

SPIRE2026

# Online Computation of Maximal Closed Substrings

Hiroki Shibata<sup>\*</sup> Haruki Umezaki<sup>\*</sup> Takuya Mieno<sup>†</sup>

Yuto Nakashima<sup>\*</sup> Shunsuke Inenaga<sup>\*</sup>

<sup>\*</sup>Kyushu University

<sup>†</sup>The University of Electro-Communications



slides

# Closed Strings

1/18

## Definition 1: Border and Closed Strings

A substring of  $w$  that appears as both a proper prefix and a proper suffix of  $w$  is called a **border** of  $w$ .

A string is **closed** when its longest border appears exactly twice.

Closed



# Closed Strings

1/18

## Definition 1: Border and Closed Strings

A substring of  $w$  that appears as both a proper prefix and a proper suffix of  $w$  is called a **border** of  $w$ .

A string is **closed** when its longest border appears exactly twice.

Not Closed

a b a a b a b a b a



# Maximal Closed Substring

2/18

## Definition 2: Maximal Closed Substrings

A closed substring is **left-maximal** (resp. **right-maximal**) if extending it to the left (resp. right) by one character gives a non-closed string.

A closed substring that is both left-maximal and right-maximal is called a **maximal closed substring (MCS)**.

a a b a d a b a c d      **MCS**

a a b a d a b a c d      **Right extension**

a a b a d a b a c d      **Left extension**

## Definition 3: Closed Repeats

Two occurrences of the same string form a **closed repeat** if

1. No other occurrence starts between them.
2. Extending both to the left by one character breaks equality.
3. Extending both to the right by one character breaks equality.

We call a repeat satisfying condition 1, 2, or 3 **consecutive**, **left-maximal**, or **right-maximal**, respectively.

a b a b c b a b d b a b

# Closed Repeats

3/18

## Definition 3: Closed Repeats

Two occurrences of the same string form a **closed repeat** if

1. No other occurrence starts between them.
2. Extending both to the left by one character breaks equality.
3. Extending both to the right by one character breaks equality.

We call a repeat satisfying condition 1, 2, or 3 **consecutive**, **left-maximal**, or **right-maximal**, respectively.

a b a b    c b a b    d b a b

## Definition 3: Closed Repeats

Two occurrences of the same string form a **closed repeat** if

1. No other occurrence starts between them.
2. Extending both to the left by one character breaks equality.
3. Extending both to the right by one character breaks equality.

We call a repeat satisfying condition 1, 2, or 3 **consecutive**, **left-maximal**, or **right-maximal**, respectively.

a   b a b   c   b a b   d b a b

**Closed Repeat**

# MCSs and Closed Repeats

4/18

## Property 1: Correspondence

MCSs are in **one-to-one correspondence** with closed repeats. Specifically, the two occurrences of the longest border of an MCS form the corresponding closed repeat.

a b a b c b a b d b a b

**babcbab** is an MCS

**bab** is a closed repeat

[Badkobeh et al., 2026] and [Kosolobov 2024] established the following results:

- A string of length  $n$  contains  $O(n \log n)$  MCSs, and this bound is tight.
  - The number of left-maximal (resp. right-maximal) closed substrings is also  $O(n \log n)$ .
- Computing all MCSs and closed repeats offline in  $O(n \log n)$  time.

## Our Goal

Maintain the set of MCSs **online** in  $O(n \log n)$  total time.

# Online Computation

6/18

- We reverse the text so that each new character is prepended.
- We compute **right-maximal consecutive repeats (RMCR)** rather than MCSs directly.
  - Existing repeats remain consecutive and right-maximal after prepending a character.
  - We also track each RMCR's left-maximality to maintain MCSs.



