X$ with its center on the left of the Case 1:



a
b
X
C
cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg
$Q \in \mathcal{Q}_X$ with center on the left

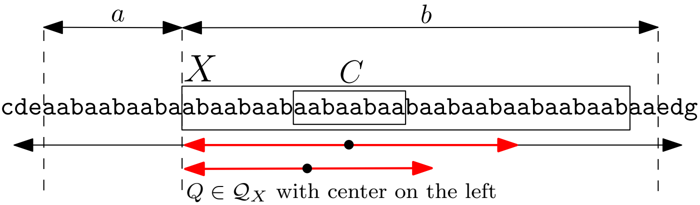
No LCP query is required to extend it...



...because the periodicity breaks first on the left.

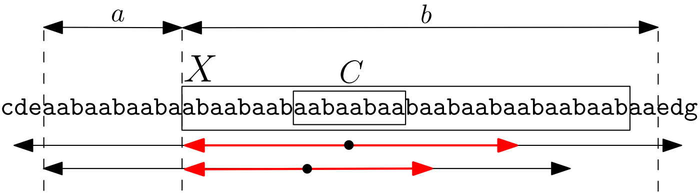
Case 2: Palindrome breaks when periodicity breaks (left)

Consider $Q \in \mathcal{Q}_X$ with its center on the left of the Case 1:



a
b
X
C
cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg
$Q \in \mathcal{Q}_X$ with center on the left

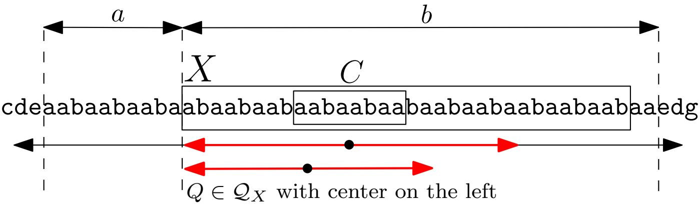
No LCP query is required to extend it...



...because the periodicity breaks first on the left.

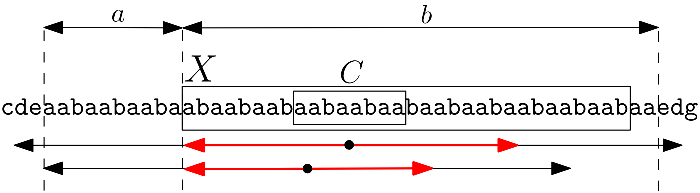
Case 2: Palindrome breaks when periodicity breaks (left)

Consider $Q \in \mathcal{Q}_X$ with its center on the left of the Case 1:



a
b
X
C
cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg
$Q \in \mathcal{Q}_X$ with center on the left

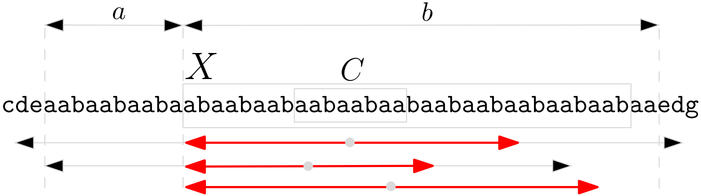
No LCP query is required to extend it...



...because the periodicity breaks first on the left.

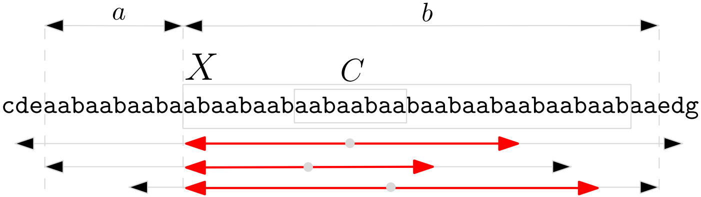
Case 3: Palindrome breaks when periodicity breaks (right)

Consider $Q \in \mathcal{Q}_X$ with its center on the right of the Case 1:



$Q \in \mathcal{Q}_X$ with center on the right

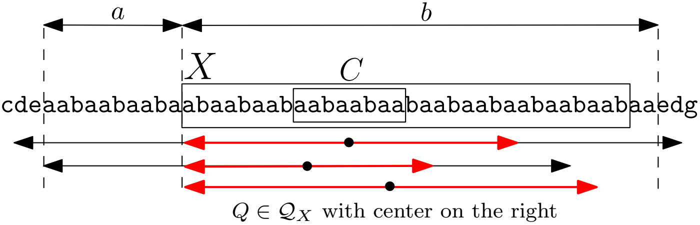
No LCP query is required to extend it...



...because the periodicity breaks first on the right.

