Recent Results on the Longest Common Substring Problem

Jakub Radoszewski

University of Warsaw, Poland and Samsung R&D, Warsaw, Poland

SPIRE 2023, Pisa, Italy, 26-28 September 2023

Longest Common Substring

Definition (Longest Common Substring Problem)

Input: Two strings S and T of length at most n

Output: The longest string U that is a substring of both S and T

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

Longest Common Substring

Definition (Longest Common Substring Problem)

Input: Two strings S and T of length at most n

Output: The longest string U that is a substring of both S and T

`c b b a a b c a b c a b c b a`

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

This is not the Longest Common **Subsequence** problem

Longest Common Factor

Definition (Longest Common Factor (LCF) Problem)

Input: Two strings S and T of length at most n

Output: The longest string U that is a factor (substring) of both S and T

c b b a a b c a b c a b c b a

d a a b a a b c b b a

This is not the Longest Common **Subsequence** problem

Longest Common Factor

Definition (Longest Common Factor (LCF) Problem)

Input: Two strings S and T of length at most n

Output: The longest string U that is a factor (substring) of both S and T

`c b b a a b c a b c a b c b a`

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

This is not the Longest Common **Subsequence** problem

Motivation for LCF:

- An elementary string similarity measure
- Computationally easier than LCS [1,2]

[1] A. Abboud, A. Backurs, V. Vassilevska Williams:
Tight Hardness Results for LCF and Other Sequence Similarity Measures. FOCS 2015

[2] K. Bringmann, M. Künnemann: Quadratic Conditional Lower Bounds
for String Problems and Dynamic Time Warping. FOCS 2015

Early Results on LCF

History of the LCF problem:

- Donald E. Knuth conjectured no $o(n \log n)$-time solution for LCF
- $O(n)$ time LCF for $O(1)$ alphabet using the suffix tree [1]

[1] P. Weiner: Linear Pattern Matching Algorithms, SWAT 1973

Early Results on LCF

History of the LCF problem:

- Donald E. Knuth conjectured no $o(n \log n)$-time solution for LCF
- $O(n)$ time LCF for $O(1)$ alphabet using the suffix tree [1]

b a b a a	a a a b a
S	T

[1] P. Weiner: Linear Pattern Matching Algorithms, SWAT 1973

Early Results on LCF

History of the LCF problem:

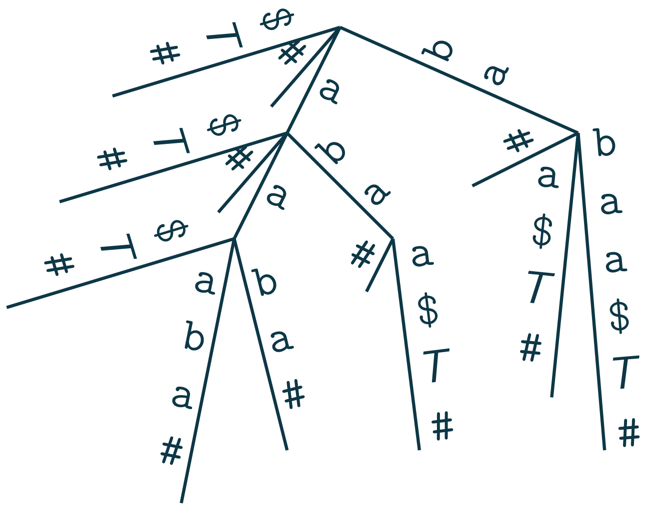
- Donald E. Knuth conjectured no $o(n \log n)$-time solution for LCF
- $O(n)$ time LCF for $O(1)$ alphabet using the suffix tree [1]

```
b a b a a $ a a a b a #
    S           T
```

[1] P. Weiner: Linear Pattern Matching Algorithms, SWAT 1973

History of the LCF problem:

- Donald E. Knuth conjectured no $o(n \log n)$-time solution for LCF
- $O(n)$ time LCF for $O(1)$ alphabet using the suffix tree [1]



b a b a a $ a a a b a #

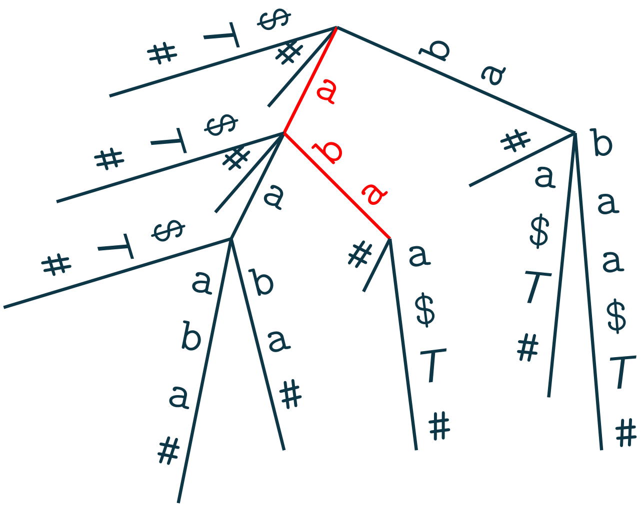
S T

[1] P. Weiner: Linear Pattern Matching Algorithms, SWAT 1973

Early Results on LCF

History of the LCF problem:

- Donald E. Knuth conjectured no $o(n \log n)$-time solution for LCF
- $O(n)$ time LCF for $O(1)$ alphabet using the suffix tree [1]



b a b a a $ a a a b a #

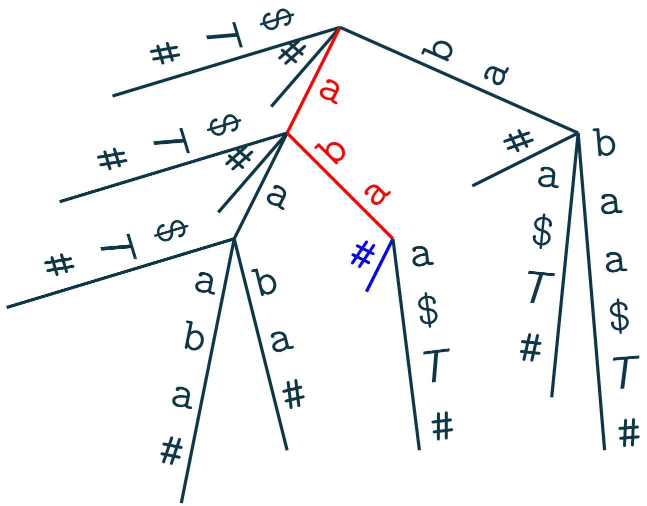
S T

[1] P. Weiner: Linear Pattern Matching Algorithms, SWAT 1973

Early Results on LCF

History of the LCF problem:

- Donald E. Knuth conjectured no $o(n \log n)$-time solution for LCF
- $O(n)$ time LCF for $O(1)$ alphabet using the suffix tree [1]



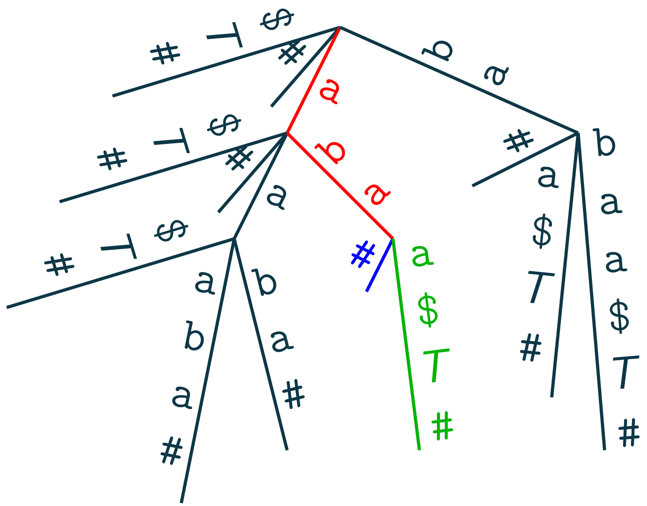
b a b a a $ a a a b a #
S T

[1] P. Weiner: Linear Pattern Matching Algorithms, SWAT 1973

Early Results on LCF

History of the LCF problem:

- Donald E. Knuth conjectured no $o(n \log n)$-time solution for LCF
- $O(n)$ time LCF for $O(1)$ alphabet using the suffix tree [1]



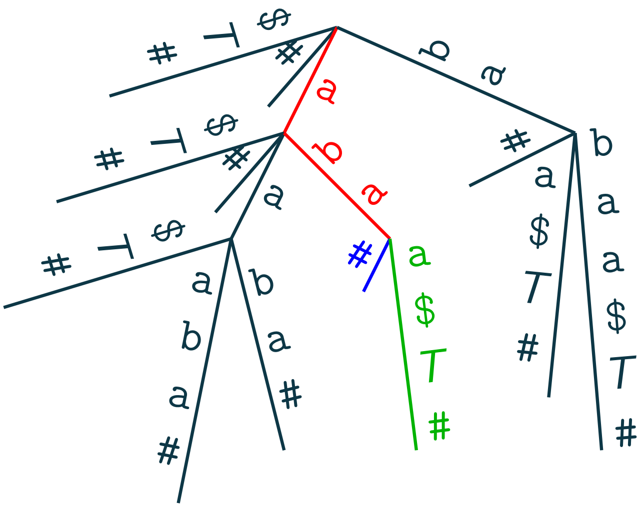
b a b a a $ a a a b a #

S T

[1] P. Weiner: Linear Pattern Matching Algorithms, SWAT 1973

History of the LCF problem:

- Donald E. Knuth conjectured no $o(n \log n)$-time solution for LCF
- $O(n)$ time LCF for $O(1)$ alphabet using the suffix tree [1]



b a b a a \$ a a a b a #

S T

- $O(n)$ time for integer alphabet $[0, n^{O(1)}]$, see [2]

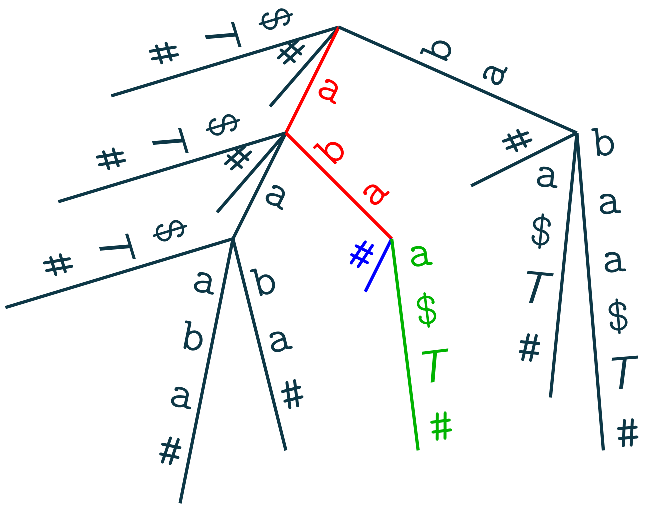
[1] P. Weiner: Linear Pattern Matching Algorithms, SWAT 1973

[2] M. Farach: Optimal Suffix Tree Construction with Large Alphabets, FOCS 1997

Early Results on LCF

History of the LCF problem:

- Donald E. Knuth conjectured no $o(n \log n)$-time solution for LCF
- $O(n)$ time LCF for $O(1)$ alphabet using the suffix tree [1]



b a b a a $ a a a b a #

S T

- $O(n)$ time for integer alphabet $[0, n^{O(1)}]$, see [2]
- $O(n)$ time for many strings of total length $O(n)$ [3]

[1] P. Weiner: Linear Pattern Matching Algorithms, SWAT 1973

[2] M. Farach: Optimal Suffix Tree Construction with Large Alphabets, FOCS 1997

[3] L.C.K. Hui. Color set size problem with applications to string matching. CPM 1992

Even Faster LCF

Packed setting:

- $O(1)$-sized alphabet, say `A`, `C`, `G`, `T`
- A string can be represented with $O(n/\log n)$ machine words

Even Faster LCF

Packed setting:

- $O(1)$-sized alphabet, say A, C, G, T
- A string can be represented with $O(n/\log n)$ machine words

Theorem ([1])

LCF on packed strings can be solved in $O(n/\sqrt{\log n})$ time.

[1] P. Charalampopoulos, T. Kociumaka, S.P. Pissis, **R**: Faster Algorithms for LCF. ESA 2021

Even Faster LCF

Packed setting:

- $O(1)$-sized alphabet, say A, C, G, T
- A string can be represented with $O(n/\log n)$ machine words

Theorem ([1])

LCF on packed strings can be solved in $O(n/\sqrt{\log n})$ time.

If alphabet has size σ, the algorithm works in $O(n \log \sigma / \sqrt{\log n})$ time.

[1] P. Charalampopoulos, T. Kociumaka, S.P. Pissis, **R**: Faster Algorithms for LCF. ESA 2021

Even Faster LCF

Packed setting:

- $O(1)$-sized alphabet, say A, C, G, T
- A string can be represented with $O(n/\log n)$ machine words

Theorem ([1])

LCF on packed strings can be solved in $O(n/\sqrt{\log n})$ time.

If alphabet has size σ, the algorithm works in $O(n\log\sigma/\sqrt{\log n})$ time.

Relation to packed indexing:

- $O(n\log\sigma/\log n)$-sized index can be constructed in $O(n\log\sigma/\sqrt{\log n})$ time [2,3]
- No direct computation of packed LCF from packed index is known.

[1] P. Charalampopoulos, T. Kociumaka, S.P. Pissis, **R**: Faster Algorithms for LCF. ESA 2021

[2] D. Kempa, T. Kociumaka: String Synchronizing Sets: Sublinear-time BWT Construction and Optimal LCE Data Structure. STOC 2019

[3] J.I. Munro, G. Navarro, Y. Nekrich: Text Indexing and Searching in Sublinear Time. CPM 2020

Some Details on Packed LCF

Assumptions for the talk:

- Alphabet of size $\sigma = O(1)$
- A number in $[0, n)$ encodes $\log_\sigma n = \Theta(\log n)$ letters.
- A string of length n is represented by $\Theta(n/\log n)$ numbers in $[0, n)$.

Some Details on Packed LCF

Assumptions for the talk:

- Alphabet of size $\sigma = O(1)$
- A number in $[0, n)$ encodes $\log_\sigma n = \Theta(\log n)$ letters.
- A string of length n is represented by $\Theta(n/\log n)$ numbers in $[0, n)$.

Three cases:

- Short LCF: $\leq \frac{1}{3}\log_\sigma n$
- Medium LCF
- Long LCF: $\geq \log^4 n$ [1]

We aim at $O(n/\sqrt{\log n})$ time.

[1] P. Charalampopoulos, M. Crochemore, C.S. Iliopoulos, T. Kociumaka, S.P. Pissis, **R**, W. Rytter, T. Waleń: Linear-Time Algorithm for Long LCF with k Mismatches. CPM 2018

Plan of Presentation

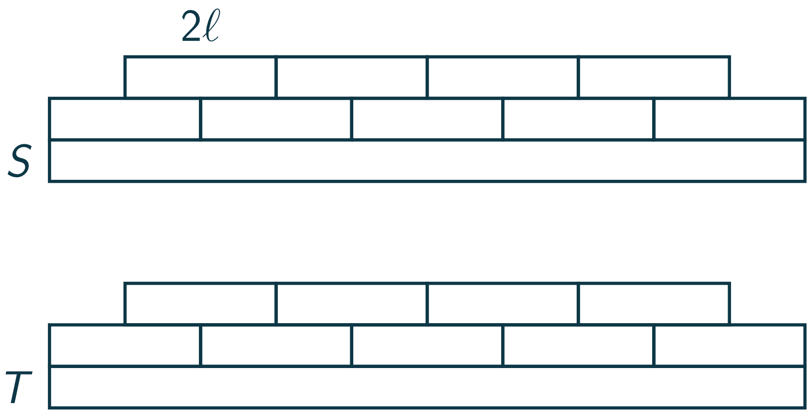
- Packed LCF:
  - Short LCF
  - Long LCF
  - Medium-length LCF
- Approximate LCF
- Small-space LCF
- Compressed LCF
- Dynamic LCF
- Internal LCF

Plan of Presentation

- **Packed LCF:**
  - **Short LCF**
  - Long LCF
  - Medium-length LCF
- Approximate LCF
- Small-space LCF
- Compressed LCF
- Dynamic LCF
- Internal LCF

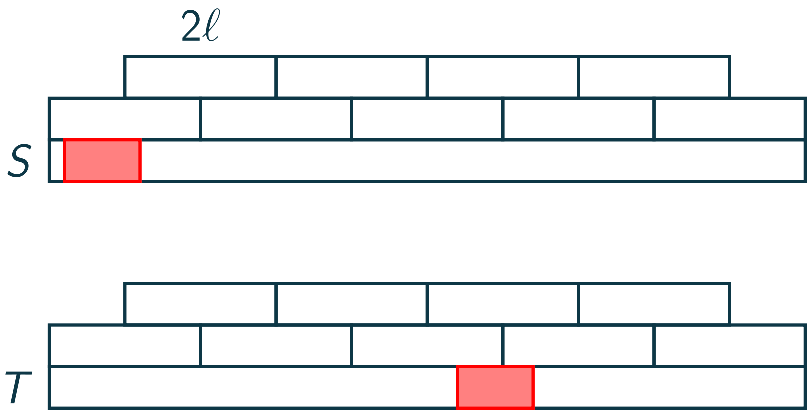
Short LCF ($\leq \frac{1}{3}\log_\sigma n$)

- Let $\ell = \frac{1}{3}\log_\sigma n$. Split each string into overlapping substrings of length 2ℓ.
- The LCF is a substring of one length-2ℓ substring in S and in T.



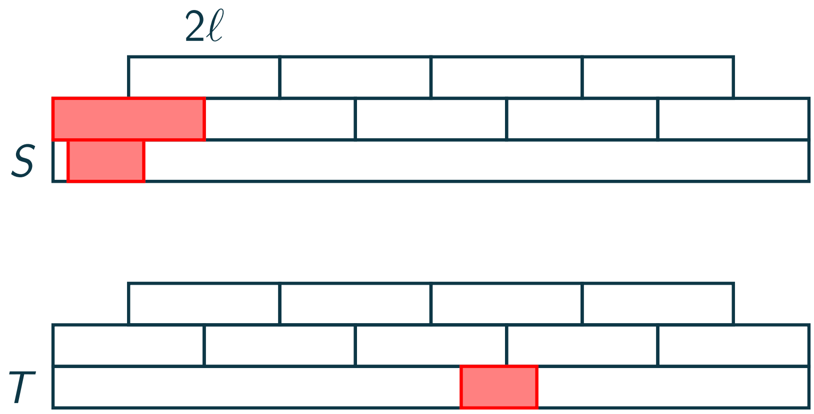
Short LCF ($\leq \frac{1}{3}\log_\sigma n$)

- Let $\ell = \frac{1}{3}\log_\sigma n$. Split each string into overlapping substrings of length 2ℓ.
- The LCF is a substring of one length-2ℓ substring in S and in T.



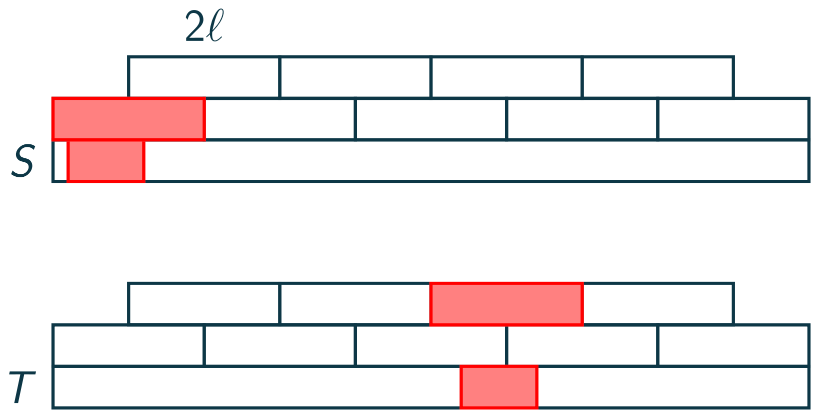
Short LCF ($\leq \frac{1}{3}\log_\sigma n$)

- Let $\ell = \frac{1}{3}\log_\sigma n$. Split each string into overlapping substrings of length 2ℓ.
- The LCF is a substring of one length-2ℓ substring in S and in T.



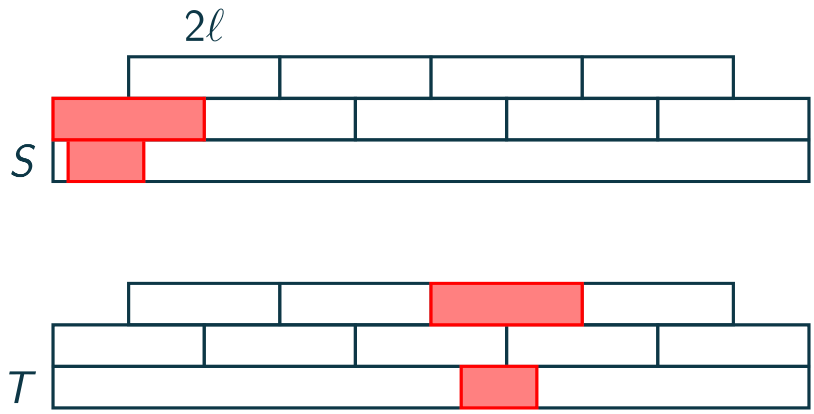
Short LCF ($\leq \frac{1}{3}\log_\sigma n$)

- Let $\ell = \frac{1}{3}\log_\sigma n$. Split each string into overlapping substrings of length 2ℓ.
- The LCF is a substring of one length-2ℓ substring in S and in T.



Short LCF ($\leq \frac{1}{3} \log_\sigma n$)

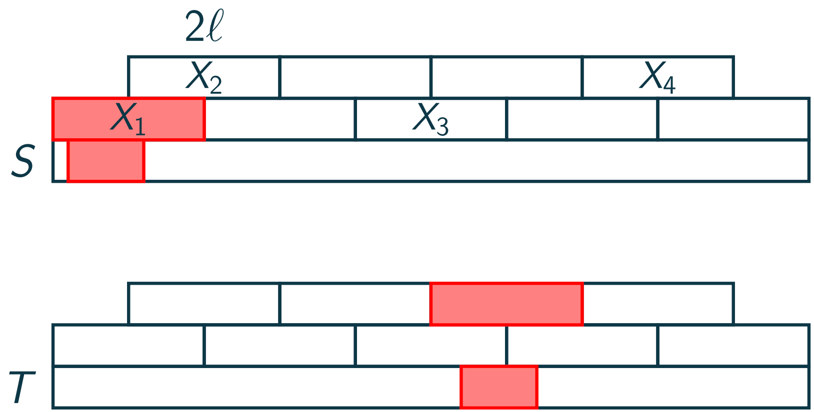
- Let $\ell = \frac{1}{3} \log_\sigma n$. Split each string into overlapping substrings of length 2ℓ.
- The LCF is a substring of one length-2ℓ substring in S and in T.



- $\leq \sigma^{2\ell} = \sigma^{\frac{2}{3} \log_\sigma n} = n^{2/3}$ distinct length-2ℓ substrings of S and T
- They can be computed in $O(n/\log n + n^{2/3})$ time (bitmask tricks)

Short LCF ($\leq \frac{1}{3} \log_\sigma n$)

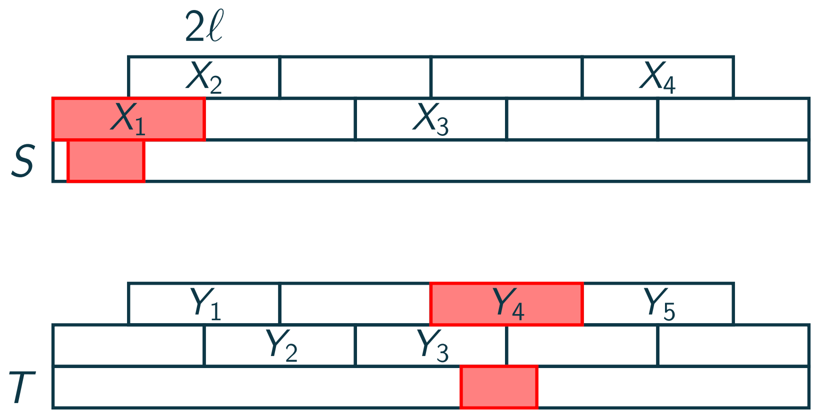
- Let $\ell = \frac{1}{3} \log_\sigma n$. Split each string into overlapping substrings of length 2ℓ.
- The LCF is a substring of one length-2ℓ substring in S and in T.



- $\leq \sigma^{2\ell} = \sigma^{\frac{2}{3} \log_\sigma n} = n^{2/3}$ distinct length-2ℓ substrings of S and T
- They can be computed in $O(n/\log n + n^{2/3})$ time (bitmask tricks)
- Let $X_1, \ldots, X_p$ – distinct length-2ℓ substrings in S

Short LCF ($\leq \frac{1}{3}\log_\sigma n$)

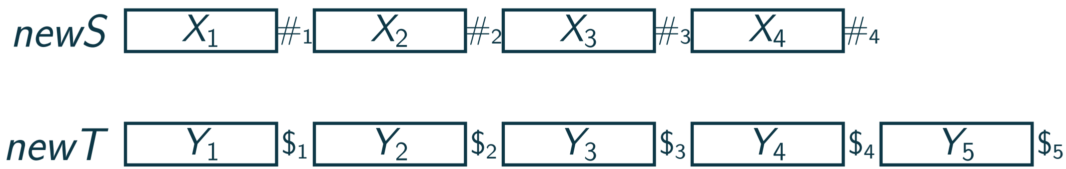
- Let $\ell = \frac{1}{3}\log_\sigma n$. Split each string into overlapping substrings of length 2ℓ.
- The LCF is a substring of one length-2ℓ substring in S and in T.



- $\leq \sigma^{2\ell} = \sigma^{\frac{2}{3}\log_\sigma n} = n^{2/3}$ distinct length-2ℓ substrings of S and T
- They can be computed in $O(n/\log n + n^{2/3})$ time (bitmask tricks)
- Let $X_1, \ldots, X_p$ – distinct length-2ℓ substrings in S
- Let $Y_1, \ldots, Y_q$ – distinct length-2ℓ substrings in T

Short LCF ($\leq \frac{1}{3}\log_\sigma n$)

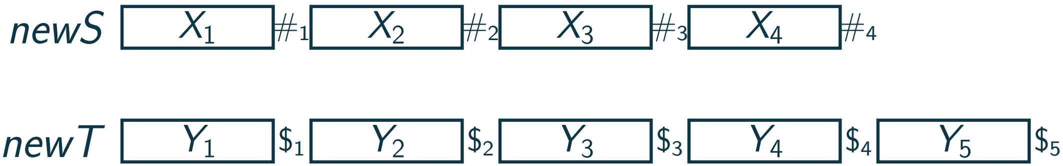
- Let $\ell = \frac{1}{3}\log_\sigma n$. Split each string into overlapping substrings of length 2ℓ.
- The LCF is a substring of one length-2ℓ substring in S and in T.



- $\leq \sigma^{2\ell} = \sigma^{\frac{2}{3}\log_\sigma n} = n^{2/3}$ distinct length-2ℓ substrings of S and T
- They can be computed in $O(n/\log n + n^{2/3})$ time (bitmask tricks)
- Let $X_1, \ldots, X_p$ – distinct length-2ℓ substrings in S
- Let $Y_1, \ldots, Y_q$ – distinct length-2ℓ substrings in T

Short LCF ($\leq \frac{1}{3}\log_\sigma n$)

- Let $\ell = \frac{1}{3}\log_\sigma n$. Split each string into overlapping substrings of length 2ℓ.
- The LCF is a substring of one length-2ℓ substring in S and in T.

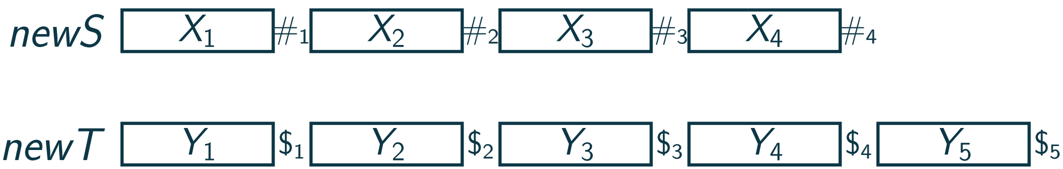


$$|newS|,\ |newT| \ \leq\ n^{2/3}(\tfrac{2}{3}\log n + 1)$$

- $\leq \sigma^{2\ell} = \sigma^{\frac{2}{3}\log_\sigma n} = n^{2/3}$ distinct length-2ℓ substrings of S and T
- They can be computed in $O(n/\log n + n^{2/3})$ time (bitmask tricks)
- Let $X_1, \ldots, X_p$ – distinct length-2ℓ substrings in S
- Let $Y_1, \ldots, Y_q$ – distinct length-2ℓ substrings in T

Short LCF ($\leq \frac{1}{3}\log_\sigma n$)

- Let $\ell = \frac{1}{3}\log_\sigma n$. Split each string into overlapping substrings of length 2ℓ.
- The LCF is a substring of one length-2ℓ substring in S and in T.



$|newS|,\ |newT| \leq n^{2/3}(\frac{2}{3}\log n + 1)$

LCF of $newS$ and $newT$ can be computed in $O(n^{2/3}\log n) = o(n/\log n)$ time.

- $\leq \sigma^{2\ell} = \sigma^{\frac{2}{3}\log_\sigma n} = n^{2/3}$ distinct length-2ℓ substrings of S and T
- They can be computed in $O(n/\log n + n^{2/3})$ time (bitmask tricks)
- Let $X_1, \ldots, X_p$ – distinct length-2ℓ substrings in S
- Let $Y_1, \ldots, Y_q$ – distinct length-2ℓ substrings in T

Plan of Presentation

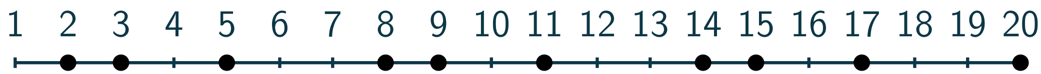
- **Packed LCF:**
  - Short LCF
  - **Long LCF**
  - Medium-length LCF
- Approximate LCF
- Small-space LCF
- Compressed LCF
- Dynamic LCF
- Internal LCF

Difference Cover

Definition (Difference cover)

A set D is a d-cover if there is a function *shift* such that for any $i, j > 0$ we have $0 \leq \mathit{shift}(i,j) < d$ and $i + \mathit{shift}(i,j)$, $j + \mathit{shift}(i,j) \in D$.

Example. $\{2, 3, 5, 8, 9, 11, 14, 15, 19, 20, \ldots\}$ is a 6-cover:

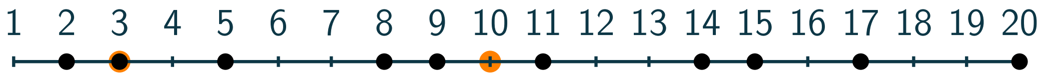


Difference Cover

Definition (Difference cover)

A set D is a d-cover if there is a function *shift* such that for any $i, j > 0$ we have $0 \le shift(i,j) < d$ and $i + shift(i,j),\ j + shift(i,j) \in D$.

