Efficient Enumeration of Distinct Factors Using Package Representations

Panagiotis Charalampopoulos[1] Tomasz Kociumaka[2]
Jakub Radoszewski[3] Wojciech Rytter[3]
Tomasz Waleń[3] Wiktor Zuba[3]

[1]King's College London, UK

[2]University of California, Berkeley, USA

[3]University of Warsaw, Poland

SPIRE 2020
13 October 2020

Package Representations

Sometimes interesting subsets of factors of a string S of length n can be described concisely (e.g. property pattern matching).

Package Representations

Sometimes interesting subsets of factors of a string S of length n can be described concisely (e.g. property pattern matching).

We show how to enumerate and count distinct factors represented compactly by *package representations*.

Package Representations

Sometimes interesting subsets of factors of a string S of length n can be described concisely (e.g. property pattern matching).

We show how to enumerate and count distinct factors represented compactly by *package representations*.

A package (i, ℓ, k) represents the factors of S of length ℓ that start in the interval $[i, i + k]$.

Package Representations

Sometimes interesting subsets of factors of a string S of length n can be described concisely (e.g. property pattern matching).

We show how to enumerate and count distinct factors represented compactly by *package representations*.

A package (i, ℓ, k) represents the factors of S of length ℓ that start in the interval $[i, i+k]$.

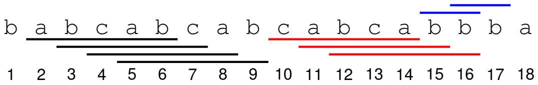
b	a	b	c	a	b	c	a	b	c	a	b	c	a	b	b	b	a
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Package Representations

Sometimes interesting subsets of factors of a string S of length n can be described concisely (e.g. property pattern matching).

We show how to enumerate and count distinct factors represented compactly by *package representations*.

A package (i, ℓ, k) represents the factors of S of length ℓ that start in the interval $[i, i+k]$.



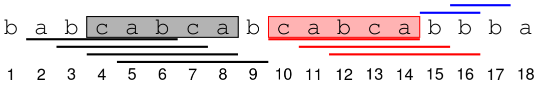
Set of packages $\{(15, 2, 1), (2, 5, 3), (10, 5, 2)\}$.

Package Representations

Sometimes interesting subsets of factors of a string S of length n can be described concisely (e.g. property pattern matching).

We show how to enumerate and count distinct factors represented compactly by *package representations*.

A package (i, ℓ, k) represents the factors of S of length ℓ that start in the interval $[i, i+k]$.

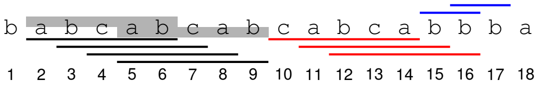


Set of packages $\{(15, 2, 1), (2, 5, 3), (10, 5, 2)\}$.

Sometimes interesting subsets of factors of a string S of length n can be described concisely (e.g. property pattern matching).

We show how to enumerate and count distinct factors represented compactly by *package representations*.

A package (i, ℓ, k) represents the factors of S of length ℓ that start in the interval $[i, i + k]$.



Set of packages $\{(15, 2, 1), (2, 5, 3), (10, 5, 2)\}$.

Period

A positive integer p is a *period* of a string S if $S[i] = S[i+p]$ for all $i = 1, \ldots, |S| - p$.

Preliminaries

Period

A positive integer p is a *period* of a string S if $S[i] = S[i+p]$ for all $i = 1, \ldots, |S| - p$.
The smallest period $\mathrm{per}(S)$ is *the period* of S.

Period

A positive integer p is a *period* of a string S if $S[i] = S[i+p]$ for all $i = 1, \ldots, |S| - p$.
The smallest period $\text{per}(S)$ is *the period* of S.

A string S is *periodic* if $\text{per}(S) \leq |S|/2$.

Period

A positive integer p is a *period* of a string S if $S[i] = S[i+p]$ for all $i = 1, \ldots, |S| - p$.
The smallest period $\text{per}(S)$ is *the period* of S.

A string S is *periodic* if $\text{per}(S) \leq |S|/2$.

E.g. *abcabcab* is periodic with period 3.

Squares and Runs I

Squares

A square is a non-empty string of the form UU; e.g. *abcabc*.

Squares and Runs I

Squares

A square is a non-empty string of the form UU; e.g. $abcabc$.

Runs

A run in a string S is a pair $(S[a\,..\,b], p)$ such that:

Squares and Runs I

Squares

A square is a non-empty string of the form UU; e.g. $abcabc$.

Runs

A run in a string S is a pair $(S[a \,..\, b], p)$ such that:

- the substring $S[a \,..\, b]$ is periodic with shortest period p;

Squares and Runs I

Squares

A square is a non-empty string of the form UU; e.g. $abcabc$.

Runs

A run in a string S is a pair $(S[a\mathinner{.\,.}b], p)$ such that:

- the substring $S[a\mathinner{.\,.}b]$ is periodic with shortest period p;
- $S[a\mathinner{.\,.}b]$ cannot be extended to the left nor to the right without violating the above property.

Squares and Runs I

Squares

A square is a non-empty string of the form UU; e.g. $abcabc$.

Runs

A run in a string S is a pair $(S[a\mathinner{..}b], p)$ such that:

- the substring $S[a\mathinner{..}b]$ is periodic with shortest period p;
- $S[a\mathinner{..}b]$ cannot be extended to the left nor to the right without violating the above property.

b	a	b	c	a	b	c	a	b	c	a	b	c	a	b	b	b	a
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

run $(S[2\mathinner{..}15], 3)$

Squares and Runs I

Squares

A square is a non-empty string of the form UU; e.g. $abcabc$.

Runs

A run in a string S is a pair $(S[a\,..\,b], p)$ such that:

- the substring $S[a\,..\,b]$ is periodic with shortest period p;
- $S[a\,..\,b]$ cannot be extended to the left nor to the right without violating the above property.

b	a	b	c	a	b	c	a	b	c	a	b	**c**	a	b	b	b	a
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

run $(S[2\,..\,15], 3)$

Squares and Runs I

Squares

A square is a non-empty string of the form UU; e.g. $abcabc$.

Runs

A run in a string S is a pair $(S[a\mathinner{.\,.}b], p)$ such that:

- the substring $S[a\mathinner{.\,.}b]$ is periodic with shortest period p;
- $S[a\mathinner{.\,.}b]$ cannot be extended to the left nor to the right without violating the above property.

b	a	b	**c**	a	b	c	a	b	c	a	b	c	a	b	b	b	a
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

run $(S[2\mathinner{.\,.}15], 3)$

Squares and Runs I

Squares

A square is a non-empty string of the form UU; e.g. $abcabc$.

Runs

A run in a string S is a pair $(S[a\,..\,b], p)$ such that:

- the substring $S[a\,..\,b]$ is periodic with shortest period p;
- $S[a\,..\,b]$ cannot be extended to the left nor to the right without violating the above property.

b	a	b	c	a	b	c	a	b	c	a	b	c	a	b	b	b	a
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

run $(S[2\,..\,15], 3)$

Squares and Runs I

Squares

A square is a non-empty string of the form UU; e.g. *abcabc*.