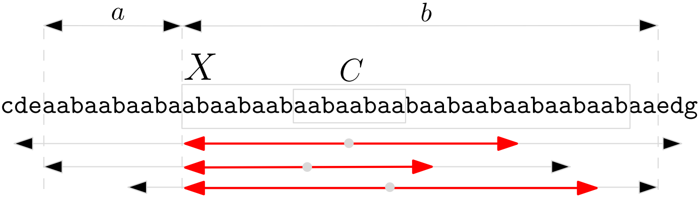
Case 3: Palindrome breaks when periodicity breaks (right)

Consider $Q \in \mathcal{Q}_X$ with its center on the right of the Case 1:



a
b
X
C
cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg
$Q \in \mathcal{Q}_X$ with center on the right

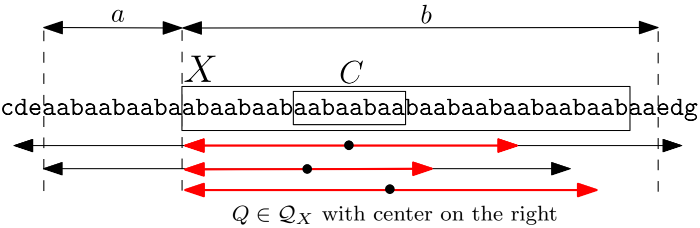
No LCP query is required to extend it...



...because the periodicity breaks first on the right.

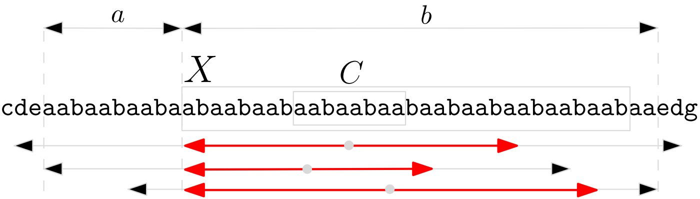
Case 3: Palindrome breaks when periodicity breaks (right)

Consider $Q \in \mathcal{Q}_X$ with its center on the right of the Case 1:



a
b
X
C
cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg
$Q \in \mathcal{Q}_X$ with center on the right

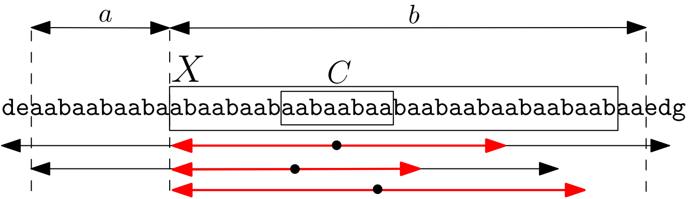
No LCP query is required to extend it...



...because the periodicity breaks first on the right.

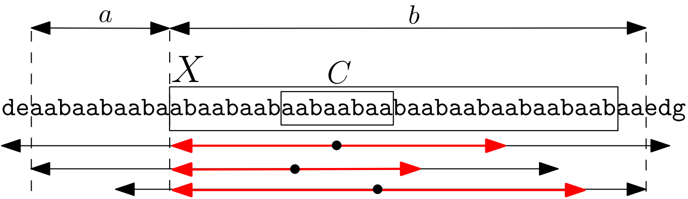
Case 3: Palindrome breaks when periodicity breaks (right)

Consider $Q \in \mathcal{Q}_X$ with its center on the right of the Case 1:



$Q \in \mathcal{Q}_X$ with center on the right

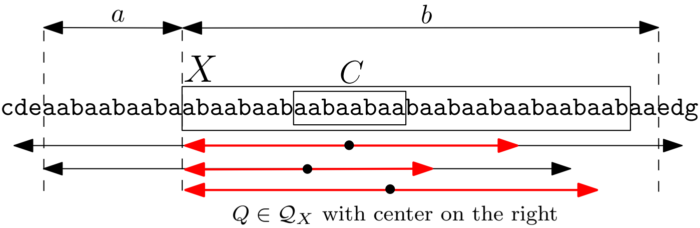
No LCP query is required to extend it...



...because the periodicity breaks first on the right.

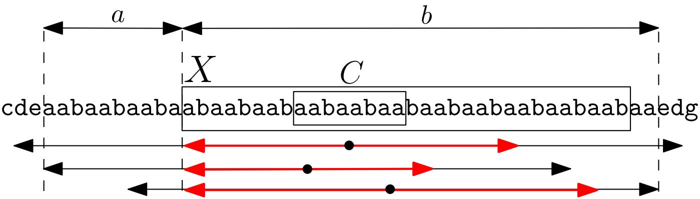
Case 3: Palindrome breaks when periodicity breaks (right)

Consider $Q \in \mathcal{Q}_X$ with its center on the right of the Case 1:



a
b
X
C
cdeaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaedg
$Q \in \mathcal{Q}_X$ with center on the right

No LCP query is required to extend it...



...because the periodicity breaks first on the right.

Wrapping-up

By picking the longest of the three palindromes (Cases 1-3), we obtain:

Lemma

The longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $\mathcal{O}(1)$ LCP queries on $S\#S^R\$$.

For each chunk C, we take the longer of:

- the palindrome computed by an application of the above lemma
- the longest palindrome precomputed for X with center in C

We have $\mathcal{O}(n/\ell) = \mathcal{O}(n/\log_\sigma n)$ chunks.