Example. $\{2, 3, 5, 8, 9, 11, 14, 15, 19, 20, \ldots\}$ is a 6-cover:

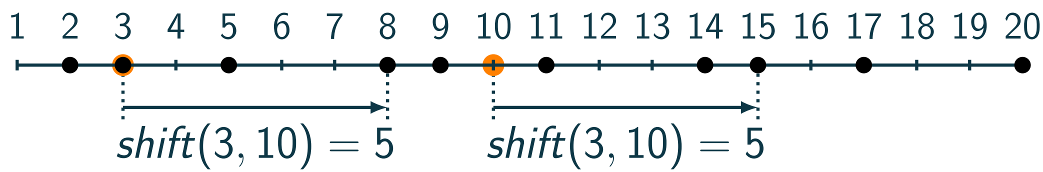


Difference Cover

Definition (Difference cover)

A set D is a d-cover if there is a function *shift* such that for any $i, j > 0$ we have $0 \leq shift(i,j) < d$ and $i + shift(i,j)$, $j + shift(i,j) \in D$.

Example. $\{2, 3, 5, 8, 9, 11, 14, 15, 19, 20, \ldots\}$ is a 6-cover:

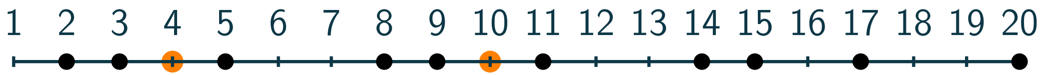


Difference Cover

Definition (Difference cover)

A set D is a d-cover if there is a function *shift* such that for any $i, j > 0$ we have $0 \leq \mathit{shift}(i, j) < d$ and $i + \mathit{shift}(i, j)$, $j + \mathit{shift}(i, j) \in D$.

Example. $\{2, 3, 5, 8, 9, 11, 14, 15, 19, 20, \ldots\}$ is a 6-cover:

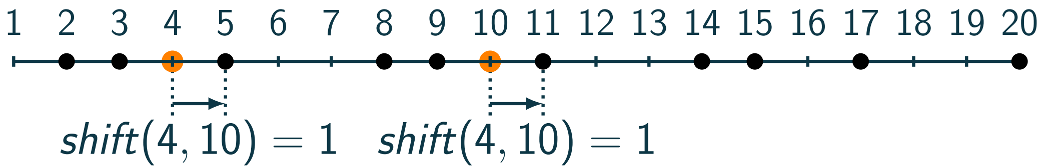


Difference Cover

Definition (Difference cover)

A set D is a d-cover if there is a function *shift* such that for any $i, j > 0$ we have $0 \leq \mathit{shift}(i,j) < d$ and $i + \mathit{shift}(i,j)$, $j + \mathit{shift}(i,j) \in D$.

Example. $\{2, 3, 5, 8, 9, 11, 14, 15, 19, 20, \ldots\}$ is a 6-cover:

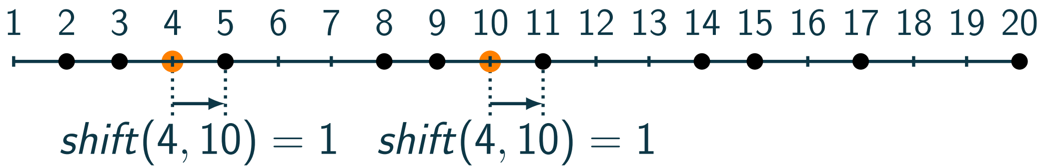


Difference Cover

Definition (Difference cover)

A set D is a d-cover if there is a function *shift* such that for any $i, j > 0$ we have $0 \le \mathit{shift}(i,j) < d$ and $i + \mathit{shift}(i,j)$, $j + \mathit{shift}(i,j) \in D$.

Example. $\{2, 3, 5, 8, 9, 11, 14, 15, 19, 20, \dots\}$ is a 6-cover:



Lemma ([1])

A d-cover D such that $D \cap [1, n]$ is of size $O(n/\sqrt{d})$ can be constructed in $O(n/\sqrt{d})$ time.

[1] M. Maekawa: A Square Root N Algorithm for Mutual Exclusion in Decentralized Systems. ACM Trans. Comput. Syst., 1985

Long LCF ($\geq \log^4 n$)

Definition (Difference cover)

A set D is a d-cover if there is a function *shift* such that for any $i, j > 0$ we have $0 \leq shift(i,j) < d$ and $i + shift(i,j), j + shift(i,j) \in D$.

Lemma (Maekawa, 1985)

A d-cover D such that $D \cap [1, n]$ is of size $O(n/\sqrt{d})$ can be constructed in $O(n/\sqrt{d})$ time.

Long LCF ($\geq \log^4 n$)

Definition (Difference cover)

A set D is a d-cover if there is a function *shift* such that for any $i, j > 0$ we have $0 \leq shift(i,j) < d$ and $i + shift(i,j), j + shift(i,j) \in D$.

Lemma (Maekawa, 1985)

A d-cover D such that $D \cap [1, n]$ is of size $O(n/\sqrt{d})$ can be constructed in $O(n/\sqrt{d})$ time.

- Use the lemma for ST, $2n$ pos., and $d = \ell$; the ℓ-cover has size $O(n/\log^2 n)$.

Long LCF ($\geq \log^4 n$)

Definition (Difference cover)

A set D is a d-cover if there is a function *shift* such that for any $i, j > 0$ we have $0 \leq \mathit{shift}(i,j) < d$ and $i + \mathit{shift}(i,j), j + \mathit{shift}(i,j) \in D$.

Lemma (Maekawa, 1985)

A d-cover D such that $D \cap [1, n]$ is of size $O(n/\sqrt{d})$ can be constructed in $O(n/\sqrt{d})$ time.

- Use the lemma for ST, $2n$ pos., and $d = \ell$; the ℓ-cover has size $O(n/\log^2 n)$.
- Let $\ell = \log^4 n$. An ℓ-cover is useful in computing LCF of length $\geq \ell$:

```
c b b a a b c a b c a b c b a
  •     •     •     •
  2     5     8     11

d a a b a a b c b b a
            •     •
            6     9
```

subset of 4-cover

Long LCF ($\geq \log^4 n$)

Definition (Difference cover)

A set D is a d-cover if there is a function *shift* such that for any $i, j > 0$ we have $0 \leq \mathit{shift}(i,j) < d$ and $i + \mathit{shift}(i,j), j + \mathit{shift}(i,j) \in D$.

Lemma (Maekawa, 1985)

A d-cover D such that $D \cap [1,n]$ is of size $O(n/\sqrt{d})$ can be constructed in $O(n/\sqrt{d})$ time.

- Use the lemma for ST, $2n$ pos., and $d = \ell$; the ℓ-cover has size $O(n/\log^2 n)$.
- Let $\ell = \log^4 n$. An ℓ-cover is useful in computing LCF of length $\geq \ell$:

c b b a a b c a b c a b c b a

2 5 8 11

d a a b a a b c b b a

6 9

subset of 4-cover

Long LCF ($\geq \log^4 n$)

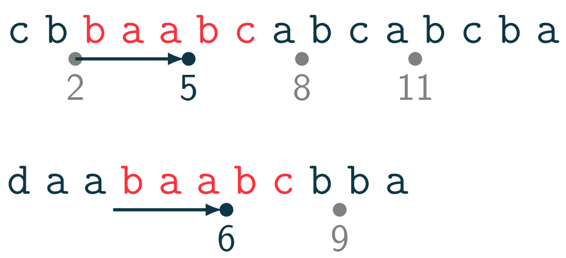
Definition (Difference cover)

A set D is a d-cover if there is a function *shift* such that for any $i, j > 0$ we have $0 \leq shift(i,j) < d$ and $i + shift(i,j), j + shift(i,j) \in D$.

Lemma (Maekawa, 1985)

A d-cover D such that $D \cap [1,n]$ is of size $O(n/\sqrt{d})$ can be constructed in $O(n/\sqrt{d})$ time.

- Use the lemma for ST, $2n$ pos., and $d = \ell$; the ℓ-cover has size $O(n/\log^2 n)$.
- Let $\ell = \log^4 n$. An ℓ-cover is useful in computing LCF of length $\geq \ell$:



subset of 4-cover

Long LCF ($\geq \log^4 n$)

Definition (Difference cover)

A set D is a d-cover if there is a function *shift* such that for any $i, j > 0$ we have $0 \leq shift(i,j) < d$ and $i + shift(i,j), j + shift(i,j) \in D$.

Lemma (Maekawa, 1985)

A d-cover D such that $D \cap [1, n]$ is of size $O(n/\sqrt{d})$ can be constructed in $O(n/\sqrt{d})$ time.

- Use the lemma for ST, $2n$ pos., and $d = \ell$; the ℓ-cover has size $O(n/\log^2 n)$.
- Let $\ell = \log^4 n$. An ℓ-cover is useful in computing LCF of length $\geq \ell$:

c b b a a b c a b c a b c b a

2 5 8 11

d a a b a a b c b b a

6 9

subset of 4-cover

Long LCF ($\geq \log^4 n$)

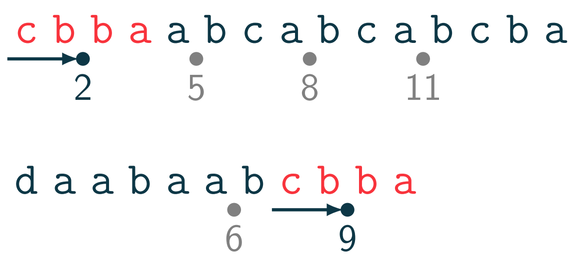
Definition (Difference cover)

A set D is a d-cover if there is a function *shift* such that for any $i, j > 0$ we have $0 \leq shift(i,j) < d$ and $i + shift(i,j), j + shift(i,j) \in D$.

Lemma (Maekawa, 1985)

A d-cover D such that $D \cap [1, n]$ is of size $O(n/\sqrt{d})$ can be constructed in $O(n/\sqrt{d})$ time.

- Use the lemma for ST, $2n$ pos., and $d = \ell$; the ℓ-cover has size $O(n/\log^2 n)$.
- Let $\ell = \log^4 n$. An ℓ-cover is useful in computing LCF of length $\geq \ell$:



subset of 4-cover

Long LCF ($\geq \log^4 n$)

Definition (Difference cover)

A set D is a d-cover if there is a function *shift* such that for any $i, j > 0$ we have $0 \leq \mathit{shift}(i,j) < d$ and $i + \mathit{shift}(i,j), j + \mathit{shift}(i,j) \in D$.

Lemma (Maekawa, 1985)

A d-cover D such that $D \cap [1,n]$ is of size $O(n/\sqrt{d})$ can be constructed in $O(n/\sqrt{d})$ time.

- Use the lemma for ST, $2n$ pos., and $d = \ell$; the ℓ-cover has size $O(n/\log^2 n)$.
- Let $\ell = \log^4 n$. An ℓ-cover is useful in computing LCF of length $\geq \ell$:

```
c b b a a b c a b c a b c b a
  •     •     •     •
  2     5     8     11

d a a b a a b c b b a
          •     •
          6     9
```

subset of 4-cover

Long LCF ($\geq \log^4 n$)

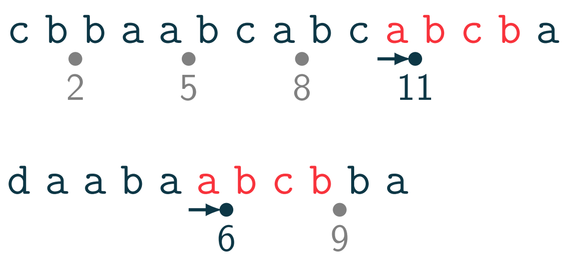
Definition (Difference cover)

A set D is a d-cover if there is a function *shift* such that for any $i, j > 0$ we have $0 \leq \mathit{shift}(i,j) < d$ and $i + \mathit{shift}(i,j), j + \mathit{shift}(i,j) \in D$.

Lemma (Maekawa, 1985)

A d-cover D such that $D \cap [1, n]$ is of size $O(n/\sqrt{d})$ can be constructed in $O(n/\sqrt{d})$ time.

- Use the lemma for ST, $2n$ pos., and $d = \ell$; the ℓ-cover has size $O(n/\log^2 n)$.
- Let $\ell = \log^4 n$. An ℓ-cover is useful in computing LCF of length $\geq \ell$:



subset of 4-cover

Long LCF ($\geq \log^4 n$)

- LCF anchored at a pair of elements of a difference cover

```
1 c b b a a b c a b c a b c b a 1
     •     •     •     •
     2     5     8     11

2 d a a b a a b c b b a 2
             •     •
             6     9
```

Long LCF ($\geq \log^4 n$)

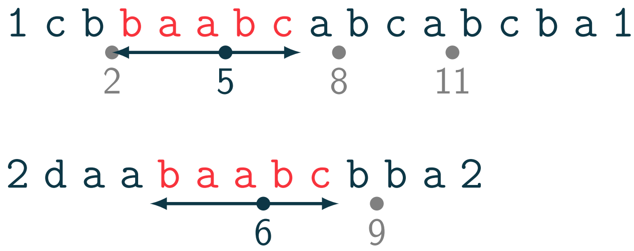
- LCF anchored at a pair of elements of a difference cover

```
1 c b b a a b c a b c a b c b a 1
       •     •     •     •
       2     5     8     11

2 d a a b a a b c b b a 2
               •     •
               6     9
```

Long LCF ($\geq \log^4 n$)

- LCF anchored at a pair of elements of a difference cover



Long LCF ($\geq \log^4 n$)

- LCF anchored at a pair of elements of a difference cover

1 c b b a a b c a b c a b c b a 1

2 5 8 11

2 d a a b a a b c b b a 2

6 9

Reversed prefixes:

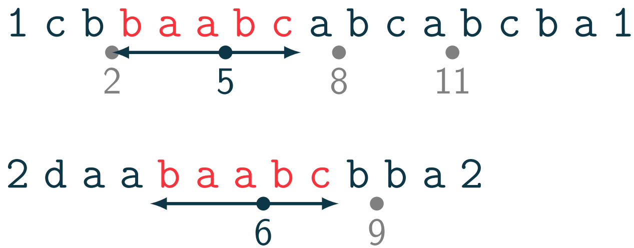
2_S bc1
5_S aabbc1
8_S acbaabbc1
11_S acbacbaabbc1
6_T aabaad2
9_T bcbaabaad2

Suffixes:

2_S baabcabcabcba1
5_S bcabcabcba1
8_S bcabcba1
11_S bcba1
6_T bcbba2
9_T ba2

Long LCF ($\geq \log^4 n$)

- LCF anchored at a pair of elements of a difference cover



Reversed prefixes:

2_S bc1
5_S aabbc1
8_S acbaabbc1
11_S acbacbaabbc1
6_T aabaad2
9_T bcbaabaad2

Suffixes:

2_S baabcabcabcba1
5_S bcabcabcba1
8_S bcabcba1
11_S bcba1
6_T bcbba2
9_T ba2

Long LCF ($\geq \log^4 n$)

- LCF anchored at a pair of elements of a difference cover

1 c b b a a b c a b c a b c b a 1

2 5 8 11

2 d a a b a a b c b b a 2

6 9

Reversed prefixes:		Suffixes:		
2_S	bc1	2_S	baabcabcabcba1	
5_S	aabbc1	5_S	bcabcabcba1	
8_S	acbaabbc1	8_S	bcabcba1	$\mathcal{F}_S$
11_S	acbacbaabbc1	11_S	bcba1	
6_T	aabaad2	6_T	bcbba2	$\mathcal{F}_T$
9_T	bcbaabaad2	9_T	ba2	

MaxPairLCP problem:

Input: two families of pairs of strings, $\mathcal{F}_S$ and $\mathcal{F}_T$, of total size N

Output: $\max\{LCP(P_1, Q_1) + LCP(P_2, Q_2) : (P_1, P_2) \in \mathcal{F}_S, (Q_1, Q_2) \in \mathcal{F}_T\}$

Representation of MaxPairLCP

Reversed prefixes:

2_S `bc1`
5_S `aabbc1`
8_S `acbaabbc1`
11_S `acbacbaabbc1`
6_T `aabaad2`
9_T `bcbaabaad2`

Suffixes:

2_S `baabcabcabcba1`
5_S `bcabcabcba1`
8_S `bcabcba1`
11_S `bcba1`
6_T `bcbba2`
9_T `ba2`

Representation of MaxPairLCP

Reversed prefixes:

aabaad2 (6_T)

aabbc1 (5_S)

acbaabbc1 (8_S)

acbacbaabbc1 (11_S)

bc1 (2_S)

bcbaabaad2 (9_T)

Suffixes:

ba2 (9_T)

baabcabcabcba1 (2_S)

bcabcabcba1 (5_S)

bcabcba1 (8_S)

bcba1 (11_S)

bcbba2 (6_T)

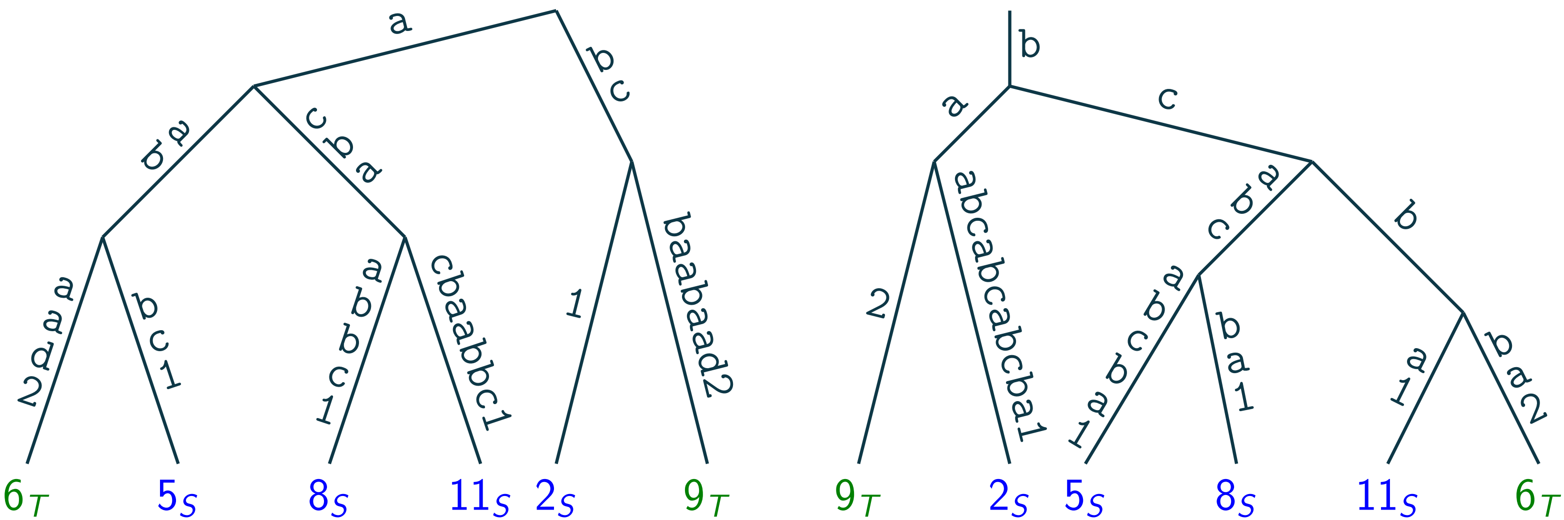
Representation of MaxPairLCP

Reversed prefixes:

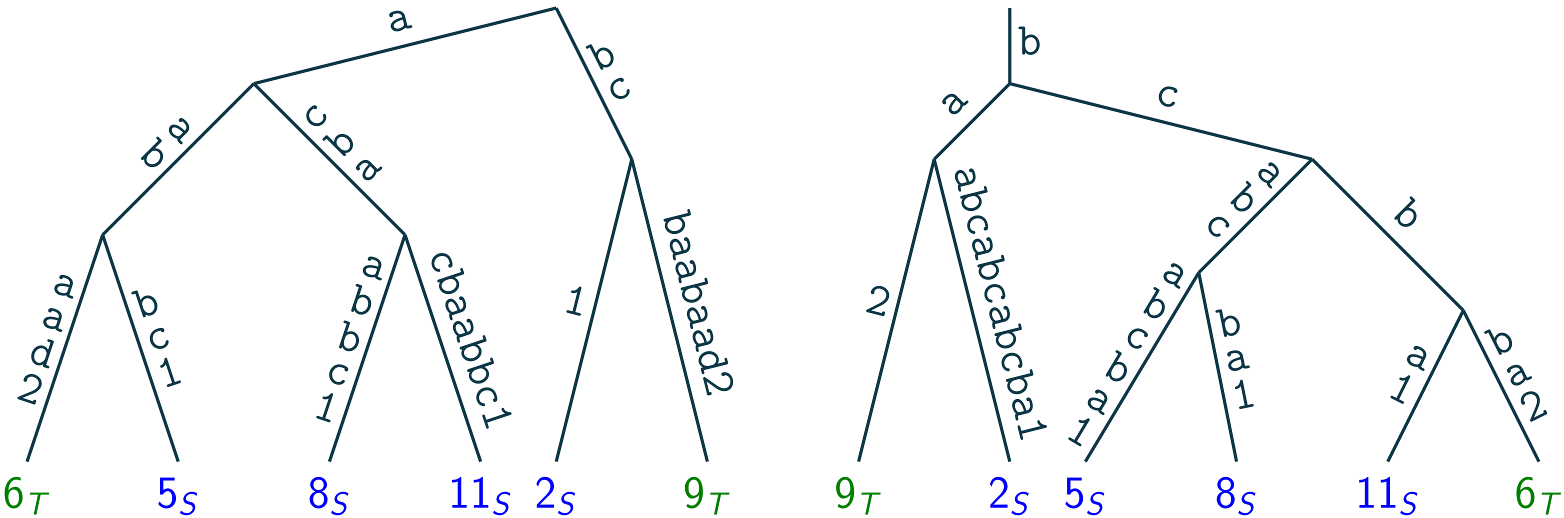
aabaad2 (6_T)
aabbc1 (5_S)
acbaabbc1 (8_S)
acbacbaabbc1 (11_S)
bc1 (2_S)
bcbaabaad2 (9_T)

Suffixes:

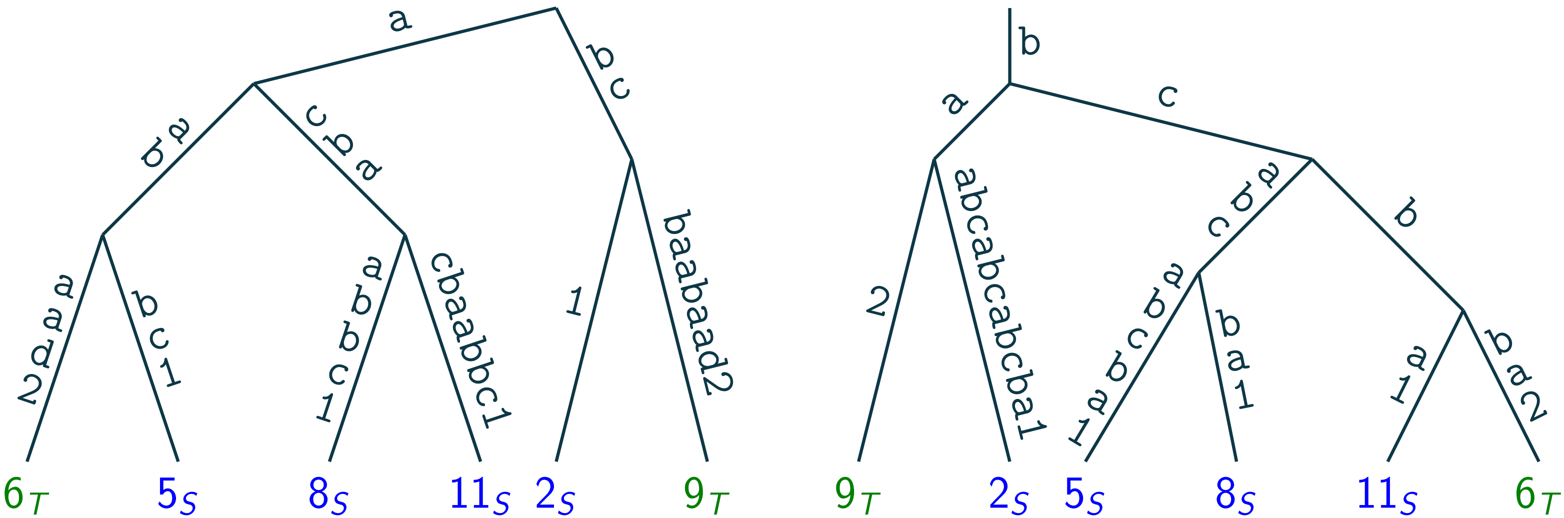
ba2 (9_T)
baabcabcabcba1 (2_S)
bcabcabcba1 (5_S)
bcabcba1 (8_S)
bcba1 (11_S)
bcbba2 (6_T)



Representation of MaxPairLCP

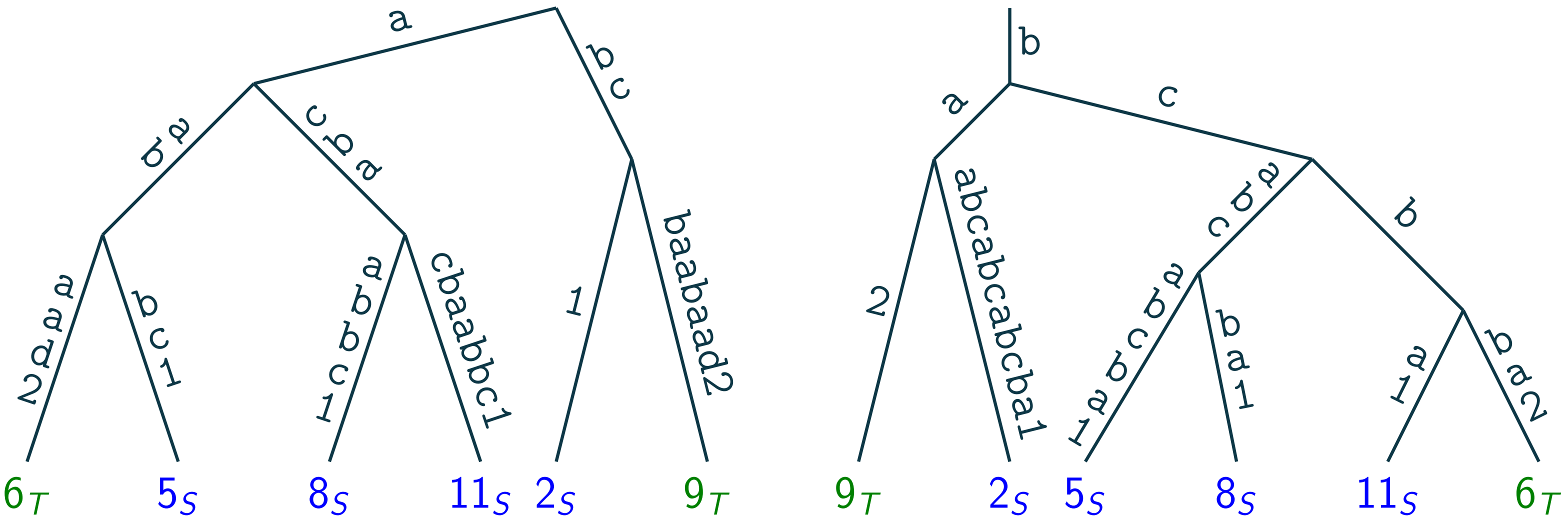


Representation of MaxPairLCP



- The difference cover has size $N = O(n/\log^2 n)$.
- The tries after compactification have $O(N)$ leaves and size $O(N)$.

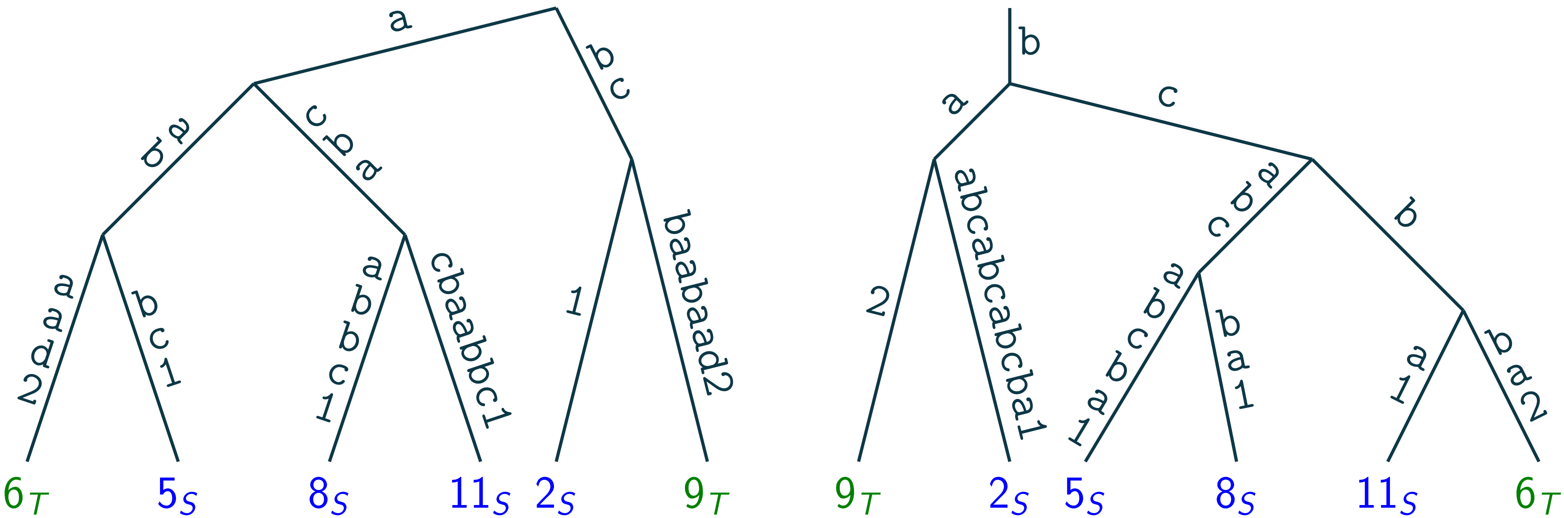
Representation of MaxPairLCP



- The difference cover has size $N = O(n/\log^2 n)$.
- The tries after compactification have $O(N)$ leaves and size $O(N)$.

Lemma. The tries can be constructed in $O(N \log N + n/\log_\sigma n)$ time.