Runs

A generalised run in a string S is a pair $(S[a\,.\,.\,b], p)$ such that:

- the substring $S[a\,.\,.\,b]$ is periodic with shortest/a period p;
- $S[a\,.\,.\,b]$ cannot be extended to the left nor to the right without violating the above property.

b	a	b	c	a	b	c	a	b	c	a	b	c	a	b	b	b	a
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

gen. run $(S[2\,.\,.\,15], 3)$

Squares and Runs I

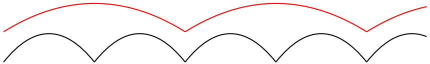
Squares

A square is a non-empty string of the form UU; e.g. $abcabc$.

Runs

A generalised run in a string S is a pair $(S[a \mathinner{..} b], p)$ such that:

- the substring $S[a \mathinner{..} b]$ is periodic with shortest/a period p;
- $S[a \mathinner{..} b]$ cannot be extended to the left nor to the right without violating the above property.



b	a	b	c	a	b	c	a	b	c	a	b	c	a	b	b	b	a
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

gen. run $(S[2 \mathinner{..} 15], 6)$
gen. run $(S[2 \mathinner{..} 15], 3)$

Squares and Runs I

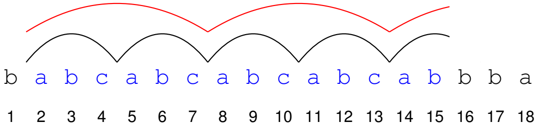
Squares

A square is a non-empty string of the form UU; e.g. *abcabc*.

Runs

A generalised run in a string S is a pair $(S[a\,..\,b], p)$ such that:

- the substring $S[a\,..\,b]$ is periodic with shortest/a period p;
- $S[a\,..\,b]$ cannot be extended to the left nor to the right without violating the above property.



gen. run $(S[2\,..\,15], 6)$

gen. run $(S[2\,..\,15], 3)$

Each occurrence of a square UU in S is contained in a unique generalised run $(S[a\,..\,b], |U|)$.

Squares and Runs I

Squares

A square is a non-empty string of the form UU; e.g. $abcabc$.

Runs

A generalised run in a string S is a pair $(S[a..b], p)$ such that:

- the substring $S[a..b]$ is periodic with shortest/a period p;
- $S[a..b]$ cannot be extended to the left nor to the right without violating the above property.

b	a	b	c	a	b	c	a	b	c	a	b	c	a	b	b	b	a
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

gen. run $(S[2..15], 6)$

gen. run $(S[2..15], 3)$

Each occurrence of a square UU in S is contained in a unique generalised run $(S[a..b], |U|)$.

Squares and Runs I

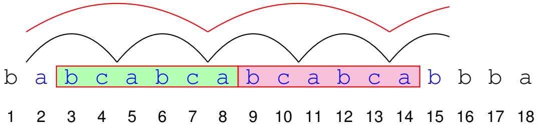
Squares

A square is a non-empty string of the form UU; e.g. *abcabc*.

Runs

A generalised run in a string S is a pair $(S[a \mathinner{.\,.} b], p)$ such that:

- the substring $S[a \mathinner{.\,.} b]$ is periodic with shortest/a period p;
- $S[a \mathinner{.\,.} b]$ cannot be extended to the left nor to the right without violating the above property.



gen. run $(S[2 \mathinner{.\,.} 15], 6)$
gen. run $(S[2 \mathinner{.\,.} 15], 3)$

Each occurrence of a square UU in S is contained in a unique generalised run $(S[a \mathinner{.\,.} b], |U|)$.

Squares and Runs II

Theorem

[Fraenkel-Simpson, J. Comb. Theory A 1996; Gusfield-Stoye, JCSS 2014]

A string of length n has $\mathcal{O}(n)$ distinct squares and they can be computed in $\mathcal{O}(n)$ time.

Squares and Runs II

Theorem
[Fraenkel-Simpson, J. Comb. Theory A 1996; Gusfield-Stoye, JCSS 2014]

A string of length n has $\mathcal{O}(n)$ distinct squares and they can be computed in $\mathcal{O}(n)$ time.

Theorem [Kolpakov-Kucherov, FOCS 1999]

A string of length n has $\mathcal{O}(n)$ runs and they can be computed in $\mathcal{O}(n)$ time.

Squares and Runs II

Theorem
[Fraenkel-Simpson, J. Comb. Theory A 1996; Gusfield-Stoye, JCSS 2014]

A string of length n has $\mathcal{O}(n)$ distinct squares and they can be computed in $\mathcal{O}(n)$ time.

Theorem [Kolpakov-Kucherov, FOCS 1999]

A string of length n has $\mathcal{O}(n)$ runs and they can be computed in $\mathcal{O}(n)$ time.

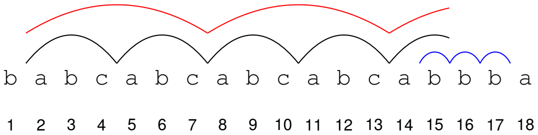
In particular, an algorithm of [Crochemore et al., TCS 2014] extracts the distinct squares of a string from its runs in $\mathcal{O}(n)$ time.

A Package Representation for Squares

A package (i, ℓ, k) represents the factors of S of length ℓ that start in the interval $[i, i + k]$.

A Package Representation for Squares

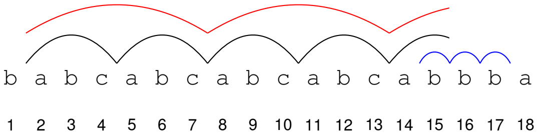
A package (i, ℓ, k) represents the factors of S of length ℓ that start in the interval $[i, i + k]$.



gen. run $(S[2\,..\,15], 6)$
gen. run $(S[2\,..\,15], 3)$
gen. run $(S[15\,..\,17], 1)$

A Package Representation for Squares

A package (i, ℓ, k) represents the factors of S of length ℓ that start in the interval $[i, i + k]$.

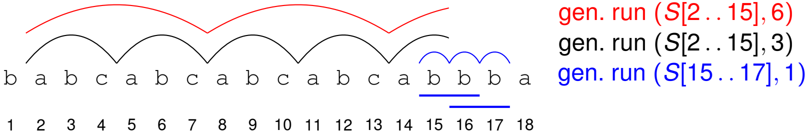


gen. run $(S[2\,..\,15], 6)$
gen. run $(S[2\,..\,15], 3)$
gen. run $(S[15\,..\,17], 1)$

The three generalised runs generate the following package representation of all squares: $\{\qquad\qquad\qquad\qquad\}$.

A Package Representation for Squares

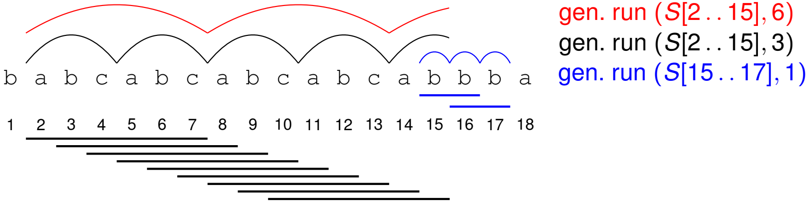
A package (i, ℓ, k) represents the factors of S of length ℓ that start in the interval $[i, i + k]$.



The three generalised runs generate the following package representation of all squares: $\{(15, 2, 1) \qquad \}$.

A Package Representation for Squares

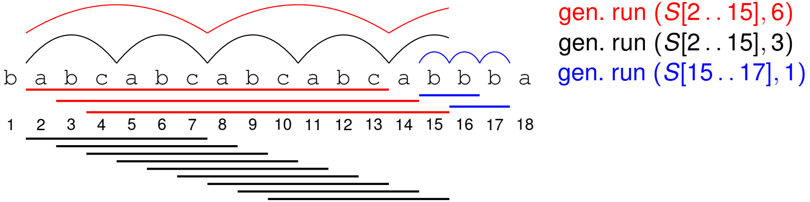
A package (i, ℓ, k) represents the factors of S of length ℓ that start in the interval $[i, i + k]$.