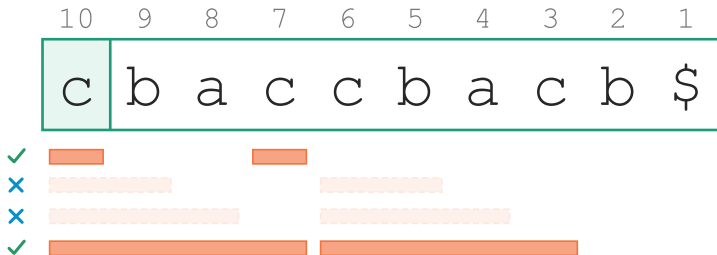
# Characterization of RMCRs

7/18

## Observation 1

For each RMCR created by prepending  $c$  to  $w$ :

1. The left occurrence is a prefix of  $cw$ .
2. The right occurrence is the leftmost occurrence in  $w$ .
3. Extending this prefix to the right changes its leftmost occurrence in  $w$ .

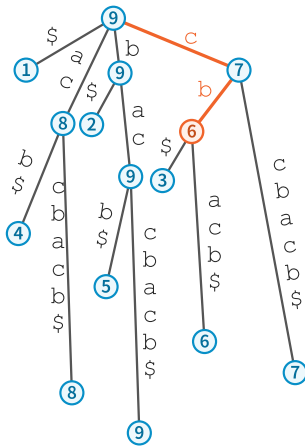
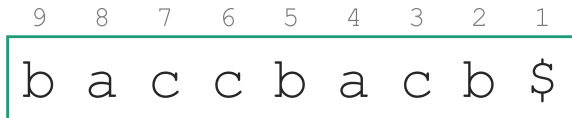


# Suffix Trees

8/18

We consider the **suffix tree** of the input string  $w$ .

- Each vertex stores the position of its leftmost occurrence.

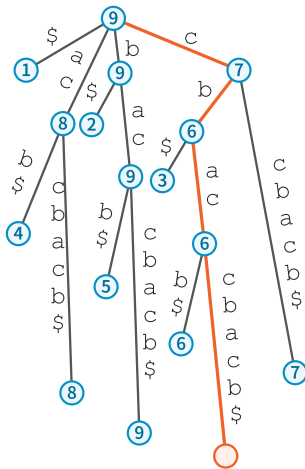


# Updating the Suffix Tree

9/18

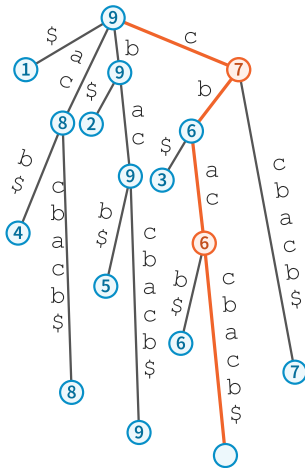
Prepending  $c$  to  $w$  adds **a new leaf for  $cw$**  to the suffix tree.

We compute new RMCRs when adding this leaf.



## 10/18

Diagram illustrating the construction of a suffix array for the string "cabcacbacb\$". The string is shown with indices 10 down to 1 above each character. Below the string, four horizontal bars represent the sorted order of suffixes. The first bar (green) starts at index 10 and ends at index 9. The second bar (red) starts at index 9 and ends at index 8. The third bar (blue) starts at index 8 and ends at index 7. The fourth bar (orange) starts at index 7 and ends at index 6. The fifth bar (yellow) starts at index 6 and ends at index 5. The sixth bar (purple) starts at index 5 and ends at index 4. The seventh bar (brown) starts at index 4 and ends at index 3. The eighth bar (pink) starts at index 3 and ends at index 2. The ninth bar (gray) starts at index 2 and ends at index 1. The tenth bar (light blue) starts at index 1 and ends at index 0. The bars are sorted by their starting index, which corresponds to the suffix array order.