Representation of MaxPairLCP



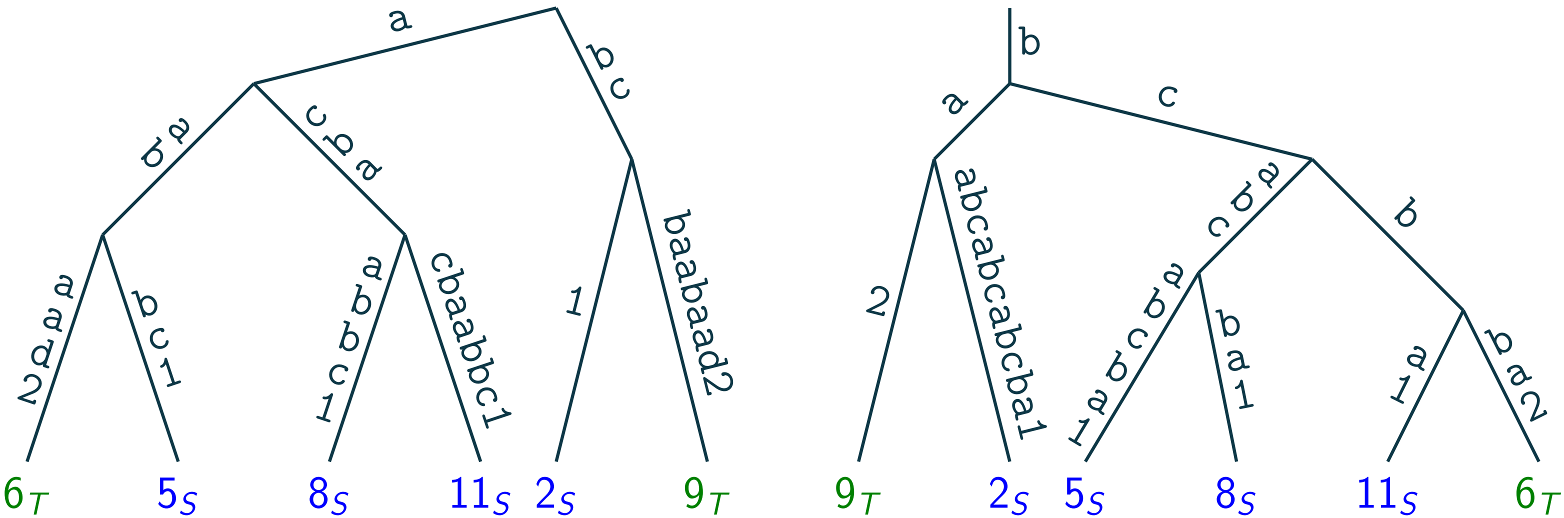
- The difference cover has size $N = O(n/\log^2 n)$.
- The tries after compactification have $O(N)$ leaves and size $O(N)$.

Lemma. The tries can be constructed in $O(N \log N + n/\log_\sigma n)$ time.

Proof. Merge Sort of strings and $O(1)$-time LCP-queries from [1].

[1] D. Kempa, T. Kociumaka: String Synchronizing Sets: Sublinear-time BWT Construction and Optimal LCE Data Structure. STOC 2019

Solution to MaxPairLCP



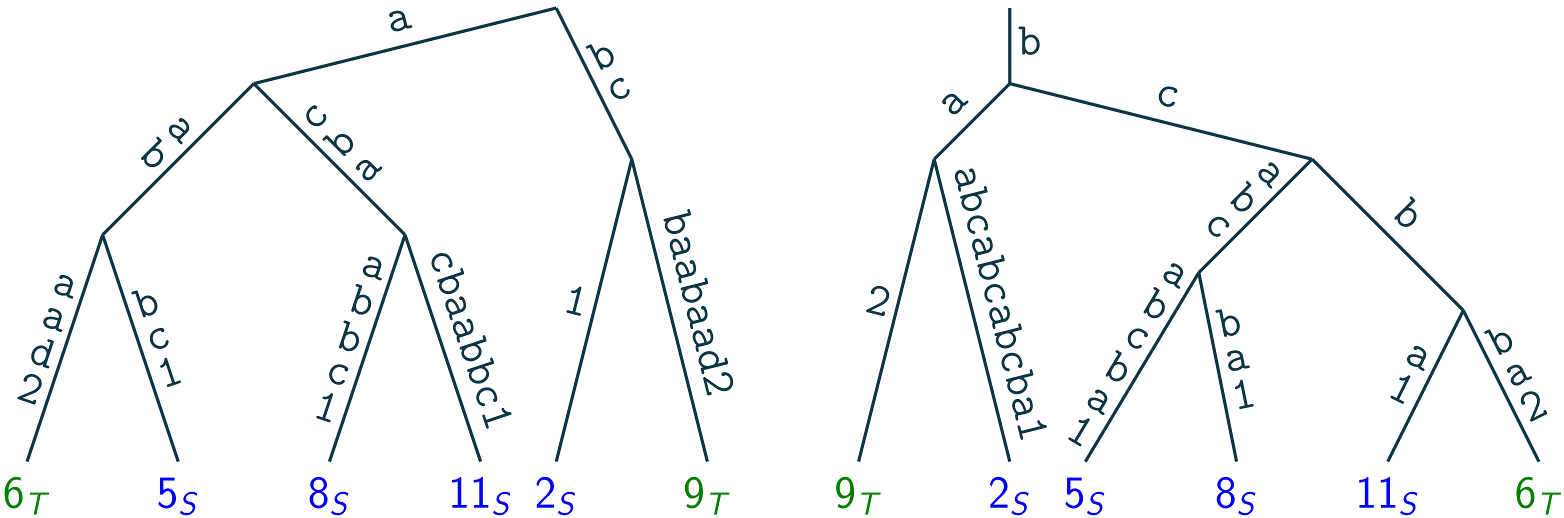
1 c b b a a b c a b c a b c b a 1

2 5 8 11

2 d a a b a a b c b b a 2

6 9

Solution to MaxPairLCP



Goal:

- Nodes u, v from both tries
- Their subtrees have a pair of equal leaves, one from S and one from T
- $depth(u) + depth(v)$ is max

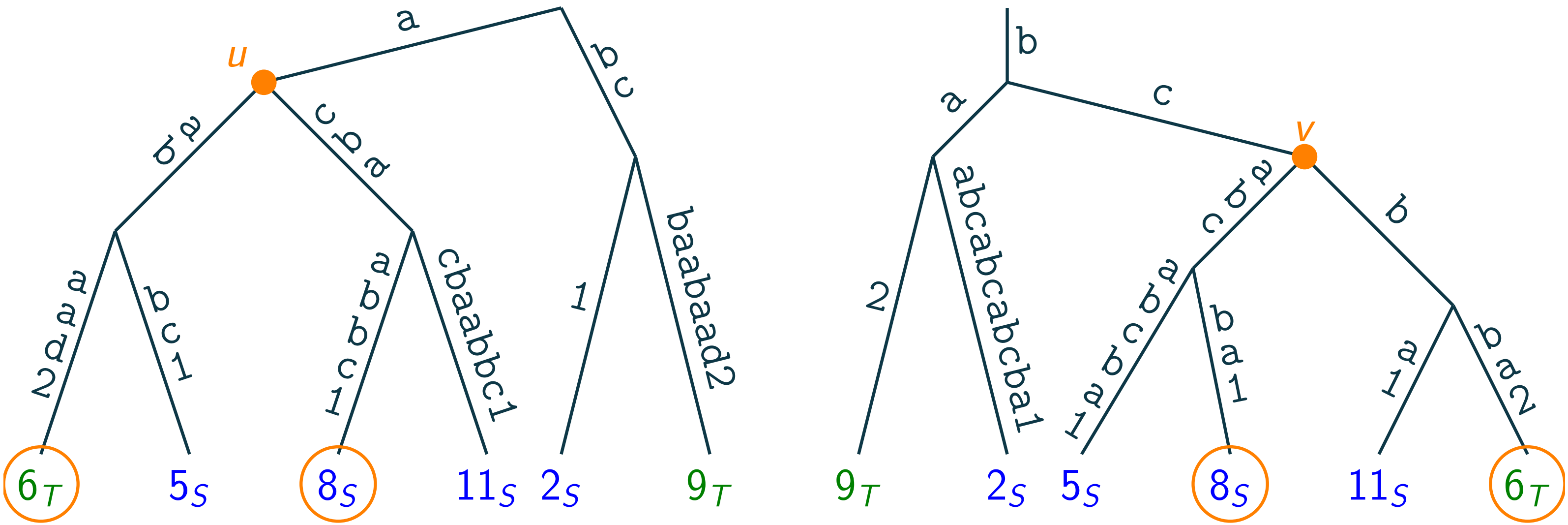
1 c b b a a b c a b c a b c b a 1

2 5 8 11

2 d a a b a a b c b b a 2

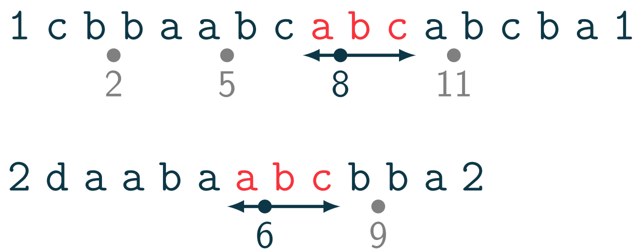
6 9

Solution to MaxPairLCP

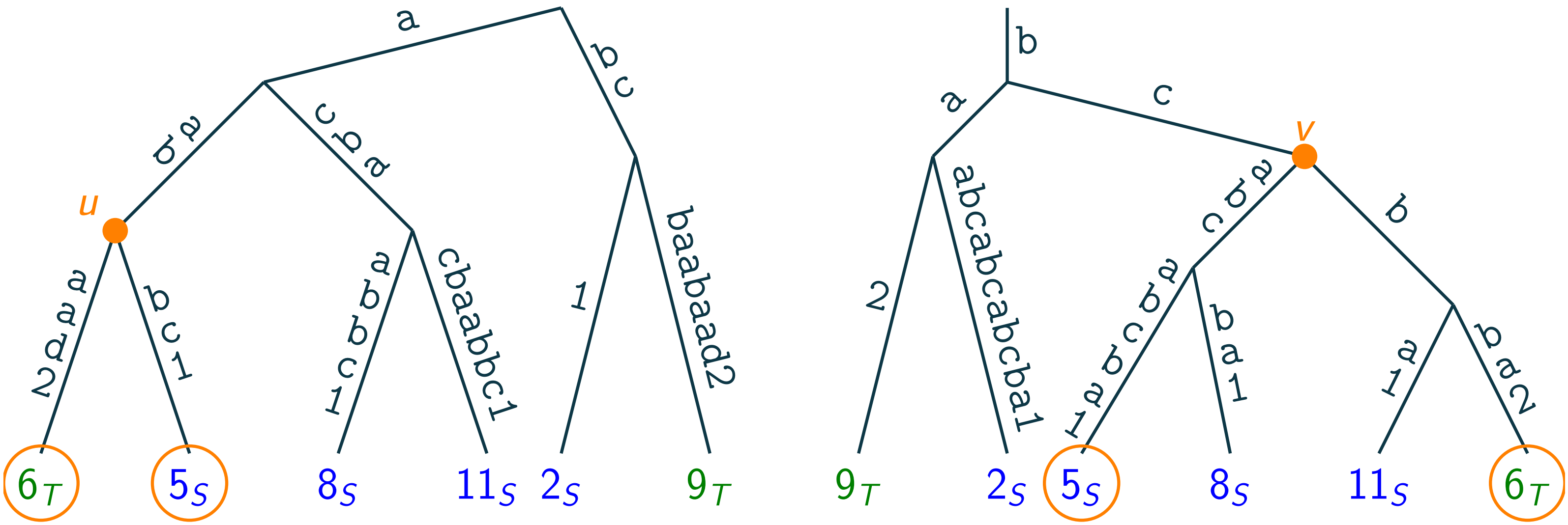


Goal:

- Nodes u, v from both tries
- Their subtrees have a pair of equal leaves, one from S and one from T
- $depth(u) + depth(v)$ is max

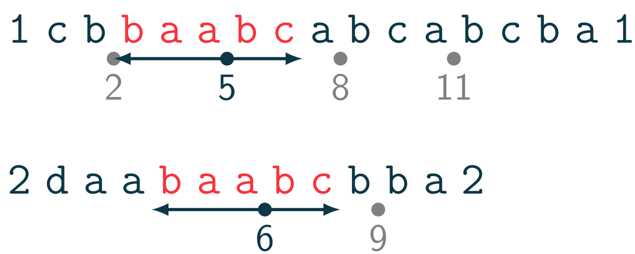


Solution to MaxPairLCP

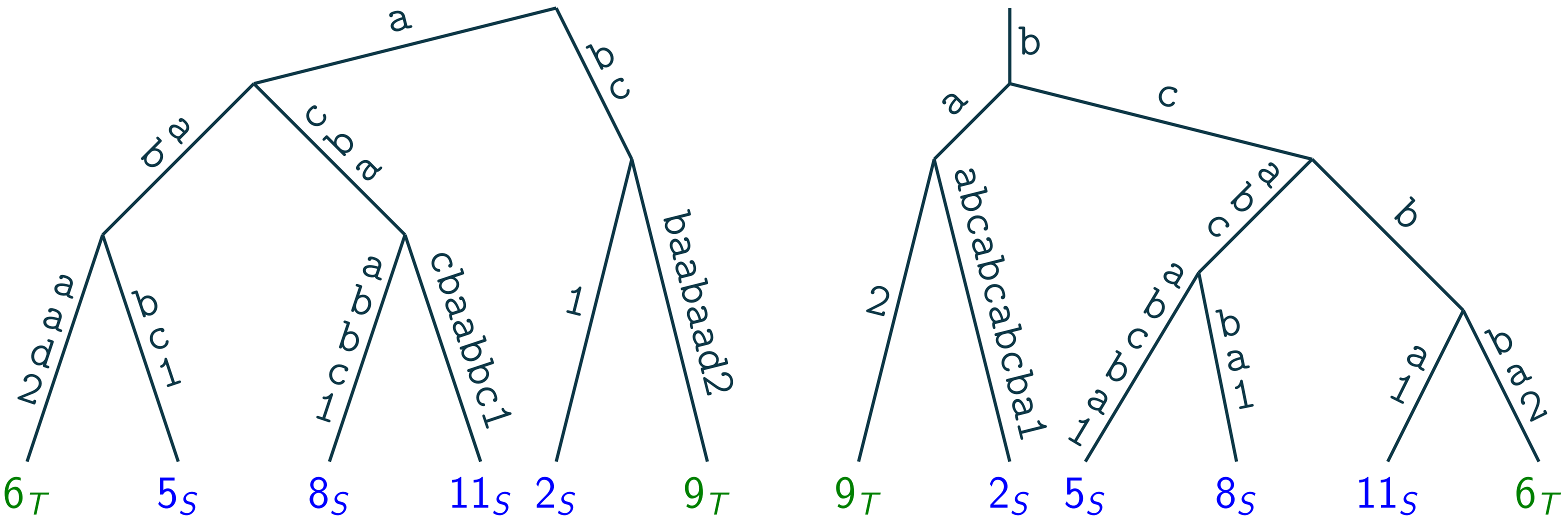


Goal:

- Nodes u, v from both tries
- Their subtrees have a pair of equal leaves, one from S and one from T
- $depth(u) + depth(v)$ is max



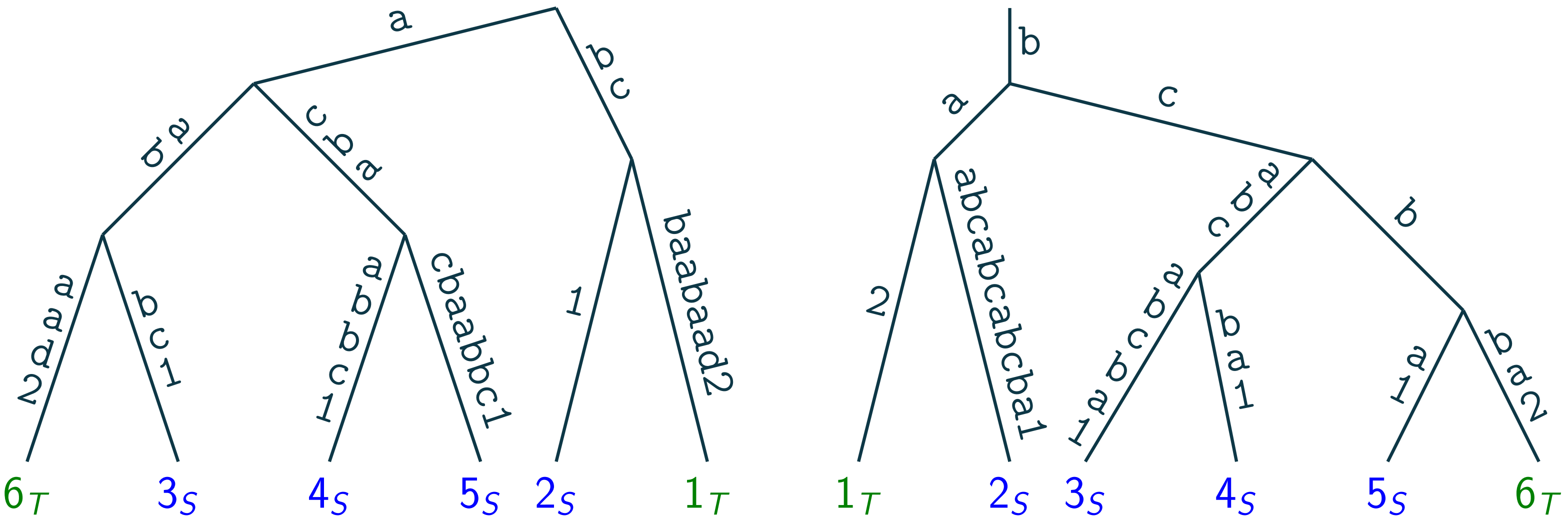
Solution to MaxPairLCP



Goal:

- Nodes u, v from both tries
- Their subtrees have a pair of equal leaves, one from S and one from T
- $depth(u) + depth(v)$ is max

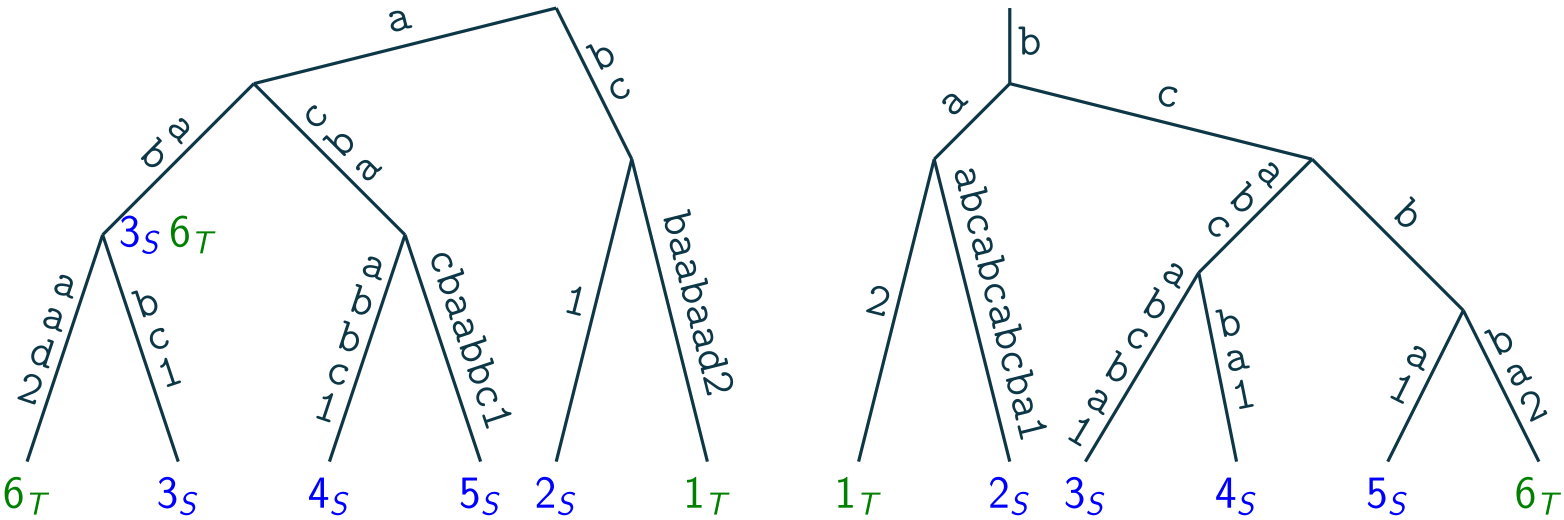
Solution to MaxPairLCP



Goal:

- Nodes u, v from both tries
- Their subtrees have a pair of equal leaves, one from S and one from T
- $depth(u) + depth(v)$ is max

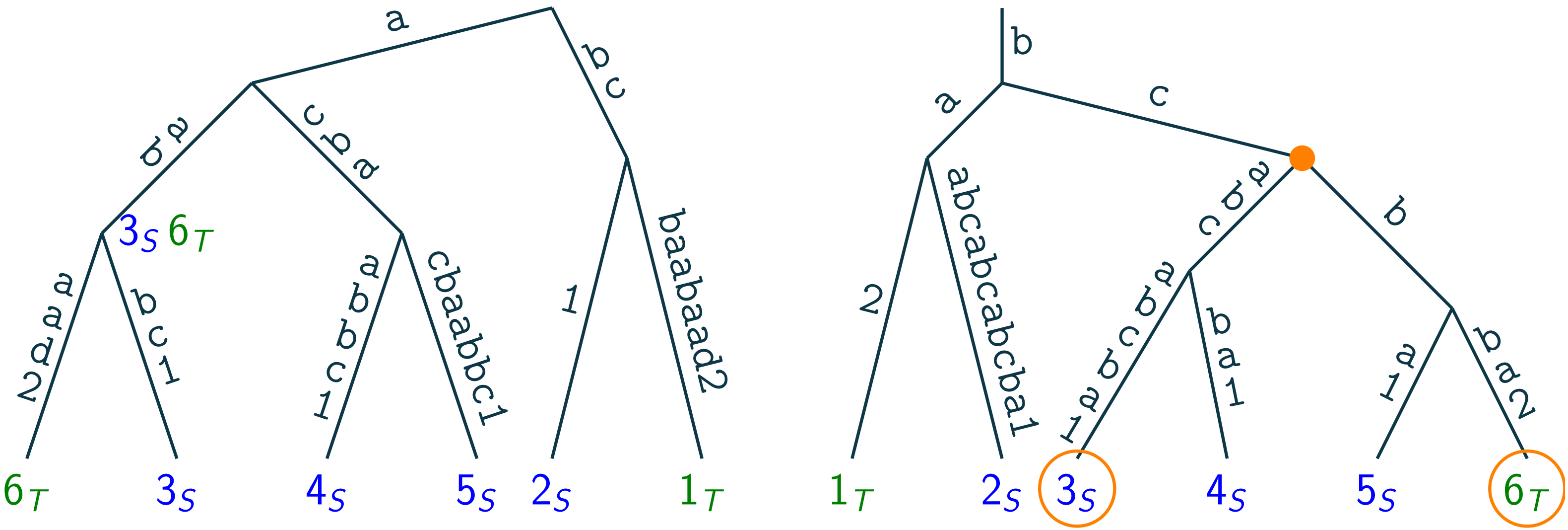
Solution to MaxPairLCP



Goal:

- Nodes u, v from both tries
- Their subtrees have a pair of equal leaves, one from S and one from T
- $depth(u) + depth(v)$ is max

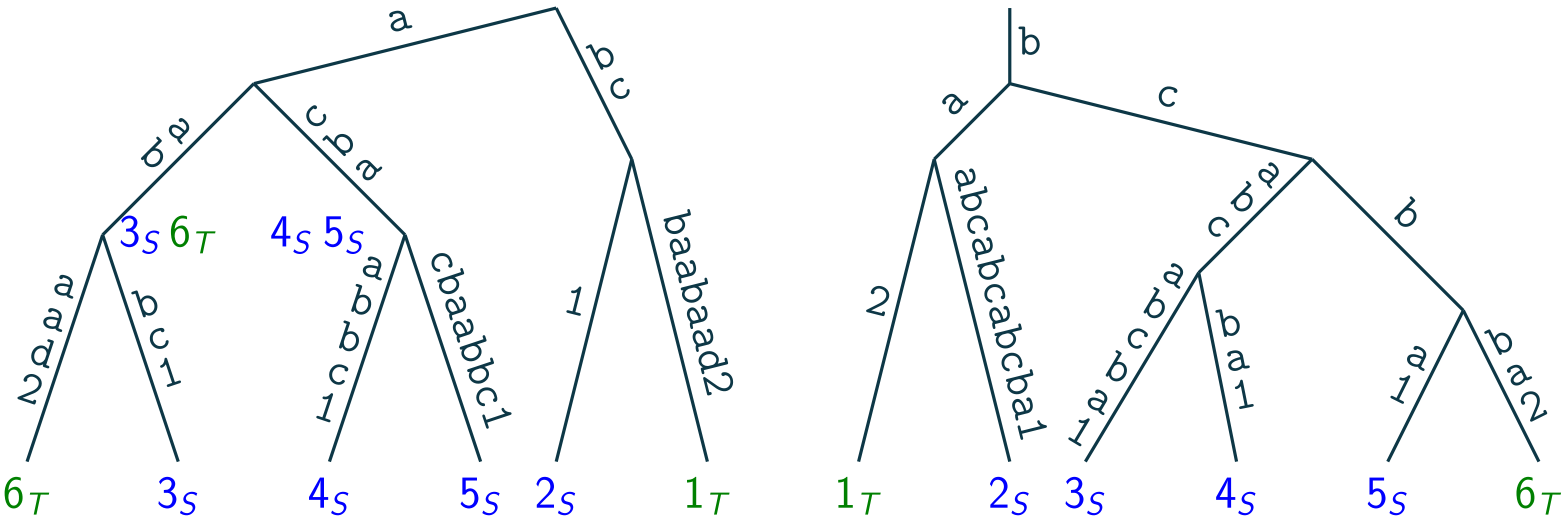
Solution to MaxPairLCP



Goal:

- Nodes u, v from both tries
- Their subtrees have a pair of equal leaves, one from S and one from T
- $depth(u) + depth(v)$ is max

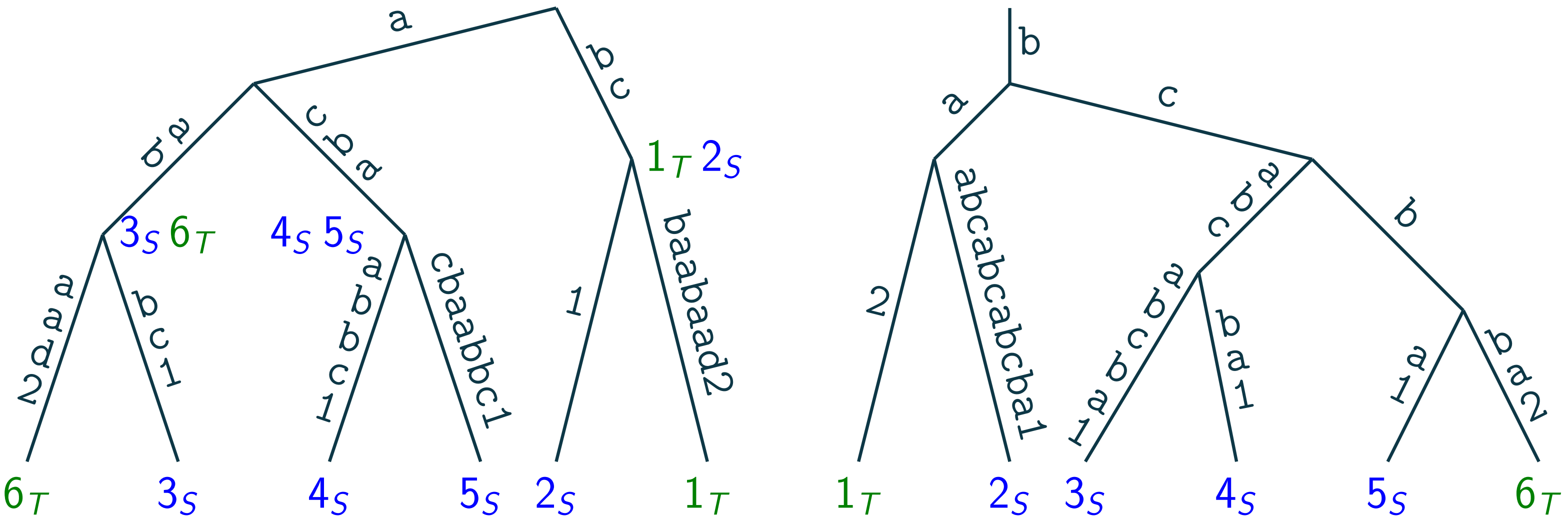
Solution to MaxPairLCP



Goal:

- Nodes u, v from both tries
- Their subtrees have a pair of equal leaves, one from S and one from T
- $depth(u) + depth(v)$ is max

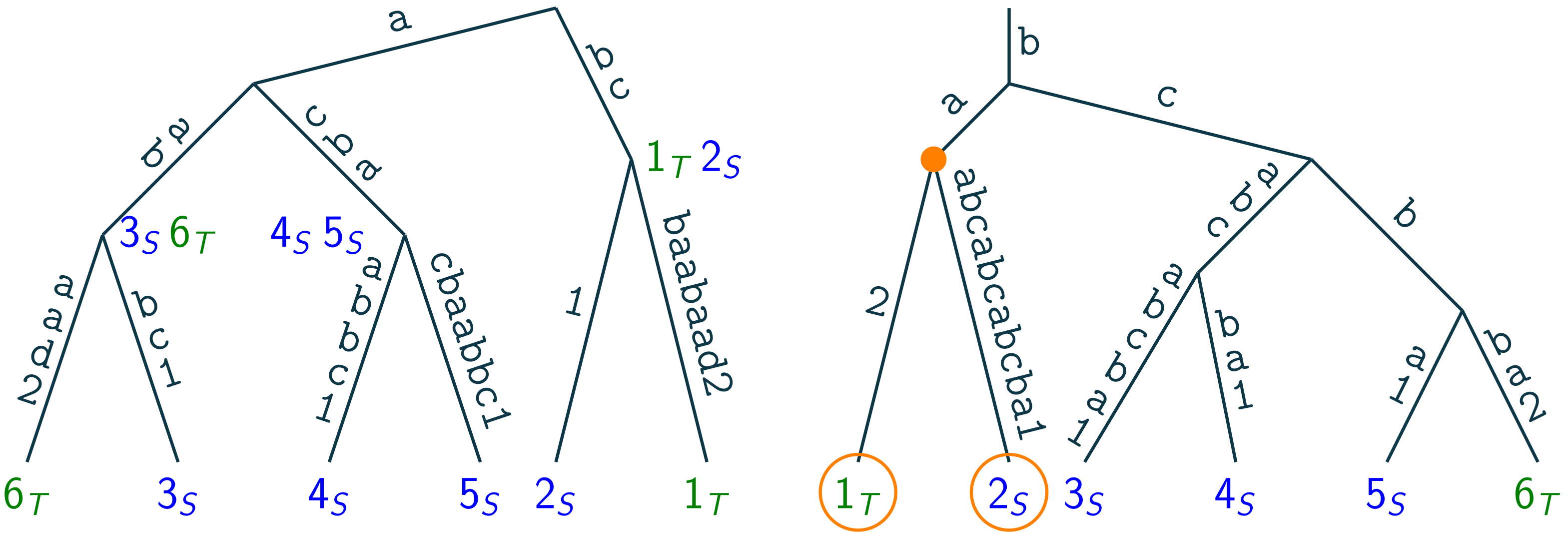
Solution to MaxPairLCP



Goal:

- Nodes u, v from both tries
- Their subtrees have a pair of equal leaves, one from S and one from T
- $depth(u) + depth(v)$ is max