The three generalised runs generate the following package representation of all squares: $\{(15, 2, 1), (2, 6, 8) \qquad\qquad \}$.

A Package Representation for Squares

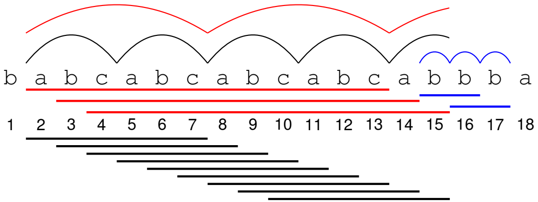
A package (i, ℓ, k) represents the factors of S of length ℓ that start in the interval $[i, i + k]$.



The three generalised runs generate the following package representation of all squares: $\{(15, 2, 1), (2, 6, 8), (2, 12, 2)\}$.

A Package Representation for Squares

A package (i, ℓ, k) represents the factors of S of length ℓ that start in the interval $[i, i+k]$.



gen. run $(S[2\,..\,15], 6)$
gen. run $(S[2\,..\,15], 3)$
gen. run $(S[15\,..\,17], 1)$

The three generalised runs generate the following package representation of all squares: $\{(15, 2, 1), (2, 6, 8), (2, 12, 2)\}$.

A string of length n has $\mathcal{O}(n)$ generalised runs and each of them yields one package.

Package Representations

Consider a set $\mathcal{F}$ of m (disjoint) packages (i, ℓ, k).

Package Representations

Consider a set $\mathcal{F}$ of m (disjoint) packages (i, ℓ, k).

$$\text{Factors}(\mathcal{F}) = \{S[j \mathinner{.\,.} j+\ell) : j \in [i, i+k] \text{ and } (i, \ell, k) \in \mathcal{F}\}.$$

Package Representations

Consider a set $\mathcal{F}$ of m (disjoint) packages (i, ℓ, k).

$$\text{Factors}(\mathcal{F}) = \{S[j\mathinner{.\,.} j+\ell) : j \in [i, i+k] \text{ and } (i,\ell,k) \in \mathcal{F}\}.$$

We consider the problems of computing

- $\text{Factors}(\mathcal{F})$,
- $|\text{Factors}(\mathcal{F})|$.

Package Representations

Consider a set $\mathcal{F}$ of m (disjoint) packages (i, ℓ, k).

$$\text{Factors}(\mathcal{F}) = \{S[j\mathinner{.\,.} j+\ell) \,:\, j \in [i, i+k] \text{ and } (i, \ell, k) \in \mathcal{F}\}.$$

We consider the problems of computing

- Factors($\mathcal{F}$),
- $|$Factors($\mathcal{F}$)$|$.

Remark

This is related to computing the *subword complexity* of S.

A Special Case

$\mathcal{F}$ is a special package representation if every occurrence of every factor represented by $\mathcal{F}$ is captured by some package in $\mathcal{F}$.

A Special Case

$\mathcal{F}$ is a special package representation if every occurrence of every factor represented by $\mathcal{F}$ is captured by some package in $\mathcal{F}$. (Our package representation for squares is special.)

A Special Case

$\mathcal{F}$ is a special package representation if every occurrence of every factor represented by $\mathcal{F}$ is captured by some package in $\mathcal{F}$. (Our package representation for squares is special.)

Aim: Compute leftmost occurrences.

A Special Case

$\mathcal{F}$ is a special package representation if every occurrence of every factor represented by $\mathcal{F}$ is captured by some package in $\mathcal{F}$. (Our package representation for squares is special.)

Aim: Compute leftmost occurrences.

We use the longest previous factor array $LPF[1\,..\,n]$.

b	a	b	c	a	b	c	a	b	c	a	b	c	a	b	b	b	a
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

$LPF[3] = 1$

A Special Case

$\mathcal{F}$ is a special package representation if every occurrence of every factor represented by $\mathcal{F}$ is captured by some package in $\mathcal{F}$. (Our package representation for squares is special.)

Aim: Compute leftmost occurrences.

We use the longest previous factor array $LPF[1\,..\,n]$.

b	a	b	c	a	b	c	a	b	c	a	b	c	a	b	b	b	a
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

$LPF[6] = 10$

A Special Case

$\mathcal{F}$ is a special package representation if every occurrence of every factor represented by $\mathcal{F}$ is captured by some package in $\mathcal{F}$. (Our package representation for squares is special.)

Aim: Compute leftmost occurrences.

We use the longest previous factor array $LPF[1\,..\,n]$.

b	a	b	c	a	b	c	a	b	c	a	b	c	a	b	b	b	a
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

$LPF[6] = 10$
not leftmost

A Special Case

$\mathcal{F}$ is a special package representation if every occurrence of every factor represented by $\mathcal{F}$ is captured by some package in $\mathcal{F}$. (Our package representation for squares is special.)

Aim: Compute leftmost occurrences.

We use the longest previous factor array $LPF[1\,..\,n]$.

b	a	b	c	a	b	c	a	b	c	a	b	c	a	b	b	b	a
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

$LPF[6] = 10$
leftmost

A Special Case

$\mathcal{F}$ is a special package representation if every occurrence of every factor represented by $\mathcal{F}$ is captured by some package in $\mathcal{F}$. (Our package representation for squares is special.)

Aim: Compute leftmost occurrences.

We use the longest previous factor array $LPF[1\mathinner{.\,.} n]$.

$$\text{Smaller}_\ell = \{\, j \in [1, n] \ : \ LPF[j] < \ell \,\}$$

A Special Case

$\mathcal{F}$ is a special package representation if every occurrence of every factor represented by $\mathcal{F}$ is captured by some package in $\mathcal{F}$. (Our package representation for squares is special.)

Aim: Compute leftmost occurrences.

We use the longest previous factor array $LPF[1\,..\,n]$.

$$\mathsf{Smaller}_\ell = \{\, j \in [1, n] \;:\; LPF[j] < \ell \,\}$$

Observation

If $\mathcal{F}$ is special,

$$\mathsf{Factors}(\mathcal{F}) = \bigcup_{(i,\ell,k)\,\in\mathcal{F}} \{S[j\,..\,j+\ell) \;:\; j \in [i, i+k] \cap \mathsf{Smaller}_\ell\}.$$

Reporting in the Special Case

Algorithm 1: High-level structure of the algorithm.

$U := [1, n]$; $\mathcal{P} := \emptyset$

for $\ell := n$ **down to** 1 **do**

  $U := U \setminus \{j \; : \; LPF[j] = \ell\}$; // $U = \text{Smaller}_\ell$

  foreach $(i, \ell, k) \in \mathcal{F}$ **do**

    foreach $j \in [i, i+k] \cap U$ **do**

      $\mathcal{P} := \mathcal{P} \cup \{S[j \mathinner{.\,.} j+\ell)\}$; // End: $\mathcal{P} = \text{Factors}(\mathcal{F})$

Reporting in the Special Case

Algorithm 1: High-level structure of the algorithm.