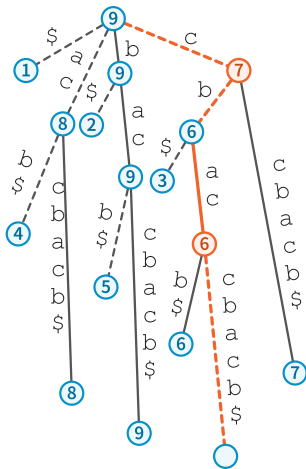
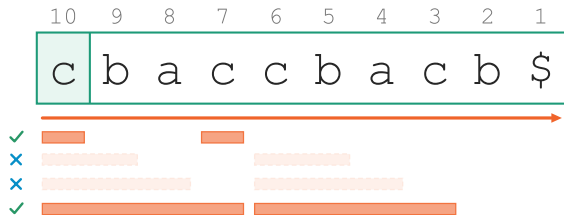
# Classifying Suffix Tree Edges

11/18

We classify suffix tree edges as follows:

- **Heavy Edge**: connecting nodes with the same label.
- **Light Edge**: connecting nodes with different labels.

Each new RMCR corresponds to **an upper endpoint of a light edge** on the root-to-new-leaf path.







# Changing Edge Types

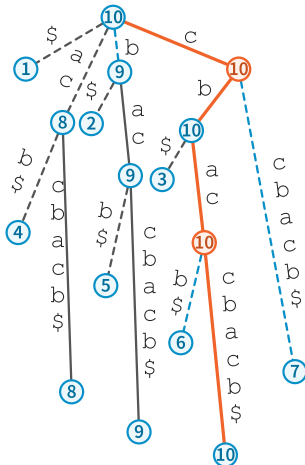
12/18

After computing new RMCRs, we update the edge types as follows:

- Each edge on the **root-to-leaf path** becomes heavy.
- Each **heavy edge** branching off this path becomes light.

## Important Observation

This update procedure is exactly the same as that used in **link-cut trees**!

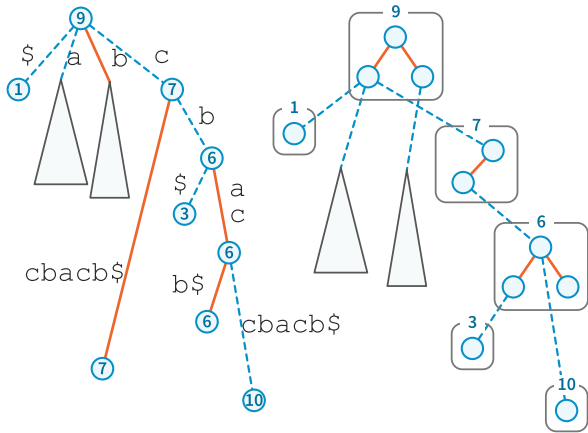


# Link-Cut Trees

13/18

A dynamic tree data structure that classifies edges as heavy or light and represents:

- Each **maximal heavy path** by a splay tree.
  - It consists of vertices with the same label.
- Each **light edge** by an edge connecting splay trees.

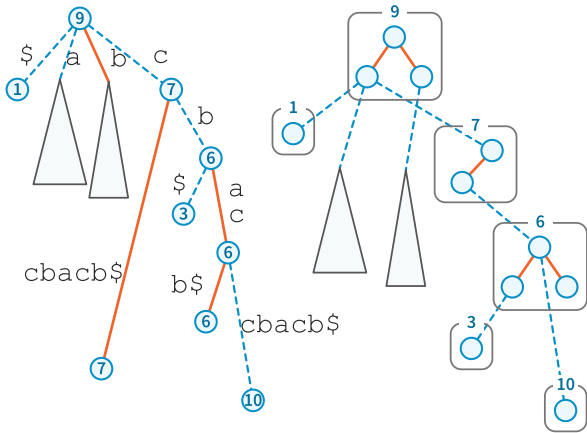


# Updating Link-Cut Trees

14/18

Update the heavy paths and their splay trees:

- Split off the parts outside the root-to-new-leaf path.
- Join the trees to form a single splay tree representing the path.