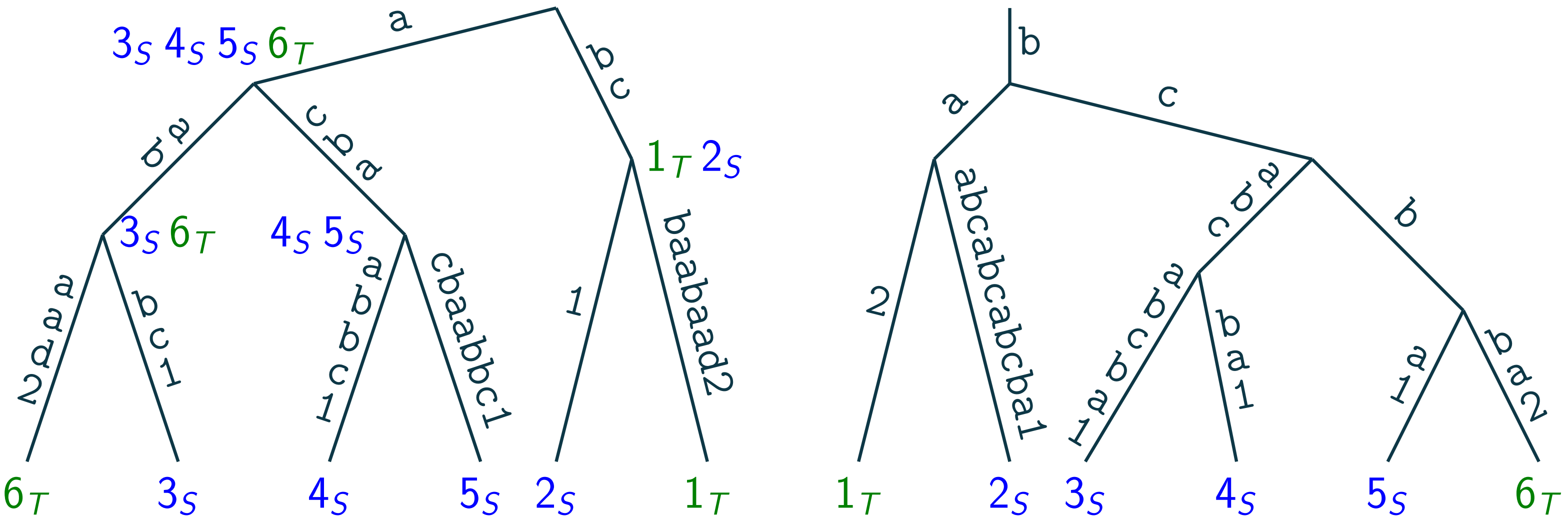
Solution to MaxPairLCP



Goal:

- Nodes u, v from both tries
- Their subtrees have a pair of equal leaves, one from S and one from T
- $depth(u) + depth(v)$ is max

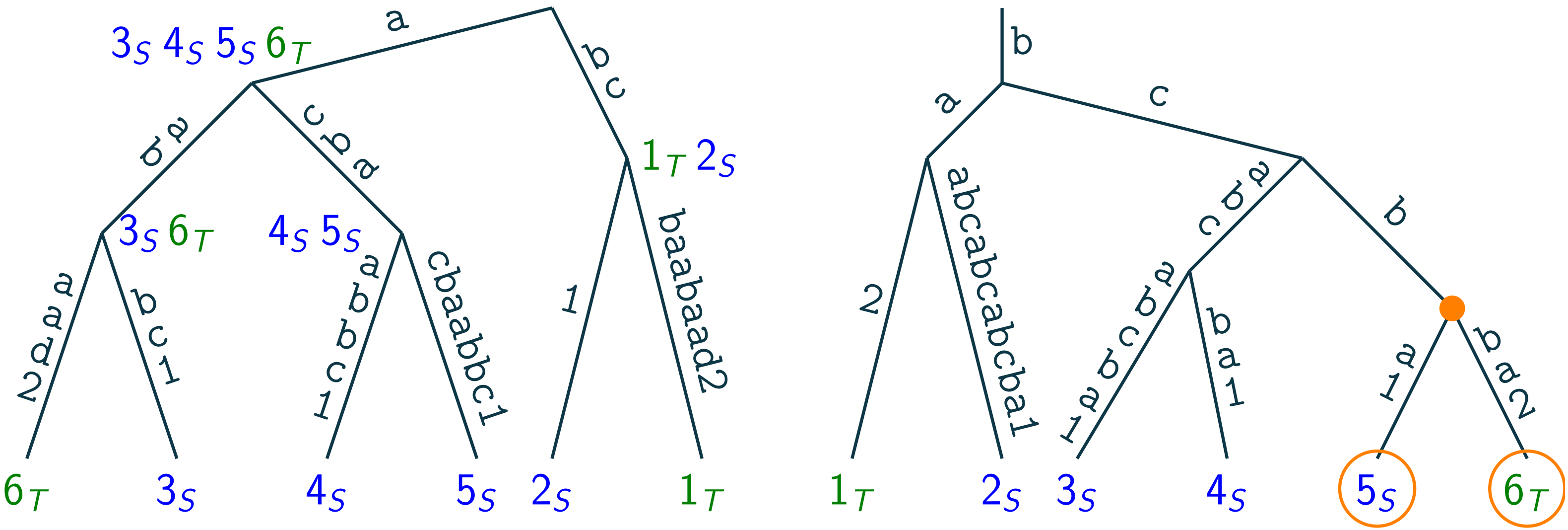
Solution to MaxPairLCP



Goal:

- Nodes u, v from both tries
- Their subtrees have a pair of equal leaves, one from S and one from T
- $depth(u) + depth(v)$ is max

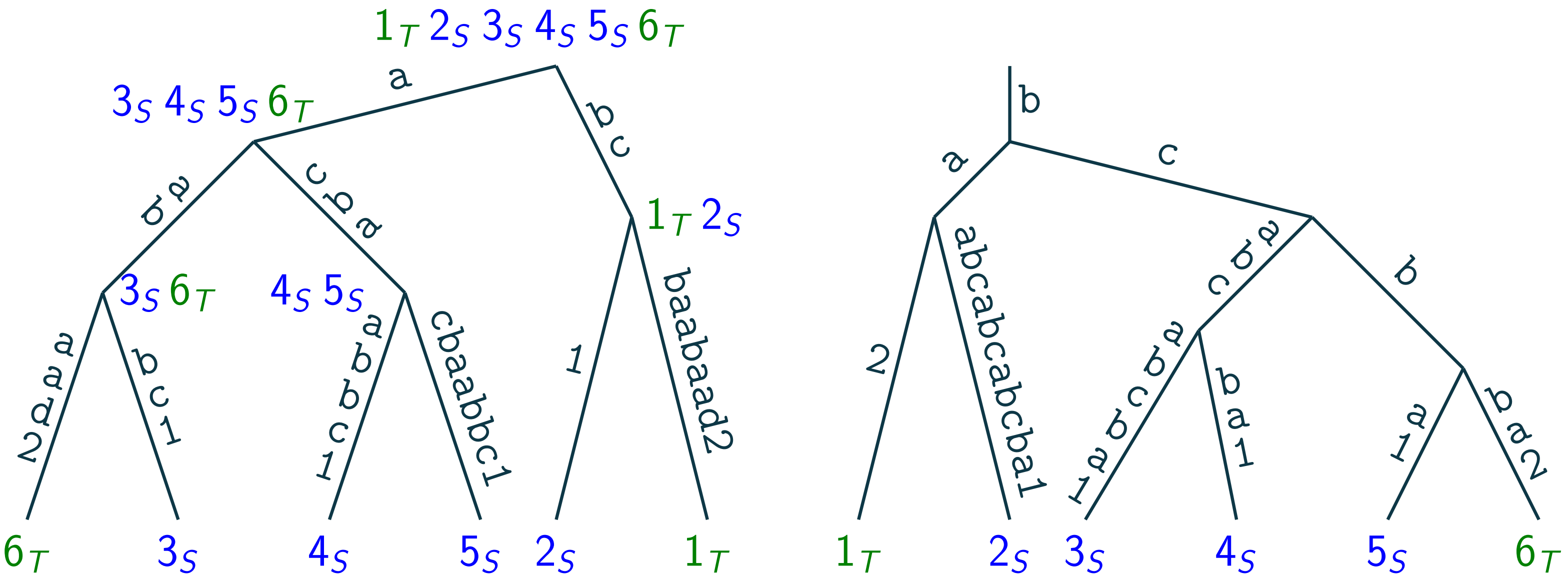
Solution to MaxPairLCP



Goal:

- Nodes u, v from both tries
- Their subtrees have a pair of equal leaves, one from S and one from T
- $depth(u) + depth(v)$ is max

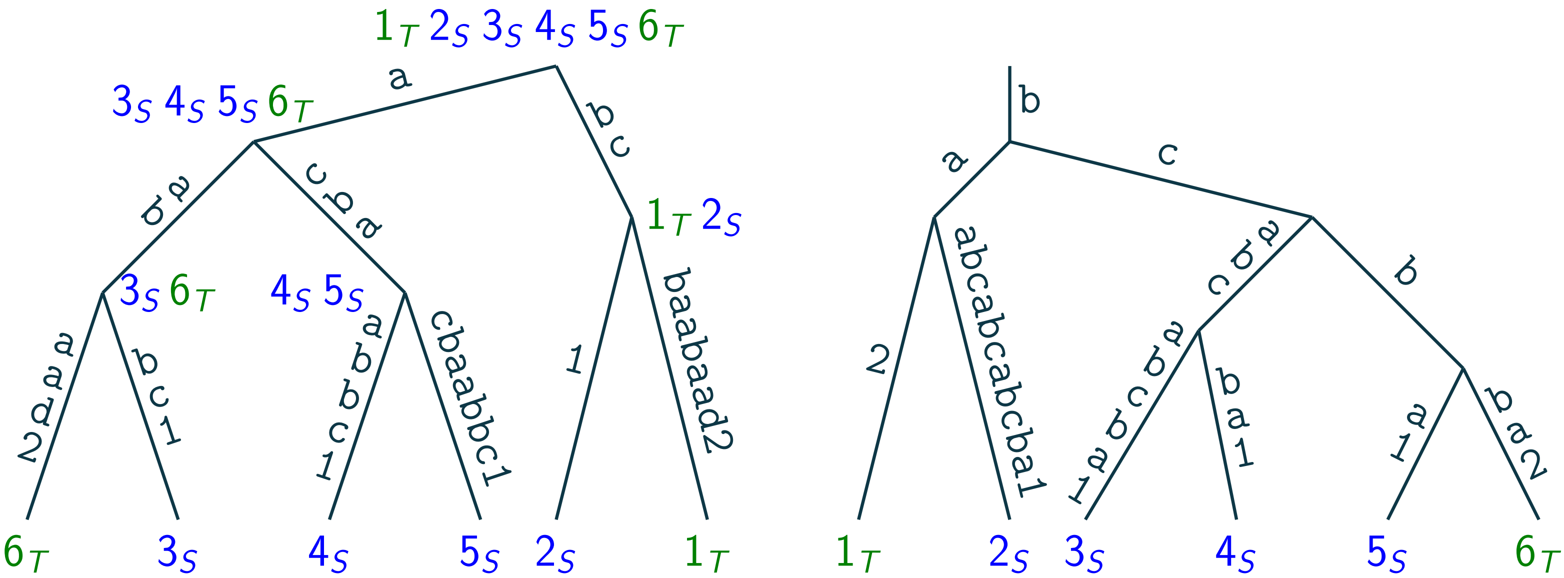
Solution to MaxPairLCP



Goal:

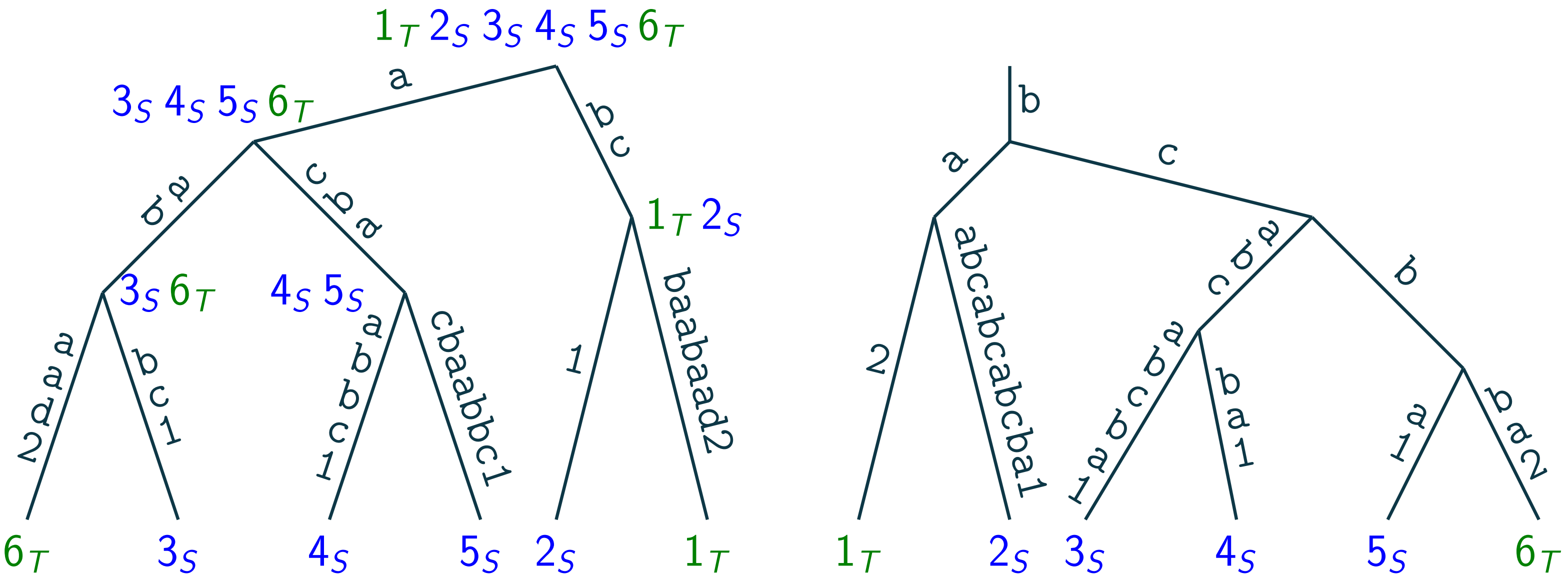
- Nodes u, v from both tries
- Their subtrees have a pair of equal leaves, one from S and one from T
- $depth(u) + depth(v)$ is max

Solution to MaxPairLCP



- Sorted lists can be stored using balanced BSTs (e.g. AVLs)

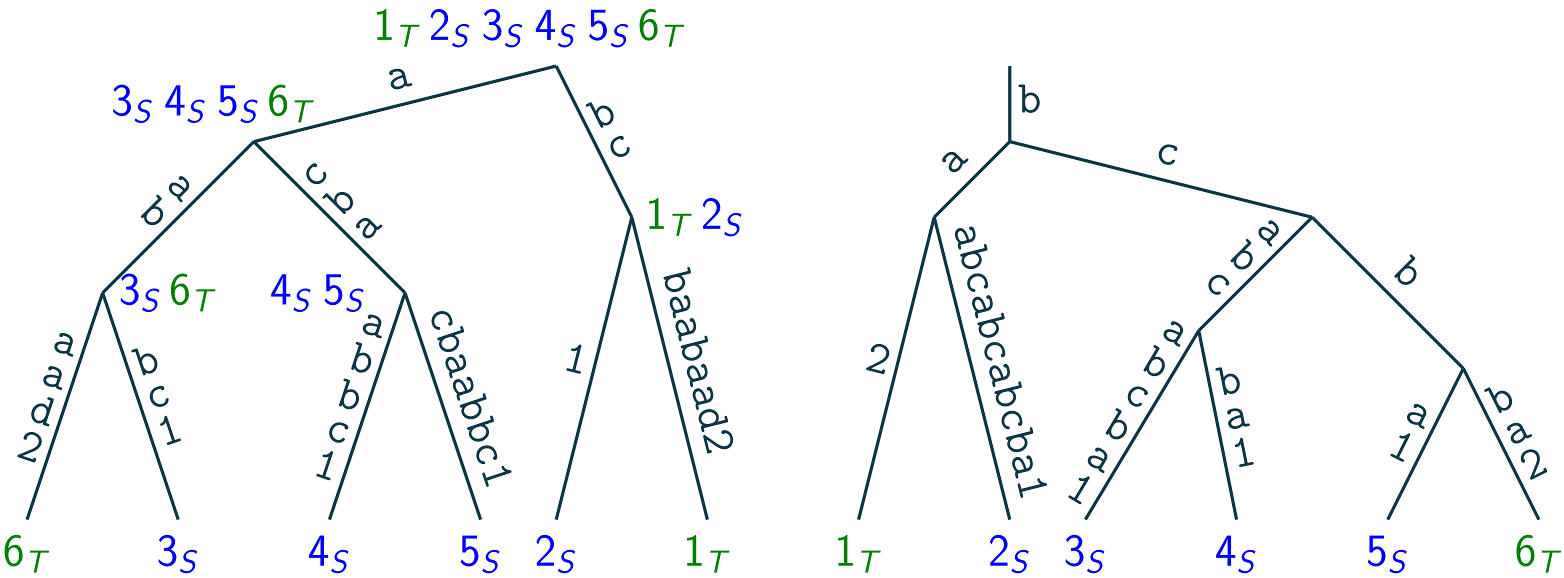
Solution to MaxPairLCP



- Sorted lists can be stored using balanced BSTs (e.g. AVLs)
- Merging AVLs takes $O(N \log N)$ total time ($N =$ size of trie) [1]

[1] M.R. Brown, R.E. Tarjan: A Fast Merging Algorithm. J. ACM, 1979

Solution to MaxPairLCP

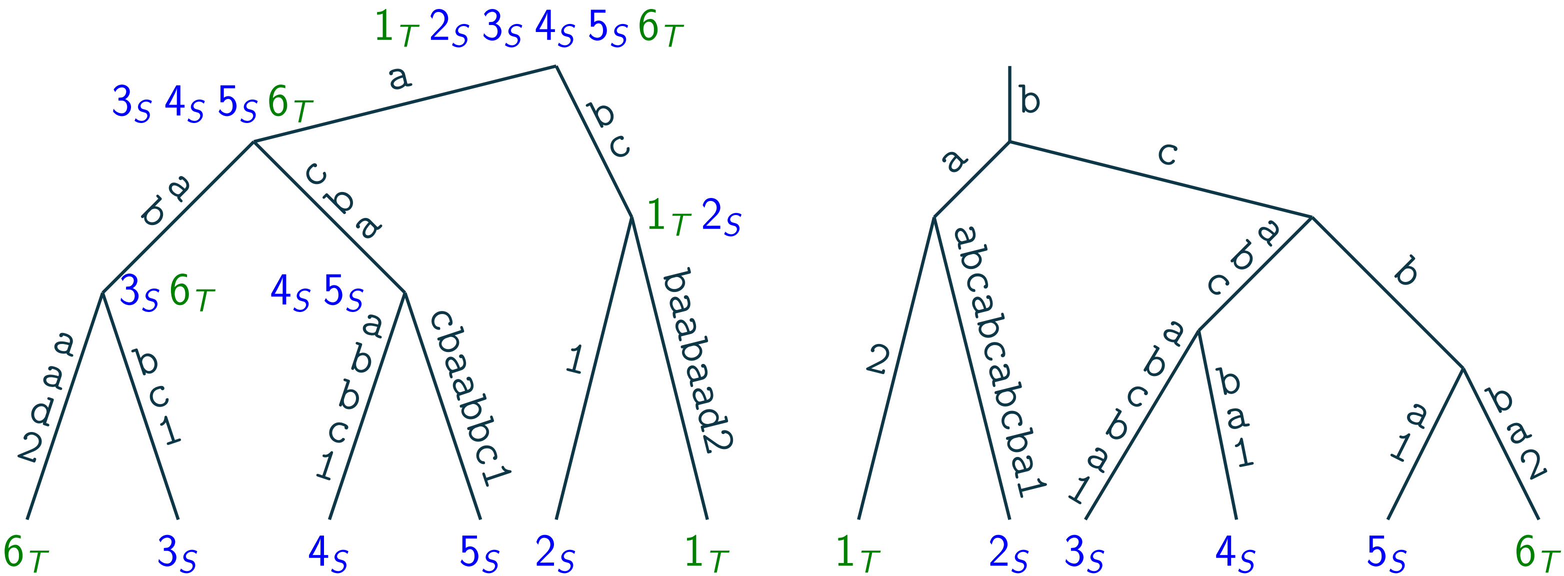


- Sorted lists can be stored using balanced BSTs (e.g. AVLs)
- Merging AVLs takes $O(N \log N)$ total time ($N =$ size of trie) [1]
- LCAs can be computed in $O(1)$ time after $O(N)$-time preprocessing [2]

[1] M.R. Brown, R.E. Tarjan: A Fast Merging Algorithm. J. ACM, 1979

[2] M.A. Bender, M. Farach-Colton: The LCA Problem Revisited. LATIN 2000

Solution to MaxPairLCP



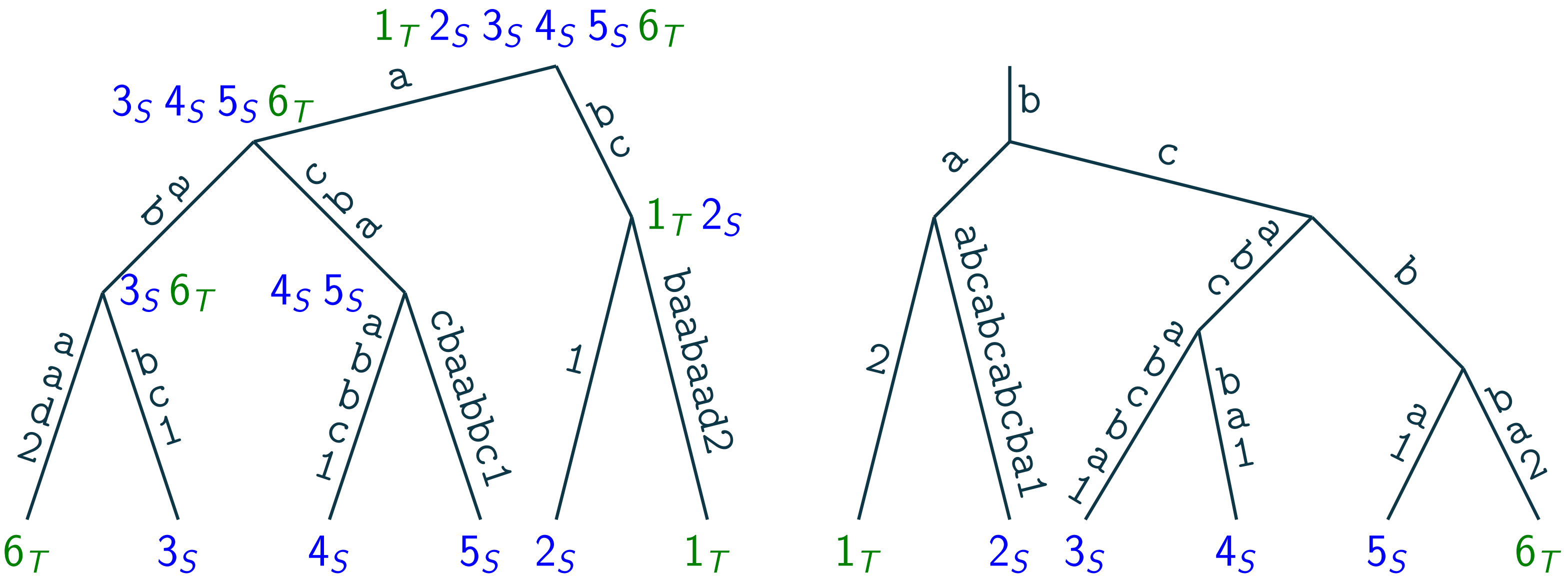
- Sorted lists can be stored using balanced BSTs (e.g. AVLs)
- Merging AVLs takes $O(N \log N)$ total time ($N =$ size of trie) [1]
- LCAs can be computed in $O(1)$ time after $O(N)$-time preprocessing [2]

MaxPairLCP in $O(N \log N + n / \log_\sigma n)$ time

[1] M.R. Brown, R.E. Tarjan: A Fast Merging Algorithm. J. ACM, 1979

[2] M.A. Bender, M. Farach-Colton: The LCA Problem Revisited. LATIN 2000

Solution to MaxPairLCP



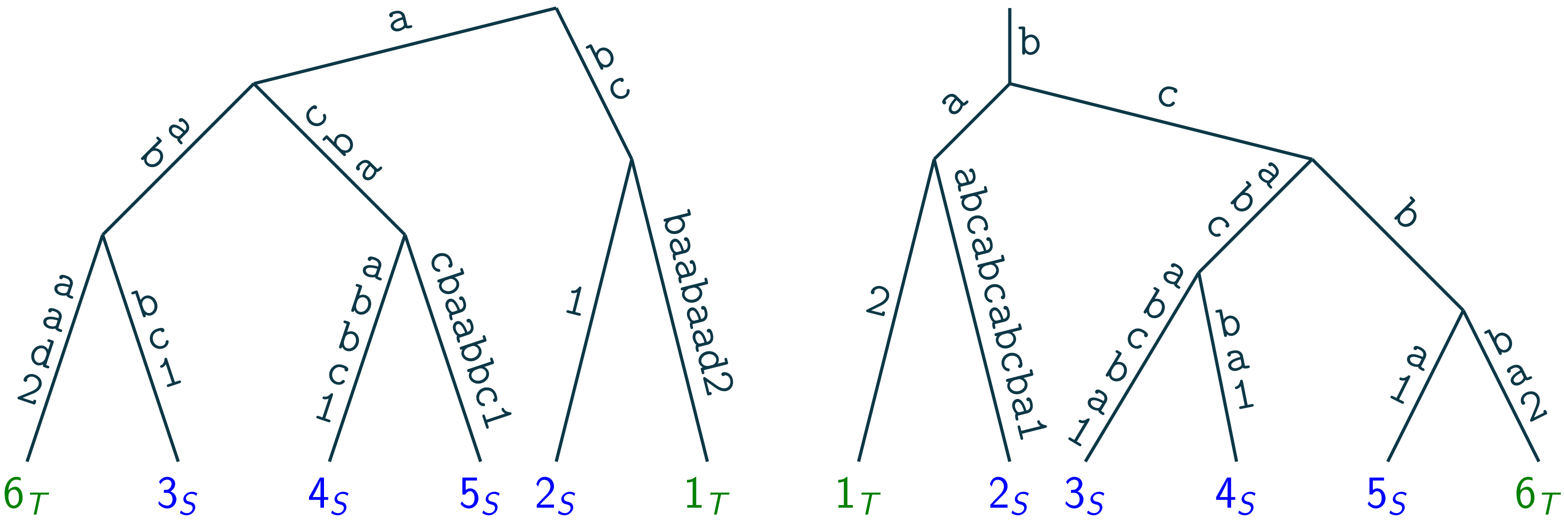
- Sorted lists can be stored using balanced BSTs (e.g. AVLs)
- Merging AVLs takes $O(N \log N)$ total time ($N =$ size of trie) [1]
- LCAs can be computed in $O(1)$ time after $O(N)$-time preprocessing [2]

MaxPairLCP in $O(N \log N + n/\log_\sigma n)$ time $\Rightarrow$ Long LCF in $O(n/\log n)$ time.