$U := [1, n]$; $\mathcal{P} := \emptyset$

for $\ell := n$ **down to** 1 **do**

  $U := U \setminus \{j \ : \ LPF[j] = \ell\}$; `//` $U = \text{Smaller}_\ell$

  foreach $(i, \ell, k) \in \mathcal{F}$ **do**

    foreach $j \in [i, i+k] \cap U$ **do**

      $\mathcal{P} := \mathcal{P} \cup \{S[j \mathinner{.\,.} j + \ell)\}$; `// End:` $\mathcal{P} = \text{Factors}(\mathcal{F})$

We show an implementation of this idea in $\mathcal{O}(n + m + |\text{output}|)$, using the Union-Find data structure of [Gabow-Tarjan, JCSS 1985].

Counting in the Special Case I

For each $(i, \ell, k) \in \mathcal{F}$, it suffices to count the number of elements in $LPF[i \mathinner{.\,.} i + k]$ that are smaller than ℓ.

Counting in the Special Case I

For each $(i, \ell, k) \in \mathcal{F}$, it suffices to count the number of elements in $LPF[i \mathinner{.\,.} i+k]$ that are smaller than ℓ.

Consider the following queries:

$$\text{Smaller}_\ell[i] = |\{j \in [1, i] : LPF[j] < \ell\}|.$$

Counting in the Special Case I

For each $(i, \ell, k) \in \mathcal{F}$, it suffices to count the number of elements in $LPF[i\,..\,i+k]$ that are smaller than ℓ.

Consider the following queries:

$$\text{Smaller}_\ell[i] = |\{j \in [1, i] : LPF[j] < \ell\}|.$$

$$|\text{Factors}(\mathcal{F})| = \sum_{(i,\ell,k)\in\mathcal{F}} \text{Smaller}_\ell[i+k] - \text{Smaller}_\ell[i-1].$$

Counting in the Special Case I

For each $(i, \ell, k) \in \mathcal{F}$, it suffices to count the number of elements in $LPF[i \mathinner{.\,.} i+k]$ that are smaller than ℓ.

Consider the following queries:

$$\text{Smaller}_\ell[i] = |\{j \in [1, i] : LPF[j] < \ell\}|.$$

$$|\text{Factors}(\mathcal{F})| = \sum_{(i,\ell,k)\in\mathcal{F}} \text{Smaller}_\ell[i+k] - \text{Smaller}_\ell[i-1].$$

We obtain an $\mathcal{O}(n+m)$-time algorithm by showing how to optimally answer these queries.

Counting in the Special Case II

Maintain array $A[1\,..\,n]$ such that during the ith phase:

$$A[\ell] = \begin{cases} i - \mathrm{Smaller}_\ell[i] & \text{if } \ell > LPF[i], \\ \mathrm{Smaller}_\ell[i] & \text{if } \ell \le LPF[i]. \end{cases}$$

Counting in the Special Case II

Maintain array $A[1\,..\,n]$ such that during the ith phase:

$$A[\ell] = \begin{cases} |\{j \in [1,i] : LPF[j] \geq \ell\}| & \text{if } \ell > LPF[i], \\ |\{j \in [1,i] : LPF[j] < \ell\}| & \text{if } \ell \leq LPF[i]. \end{cases}$$

Counting in the Special Case II

Maintain array $A[1\,..\,n]$ such that during the ith phase:

$$A[\ell] = \begin{cases} |\{j \in [1, i] : LPF[j] \geq \ell\}| & \text{if } \ell > LPF[i], \\ |\{j \in [1, i] : LPF[j] < \ell\}| & \text{if } \ell \leq LPF[i]. \end{cases}$$

In the transition from the ith phase to the $(i + 1)$th phase, $A[\ell]$ remains unchanged for:

Counting in the Special Case II

Maintain array $A[1\,..\,n]$ such that during the ith phase:

$$A[\ell] = \begin{cases} |\{j \in [1,i] : LPF[j] \geq \ell\}| & \text{if } \ell > LPF[i], \\ |\{j \in [1,i] : LPF[j] < \ell\}| & \text{if } \ell \leq LPF[i]. \end{cases}$$

In the transition from the ith phase to the $(i+1)$th phase, $A[\ell]$ remains unchanged for:

- $\ell > \max(LPF[i+1], LPF[i])$, and

Counting in the Special Case II

Maintain array $A[1..n]$ such that during the ith phase:

$$A[\ell] = \begin{cases} |\{j \in [1,i] : LPF[j] \geq \ell\}| & \text{if } \ell > LPF[i], \\ |\{j \in [1,i] : LPF[j] < \ell\}| & \text{if } \ell \leq LPF[i]. \end{cases}$$

In the transition from the ith phase to the $(i+1)$th phase, $A[\ell]$ remains unchanged for:

- $\ell > \max(LPF[i+1], LPF[i])$, and
- $\ell \leq \min(LPF[i+1], LPF[i])$.

Counting in the Special Case II

Maintain array $A[1\,..\,n]$ such that during the ith phase:

$$A[\ell] = \begin{cases} |\{j \in [1,i] : LPF[j] \geq \ell\}| & \text{if } \ell > LPF[i], \\ |\{j \in [1,i] : LPF[j] < \ell\}| & \text{if } \ell \leq LPF[i]. \end{cases}$$

In the transition from the ith phase to the $(i+1)$th phase, $A[\ell]$ remains unchanged for:

- $\ell > \max(LPF[i+1], LPF[i])$, and
- $\ell \leq \min(LPF[i+1], LPF[i])$.

Number of updates to A:

$$|LPF[i+1] - LPF[i]|$$

Counting in the Special Case II

Maintain array $A[1\,..\,n]$ such that during the ith phase:

$$A[\ell] = \begin{cases} |\{j \in [1,i] : LPF[j] \geq \ell\}| & \text{if } \ell > LPF[i], \\ |\{j \in [1,i] : LPF[j] < \ell\}| & \text{if } \ell \leq LPF[i]. \end{cases}$$

In the transition from the ith phase to the $(i+1)$th phase, $A[\ell]$ remains unchanged for:

- $\ell > \max(LPF[i+1], LPF[i])$, and
- $\ell \leq \min(LPF[i+1], LPF[i])$.

Number of updates to A:

$\sum_{i=1}^{n-1} |LPF[i+1] - LPF[i]| = \mathcal{O}(n)$.

Applications

- **Powers.** $(abc)^{8/3} = abcabcab$

Applications

- **Powers.** $(abc)^{8/3} = abcabcab$, $(abc)^{3/2}$ is undefined.

Applications

- **Powers.** $(abc)^{8/3} = abcabcab$, $(abc)^{3/2}$ is undefined.

  Result: All distinct γ-powers in a length-n string can be
  - counted in $\mathcal{O}(\frac{\gamma}{\gamma-1}n)$ time, and
  - reported in $\mathcal{O}(\frac{\gamma}{\gamma-1}n + |\text{output}|)$ time.

Applications

- **Powers.** $(abc)^{8/3} = abcabcab$, $(abc)^{3/2}$ is undefined.

  Result: All distinct γ-powers in a length-n string can be
  - counted in $\mathcal{O}(\frac{\gamma}{\gamma-1}n)$ time, and
  - reported in $\mathcal{O}(\frac{\gamma}{\gamma-1}n + |\text{output}|)$ time.

- **Antipowers.** A *k-antipower* (for an integer $k \geq 2$) is a concatenation of k pairwise distinct strings of the same length [Fici et al., ICALP 2016], e.g. *abbcaaba* is a 4-antipower.

Applications

- **Powers.** $(abc)^{8/3} = abcabcab$, $(abc)^{3/2}$ is undefined.