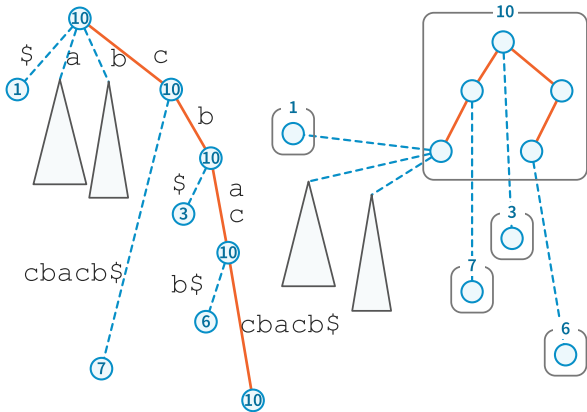


# Updating Link-Cut Trees

14/18

Update the heavy paths and their splay trees:

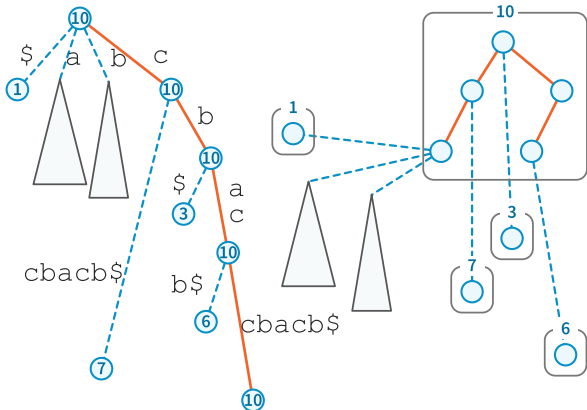
- Split off the parts outside the root-to-new-leaf path.
- Join the trees to form a single splay tree representing the path.



## 14/18

- Split off the parts outside the root-to-new-leaf path.
- Join the trees to form a single splay tree representing the path.

**We can update a link-cut tree in amortized  $O(\log n)$  time per operation.**



**LCST:** A suffix tree represented by a link-cut tree.

Updating the LCST and Computing RMCRs:

1. **Add a new leaf** corresponding to the new string  $cw$ .
  - $O(\log \min\{\sigma, n\}) \subseteq O(\log n)$  time (Weiner's algorithm)
2. **Update the underlying link-cut tree** while recording the light edges on the root-to-new-leaf path.
  - amortized  $O(\log n)$  time per operation (LCT time bound)
3. **Compute the corresponding RMCRs** from the recorded light edges.
  - amortized  $O(\log n)$  time per operation (from the LCT time bound)

# Computing Closed Repeats and MCSs

---

16/18

Each closed repeat corresponds to a **left-maximal RMCR**.

- **Each new RMCR is left-maximal** because its left occurrence is a prefix.
- **A closed repeat corresponding to a prefix may lose left-maximality** when a new character is added.
  - There are  $O(n \log n)$  checks in total, and each check takes constant time.
- Other closed repeats do not lose left-maximality.

# Computing Closed Repeats and MCSs

16/18

Each closed repeat corresponds to a **left-maximal RMCR**.

- **Each new RMCR is left-maximal** because its left occurrence is a prefix.
- **A closed repeat corresponding to a prefix may lose left-maximality** when a new character is added.
  - There are  $O(n \log n)$  checks in total, and each check takes constant time.
- Other closed repeats do not lose left-maximality.

## Main Theorem

We can maintain closed repeats online in  $O(n \log n)$  total time.

Since closed repeats are in one-to-one correspondence with MCSs, **we can also compute MCSs online in  $O(n \log n)$  total time.**

Link-cut suffix trees can be viewed as an online data structure that **maintains the position of the most recent occurrence** of each substring.

Applications:

- **Most Recent Match**: Find the most recent occurrence of a pattern in a text that grows by appending characters.
- **Rightmost LZ Factorization**: A variant of LZ factorization that uses the rightmost occurrence as the copy source.

Although both problems already have efficient solutions [[Larsson, 2014](#); [Sumiyoshi et al., 2024](#)] and LCSTs do not improve their time bounds, LCSTs offer **a general and relatively simple approach**.

- We introduce **an online algorithm for computing MCSs** that runs in  $O(n \log n)$  total time.
  - We can also compute closed repeats using their one-to-one correspondence with MCSs.
- Our algorithm is based on **link-cut suffix trees**, a simple combination of suffix trees and link-cut trees.
- LCSTs have **other applications**, such as most recent match and rightmost LZ factorization.