[1] M.R. Brown, R.E. Tarjan: A Fast Merging Algorithm. J. ACM, 1979

[2] M.A. Bender, M. Farach-Colton: The LCA Problem Revisited. LATIN 2000

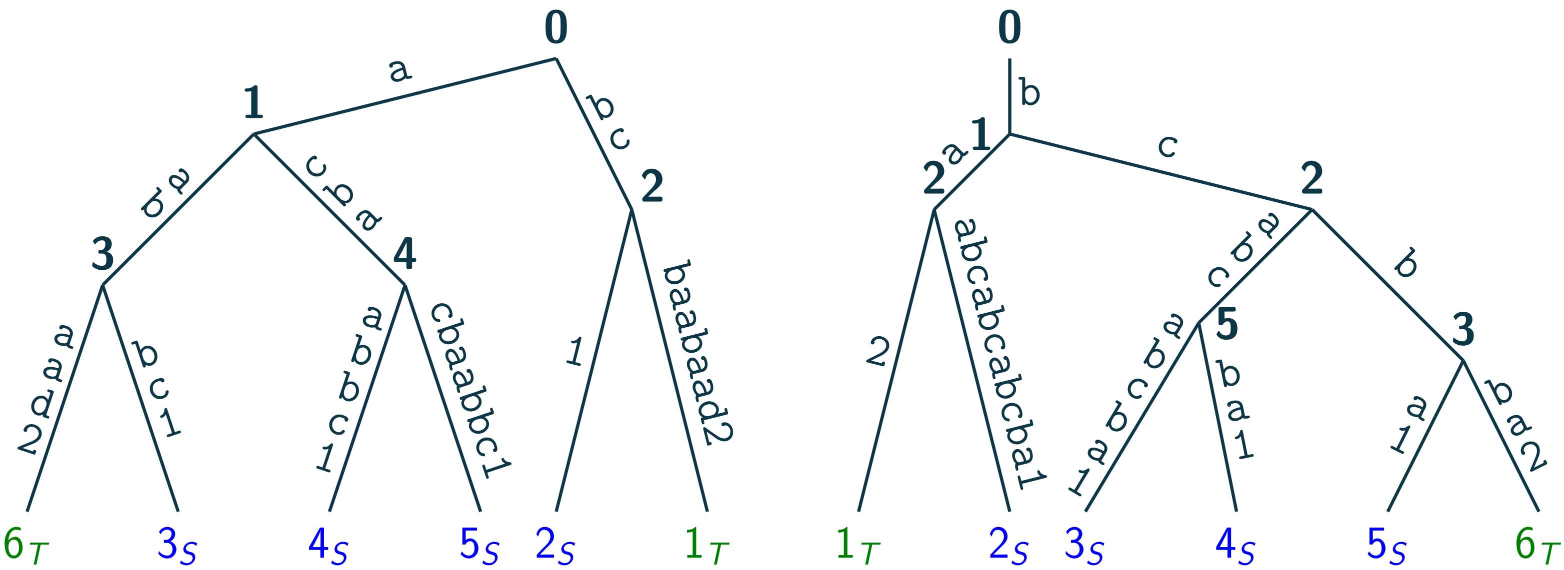
Colored Trees Problem



Goal:

- Nodes u, v from both tries
- Their subtrees have a pair of equal leaves, one from S and one from T
- $depth(u) + depth(v)$ is max

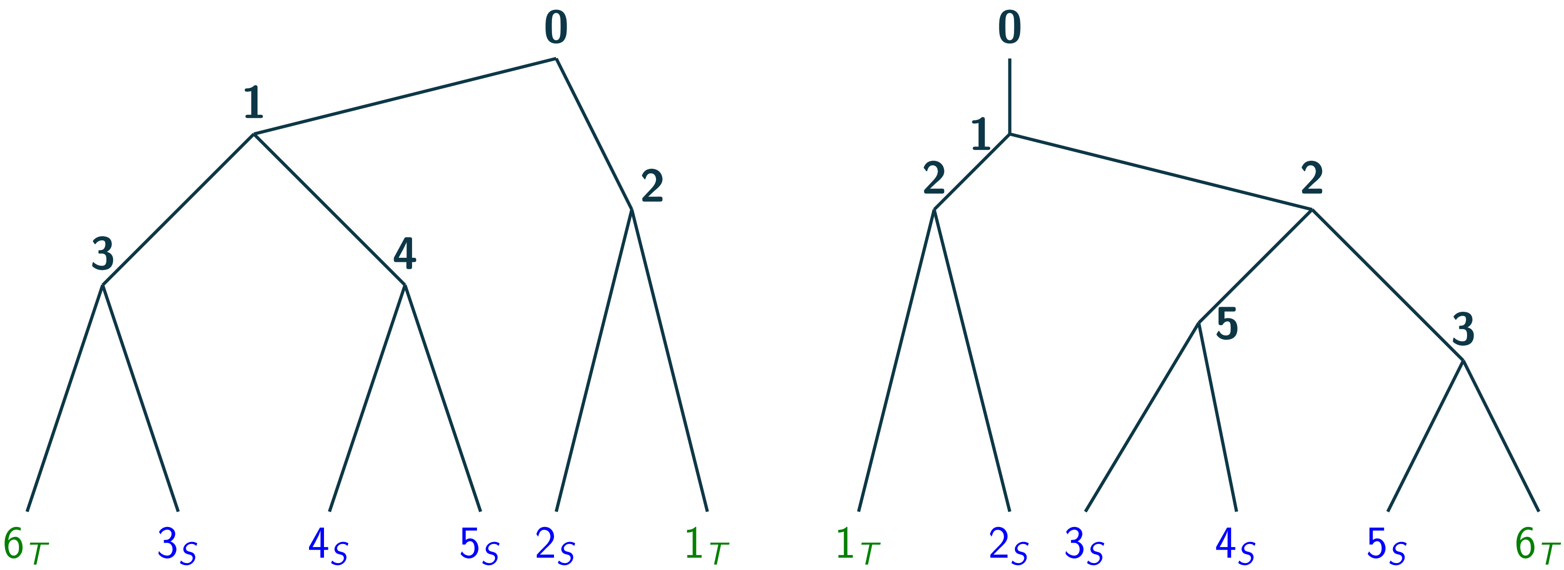
Colored Trees Problem



Goal:

- Nodes u, v from both tries
- Their subtrees have a pair of equal leaves, one from S and one from T
- $depth(u) + depth(v)$ is max

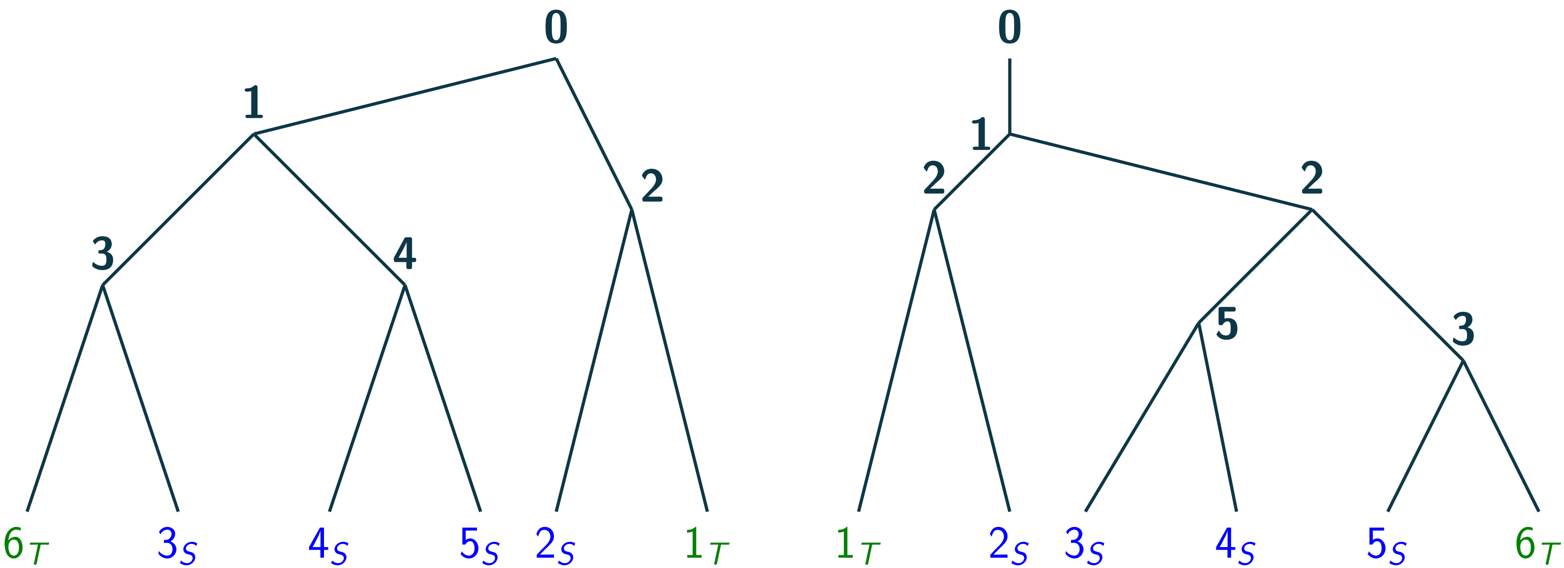
Colored Trees Problem



ColoredTrees problem

- Nodes u, v from both trees
- Their subtrees have a pair of equal leaves of different colors
- $depth(u) + depth(v)$ is max

Colored Trees Problem



COLOREDTREES problem

- Nodes u, v from both trees
- Their subtrees have a pair of equal leaves of different colors
- $depth(u) + depth(v)$ is max

COLOREDTREES problem can be solved in $O(N \log N)$ time (N is the size of trees).

Plan of Presentation

- **Packed LCF:**
    - Short LCF
    - Long LCF
    - **Medium-length LCF**
- Approximate LCF
- Small-space LCF
- Compressed LCF
- Dynamic LCF
- Internal LCF

Medium-Length LCF

Three cases:

- Short LCF: $\leq \frac{1}{3}\log_\sigma n$: $o(n/\log n)$ time
- Medium-length LCF: $O(n/\sqrt{\log n})$ time?
- Long LCF: $\geq \log^4 n$: $O(n/\log n)$ time

Medium-Length LCF

Three cases:

- Short LCF: $\leq \frac{1}{3}\log_\sigma n$: $o(n/\log n)$ time
- Medium-length LCF: $O(n/\sqrt{\log n})$ time?
- Long LCF: $\geq \log^4 n$: $O(n/\log n)$ time

MaxPairLCP in $O(N \log N + n/\log n)$ time

Medium-Length LCF

Three cases:

- Short LCF: $\leq \frac{1}{3}\log_\sigma n$: $o(n/\log n)$ time
- Medium-length LCF: $O(n/\sqrt{\log n})$ time?
- Long LCF: $\geq \log^4 n$: $O(n/\log n)$ time

MaxPairLCP in $O(N \log N + n/\log n)$ time

- We don't know how to beat $O(N \log N)$ in general

Medium-Length LCF

Three cases:

- Short LCF: $\leq \frac{1}{3} \log_\sigma n$: $o(n/\log n)$ time
- Medium-length LCF: $O(n/\sqrt{\log n})$ time?
- Long LCF: $\geq \log^4 n$: $O(n/\log n)$ time

MaxPairLCP in $O(N \log N + n/\log n)$ time

- We don't know how to beat $O(N \log N)$ in general
- We will consider special cases of MaxPairLCP

Medium-Length LCF

Three cases:

- Short LCF: $\leq \frac{1}{3}\log_\sigma n$: $o(n/\log n)$ time
- Medium-length LCF: $O(n/\sqrt{\log n})$ time?
- Long LCF: $\geq \log^4 n$: $O(n/\log n)$ time

MaxPairLCP in $O(N \log N + n/\log n)$ time

- We don't know how to beat $O(N \log N)$ in general
- We will consider special cases of MaxPairLCP
- and replace difference covers with string synchronizing sets [1]

[1] D. Kempa, T. Kociumaka: String Synchronizing Sets: Sublinear-time BWT Construction and Optimal LCE Data Structure. STOC 2019

String Synchronizing Set

A τ-synchronizing set of a string W of length n is a set of positions X that is:

- small: $|X| = O(n/\tau)$
- consistent: whether $i \in X$ depends only on $W[i..i+2\tau)$
- dense: $X \cap [i..i+\tau) = \emptyset$ for $i \in [1..n-3\tau+2]$
  iff $per(T[i..i+3\tau-1]) > \tau/3$
- fast to construct: in $O(n/\tau)$ time if $\tau \le \frac{1}{5}\log_\sigma n$, otherwise $O(n)$ time

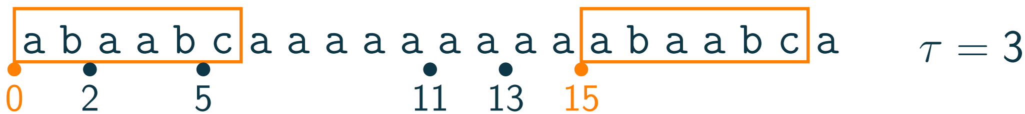
a b a a b c a a a a a a a a a a b a a b c a $\qquad \tau = 3$

0 2 5 11 13 15

String Synchronizing Set

A τ-synchronizing set of a string W of length n is a set of positions X that is:

- small: $|X| = O(n/\tau)$
- consistent: whether $i \in X$ depends only on $W[i..i+2\tau)$
- dense: $X \cap [i..i+\tau) = \emptyset$ for $i \in [1..n-3\tau+2]$
  iff $per(T[i..i+3\tau-1]) > \tau/3$
- fast to construct: in $O(n/\tau)$ time if $\tau \leq \frac{1}{5}\log_\sigma n$, otherwise $O(n)$ time



String Synchronizing Set

A τ-synchronizing set of a string W of length n is a set of positions X that is:

- small: $|X| = O(n/\tau)$
- consistent: whether $i \in X$ depends only on $W[i..i+2\tau)$
- dense: $X \cap [i..i+\tau) = \emptyset$ for $i \in [1..n-3\tau+2]$
  iff $per(T[i..i+3\tau-1]) > \tau/3$
- fast to construct: in $O(n/\tau)$ time if $\tau \leq \frac{1}{5}\log_\sigma n$, otherwise $O(n)$ time

a b a a b c a a a a a a a a a a b a a b c a $\tau = 3$

0 2 5 11 13 15

String Synchronizing Set

A τ-synchronizing set of a string W of length n is a set of positions X that is:

- small: $|X| = O(n/\tau)$
- consistent: whether $i \in X$ depends only on $W[i..i+2\tau)$
- dense: $X \cap [i..i+\tau) = \emptyset$ for $i \in [1..n-3\tau+2]$ iff $per(T[i..i+3\tau-1]) > \tau/3$
- fast to construct: in $O(n/\tau)$ time if $\tau \leq \frac{1}{5}\log_\sigma n$, otherwise $O(n)$ time

a b a a b c a a a a a a a a a a b a a b c a $\quad \tau = 3$

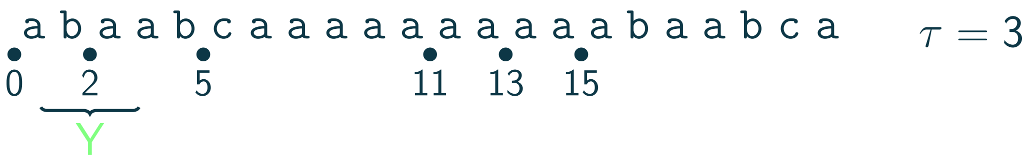
0 2 5 11 13 15

Y

String Synchronizing Set

A τ-synchronizing set of a string W of length n is a set of positions X that is:

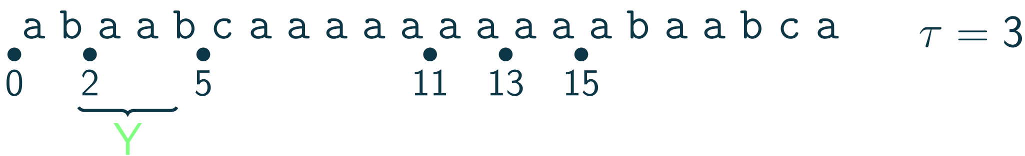
- small: $|X| = O(n/\tau)$
- consistent: whether $i \in X$ depends only on $W[i..i+2\tau)$
- dense: $X \cap [i..i+\tau) = \emptyset$ for $i \in [1..n-3\tau+2]$ iff $per(T[i..i+3\tau-1]) > \tau/3$
- fast to construct: in $O(n/\tau)$ time if $\tau \leq \frac{1}{5}\log_\sigma n$, otherwise $O(n)$ time



String Synchronizing Set

A τ-synchronizing set of a string W of length n is a set of positions X that is:

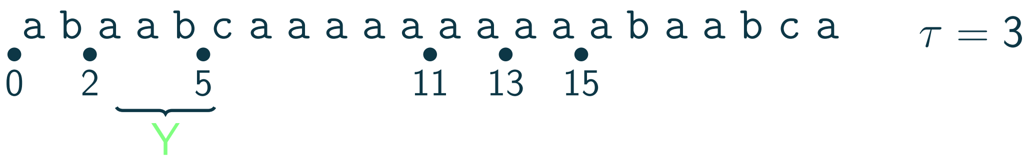
- small: $|X| = O(n/\tau)$
- consistent: whether $i \in X$ depends only on $W[i..i+2\tau)$
- dense: $X \cap [i..i+\tau) = \emptyset$ for $i \in [1..n-3\tau+2]$
  iff $per(T[i..i+3\tau-1]) > \tau/3$
- fast to construct: in $O(n/\tau)$ time if $\tau \le \frac{1}{5}\log_\sigma n$, otherwise $O(n)$ time



String Synchronizing Set

A τ-synchronizing set of a string W of length n is a set of positions X that is:

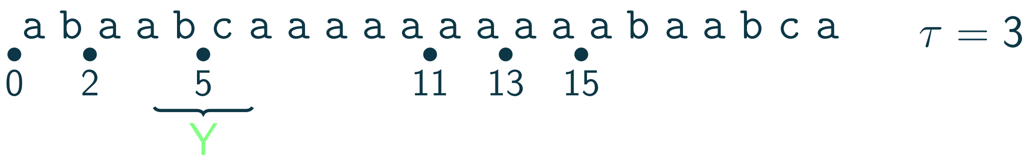
- small: $|X| = O(n/\tau)$
- consistent: whether $i \in X$ depends only on $W[i..i+2\tau)$
- dense: $X \cap [i..i+\tau) = \emptyset$ for $i \in [1..n-3\tau+2]$
  iff $per(T[i..i+3\tau-1]) > \tau/3$
- fast to construct: in $O(n/\tau)$ time if $\tau \leq \frac{1}{5}\log_\sigma n$, otherwise $O(n)$ time



String Synchronizing Set

A τ-synchronizing set of a string W of length n is a set of positions X that is:

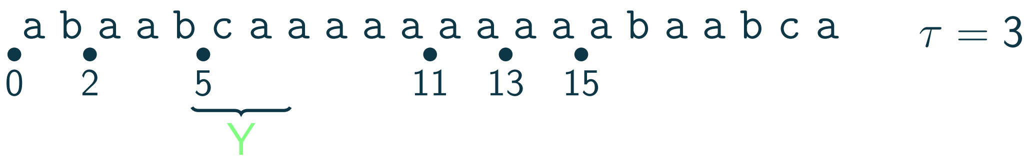
- small: $|X| = O(n/\tau)$
- consistent: whether $i \in X$ depends only on $W[i..i+2\tau)$
- dense: $X \cap [i..i+\tau) = \emptyset$ for $i \in [1..n-3\tau+2]$ iff $per(T[i..i+3\tau-1]) > \tau/3$
- fast to construct: in $O(n/\tau)$ time if $\tau \le \frac{1}{5}\log_\sigma n$, otherwise $O(n)$ time



String Synchronizing Set

A τ-synchronizing set of a string W of length n is a set of positions X that is:

- small: $|X| = O(n/\tau)$
- consistent: whether $i \in X$ depends only on $W[i..i+2\tau)$
- dense: $X \cap [i..i+\tau) = \emptyset$ for $i \in [1..n-3\tau+2]$
  iff $per(T[i..i+3\tau-1]) > \tau/3$
- fast to construct: in $O(n/\tau)$ time if $\tau \leq \frac{1}{5}\log_\sigma n$, otherwise $O(n)$ time



String Synchronizing Set

A τ-synchronizing set of a string W of length n is a set of positions X that is:

- small: $|X| = O(n/\tau)$
- consistent: whether $i \in X$ depends only on $W[i..i+2\tau)$
- dense: $X \cap [i..i+\tau) = \emptyset$ for $i \in [1..n-3\tau+2]$
  iff $per(T[i..i+3\tau-1]) > \tau/3$
- fast to construct: in $O(n/\tau)$ time if $\tau \le \frac{1}{5}\log_\sigma n$, otherwise $O(n)$ time

a b a a b c a a a a a a a a a a b a a b c a $\tau = 3$

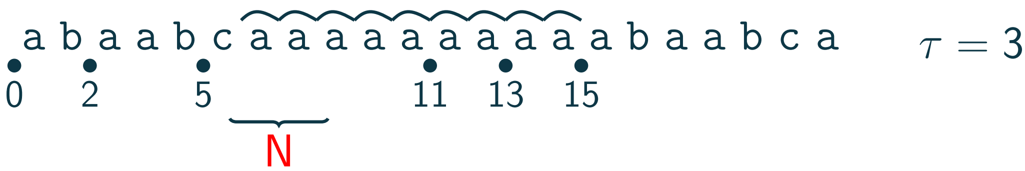
0 2 5 11 13 15

N

String Synchronizing Set

A τ-synchronizing set of a string W of length n is a set of positions X that is:

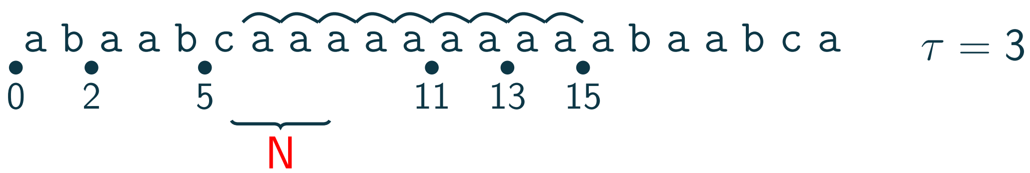
- small: $|X| = O(n/\tau)$
- consistent: whether $i \in X$ depends only on $W[i..i+2\tau)$
- dense: $X \cap [i..i+\tau) = \emptyset$ for $i \in [1..n-3\tau+2]$
  iff $per(T[i..i+3\tau-1]) > \tau/3$
- fast to construct: in $O(n/\tau)$ time if $\tau \le \frac{1}{5}\log_\sigma n$, otherwise $O(n)$ time



String Synchronizing Set

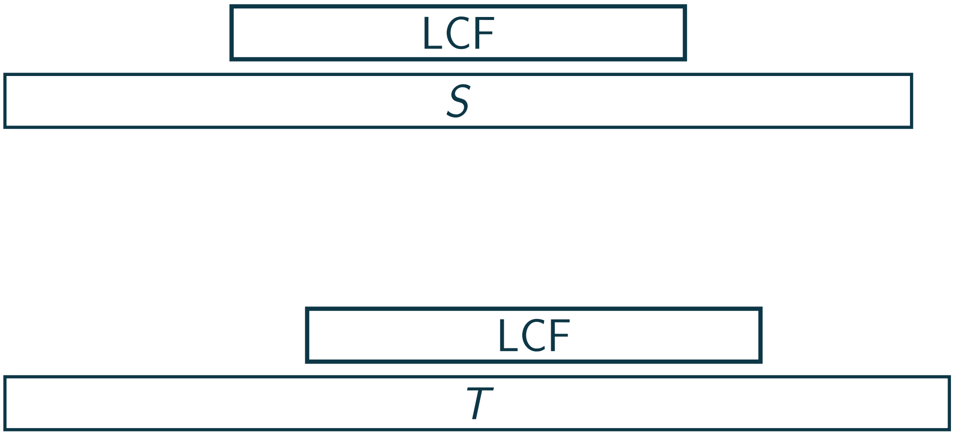
A τ-synchronizing set of a string W of length n is a set of positions X that is:

- small: $|X| = O(n/\tau)$
- consistent: whether $i \in X$ depends only on $W[i..i+2\tau)$
- dense: $X \cap [i..i+\tau) = \emptyset$ for $i \in [1..n-3\tau+2]$
  iff $per(T[i..i+3\tau-1]) > \tau/3$
- fast to construct: in $O(n/\tau)$ time if $\tau \le \frac{1}{5}\log_\sigma n$, otherwise $O(n)$ time
- fast to construct: in $O(n/\log n)$ time if $\tau = \lfloor \frac{1}{5}\log_\sigma n \rfloor$



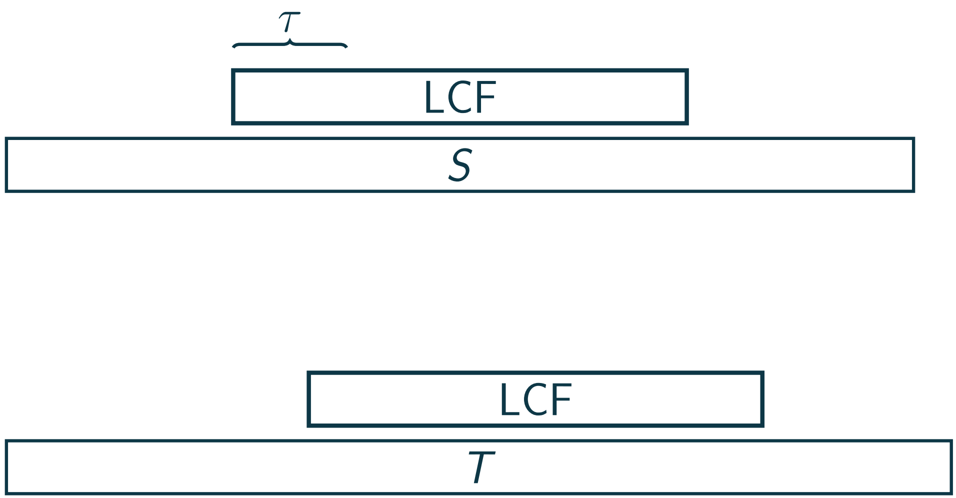
Computing Anchors

Assume the length of LCF is in $[\tau, \ell]$.



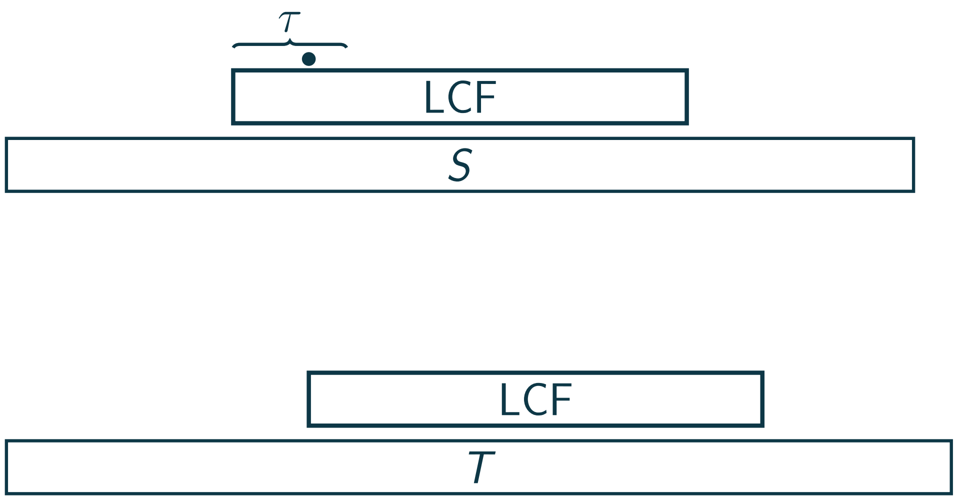
Computing Anchors

Assume the length of LCF is in $[\tau, \ell]$.



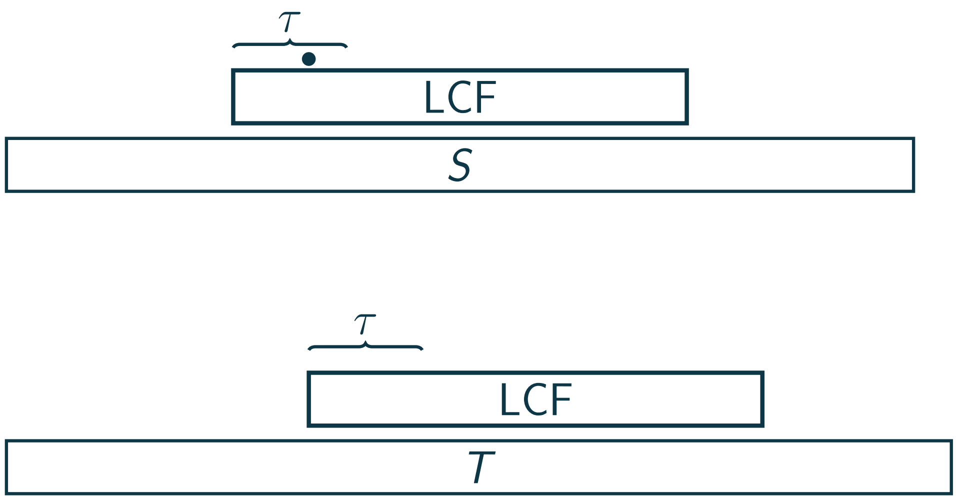
Computing Anchors

Assume the length of LCF is in $[\tau, \ell]$.



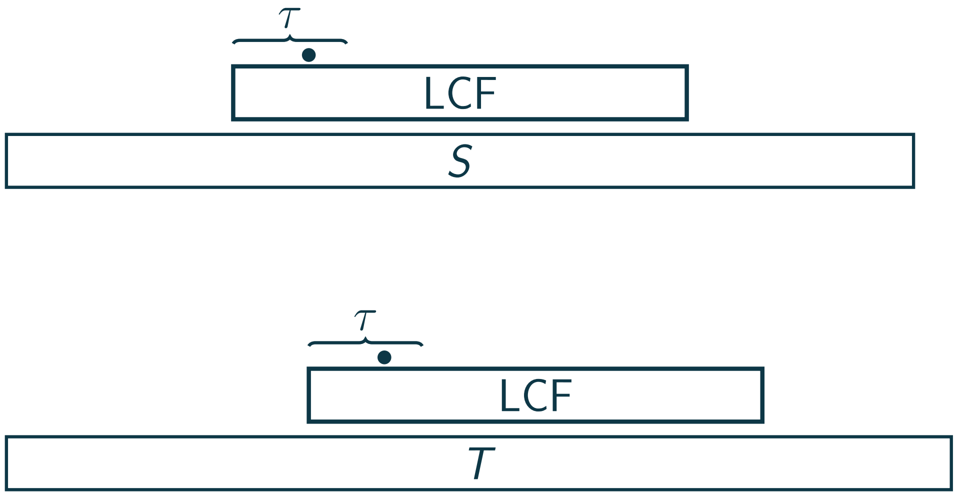
Computing Anchors

Assume the length of LCF is in $[\tau, \ell]$.



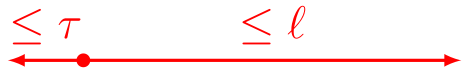
Computing Anchors

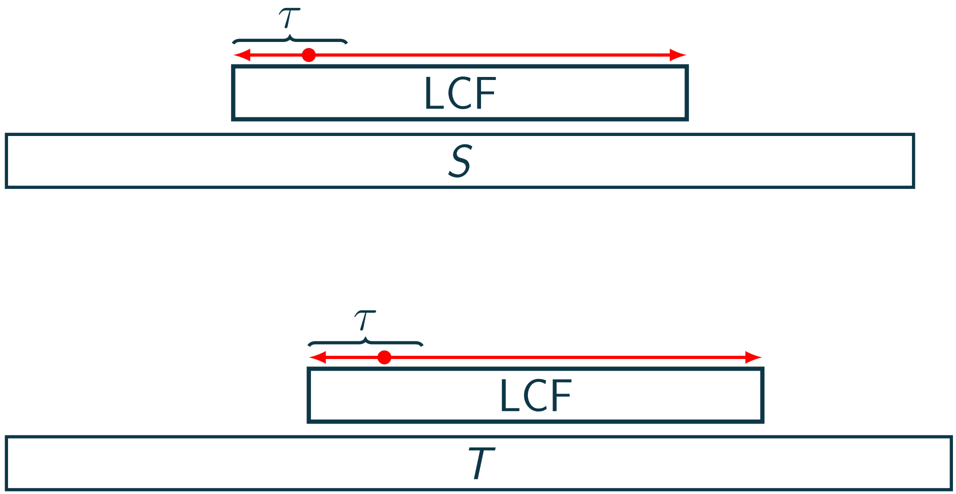
Assume the length of LCF is in $[\tau, \ell]$.



Computing Anchors

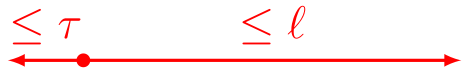
Assume the length of LCF is in $[\tau, \ell]$.

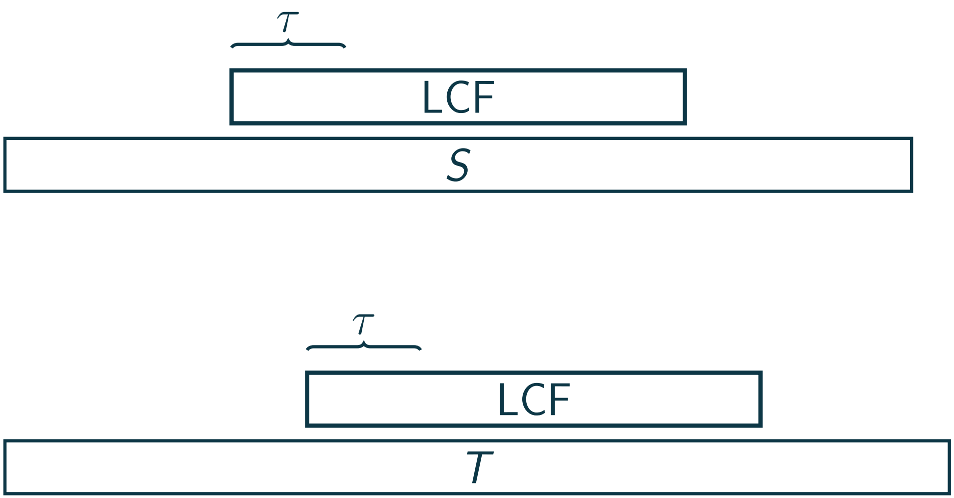




Computing Anchors

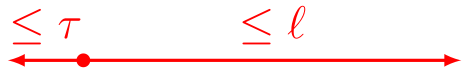
Assume the length of LCF is in $[\tau, \ell]$.

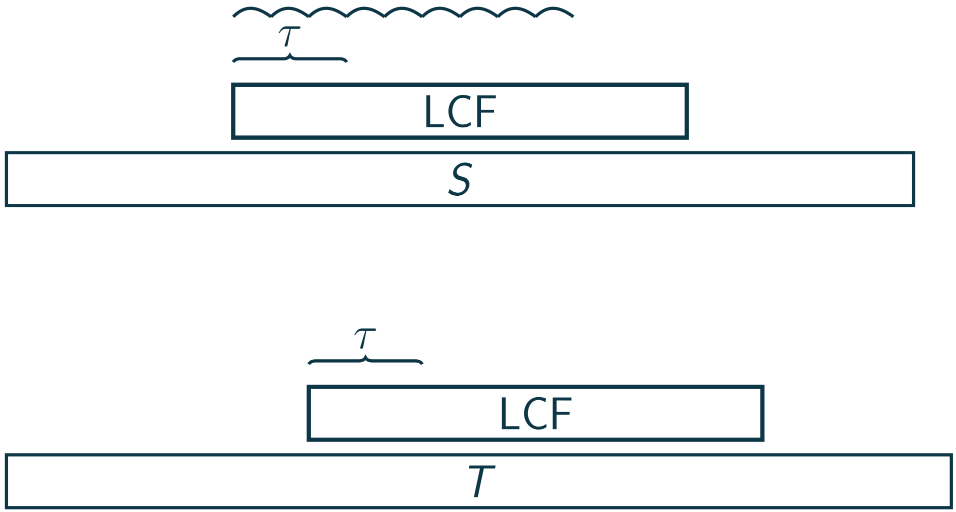




Computing Anchors

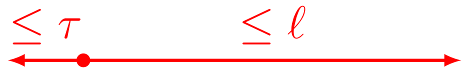
Assume the length of LCF is in $[\tau, \ell]$.