  Result: All distinct γ-powers in a length-n string can be
  - counted in $\mathcal{O}(\frac{\gamma}{\gamma-1}n)$ time, and
  - reported in $\mathcal{O}(\frac{\gamma}{\gamma-1}n + |\text{output}|)$ time.

- **Antipowers.** A *k-antipower* (for an integer $k \geq 2$) is a concatenation of k pairwise distinct strings of the same length [Fici et al., ICALP 2016], e.g. *abbcaaba* is a 4-antipower.

  Result: All distinct k-antipowers in a length-n string can be
  - counted in $\mathcal{O}(nk^2)$ time, and
  - reported in $\mathcal{O}(nk^2 + |\text{output}|)$ time.

Applications

- **Powers.** $(abc)^{8/3} = abcabcab$, $(abc)^{3/2}$ is undefined.

  Result: All distinct γ-powers in a length-n string can be
  - counted in $\mathcal{O}(\frac{\gamma}{\gamma-1}n)$ time, and
  - reported in $\mathcal{O}(\frac{\gamma}{\gamma-1}n + |\text{output}|)$ time.

- **Antipowers.** A *k-antipower* (for an integer $k \geq 2$) is a concatenation of k pairwise distinct strings of the same length [Fici et al., ICALP 2016], e.g. $abbcaaba$ is a 4-antipower.

  Result: All distinct k-antipowers in a length-n string can be
  - counted in $\mathcal{O}(nk^2)$ time, and
  - reported in $\mathcal{O}(nk^2 + |\text{output}|)$ time.

For counting distinct k-antipowers, we improve over the $\mathcal{O}(nk^4 \log n \log k)$-time algorithm of [Kociumaka et al., arxiv].

The General Case: Synchronisers

Let us assume that S is cube-free, i.e. it has no non-empty factor of the form UUU.

The General Case: Synchronisers

Let us assume that S is cube-free, i.e. it has no non-empty factor of the form UUU.

Theorem [Kempa-Kociumaka, STOC 2019]

For a cube-free string of length n, and an integer $\tau \leq n/2$, we can compute in $\mathcal{O}(n)$ time a set Sync of size $\mathcal{O}(n/\tau)$ such that:

The General Case: Synchronisers

Let us assume that S is cube-free, i.e. it has no non-empty factor of the form UUU.

Theorem [Kempa-Kociumaka, STOC 2019]

For a cube-free string of length n, and an integer $\tau \leq n/2$, we can compute in $\mathcal{O}(n)$ time a set Sync of size $\mathcal{O}(n/\tau)$ such that:

1. If $S[i \mathinner{.\,.} i+2\tau) = S[j \mathinner{.\,.} j+2\tau)$, then $i \in \text{Sync} \Leftrightarrow j \in \text{Sync}$.

The General Case: Synchronisers

Let us assume that S is cube-free, i.e. it has no non-empty factor of the form UUU.

Theorem [Kempa-Kociumaka, STOC 2019]

For a cube-free string of length n, and an integer $\tau \leq n/2$, we can compute in $\mathcal{O}(n)$ time a set Sync of size $\mathcal{O}(n/\tau)$ such that:

1. If $S[i\mathinner{.\,.} i+2\tau) = S[j\mathinner{.\,.} j+2\tau)$, then $i \in \mathrm{Sync} \Leftrightarrow j \in \mathrm{Sync}$.
2. For $i \in [1, n-3\tau+2]$, $\mathrm{Sync} \cap [i, i+\tau) \neq \emptyset$.

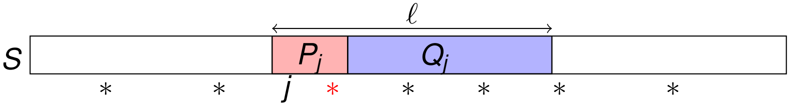
The General Case: Synchronisers

Let us assume that S is cube-free, i.e. it has no non-empty factor of the form UUU.

Theorem [Kempa-Kociumaka, STOC 2019]

For a cube-free string of length n, and an integer $\tau \leq n/2$, we can compute in $\mathcal{O}(n)$ time a set Sync of size $\mathcal{O}(n/\tau)$ such that:

1. If $S[i\mathinner{.\,.} i+2\tau) = S[j\mathinner{.\,.} j+2\tau)$, then $i \in \mathrm{Sync} \Leftrightarrow j \in \mathrm{Sync}$.
2. For $i \in [1, n-3\tau+2]$, $\mathrm{Sync} \cap [i, i+\tau) \neq \emptyset$.



Idea: Assign each factor with $\ell \in [3\tau, 9\tau)$ to its first τ-synchroniser.

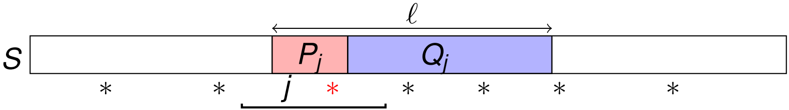
The General Case: Synchronisers

Let us assume that S is cube-free, i.e. it has no non-empty factor of the form UUU.

Theorem [Kempa-Kociumaka, STOC 2019]

For a cube-free string of length n, and an integer $\tau \leq n/2$, we can compute in $\mathcal{O}(n)$ time a set Sync of size $\mathcal{O}(n/\tau)$ such that:

1. If $S[i\mathinner{..}i+2\tau) = S[j\mathinner{..}j+2\tau)$, then $i \in \text{Sync} \Leftrightarrow j \in \text{Sync}$.
2. For $i \in [1, n-3\tau+2]$, $\text{Sync} \cap [i, i+\tau) \neq \emptyset$.



We may have to split packages, ending up with $\mathcal{O}(n/\tau)$ more for each ℓ.

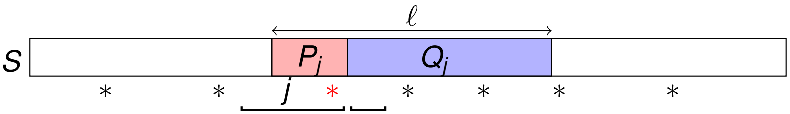
The General Case: Synchronisers

Let us assume that S is cube-free, i.e. it has no non-empty factor of the form UUU.

Theorem [Kempa-Kociumaka, STOC 2019]

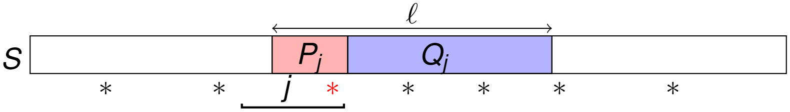
For a cube-free string of length n, and an integer $\tau \leq n/2$, we can compute in $\mathcal{O}(n)$ time a set Sync of size $\mathcal{O}(n/\tau)$ such that:

1. If $S[i\,..\,i+2\tau) = S[j\,..\,j+2\tau)$, then $i \in \text{Sync} \Leftrightarrow j \in \text{Sync}$.
2. For $i \in [1, n-3\tau+2]$, $\text{Sync} \cap [i, i+\tau) \neq \emptyset$.