Using an optimal LCP data structure[2], we obtain our final result:

Theorem

A longest palindrome in S can be computed in $\mathcal{O}(n/\log_\sigma n)$ time.

[2] Kempa and Kociumaka: String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. STOC 2019.

Wrapping-up

By picking the longest of the three palindromes (Cases 1-3), we obtain:

Lemma

The longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $\mathcal{O}(1)$ LCP queries on $S\#S^R\$$.

For each chunk C, we take the longer of:

- the palindrome computed by an application of the above lemma
- the longest palindrome precomputed for X with center in C

We have $\mathcal{O}(n/\ell) = \mathcal{O}(n/\log_\sigma n)$ chunks.

Using an optimal LCP data structure[2], we obtain our final result:

Theorem

A longest palindrome in S can be computed in $\mathcal{O}(n/\log_\sigma n)$ time.

[2] Kempa and Kociumaka: String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. STOC 2019.

Wrapping-up

By picking the longest of the three palindromes (Cases 1-3), we obtain:

Lemma

The longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $\mathcal{O}(1)$ LCP queries on $S\#S^R\$$.

For each chunk C, we take the longer of:

- the palindrome computed by an application of the above lemma
- the longest palindrome precomputed for X with center in C

We have $\mathcal{O}(n/\ell) = \mathcal{O}(n/\log_\sigma n)$ chunks.

Using an optimal LCP data structure[2], we obtain our final result:

Theorem

A longest palindrome in S can be computed in $\mathcal{O}(n/\log_\sigma n)$ time.

[2] Kempa and Kociumaka: String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. STOC 2019.

Wrapping-up

By picking the longest of the three palindromes (Cases 1-3), we obtain:

Lemma

The longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $\mathcal{O}(1)$ LCP queries on $S\#S^R\$$.

For each chunk C, we take the longer of:

- the palindrome computed by an application of the above lemma
- the longest palindrome precomputed for X with center in C

We have $\mathcal{O}(n/\ell) = \mathcal{O}(n/\log_\sigma n)$ chunks.

Using an optimal LCP data structure[2], we obtain our final result:

Theorem

A longest palindrome in S can be computed in $\mathcal{O}(n/\log_\sigma n)$ time.

[2] Kempa and Kociumaka: String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. STOC 2019.

Wrapping-up

By picking the longest of the three palindromes (Cases 1-3), we obtain:

Lemma

The longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $\mathcal{O}(1)$ LCP queries on $S\#S^R\$$.

For each chunk C, we take the longer of:

- the palindrome computed by an application of the above lemma
- the longest palindrome precomputed for X with center in C

We have $\mathcal{O}(n/\ell) = \mathcal{O}(n/\log_\sigma n)$ chunks.

Using an optimal LCP data structure[2], we obtain our final result:

Theorem

A longest palindrome in S can be computed in $\mathcal{O}(n/\log_\sigma n)$ time.

[2] Kempa and Kociumaka: String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. STOC 2019.

Wrapping-up

By picking the longest of the three palindromes (Cases 1-3), we obtain:

Lemma

The longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $\mathcal{O}(1)$ LCP queries on $S\#S^R\$$.

For each chunk C, we take the longer of:

- the palindrome computed by an application of the above lemma
- the longest palindrome precomputed for X with center in C

We have $\mathcal{O}(n/\ell) = \mathcal{O}(n/\log_\sigma n)$ chunks.

Using an optimal LCP data structure[2], we obtain our final result:

Theorem

A longest palindrome in S can be computed in $\mathcal{O}(n/\log_\sigma n)$ time.

[2] Kempa and Kociumaka: String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. STOC 2019

Wrapping-up

By picking the longest of the three palindromes (Cases 1-3), we obtain:

Lemma

The longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $\mathcal{O}(1)$ LCP queries on $S\#S^R\$$.

For each chunk C, we take the longer of:

- the palindrome computed by an application of the above lemma
- the longest palindrome precomputed for X with center in C

We have $\mathcal{O}(n/\ell) = \mathcal{O}(n/\log_\sigma n)$ chunks.

Using an optimal LCP data structure[2], we obtain our final result:

Theorem

A longest palindrome in S can be computed in $\mathcal{O}(n/\log_\sigma n)$ time.

[2] Kempa and Kociumaka: String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. STOC 2019

Wrapping-up

By picking the longest of the three palindromes (Cases 1-3), we obtain:

Lemma

The longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $\mathcal{O}(1)$ LCP queries on $S\#S^R\$$.

For each chunk C, we take the longer of:

- the palindrome computed by an application of the above lemma
- the longest palindrome precomputed for X with center in C

We have $\mathcal{O}(n/\ell) = \mathcal{O}(n/\log_\sigma n)$ chunks.

Using an optimal LCP data structure[2], we obtain our final result:

Theorem

A longest palindrome in S can be computed in $\mathcal{O}(n/\log_\sigma n)$ time.

[2] Kempa and Kociumaka: String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. STOC 2019