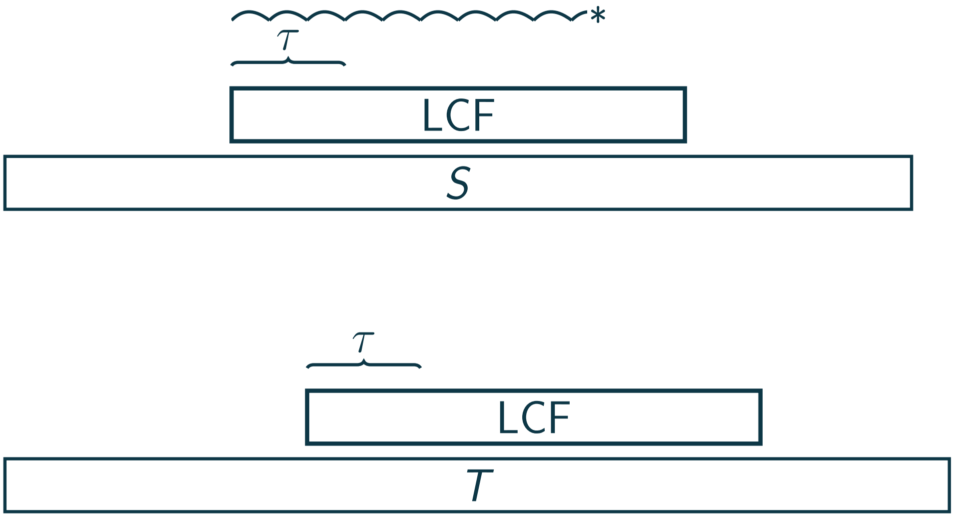




Computing Anchors

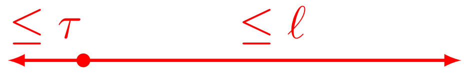
Assume the length of LCF is in $[\tau, \ell]$.

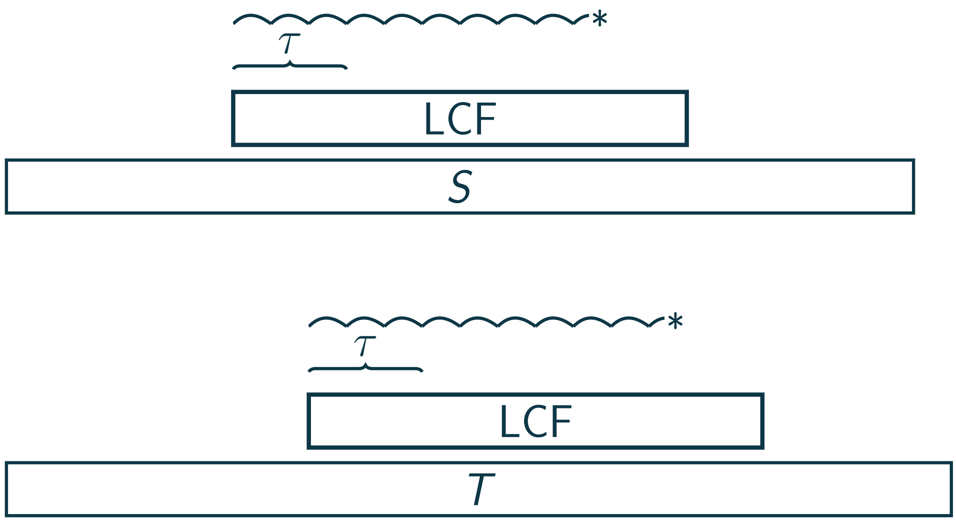




Computing Anchors

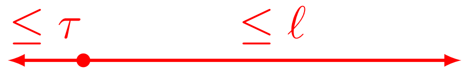
Assume the length of LCF is in $[\tau, \ell]$.

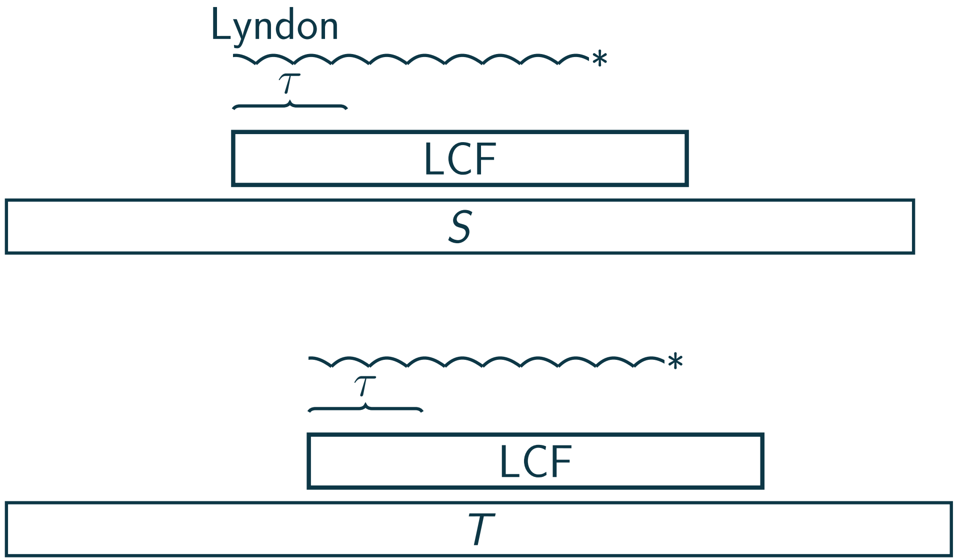




Computing Anchors

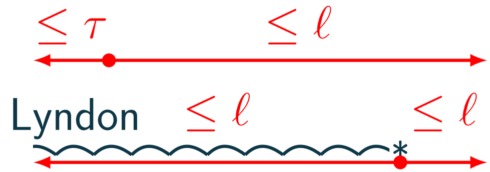
Assume the length of LCF is in $[\tau, \ell]$.

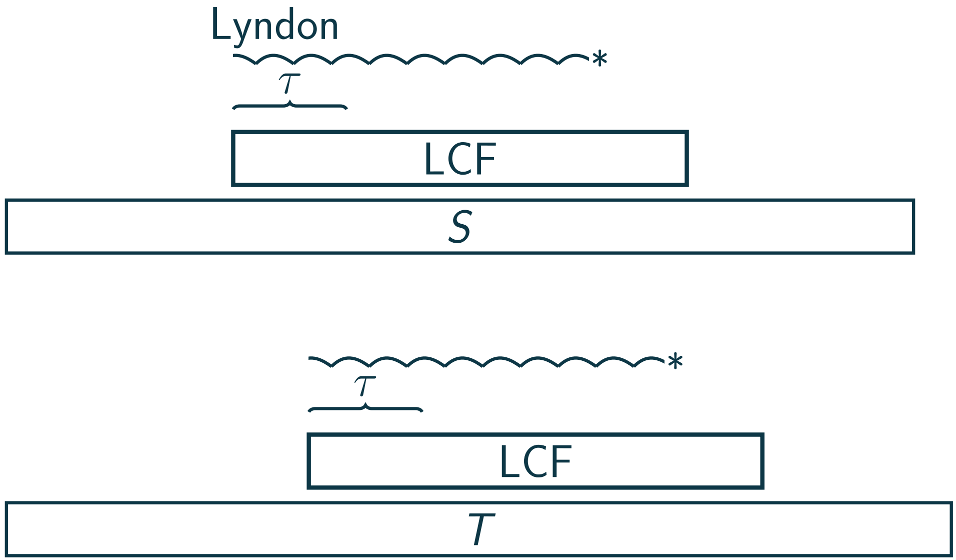




Computing Anchors

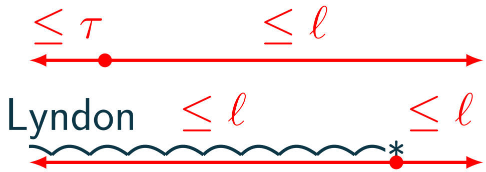
Assume the length of LCF is in $[\tau, \ell]$.

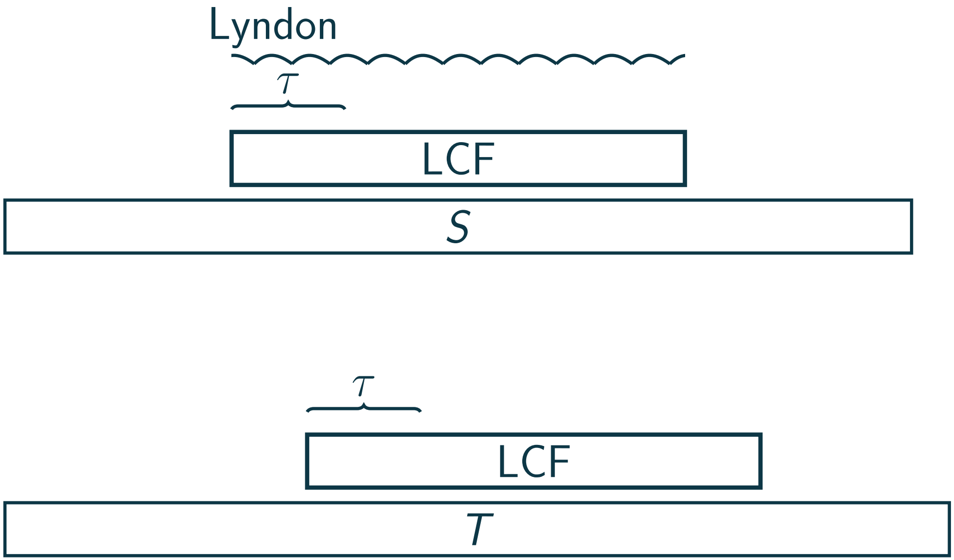




Computing Anchors

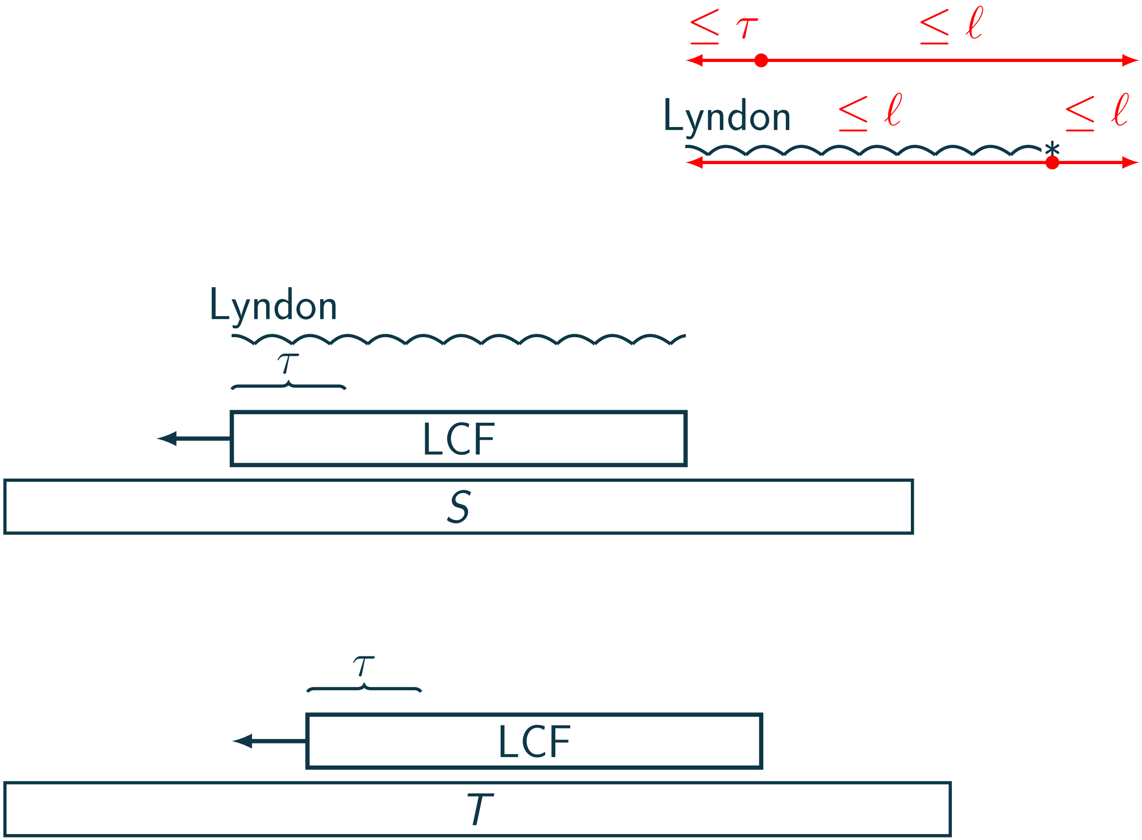
Assume the length of LCF is in $[\tau, \ell]$.





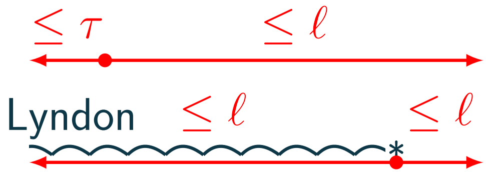
Computing Anchors

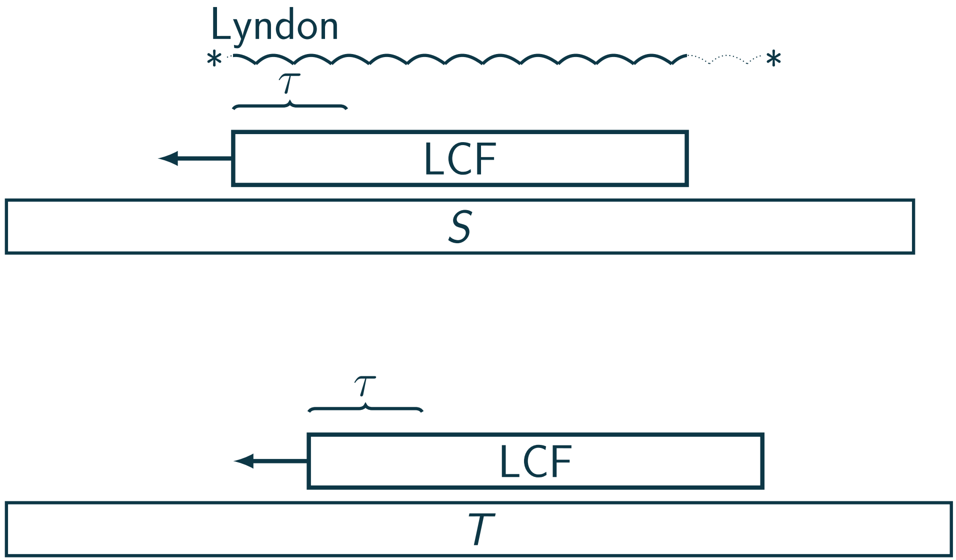
Assume the length of LCF is in $[\tau, \ell]$.



Computing Anchors

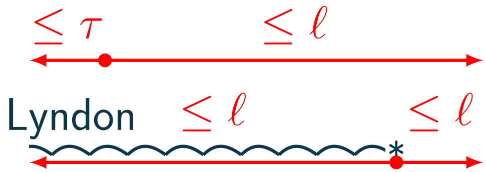
Assume the length of LCF is in $[\tau, \ell]$.

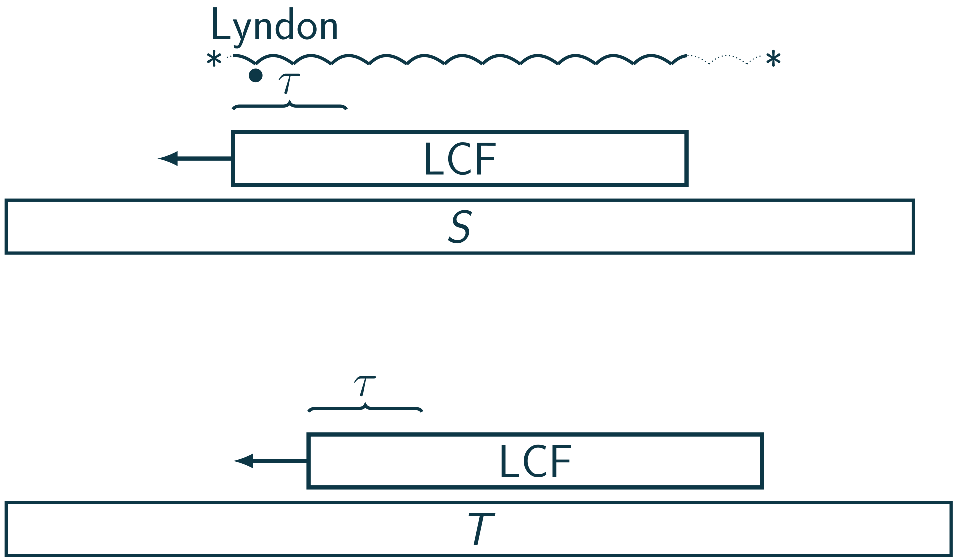




Computing Anchors

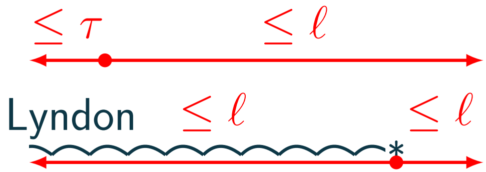
Assume the length of LCF is in $[\tau, \ell]$.

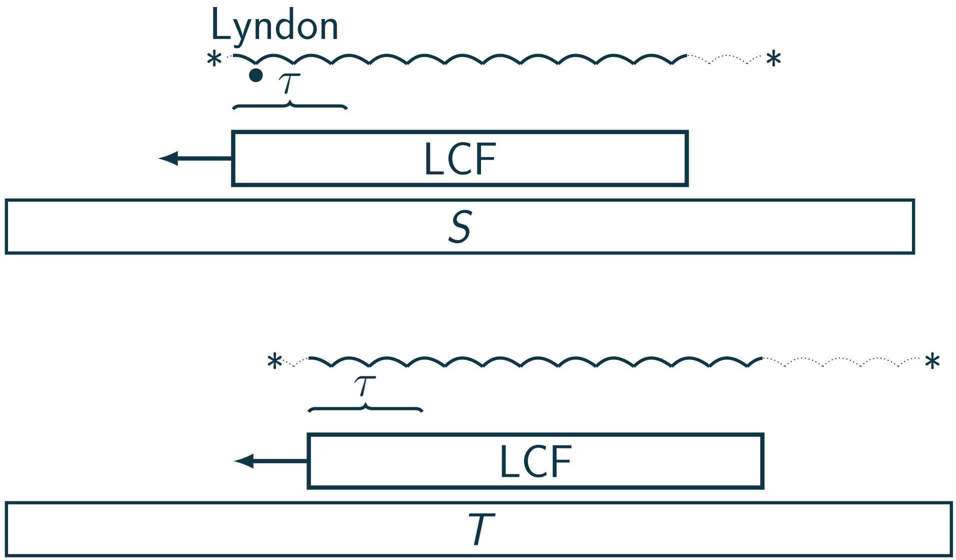




Computing Anchors

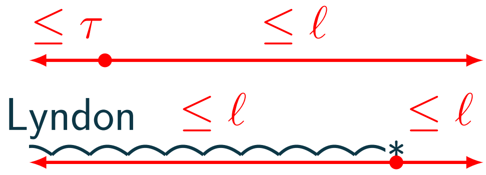
Assume the length of LCF is in $[\tau, \ell]$.

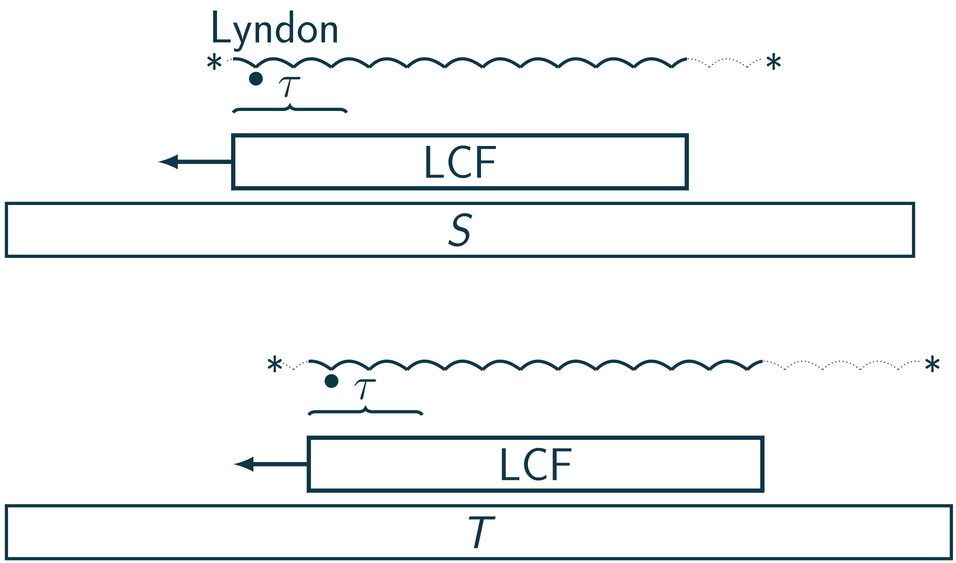




Computing Anchors

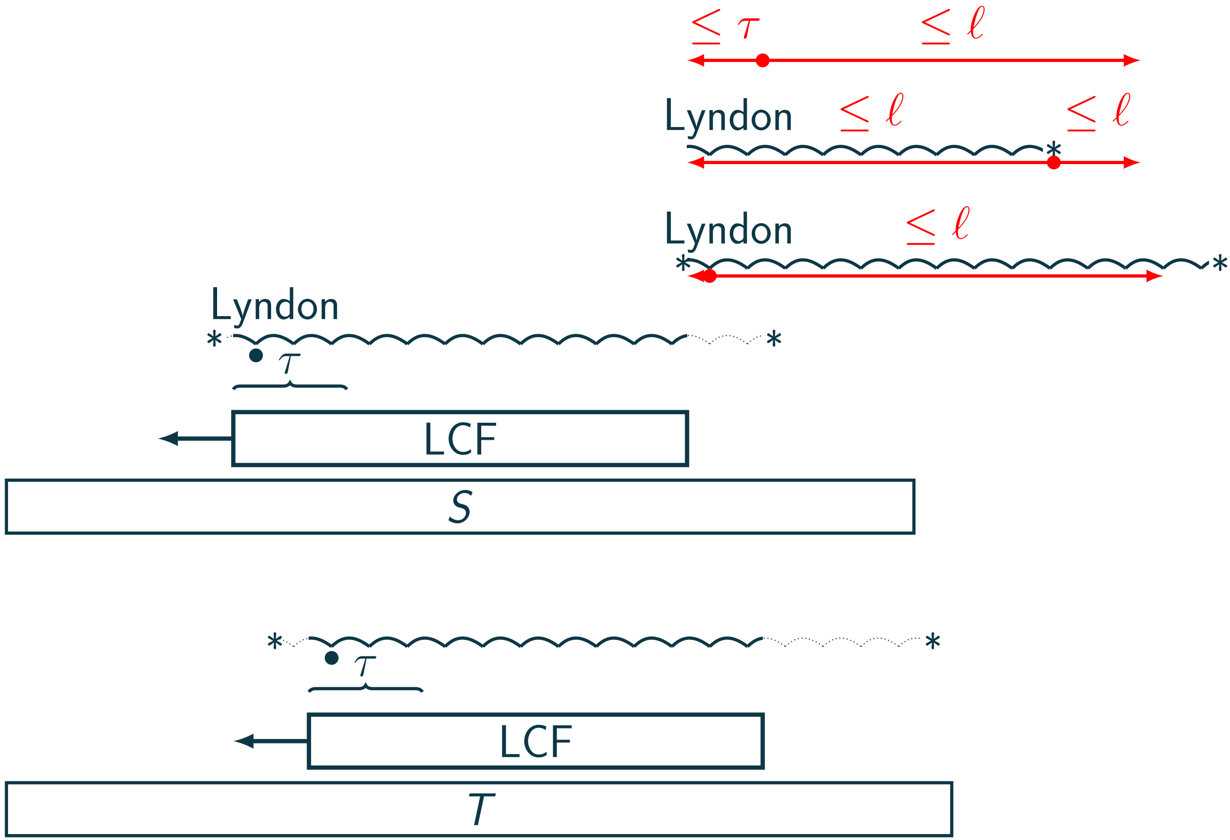
Assume the length of LCF is in $[\tau, \ell]$.





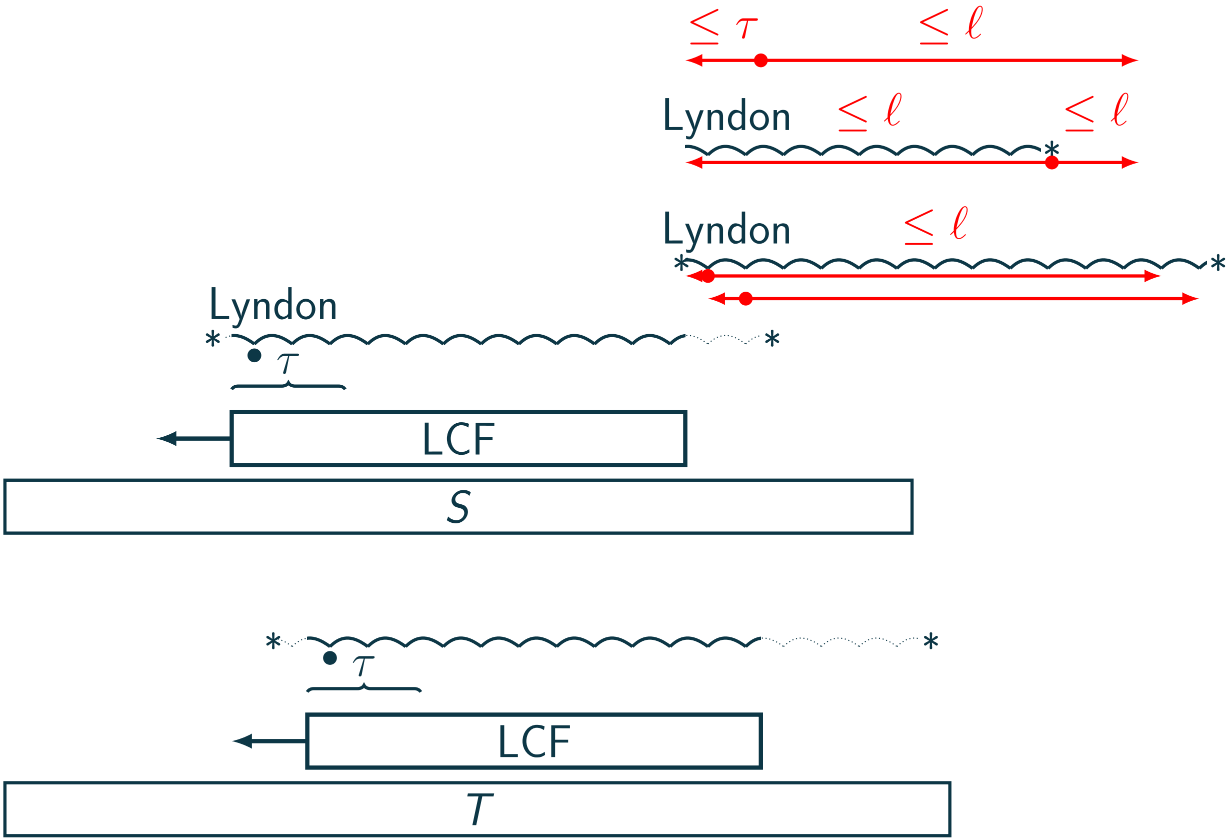
Computing Anchors

Assume the length of LCF is in $[\tau, \ell]$.



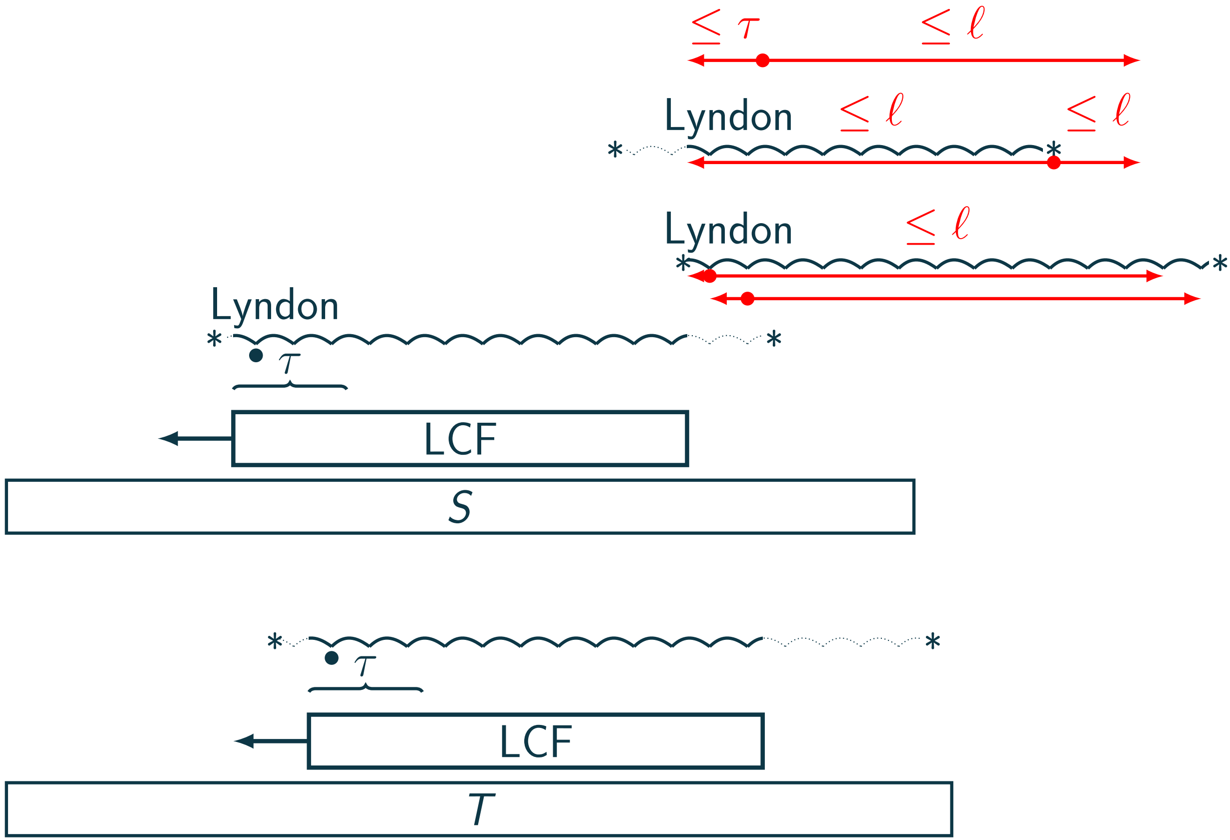
Computing Anchors

Assume the length of LCF is in $[\tau, \ell]$.