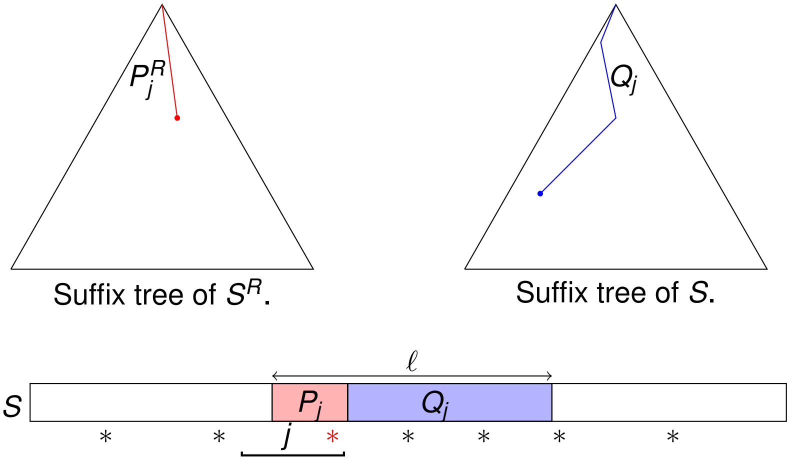
We may have to split packages, ending up with $\mathcal{O}(n/\tau)$ more for each ℓ.

The General Case: Synchronisers



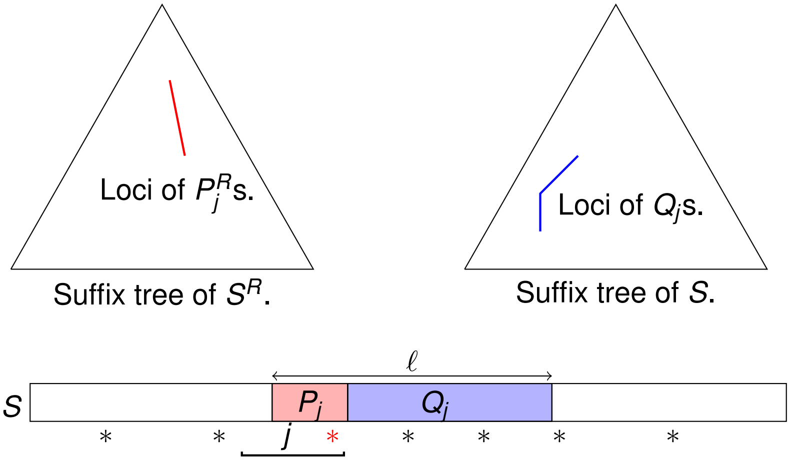
Then, for each package, the loci of the relevant Q_js (resp. P_j^Rs) correspond to a path in the suffix tree of S (resp. S^R).

The General Case: Synchronisers



Then, for each package, the loci of the relevant Q_js (resp. P_j^Rs) correspond to a path in the suffix tree of S (resp. S^R).

The General Case: Synchronisers



Then, for each package, the loci of the relevant Q_js (resp. P_j^Rs) correspond to a path in the suffix tree of S (resp. S^R).

The General Case: A Problem on Trees

Input: Two compact trees $\mathcal{T}$ and $\mathcal{T}'$ of total size N, and a set Π of pairs (π, π') of equal-length paths, with π going downwards in $\mathcal{T}$ and π' going upwards in $\mathcal{T}'$.

The General Case: A Problem on Trees

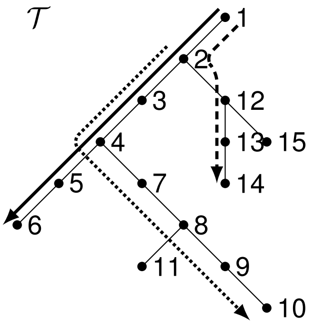
Input: Two compact trees $\mathcal{T}$ and $\mathcal{T}'$ of total size N, and a set Π of pairs (π, π') of equal-length paths, with π going downwards in $\mathcal{T}$ and π' going upwards in $\mathcal{T}'$.

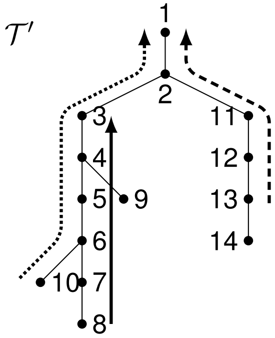
Output: $|\bigcup_{(\pi,\pi')\in\Pi} \text{Induced}(\pi, \pi')|$, where $\text{Induced}(\pi, \pi')$ is the set of pairs of (explicit or implicit) nodes (u, u') such that, for some i, u is the ith node on π and u' is the ith node on π'.

The General Case: A Problem on Trees

Input: Two compact trees $\mathcal{T}$ and $\mathcal{T}'$ of total size N, and a set Π of pairs (π, π') of equal-length paths, with π going downwards in $\mathcal{T}$ and π' going upwards in $\mathcal{T}'$.

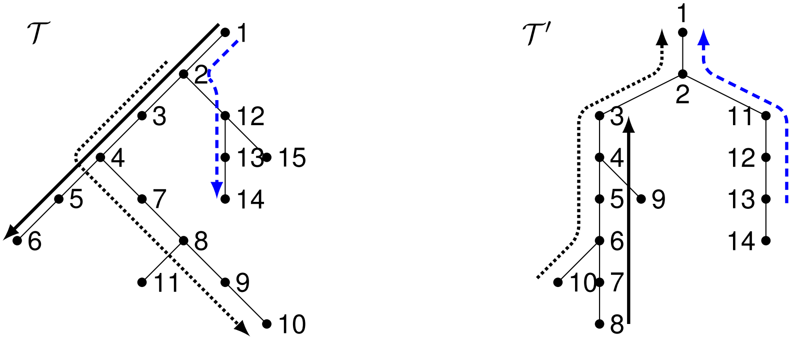
Output: $|\bigcup_{(\pi,\pi')\in\Pi} \text{Induced}(\pi, \pi')|$, where $\text{Induced}(\pi, \pi')$ is the set of pairs of (explicit or implicit) nodes (u, u') such that, for some i, u is the ith node on π and u' is the ith node on π'.





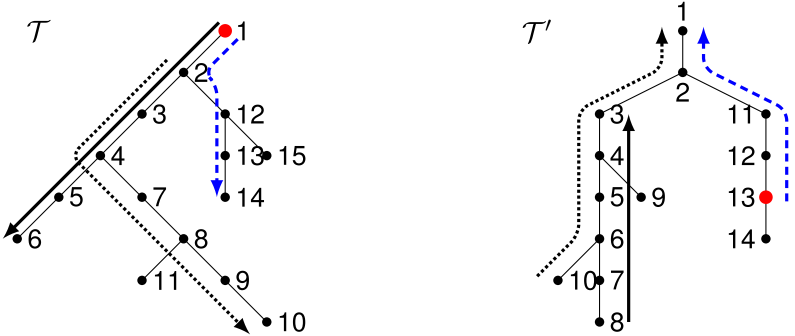
Input: Two compact trees $\mathcal{T}$ and $\mathcal{T}'$ of total size N, and a set Π of pairs (π, π') of equal-length paths, with π going downwards in $\mathcal{T}$ and π' going upwards in $\mathcal{T}'$.

Output: $|\bigcup_{(\pi,\pi')\in\Pi} \text{Induced}(\pi,\pi')|$, where $\text{Induced}(\pi,\pi')$ is the set of pairs of (explicit or implicit) nodes (u,u') such that, for some i, u is the ith node on π and u' is the ith node on π'.



Input: Two compact trees $\mathcal{T}$ and $\mathcal{T}'$ of total size N, and a set Π of pairs (π, π') of equal-length paths, with π going downwards in $\mathcal{T}$ and π' going upwards in $\mathcal{T}'$.

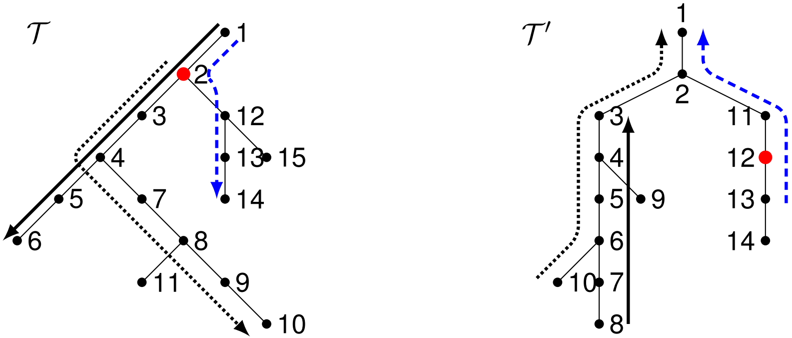
Output: $|\bigcup_{(\pi,\pi')\in\Pi} \text{Induced}(\pi, \pi')|$, where $\text{Induced}(\pi, \pi')$ is the set of pairs of (explicit or implicit) nodes (u, u') such that, for some i, u is the ith node on π and u' is the ith node on π'.



The General Case: A Problem on Trees

Input: Two compact trees $\mathcal{T}$ and $\mathcal{T}'$ of total size N, and a set Π of pairs (π, π') of equal-length paths, with π going downwards in $\mathcal{T}$ and π' going upwards in $\mathcal{T}'$.

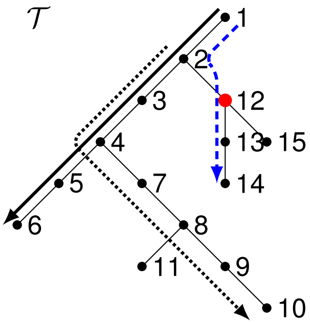
Output: $|\bigcup_{(\pi,\pi')\in\Pi} \text{Induced}(\pi, \pi')|$, where $\text{Induced}(\pi, \pi')$ is the set of pairs of (explicit or implicit) nodes (u, u') such that, for some i, u is the ith node on π and u' is the ith node on π'.

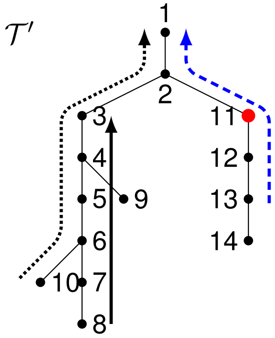


The General Case: A Problem on Trees

Input: Two compact trees $\mathcal{T}$ and $\mathcal{T}'$ of total size N, and a set Π of pairs (π, π') of equal-length paths, with π going downwards in $\mathcal{T}$ and π' going upwards in $\mathcal{T}'$.

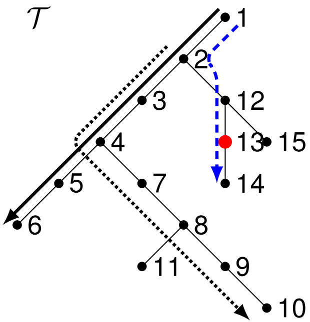
Output: $|\bigcup_{(\pi,\pi')\in\Pi} \text{Induced}(\pi, \pi')|$, where $\text{Induced}(\pi, \pi')$ is the set of pairs of (explicit or implicit) nodes (u, u') such that, for some i, u is the ith node on π and u' is the ith node on π'.

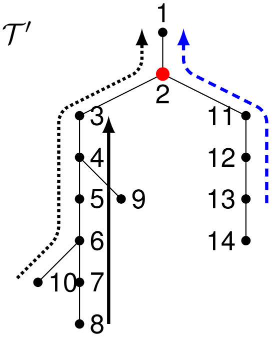




Input: Two compact trees $\mathcal{T}$ and $\mathcal{T}'$ of total size N, and a set Π of pairs (π, π') of equal-length paths, with π going downwards in $\mathcal{T}$ and π' going upwards in $\mathcal{T}'$.

Output: $|\bigcup_{(\pi,\pi')\in\Pi} \text{Induced}(\pi, \pi')|$, where $\text{Induced}(\pi, \pi')$ is the set of pairs of (explicit or implicit) nodes (u, u') such that, for some i, u is the ith node on π and u' is the ith node on π'.

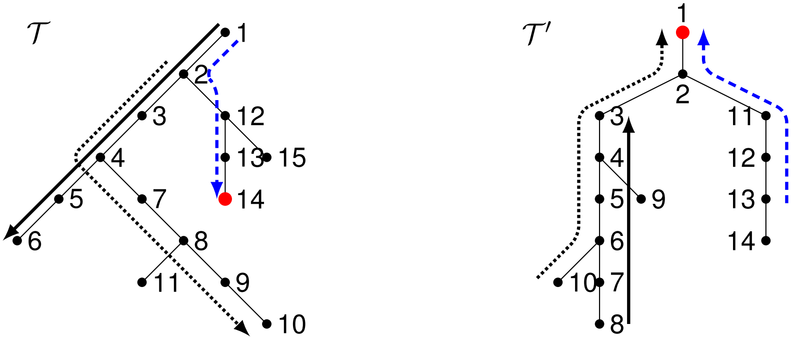




The General Case: A Problem on Trees

Input: Two compact trees $\mathcal{T}$ and $\mathcal{T}'$ of total size N, and a set Π of pairs (π, π') of equal-length paths, with π going downwards in $\mathcal{T}$ and π' going upwards in $\mathcal{T}'$.

Output: $|\bigcup_{(\pi,\pi')\in\Pi} \text{Induced}(\pi, \pi')|$, where $\text{Induced}(\pi, \pi')$ is the set of pairs of (explicit or implicit) nodes (u, u') such that, for some i, u is the ith node on π and u' is the ith node on π'.



The General Case: Wrap-up for Cube-Free Strings

Using a heavy paths decomposition of each tree, this problem can be solved in time $\mathcal{O}(N + |\Pi| \log N)$ [Kociumaka et al., arxiv].

The General Case: Wrap-up for Cube-Free Strings

Using a heavy paths decomposition of each tree, this problem can be solved in time $\mathcal{O}(N + |\Pi| \log N)$ [Kociumaka et al., arxiv].

Here, $N = \mathcal{O}(n)$.

The General Case: Wrap-up for Cube-Free Strings

Using a heavy paths decomposition of each tree, this problem can be solved in time $\mathcal{O}(N + |\Pi| \log N)$ [Kociumaka et al., arxiv].

Here, $N = \mathcal{O}(n)$.

Let us denote the number of packages representing factors of length ℓ by m_ℓ.

The General Case: Wrap-up for Cube-Free Strings

Using a heavy paths decomposition of each tree, this problem can be solved in time $\mathcal{O}(N + |\Pi| \log N)$ [Kociumaka et al., arxiv].

Here, $N = \mathcal{O}(n)$.

Let us denote the number of packages representing factors of length ℓ by m_ℓ. For each $\tau = 3^x$, for $x \in [1, \log_3 n)$, we have

$$\mathcal{O}\left(\sum_{\ell=3\tau}^{9\tau-1}\left(m_\ell + \frac{n}{\tau}\right)\right) \qquad \text{paths.}$$

The General Case: Wrap-up for Cube-Free Strings

Using a heavy paths decomposition of each tree, this problem can be solved in time $\mathcal{O}(N + |\Pi| \log N)$ [Kociumaka et al., arxiv].

Here, $N = \mathcal{O}(n)$.