Computing Anchors

Assume the length of LCF is in $[\tau, \ell]$.



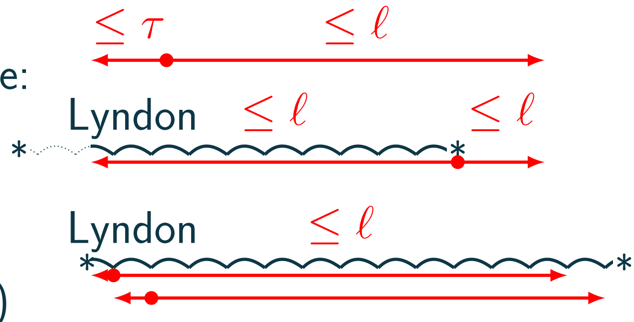
Computing Anchors

Assume the length of LCF is in $[\tau, \ell]$.

$O(n/\log n)$ anchors computed in $O(n/\log n)$ time:

- from synchronizing set
- 3 anchors from a τ-run

($\leq n/\tau$ such runs, computed in $O(n/\log n)$ time)



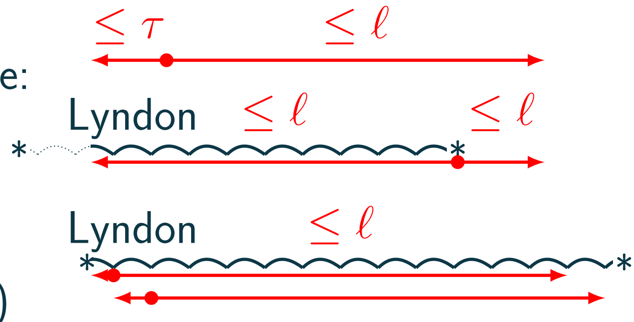
Computing Anchors

Assume the length of LCF is in $[\tau, \ell]$.

$O(n/\log n)$ anchors computed in $O(n/\log n)$ time:

- from synchronizing set
- 3 anchors from a τ-run

($\leq n/\tau$ such runs, computed in $O(n/\log n)$ time)



Two special types of instances of MaxPairLCP:

1. (τ, ℓ)-MaxPairLCP

2. MaxPairLCP with first components being prefixes of the same string

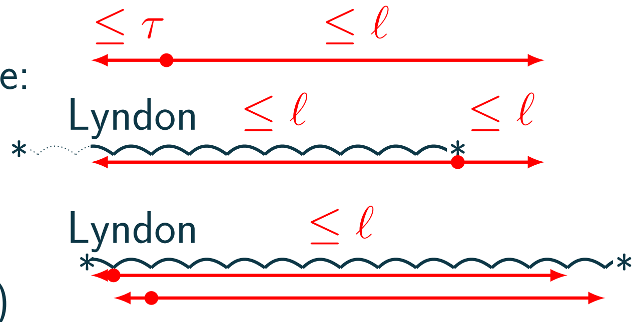
Computing Anchors

Assume the length of LCF is in $[\tau, \ell]$.

$O(n/\log n)$ anchors computed in $O(n/\log n)$ time:

- from synchronizing set
- 3 anchors from a τ-run

($\leq n/\tau$ such runs, computed in $O(n/\log n)$ time)



Two special types of instances of MaxPairLCP:

1. (τ, ℓ)-MaxPairLCP
2. MaxPairLCP with first components being prefixes of the same string

Type 2 can be solved in linear time (one tree in ColoredTrees is a path),

$O(n/\log n)$ time in total.

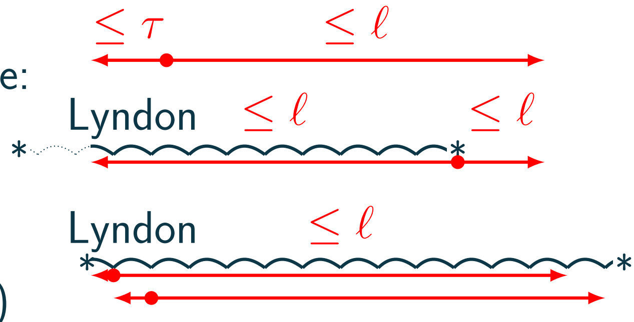
Computing Anchors

Assume the length of LCF is in $[\tau, \ell]$.

$O(n/\log n)$ anchors computed in $O(n/\log n)$ time:

- from synchronizing set
- 3 anchors from a τ-run

($\leq n/\tau$ such runs, computed in $O(n/\log n)$ time)



Two special types of instances of MaxPairLCP:

1. (τ, ℓ)-MaxPairLCP
2. MaxPairLCP with first components being prefixes of the same string

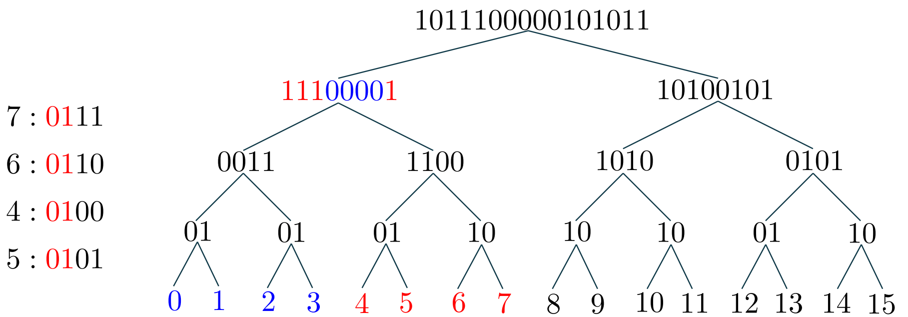
Type 2 can be solved in linear time (one tree in ColoredTrees is a path), $O(n/\log n)$ time in total.

We solve type 1 using a wavelet tree.

Application of Wavelet Trees (Sketch)

Standard wavelet tree [1]:

T 12 7 11 15 9 6 4 0 1 2 10 3 13 5 8 14

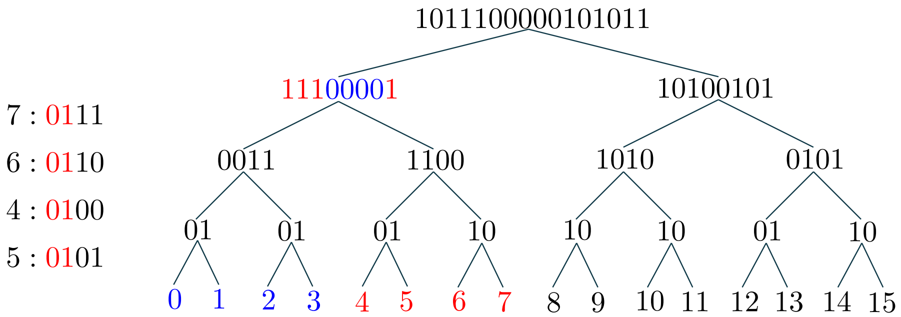


[1] R. Grossi, A. Gupta, J.S. Vitter: High-order entropy-compressed text indexes. SODA 2003

Application of Wavelet Trees (Sketch)

Standard wavelet tree [1]:

T 12 7 11 15 9 6 4 0 1 2 10 3 13 5 8 14



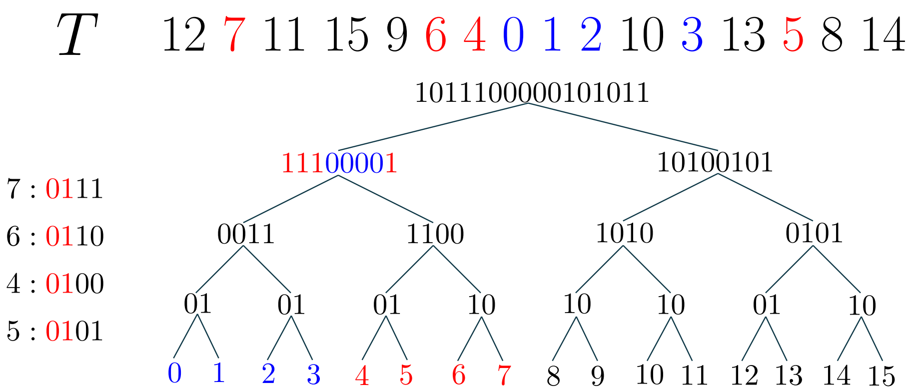
Arbitrarily shaped wavelet tree:

- Shape given by a binary trie, represents an alphabet of strings

[1] R. Grossi, A. Gupta, J.S. Vitter: High-order entropy-compressed text indexes. SODA 2003

Application of Wavelet Trees (Sketch)

Standard wavelet tree [1]:



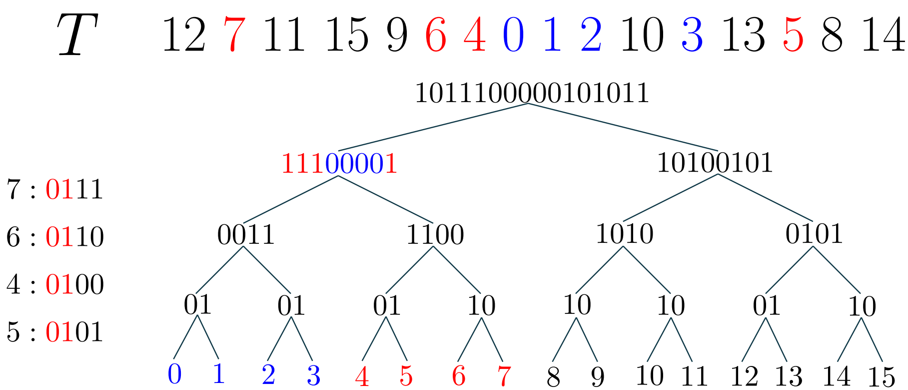
Arbitrarily shaped wavelet tree:

- Shape given by a binary trie, represents an alphabet of strings
- We use: left trie in MaxPairLCP of height $h \leq \tau$ after binarisation

[1] R. Grossi, A. Gupta, J.S. Vitter: High-order entropy-compressed text indexes. SODA 2003

Application of Wavelet Trees (Sketch)

Standard wavelet tree [1]:



Arbitrarily shaped wavelet tree:

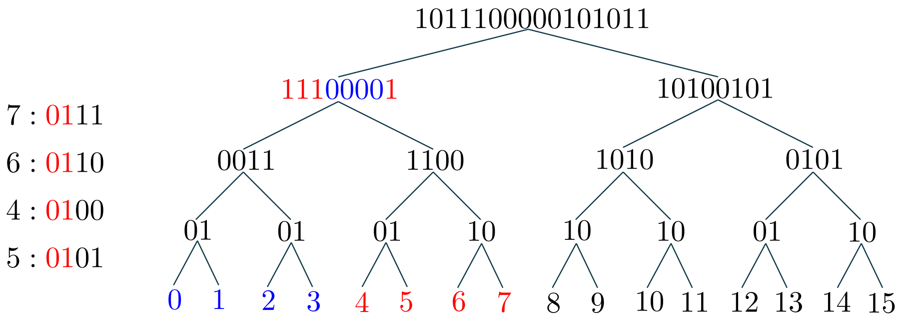
- Shape given by a binary trie, represents an alphabet of strings
- We use: left trie in MaxPairLCP of height $h \le \tau$ after binarisation
- Can be constructed in $O(Nh/\sqrt{\log n}) = O(n/\sqrt{\log n})$ time if $\sigma = 2$

[1] R. Grossi, A. Gupta, J.S. Vitter: High-order entropy-compressed text indexes. SODA 2003

Application of Wavelet Trees (Sketch)

Standard wavelet tree [1]:

T 12 7 11 15 9 6 4 0 1 2 10 3 13 5 8 14



Arbitrarily shaped wavelet tree:

- Shape given by a binary trie, represents an alphabet of strings
- We use: left trie in $\textsc{MaxPairLCP}$ of height $h \le \tau$ after binarisation
- Can be constructed in $O(Nh/\sqrt{\log n}) = O(n/\sqrt{\log n})$ time if $\sigma = 2$
- $\textsc{MaxPairLCP}$ can be solved in $O(n/\sqrt{\log n})$ time if $\ell \le 2^{\sqrt{\log n}}$

[1] R. Grossi, A. Gupta, J.S. Vitter: High-order entropy-compressed text indexes. SODA 2003

Summary of Packed LCF

Three cases:

- Short LCF: $\ell \le \frac{1}{3}\log_\sigma n$: $o(n/\log n)$ time
- Medium-length LCF $\frac{1}{5}\log_\sigma \le \ell \le 2^{\sqrt{\log n}}$: $O(n/\sqrt{\log n})$ time
- Long LCF: $\ell \ge \log^4 n$: $O(n/\log n)$ time

Summary of Packed LCF

Three cases:

- Short LCF: $\ell \leq \frac{1}{3}\log_\sigma n$: $o(n/\log n)$ time
- Medium-length LCF $\frac{1}{5}\log_\sigma \leq \ell \leq 2^{\sqrt{\log n}}$: $O(n/\sqrt{\log n})$ time
- Long LCF: $\ell \geq \log^4 n$: $O(n/\log n)$ time

OP: An $o(n/\sqrt{\log n})$-time algorithm via a completely different approach? [1]

T.M. Chan, M. Patrascu: Counting Inversions, Offline Orthogonal Range Counting, and Related Problems. SODA 2010

Plan of Presentation

- Packed LCF:
  - Short LCF
  - Long LCF
  - Medium-length LCF
- **Approximate LCF**
- Small-space LCF
- Compressed LCF
- Dynamic LCF
- Internal LCF

Approximate LCF, Small k

$\leq k$ mismatches allowed in the pair of substrings

`c b b a a b c a b c a b c b a`

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

Approximate LCF, Small k

$\leq k$ mismatches allowed in the pair of substrings

c b b a a b c a b c a b c b a

d a a b a a b c b b a

Approximate LCF, Small k

$\leq k$ mismatches allowed in the pair of substrings

```
c b b a a b c a b c a b c b a

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

- 1-mismatch LCF in $O(n \log n)$ time [1]
- k-mismatch LCF for $k = O(1)$ in $O(n \log^k n)$ time [2]
- k-mismatch LCF for $k = O(1)$ in $O(n \log^{k-0.5} n)$ time [3]

[1] T. Flouri, E. Giaquinta, K. Kobert, E. Ukkonen:
LCFs with k Mismatches. Inf. Process. Lett., 2015

[2] S.V. Thankachan, A. Apostolico, S. Aluru: A Provably Efficient Algorithm for the k-Mismatch Average Common Substring Problem. J. Comput. Biol., 2016

[3] P. Charalampopoulos, T. Kociumaka, S.P. Pissis, **R**:
Faster Algorithms for LCF. ESA 2021

Approximate LCF, Small k

$\leq k$ mismatches allowed in the pair of substrings

c b b a a b c a b c a b c b a

d a a b a a b c b b a

- 1-mismatch LCF in $O(n \log n)$ time [1] $\leftarrow$ MaxPairLCP problem
- k-mismatch LCF for $k = O(1)$ in $O(n \log^k n)$ time [2]
- k-mismatch LCF for $k = O(1)$ in $O(n \log^{k-0.5} n)$ time [3]

[1] T. Flouri, E. Giaquinta, K. Kobert, E. Ukkonen:
LCFs with k Mismatches. Inf. Process. Lett., 2015

[2] S.V. Thankachan, A. Apostolico, S. Aluru: A Provably Efficient Algorithm
for the k-Mismatch Average Common Substring Problem. J. Comput. Biol., 2016

[3] P. Charalampopoulos, T. Kociumaka, S.P. Pissis, **R**:
Faster Algorithms for LCF. ESA 2021

Approximate LCF, Small k

$\leq k$ mismatches allowed in the pair of substrings

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

- 1-mismatch LCF in $O(n \log n)$ time [1] $\leftarrow$ MaxPairLCP problem
- k-mismatch LCF for $k = O(1)$ in $O(n \log^k n)$ time [2] $\leftarrow$ k-errata trees [4]
- k-mismatch LCF for $k = O(1)$ in $O(n \log^{k-0.5} n)$ time [3]

[1] T. Flouri, E. Giaquinta, K. Kobert, E. Ukkonen:
LCFs with k Mismatches. Inf. Process. Lett., 2015

[2] S.V. Thankachan, A. Apostolico, S. Aluru: A Provably Efficient Algorithm for the k-Mismatch Average Common Substring Problem. J. Comput. Biol., 2016

[3] P. Charalampopoulos, T. Kociumaka, S.P. Pissis, **R**:
Faster Algorithms for LCF. ESA 2021

[4] R. Cole, L.-A. Gottlieb, M. Lewenstein:
Dictionary Matching and Indexing with Errors and Don't Cares. STOC 2004

Approximate LCF, Small k

$\leq k$ mismatches allowed in the pair of substrings

c b b a a b c a b c a b c b a

d a a b a a b c b b a

- 1-mismatch LCF in $O(n \log n)$ time [1] $\leftarrow$ MaxPairLCP problem
- k-mismatch LCF for $k = O(1)$ in $O(n \log^k n)$ time [2] $\leftarrow$ k-errata trees [4]
- k-mismatch LCF for $k = O(1)$ in $O(n \log^{k-0.5} n)$ time [3]

  $\leftarrow$ special case of MaxPairLCP and string synchronizing sets

[1] T. Flouri, E. Giaquinta, K. Kobert, E. Ukkonen:
LCFs with k Mismatches. Inf. Process. Lett., 2015

[2] S.V. Thankachan, A. Apostolico, S. Aluru: A Provably Efficient Algorithm
for the k-Mismatch Average Common Substring Problem. J. Comput. Biol., 2016

[3] P. Charalampopoulos, T. Kociumaka, S.P. Pissis, **R**:
Faster Algorithms for LCF. ESA 2021

[4] R. Cole, L.-A. Gottlieb, M. Lewenstein:
Dictionary Matching and Indexing with Errors and Don't Cares. STOC 2004

Approximate LCF, Small k

$\leq k$ mismatches allowed in the pair of substrings

c b b a a b c a b c a b c b a

d a a b a a b c b b a

OP: Faster algorithm?

- 1-mismatch LCF in $O(n \log n)$ time [1] $\leftarrow$ $\textsc{MaxPairLCP}$ problem
- k-mismatch LCF for $k = O(1)$ in $O(n \log^k n)$ time [2] $\leftarrow$ k-errata trees [4]
- k-mismatch LCF for $k = O(1)$ in $O(n \log^{k-0.5} n)$ time [3]

$\leftarrow$ special cases of $\textsc{MaxPairLCP}$ and string synchronizing sets

[1] T. Flouri, E. Giaquinta, K. Kobert, E. Ukkonen:
LCFs with k Mismatches. Inf. Process. Lett., 2015

[2] S.V. Thankachan, A. Apostolico, S. Aluru: A Provably Efficient Algorithm for the k-Mismatch Average Common Substring Problem. J. Comput. Biol., 2016

[3] P. Charalampopoulos, T. Kociumaka, S.P. Pissis, **R**:
Faster Algorithms for LCF. ESA 2021

[4] R. Cole, L.-A. Gottlieb, M. Lewenstein:
Dictionary Matching and Indexing with Errors and Don't Cares. STOC 2004

Approximate LCF, Large k

- $O(n^2)$-time algorithm for k-mismatch LCF [1]
- No $O(n^{2-\epsilon})$-time algorithm for k-mismatch LCF for $k = \Omega(\log n)$ [2]

[1] T. Flouri, E. Giaquinta, K. Kobert, E. Ukkonen:
LCF with *k* Mismatches. Inf. Process. Lett., 2015

[2] T. Kociumaka, **R**, T. Starikovskaya:
LCF with Approximately *k* Mismatches. Algorithmica 2019

Approximate LCF, Large k

- $O(n^2)$-time algorithm for k-mismatch LCF [1]
- No $O(n^{2-\epsilon})$-time algorithm for k-mismatch LCF for $k = \Omega(\log n)$ [2]
- 2-approximation of k-mismatch LCF in $O(n^{4/3+o(1)})$ time [3]
- No $(2-\epsilon)$-approximation of k-mismatch LCF in $O(n^{2-\delta})$ time [2]

[1] T. Flouri, E. Giaquinta, K. Kobert, E. Ukkonen:
LCF with k Mismatches. Inf. Process. Lett., 2015

[2] T. Kociumaka, **R**, T. Starikovskaya:
LCF with Approximately k Mismatches. Algorithmica 2019

[3] G. Gourdel, T. Kociumaka, **R**, T. Starikovskaya:
Approximating LCF with k mismatches: Theory and Practice. CPM 2020

Approximate LCF, Large k

- $O(n^2)$-time algorithm for k-mismatch LCF [1] $\leftarrow$ DP
- No $O(n^{2-\epsilon})$-time algorithm for k-mismatch LCF for $k = \Omega(\log n)$ [2]
- 2-approximation of k-mismatch LCF in $O(n^{4/3+o(1)})$ time [3]
- No $(2-\epsilon)$-approximation of k-mismatch LCF in $O(n^{2-\delta})$ time [2]

[1] T. Flouri, E. Giaquinta, K. Kobert, E. Ukkonen:
LCF with k Mismatches. Inf. Process. Lett., 2015

[2] T. Kociumaka, **R**, T. Starikovskaya:
LCF with Approximately k Mismatches. Algorithmica 2019

[3] G. Gourdel, T. Kociumaka, **R**, T. Starikovskaya:
Approximating LCF with k mismatches: Theory and Practice. CPM 2020

Approximate LCF, Large k

- $O(n^2)$-time algorithm for k-mismatch LCF [1] $\leftarrow$ DP
- No $O(n^{2-\epsilon})$-time algorithm for k-mismatch LCF for $k = \Omega(\log n)$ [2] $\leftarrow$ SETH
- 2-approximation of k-mismatch LCF in $O(n^{4/3+o(1)})$ time [3]
- No $(2-\epsilon)$-approximation of k-mismatch LCF in $O(n^{2-\delta})$ time [2]

[1] T. Flouri, E. Giaquinta, K. Kobert, E. Ukkonen:
LCF with k Mismatches. Inf. Process. Lett., 2015

[2] T. Kociumaka, **R**, T. Starikovskaya:
LCF with Approximately k Mismatches. Algorithmica 2019

[3] G. Gourdel, T. Kociumaka, **R**, T. Starikovskaya:
Approximating LCF with k mismatches: Theory and Practice. CPM 2020

Approximate LCF, Large k

- $O(n^2)$-time algorithm for k-mismatch LCF [1] $\leftarrow$ DP
- No $O(n^{2-\epsilon})$-time algorithm for k-mismatch LCF for $k = \Omega(\log n)$ [2] $\leftarrow$ SETH
- 2-approximation of k-mismatch LCF in $O(n^{4/3+o(1)})$ time [3]

  $\leftarrow$ LSH via "LCF with approximately k mismatches"
- No $(2-\epsilon)$-approximation of k-mismatch LCF in $O(n^{2-\delta})$ time [2]

[1] T. Flouri, E. Giaquinta, K. Kobert, E. Ukkonen:
LCF with k Mismatches. Inf. Process. Lett., 2015

[2] T. Kociumaka, **R**, T. Starikovskaya:
LCF with Approximately k Mismatches. Algorithmica 2019

[3] G. Gourdel, T. Kociumaka, **R**, T. Starikovskaya:
Approximating LCF with k mismatches: Theory and Practice. CPM 2020

Approximate LCF, Large k

- $O(n^2)$-time algorithm for k-mismatch LCF [1] $\leftarrow$ DP
- No $O(n^{2-\epsilon})$-time algorithm for k-mismatch LCF for $k = \Omega(\log n)$ [2] $\leftarrow$ SETH
- 2-approximation of k-mismatch LCF in $O(n^{4/3+o(1)})$ time [3]

  $\leftarrow$ LSH via "LCF with approximately k mismatches"
- No $(2-\epsilon)$-approximation of k-mismatch LCF in $O(n^{2-\delta})$ time [2] $\leftarrow$ SETH

[1] T. Flouri, E. Giaquinta, K. Kobert, E. Ukkonen:
LCF with k Mismatches. Inf. Process. Lett., 2015

[2] T. Kociumaka, **R**, T. Starikovskaya:
LCF with Approximately k Mismatches. Algorithmica 2019

[3] G. Gourdel, T. Kociumaka, **R**, T. Starikovskaya:
Approximating LCF with k mismatches: Theory and Practice. CPM 2020

Approximate LCF, Large k

- $O(n^2)$-time algorithm for k-mismatch LCF [1] $\leftarrow$ DP
- No $O(n^{2-\epsilon})$-time algorithm for k-mismatch LCF for $k = \Omega(\log n)$ [2] $\leftarrow$ SETH
- 2-approximation of k-mismatch LCF in $O(n^{4/3+o(1)})$ time [3]

  $\leftarrow$ LSH via "LCF with approximately k mismatches"
- No $(2-\epsilon)$-approximation of k-mismatch LCF in $O(n^{2-\delta})$ time [2] $\leftarrow$ SETH

OP: Faster approximation algorithm? ([2] had $\tilde{O}(n^{3/2})$-time algorithm)

[1] T. Flouri, E. Giaquinta, K. Kobert, E. Ukkonen:
LCF with k Mismatches. Inf. Process. Lett., 2015

[2] T. Kociumaka, **R**, T. Starikovskaya:
LCF with Approximately k Mismatches. Algorithmica 2019

[3] G. Gourdel, T. Kociumaka, **R**, T. Starikovskaya:
Approximating LCF with k mismatches: Theory and Practice. CPM 2020

Approximate LCF, Edit Distance

- k-edit LCF in $O(n \log^k n)$ time [1]
- $k^{1.5} n^2 / 2^{\Omega(\sqrt{(\log n)/k})}$-time for k-edit LCF [2]

[1] S.V. Thankachan, C. Aluru, S.P. Chockalingam, S. Aluru:
Algorithmic Framework for Approximate Matching Under Bounded Edits
with Applications to Sequence Analysis. RECOMB 2018

[2] A. Abboud, R.R. Williams, H. Yu:
More Applications of the Polynomial Method to Algorithm Design. SODA 2015

Approximate LCF, Edit Distance

- k-edit LCF in $O(n \log^k n)$ time [1]
- $k^{1.5} n^2 / 2^{\Omega(\sqrt{(\log n)/k})}$-time for k-edit LCF [2] $\leftarrow$ polynomial method

[1] S.V. Thankachan, C. Aluru, S.P. Chockalingam, S. Aluru:
Algorithmic Framework for Approximate Matching Under Bounded Edits
with Applications to Sequence Analysis. RECOMB 2018

[2] A. Abboud, R.R. Williams, H. Yu:
More Applications of the Polynomial Method to Algorithm Design. SODA 2015

Approximate LCF, Edit Distance

- k-edit LCF in $O(n \log^k n)$ time [1]
- $k^{1.5} n^2 / 2^{\Omega(\sqrt{(\log n)/k})}$-time for k-edit LCF [2] $\leftarrow$ polynomial method

OP: Faster algorithm?

[1] S.V. Thankachan, C. Aluru, S.P. Chockalingam, S. Aluru:
Algorithmic Framework for Approximate Matching Under Bounded Edits
with Applications to Sequence Analysis. RECOMB 2018

[2] A. Abboud, R.R. Williams, H. Yu:
More Applications of the Polynomial Method to Algorithm Design. SODA 2015

Plan of Presentation

- Packed LCF:
  - Short LCF
  - Long LCF
  - Medium-length LCF
- Approximate LCF
- **Small-space LCF**
- Compressed LCF
- Dynamic LCF
- Internal LCF

Small-Space LCF

- $O(s)$ space, $\tilde{O}(n^2/s)$ time for $n^{2/3} \le s \le n$ [1]
- $O(s)$ space, $O(n^2/s)$ time for $1 \le s \le n$ [2]
- $O(s)$ space, $\tilde{O}(n^2/(sL))$ time for $1 \le s \le n$ where L is the LCF [3]

[1] T. Starikovskaya and H.W. Vildhøj: Time-Space Trade-offs for the LCF Problem. CPM 2013

[2] T. Kociumaka, T. Starikovskaya, and H.W. Vildhøj:
Sublinear Space Algorithms for the LCF Problem. ESA 2014

[3] S. Ben-Nun, S. Golan, T. Kociumaka, M. Kraus:
Time-Space Tradeoffs for Finding a Long Common Substring. CPM 2020

Small-Space LCF

- $O(s)$ space, $\tilde{O}(n^2/s)$ time for $n^{2/3} \leq s \leq n$ [1] $\leftarrow$ tailored
- $O(s)$ space, $O(n^2/s)$ time for $1 \leq s \leq n$ [2] $\leftarrow$ tailored
- $O(s)$ space, $\tilde{O}(n^2/(sL))$ time for $1 \leq s \leq n$ where L is the LCF [3]

[1] T. Starikovskaya and H.W. Vildhøj: Time-Space Trade-offs for the LCF Problem. CPM 2013

[2] T. Kociumaka, T. Starikovskaya, and H.W. Vildhøj:
Sublinear Space Algorithms for the LCF Problem. ESA 2014

[3] S. Ben-Nun, S. Golan, T. Kociumaka, M. Kraus:
Time-Space Tradeoffs for Finding a Long Common Substring. CPM 2020

Small-Space LCF

- $O(s)$ space, $\tilde{O}(n^2/s)$ time for $n^{2/3} \leq s \leq n$ [1] $\leftarrow$ tailored
- $O(s)$ space, $O(n^2/s)$ time for $1 \leq s \leq n$ [2] $\leftarrow$ tailored
- $O(s)$ space, $\tilde{O}(n^2/(sL))$ time for $1 \leq s \leq n$ where L is the LCF [3]

  $\leftarrow$ string synchronizing sets as anchors and MaxPairLCP

[1] T. Starikovskaya and H.W. Vildhøj: Time-Space Trade-offs for the LCF Problem. CPM 2013

[2] T. Kociumaka, T. Starikovskaya, and H.W. Vildhøj:
Sublinear Space Algorithms for the LCF Problem. ESA 2014

[3] S. Ben-Nun, S. Golan, T. Kociumaka, M. Kraus:
Time-Space Tradeoffs for Finding a Long Common Substring. CPM 2020

Plan of Presentation

- Packed LCF:
  - Short LCF
  - Long LCF
  - Medium-length LCF
- Approximate LCF
- Small-space LCF
- **Compressed LCF**
- Dynamic LCF
- Internal LCF

LCF of Compressed Strings

Goal:

- Compute LCF of compressed strings without uncompressing them.

LCF of Compressed Strings

Goal:

- Compute LCF of compressed strings without uncompressing them.

S and T represented as **SLPs** of size n:

- $O(n^4 \log n)$ time [1]

$|S| = N$, **LZ77** parse of S consists of n phrases, T comes as a query, $|T| = m$:

- $O(n \log N)$-sized data structure over S that
  computes LCF of S and T in $\tilde{O}(m)$ time w.h.p. [2]

[1] W. Matsubara, S. Inenaga, A. Ishino, A. Shinohara, T. Nakamura, K. Hashimoto: Efficient Algorithms to Compute Compressed LCFs and Compressed Palindromes. Theor. Comput. Sci., 2009

[2] T. Gagie, P. Gawrychowski, Y. Nekrich: Heaviest Induced Ancestors and LCFs. CCCG 2013

LCF of Compressed Strings

Goal:

- Compute LCF of compressed strings without uncompressing them.

S and T represented as **SLPs** of size n:

- $O(n^4 \log n)$ time [1] $\leftarrow$ compressed overlaps and PREF table [3]

$|S| = N$, **LZ77** parse of S consists of n phrases, T comes as a query, $|T| = m$:

- $O(n \log N)$-sized data structure over S that computes LCF of S and T in $\tilde{O}(m)$ time w.h.p. [2]

[1] W. Matsubara, S. Inenaga, A. Ishino, A. Shinohara, T. Nakamura, K. Hashimoto: Efficient Algorithms to Compute Compressed LCFs and Compressed Palindromes. Theor. Comput. Sci., 2009

[2] T. Gagie, P. Gawrychowski, Y. Nekrich: Heaviest Induced Ancestors and LCFs. CCCG 2013

[3] M. Karpinski, W. Rytter, A. Shinohara: An Efficient Pattern-Matching Algorithm for Strings with Short Descriptions. Nord. J. Comput., 1997

LCF of Compressed Strings

Goal:

- Compute LCF of compressed strings without uncompressing them.

S and T represented as **SLPs** of size n:

- $O(n^4 \log n)$ time [1] $\leftarrow$ compressed overlaps and PREF table [3]

$|S| = N$, **LZ77** parse of S consists of n phrases, T comes as a query, $|T| = m$:

- $O(n \log N)$-sized data structure over S that computes LCF of S and T in $\tilde{O}(m)$ time w.h.p. [2] $\leftarrow$ Heaviest Induced Ancestor queries

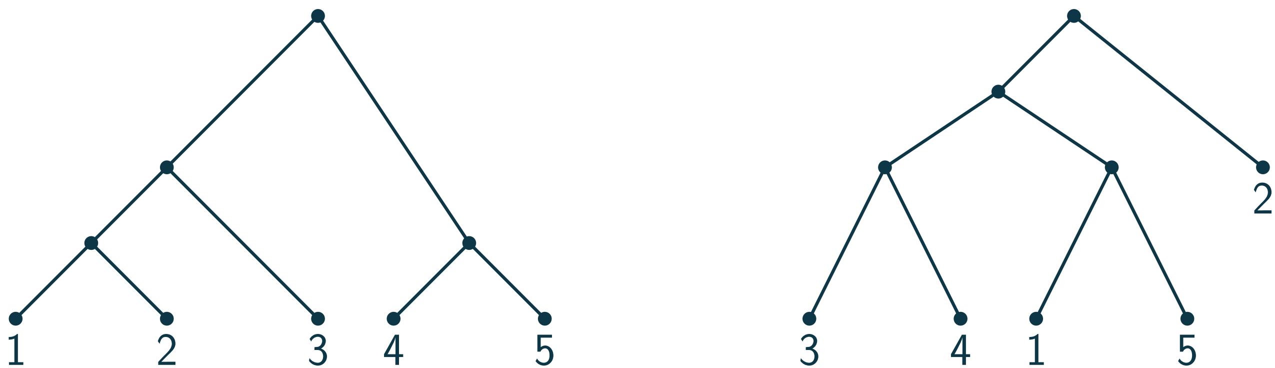
[1] W. Matsubara, S. Inenaga, A. Ishino, A. Shinohara, T. Nakamura, K. Hashimoto: Efficient Algorithms to Compute Compressed LCFs and Compressed Palindromes. Theor. Comput. Sci., 2009

[2] T. Gagie, P. Gawrychowski, Y. Nekrich: Heaviest Induced Ancestors and LCFs. CCCG 2013

[3] M. Karpinski, W. Rytter, A. Shinohara: An Efficient Pattern-Matching Algorithm for Strings with Short Descriptions. Nord. J. Comput., 1997

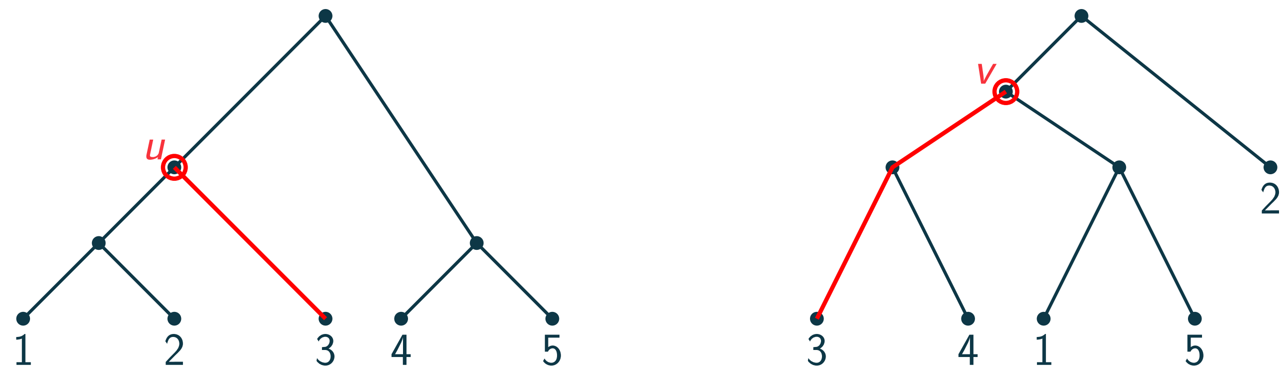
Heaviest Induced Ancestor (HIA) Queries

- Two trees on same set of leaves $\{1, \ldots, N\}$



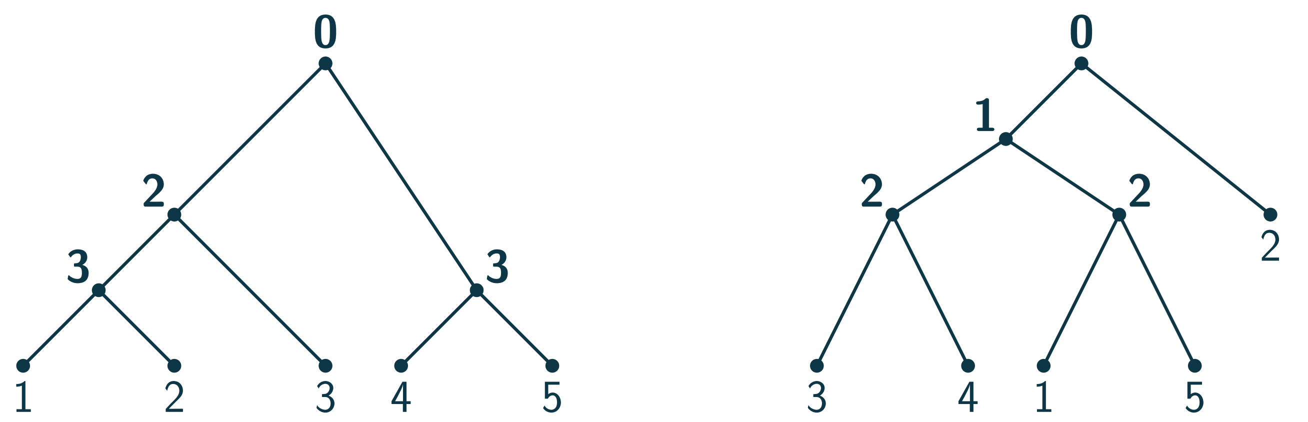
Heaviest Induced Ancestor (HIA) Queries

- Two trees on same set of leaves $\{1, \ldots, N\}$
- Pair of nodes is induced if their subtrees contain a common leaf



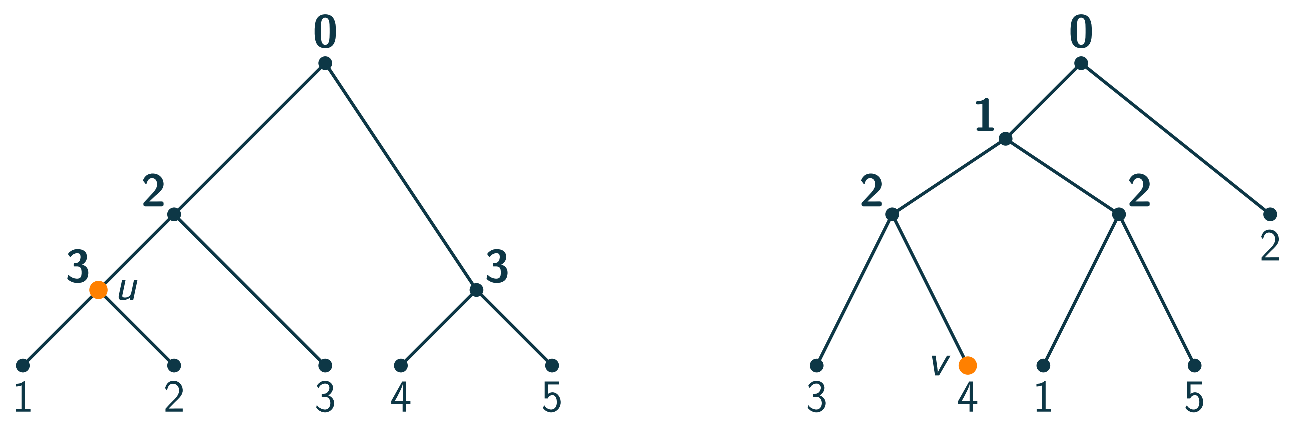
Heaviest Induced Ancestor (HIA) Queries

- Two trees on same set of leaves $\{1, \ldots, N\}$
- Pair of nodes is induced if their subtrees contain a common leaf
- Weighted nodes, non-decreasing weights going down



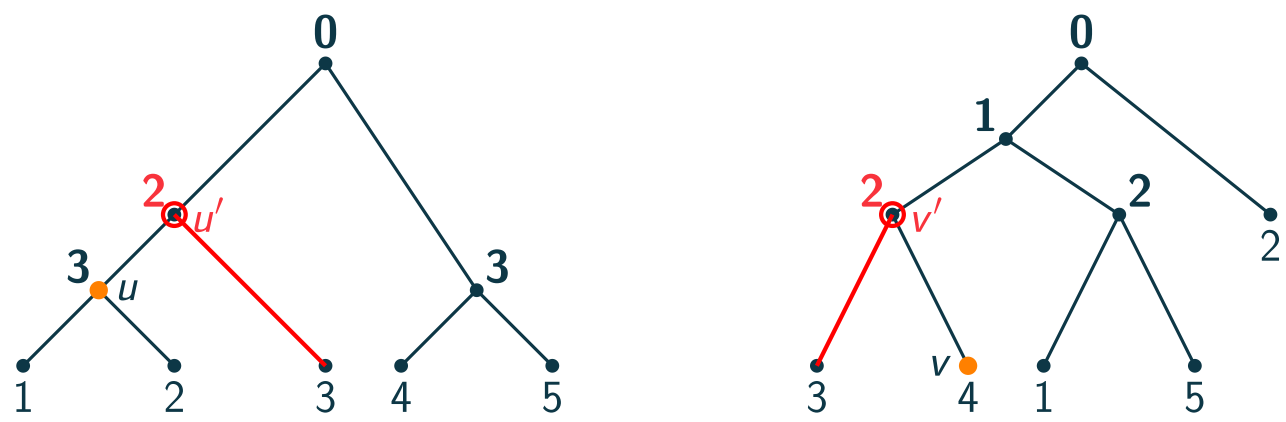
Heaviest Induced Ancestor (HIA) Queries

- Two trees on same set of leaves $\{1, \ldots, N\}$
- Pair of nodes is induced if their subtrees contain a common leaf
- Weighted nodes, non-decreasing weights going down
- Query: given nodes u, v, find their ancestors u', v' that are induced and have max total weight



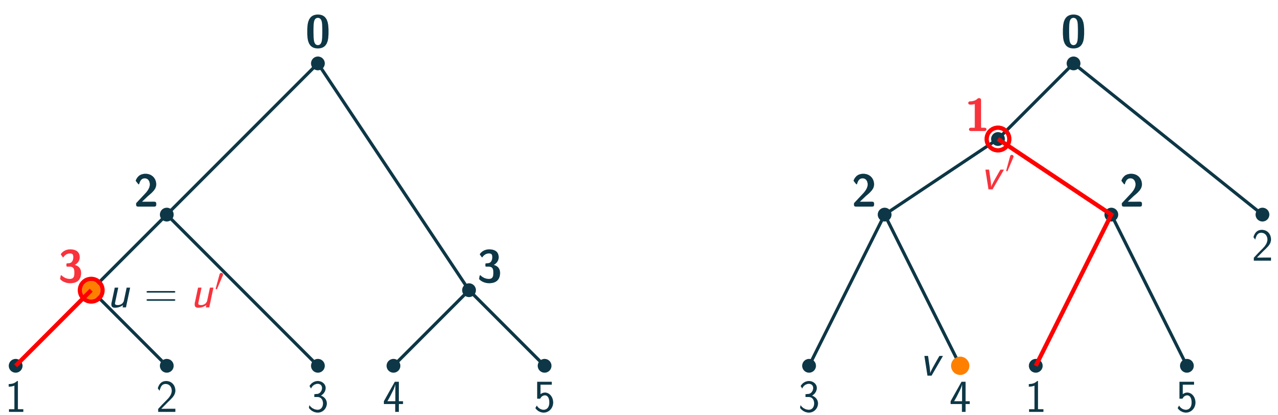
Heaviest Induced Ancestor (HIA) Queries

- Two trees on same set of leaves $\{1, \ldots, N\}$
- Pair of nodes is induced if their subtrees contain a common leaf
- Weighted nodes, non-decreasing weights going down
- Query: given nodes u, v, find their ancestors u', v' that are induced and have max total weight



Heaviest Induced Ancestor (HIA) Queries

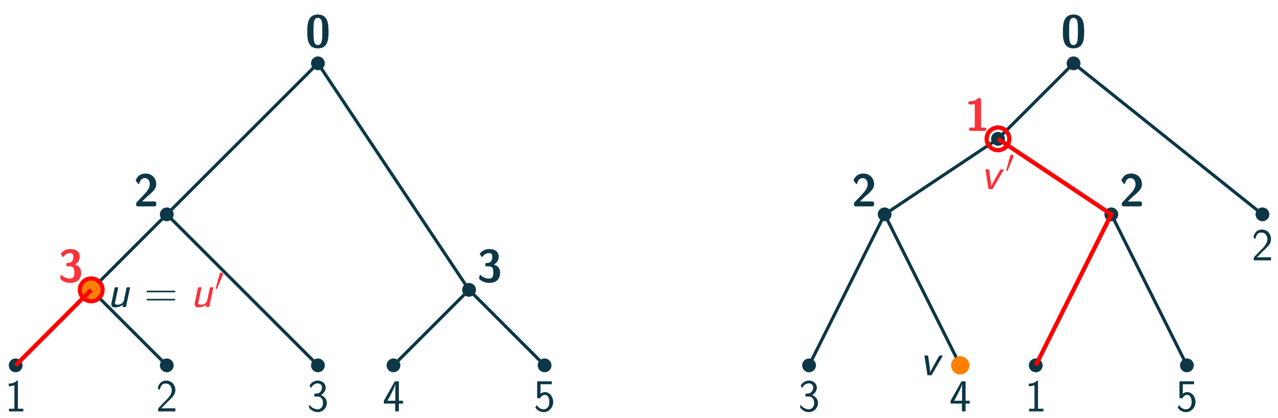
- Two trees on same set of leaves $\{1, \ldots, N\}$
- Pair of nodes is induced if their subtrees contain a common leaf
- Weighted nodes, non-decreasing weights going down
- Query: given nodes u, v, find their ancestors u', v' that are induced and have max total weight



Heaviest Induced Ancestor (HIA) Queries

- Two trees on same set of leaves $\{1, \ldots, N\}$
- Pair of nodes is induced if their subtrees contain a common leaf
- Weighted nodes, non-decreasing weights going down
- Query: given nodes u, v, find their ancestors u', v' that are induced and have max total weight

$\textsc{ColoredTrees} = N$ HIA queries



Heaviest Induced Ancestor (HIA) Queries

- Two trees on same set of leaves $\{1, \ldots, N\}$
- Pair of nodes is induced if their subtrees contain a common leaf
- Weighted nodes, non-decreasing weights going down
- Query: given nodes u, v, find their ancestors u', v' that are induced and have max total weight

$\textsc{ColoredTrees} = N$ HIA queries

- Various trade-offs for HIA queries given in [1-4], lower bound proved in [4]

[1] T. Gagie, P. Gawrychowski, Y. Nekrich: Heaviest Induced Ancestors and LCFs. CCCG 2013

[2] P. Abedin, S. Hooshmand, A. Ganguly, S.V. Thankachan:
The Heaviest Induced Ancestors Problem Revisited. CPM 2018

[3] P. Charalampopoulos, B. Dudek, P. Gawrychowski, K. Pokorski:
Optimal Near-Linear Space Heaviest Induced Ancestors. CPM 2023

[4] P. Charalampopoulos, P. Gawrychowski, K. Pokorski:
Dynamic LCF in Polylogarithmic Time. ICALP 2020

Heaviest Induced Ancestor (HIA) Queries

size	query	paper
$\tilde{O}(N)$	$\Omega(\log N/\log\log N)$	[4]
$O(N)$	$O(\log^2 N/\log\log N)$	[2]
$O(N\log N)$	$O(\log N\log\log N)$	[2]
$O(N\log^2 N)$	$O(\log N)$	[1]
$O(N\log^{2+\epsilon} N)$	$O(\log N/\log\log N)$	[3]
$O(N^{1+\epsilon})$	$O(1)$	[4]

- Various trade-offs for HIA queries given in [1-4], lower bound proved in [4]

[1] T. Gagie, P. Gawrychowski, Y. Nekrich: Heaviest Induced Ancestors and LCFs. CCCG 2013

[2] P. Abedin, S. Hooshmand, A. Ganguly, S.V. Thankachan:
The Heaviest Induced Ancestors Problem Revisited. CPM 2018

[3] P. Charalampopoulos, B. Dudek, P. Gawrychowski, K. Pokorski:
Optimal Near-Linear Space Heaviest Induced Ancestors. CPM 2023

[4] P. Charalampopoulos, P. Gawrychowski, K. Pokorski:
Dynamic LCF in Polylogarithmic Time. ICALP 2020

Heaviest Induced Ancestor (HIA) Queries

size	query	paper
$\tilde{O}(N)$	$\Omega(\log N/\log\log N)$	[4]
$O(N)$	$O(\log^2 N/\log\log N)$	[2]
$O(N\log N)$	$O(\log N\log\log N)$	[2]
$O(N\log^2 N)$	$O(\log N)$	[1]
$O(N\log^{2+\epsilon} N)$	$O(\log N/\log\log N)$	[3]
$O(N^{1+\epsilon})$	$O(1)$	[4]

- Various trade-offs for HIA queries given in [1-4], lower bound proved in [4]

[1] T. Gagie, P. Gawrychowski, Y. Nekrich: Heaviest Induced Ancestors and LCFs. CCCG 2013

[2] P. Abedin, S. Hooshmand, A. Ganguly, S.V. Thankachan:
The Heaviest Induced Ancestors Problem Revisited. CPM 2018

[3] P. Charalampopoulos, B. Dudek, P. Gawrychowski, K. Pokorski:
Optimal Near-Linear Space Heaviest Induced Ancestors. CPM 2023

[4] P. Charalampopoulos, P. Gawrychowski, K. Pokorski:
Dynamic LCF in Polylogarithmic Time. ICALP 2020

Heaviest Induced Ancestor (HIA) Queries

size	query	paper
$\tilde{O}(N)$	$\Omega(\log N / \log\log N)$	[4]
$O(N)$	$O(\log^2 N / \log\log N)$	[2]
$O(N \log N)$	$O(\log N \log\log N)$	[2]
$O(N \log^2 N)$	$O(\log N)$	[1]
$O(N \log^{2+\epsilon} N)$	$O(\log N / \log\log N)$	[3]
$O(N^{1+\epsilon})$	$O(1)$	[4]

OP: Better trade-offs?

- Various trade-offs for HIA queries given in [1-4], lower bound proved in [4]

[1] T. Gagie, P. Gawrychowski, Y. Nekrich: Heaviest Induced Ancestors and LCFs. CCCG 2013

[2] P. Abedin, S. Hooshmand, A. Ganguly, S.V. Thankachan: The Heaviest Induced Ancestors Problem Revisited. CPM 2018

[3] P. Charalampopoulos, B. Dudek, P. Gawrychowski, K. Pokorski: Optimal Near-Linear Space Heaviest Induced Ancestors. CPM 2023

[4] P. Charalampopoulos, P. Gawrychowski, K. Pokorski: Dynamic LCF in Polylogarithmic Time. ICALP 2020

Plan of Presentation

- Packed LCF:
  - Short LCF
  - Long LCF
  - Medium-length LCF
- Approximate LCF
- Small-space LCF
- Compressed LCF
- **Dynamic LCF**
- Internal LCF

Dynamic LCF

- input strings keep changing: insertions, deletions, substitutions allowed
- need to update the LCF after each operation

c b b a a b c a b c a b c b a

d a a b a a b c b b a

Dynamic LCF

- input strings keep changing: insertions, deletions, substitutions allowed
- need to update the LCF after each operation

c b b a a b c a b a a b c b a

d a a b a a b c b b a

Dynamic LCF

- input strings keep changing: insertions, deletions, substitutions allowed
- need to update the LCF after each operation

`c b b a a b c a b a a b c b a`

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

Dynamic LCF

- input strings keep changing: insertions, deletions, substitutions allowed
- need to update the LCF after each operation

c b b a a b c a b a a b c b a

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

Dynamic LCF

- input strings keep changing: insertions, deletions, substitutions allowed
- need to update the LCF after each operation

`c b b a a b c a b a a b c b a`

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

Dynamic LCF

- input strings keep changing: insertions, deletions, substitutions allowed
- need to update the LCF after each operation

c b b a a b c a b a a b c b a

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

- $\log^{O(1)} n$-time "after-edit" queries with $\tilde{O}(n)$-sized data structure [1]
- fully dynamic $\tilde{O}(n^{2/3})$-time updates [2]
- fully dynamic $\log^{O(1)} n$ amortized time substitutions [3]

[1] A. Amir, P. Charalampopoulos, C.S. Iliopoulos, S.P. Pissis, **R**:
LCF After One Edit Operation. SPIRE 2017

[2] A. Amir, P. Charalampopoulos, S.P. Pissis, **R**: Dynamic and Internal LCF. Algorithmica 2020

[3] P. Charalampopoulos, P. Gawrychowski, K. Pokorski:
Dynamic LCF in Polylogarithmic Time. ICALP 2020

Dynamic LCF

- input strings keep changing: insertions, deletions, substitutions allowed
- need to update the LCF after each operation

`c b b a a b c a b a a b c b a`

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

- $\log^{O(1)} n$-time "after-edit" queries with $\tilde{O}(n)$-sized data structure [1]

  $\leftarrow$ HIA queries
- fully dynamic $\tilde{O}(n^{2/3})$-time updates [2]
- fully dynamic $\log^{O(1)} n$ amortized time substitutions [3]

[1] A. Amir, P. Charalampopoulos, C.S. Iliopoulos, S.P. Pissis, **R**:
LCF After One Edit Operation. SPIRE 2017

[2] A. Amir, P. Charalampopoulos, S.P. Pissis, **R**: Dynamic and Internal LCF. Algorithmica 2020

[3] P. Charalampopoulos, P. Gawrychowski, K. Pokorski:
Dynamic LCF in Polylogarithmic Time. ICALP 2020

Dynamic LCF

- input strings keep changing: insertions, deletions, substitutions allowed
- need to update the LCF after each operation

```
c b b a a b c a b a a b c b a

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

- $\log^{O(1)} n$-time "after-edit" queries with $\tilde{O}(n)$-sized data structure [1]

  $\leftarrow$ HIA queries
- fully dynamic $\tilde{O}(n^{2/3})$-time updates [2] $\leftarrow$ dynamic strings, MaxPairLCP
- fully dynamic $\log^{O(1)} n$ amortized time substitutions [3]

[1] A. Amir, P. Charalampopoulos, C.S. Iliopoulos, S.P. Pissis, **R**:
LCF After One Edit Operation. SPIRE 2017

[2] A. Amir, P. Charalampopoulos, S.P. Pissis, **R**: Dynamic and Internal LCF. Algorithmica 2020

[3] P. Charalampopoulos, P. Gawrychowski, K. Pokorski:
Dynamic LCF in Polylogarithmic Time. ICALP 2020

Dynamic LCF

- input strings keep changing: insertions, deletions, substitutions allowed
- need to update the LCF after each operation

c b b a a b c a b a a b c b a

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

- $\log^{O(1)} n$-time "after-edit" queries with $\tilde{O}(n)$-sized data structure [1]

  $\leftarrow$ HIA queries
- fully dynamic $\tilde{O}(n^{2/3})$-time updates [2] $\leftarrow$ dynamic strings, MaxPairLCP
- fully dynamic $\log^{O(1)} n$ amortized time substitutions [3] $\leftarrow$ locally cons. parsing

[1] A. Amir, P. Charalampopoulos, C.S. Iliopoulos, S.P. Pissis, **R**:
LCF After One Edit Operation. SPIRE 2017

[2] A. Amir, P. Charalampopoulos, S.P. Pissis, **R**: Dynamic and Internal LCF. Algorithmica 2020

[3] P. Charalampopoulos, P. Gawrychowski, K. Pokorski:
Dynamic LCF in Polylogarithmic Time. ICALP 2020

Plan of Presentation

- Packed LCF:
  - Short LCF
  - Long LCF
  - Medium-length LCF
- Approximate LCF
- Small-space LCF
- Compressed LCF
- Dynamic LCF
- **Internal LCF**

Internal LCF

- queries about LCF of given factors of two strings

c b **b a a b c** a b c a b c b a

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

Internal LCF

- queries about LCF of given factors of two strings

c b b a a b c a b c a b c b a

d a a b a a b c b b a

Internal LCF

- queries about LCF of given factors of two strings

c b b a a b c a b c a b c b a

d a a b a a b c b b a

Internal LCF

- queries about LCF of given factors of two strings

c b b a a b c a b c a b c b a

d a a b a a b c b b a

factor of S	factor of T	space	query	comment
any	any	$\tilde{O}(n^2/t^2)$	$\tilde{O}(t)$	
any	any	$\tilde{\Omega}(n^2/t^2)$	t	clb, Set Disjointness

[1] A. Amir, P. Charalampopoulos, S.P. Pissis, **R**: Dynamic and Internal LCF. Algorithmica 2020

Internal LCF

- queries about LCF of given factors of two strings

c b b a a b c a b c a b c b a

d a a b a a b c b b a

factor of S	factor of T	space	query	comment
any	any	$\tilde{O}(n^2/t^2)$	$\tilde{O}(t)$	
any	any	$\tilde{\Omega}(n^2/t^2)$	t	clb, Set Disjointness
prefix/suffix	prefix/suffix	$O(n \log n)$	$O(\log n)$	$O(n \log^2 n)$ constr.
any	whole T	$O(n)$	$O(\log n)$	$O(n)$ constr.

[1] A. Amir, P. Charalampopoulos, S.P. Pissis, **R**: Dynamic and Internal LCF. Algorithmica 2020

Internal LCF

- queries about LCF of given factors of two strings

c b b a a b c a b c a b c b a

d a a b a a b c b b a

factor of S	factor of T	space	query	comment
any	any	$\tilde{O}(n^2/t^2)$	$\tilde{O}(t)$	
any	any	$\tilde{\Omega}(n^2/t^2)$	t	clb, Set Disjointness
prefix/suffix	prefix/suffix	$O(n \log n)$	$O(\log n)$	$O(n \log^2 n)$ constr.
any	whole T	$O(n)$	$O(\log n)$	$O(n)$ constr.

OP: Better data structures for special cases?

[1] A. Amir, P. Charalampopoulos, S.P. Pissis, **R**: Dynamic and Internal LCF. Algorithmica 2020

Summary

- Packed LCF:
  - Short LCF
  - Long LCF
  - Medium-length LCF
- Approximate LCF
- Small-space LCF
- Compressed LCF
- Dynamic LCF
- Internal LCF

Other Variants of LCF

- Longest common Abelian factor (LCAF) [1-5]

[1] A. Alatabbi, C.S. Iliopoulos, A. Langiu, and M.S. Rahman: Algorithms for LCAFs. Int. J. Found. Comput. Sci., 2016

[2] G. Badkobeh, T. Gagie, S. Grabowski, Y. Nakashima, S.J. Puglisi, and S. Sugimoto: LCAFs and large alphabets. SPIRE 2016

[3] S. Sugimoto, N. Noda, S. Inenaga, H. Bannai, M. Takeda: Computing Abelian String Regularities Based on RLE. IWOCA 2017

[4] S. Grabowski: Regular Abelian Periods and LCAFs on Run-Length Encoded Strings. SPIRE 2017

[5] S. Grabowski, T. Kociumaka, **R**: On LCAF with and without RLE. Fundam. Informaticae, 2018

Other Variants of LCF

- Longest common Abelian factor (LCAF) [1-5]
- Longest common order-preserving factor [6]

[1] A. Alatabbi, C.S. Iliopoulos, A. Langiu, and M.S. Rahman:
Algorithms for LCAFs. Int. J. Found. Comput. Sci., 2016

[2] G. Badkobeh, T. Gagie, S. Grabowski, Y. Nakashima, S.J. Puglisi, and S. Sugimoto:
LCAFs and large alphabets. SPIRE 2016

[3] S. Sugimoto, N. Noda, S. Inenaga, H. Bannai, M. Takeda:
Computing Abelian String Regularities Based on RLE. IWOCA 2017

[4] S. Grabowski: Regular Abelian Periods and LCAFs on Run-Length Encoded Strings.
SPIRE 2017

[5] S. Grabowski, T. Kociumaka, **R**:
On LCAF with and without RLE. Fundam. Informaticae, 2018

[6] M. Crochemore, C.S. Iliopoulos, T. Kociumaka, M. Kubica, A. Langiu, S.P. Pissis, **R**,
W. Rytter, T. Waleń: Order-preserving indexing. Theor. Comput. Sci., 2016

Other Variants of LCF

- Longest common Abelian factor (LCAF) [1-5]
- Longest common order-preserving factor [6]
- Longest common circular factor (LCCF) [7]

[1] A. Alatabbi, C.S. Iliopoulos, A. Langiu, and M.S. Rahman:
Algorithms for LCAFs. Int. J. Found. Comput. Sci., 2016

[2] G. Badkobeh, T. Gagie, S. Grabowski, Y. Nakashima, S.J. Puglisi, and S. Sugimoto:
LCAFs and large alphabets. SPIRE 2016

[3] S. Sugimoto, N. Noda, S. Inenaga, H. Bannai, M. Takeda:
Computing Abelian String Regularities Based on RLE. IWOCA 2017

[4] S. Grabowski: Regular Abelian Periods and LCAFs on Run-Length Encoded Strings.
SPIRE 2017

[5] S. Grabowski, T. Kociumaka, **R**:
On LCAF with and without RLE. Fundam. Informaticae, 2018

[6] M. Crochemore, C.S. Iliopoulos, T. Kociumaka, M. Kubica, A. Langiu, S.P. Pissis, **R**,
W. Rytter, T. Waleń: Order-preserving indexing. Theor. Comput. Sci., 2016

[7] M. Alzamel, M. Crochemore, C.S. Iliopoulos, T. Kociumaka, **R**, W. Rytter, J. Straszyński,
T. Waleń, W. Zuba: Quasi-Linear-Time Algorithm for LCCF. CPM 2019

Summary

- Packed LCF:
  - Short LCF
  - Long LCF
  - Medium-length LCF
- Approximate LCF
- Small-space LCF
- Compressed LCF
- Dynamic LCF
- Internal LCF
- Other variants of LCF