Let us denote the number of packages representing factors of length ℓ by m_ℓ. For each $\tau = 3^x$, for $x \in [1, \log_3 n)$, we have

$$\mathcal{O}\left(\sum_{\ell=3\tau}^{9\tau-1}\left(m_\ell + \frac{n}{\tau}\right)\right) = \mathcal{O}\left(n + \sum_{\ell=3\tau}^{9\tau-1} m_\ell\right) \text{ paths.}$$

The General Case: Wrap-up for Cube-Free Strings

Using a heavy paths decomposition of each tree, this problem can be solved in time $\mathcal{O}(N + |\Pi| \log N)$ [Kociumaka et al., arxiv].

Here, $N = \mathcal{O}(n)$.

Let us denote the number of packages representing factors of length ℓ by m_ℓ. For each $\tau = 3^x$, for $x \in [1, \log_3 n)$, we have

$$\mathcal{O}\left(\sum_{\ell=3\tau}^{9\tau-1}\left(m_\ell + \frac{n}{\tau}\right)\right) = \mathcal{O}\left(n + \sum_{\ell=3\tau}^{9\tau-1} m_\ell\right) \text{ paths.}$$

Hence, $|\Pi| = \mathcal{O}(n \log n + m)$.

The General Case: Wrap-up for Cube-Free Strings

Using a heavy paths decomposition of each tree, this problem can be solved in time $\mathcal{O}(N + |\Pi| \log N)$ [Kociumaka et al., arxiv].

Here, $N = \mathcal{O}(n)$.

Let us denote the number of packages representing factors of length ℓ by m_ℓ. For each $\tau = 3^x$, for $x \in [1, \log_3 n)$, we have

$$\mathcal{O}\left(\sum_{\ell=3\tau}^{9\tau-1}\left(m_\ell + \frac{n}{\tau}\right)\right) = \mathcal{O}\left(n + \sum_{\ell=3\tau}^{9\tau-1} m_\ell\right) \text{ paths.}$$

Hence, $|\Pi| = \mathcal{O}(n \log n + m)$.

Overall, we solve the counting version of the problem in time $\mathcal{O}(n \log^2 n + m \log n)$.

The General Case: Periodicity

We replace $\mathcal{F}$ by two sets of packages:

The General Case: Periodicity

We replace $\mathcal{F}$ by two sets of packages:

- $\mathcal{F}_p$ representing highly-periodic factors, and

The General Case: Periodicity

We replace $\mathcal{F}$ by two sets of packages:

- $\mathcal{F}_p$ representing highly-periodic factors, and
- $\mathcal{F}_a$ representing non-highly-periodic factors.

The General Case: Periodicity

We replace $\mathcal{F}$ by two sets of packages:

- $\mathcal{F}_p$ representing highly-periodic factors, and
- $\mathcal{F}_a$ representing non-highly-periodic factors.

The solution using synchronisers works for $\mathcal{F}_a$.

The General Case: Periodicity

We replace $\mathcal{F}$ by two sets of packages:

- $\mathcal{F}_p$ representing highly-periodic factors, and
- $\mathcal{F}_a$ representing non-highly-periodic factors.

The solution using synchronisers works for $\mathcal{F}_a$.

For highly-periodic factors, we reduce the problem to the same problem on trees using runs and Lyndon roots.

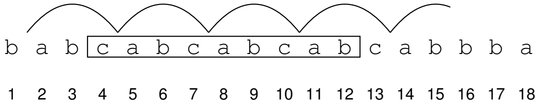
The General Case: Periodicity

We replace $\mathcal{F}$ by two sets of packages:

- $\mathcal{F}_p$ representing highly-periodic factors, and
- $\mathcal{F}_a$ representing non-highly-periodic factors.

The solution using synchronisers works for $\mathcal{F}_a$.

For highly-periodic factors, we reduce the problem to the same problem on trees using runs and Lyndon roots.



run $(S[2\,..\,15], 3)$

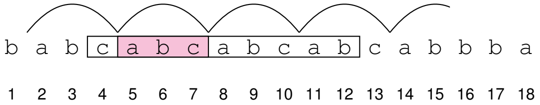
The General Case: Periodicity

We replace $\mathcal{F}$ by two sets of packages:

- $\mathcal{F}_p$ representing highly-periodic factors, and
- $\mathcal{F}_a$ representing non-highly-periodic factors.

The solution using synchronisers works for $\mathcal{F}_a$.

For highly-periodic factors, we reduce the problem to the same problem on trees using runs and Lyndon roots.



run $(S[2\,..\,15], 3)$

We replace $\mathcal{F}$ by two sets of packages:

- $\mathcal{F}_p$ representing highly-periodic factors, and
- $\mathcal{F}_a$ representing non-highly-periodic factors.

The solution using synchronisers works for $\mathcal{F}_a$.

For highly-periodic factors, we reduce the problem to the same problem on trees using runs and Lyndon roots.

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

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

run $(S[2\,..\,15], 3)$

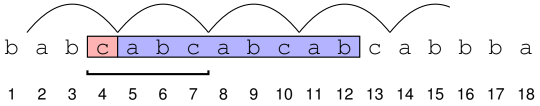
The General Case: Periodicity

We replace $\mathcal{F}$ by two sets of packages:

- $\mathcal{F}_p$ representing highly-periodic factors, and
- $\mathcal{F}_a$ representing non-highly-periodic factors.

The solution using synchronisers works for $\mathcal{F}_a$.

For highly-periodic factors, we reduce the problem to the same problem on trees using runs and Lyndon roots.



run $(S[2\,..\,15], 3)$

We ensure that each package represents factors with the same period.

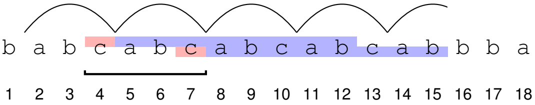
The General Case: Periodicity

We replace $\mathcal{F}$ by two sets of packages:

- $\mathcal{F}_p$ representing highly-periodic factors, and
- $\mathcal{F}_a$ representing non-highly-periodic factors.

The solution using synchronisers works for $\mathcal{F}_a$.

For highly-periodic factors, we reduce the problem to the same problem on trees using runs and Lyndon roots.



run $(S[2\,..\,15], 3)$

We ensure that each package represents factors with the same period.

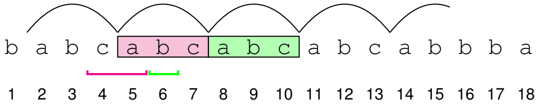
The General Case: Periodicity

We replace $\mathcal{F}$ by two sets of packages:

- $\mathcal{F}_p$ representing highly-periodic factors, and
- $\mathcal{F}_a$ representing non-highly-periodic factors.

The solution using synchronisers works for $\mathcal{F}_a$.

For highly-periodic factors, we reduce the problem to the same problem on trees using runs and Lyndon roots.



run $(S[2\,..\,15], 3)$

We ensure that each package represents factors with the same period. Each package yields at most two pairs of paths.

Our Results

For a special package representation $\mathcal{F}$ consisting of m packages and a string of length n we can compute:

Our Results

For a special package representation $\mathcal{F}$ consisting of m packages and a string of length n we can compute:

- Factors($\mathcal{F}$) in $\mathcal{O}(n + m + |\text{output}|)$ time,
- $|$Factors($\mathcal{F}$)$|$ in $\mathcal{O}(n + m)$ time.

Our Results

For a special package representation $\mathcal{F}$ consisting of m packages and a string of length n we can compute:

- Factors($\mathcal{F}$) in $\mathcal{O}(n + m + |\text{output}|)$ time,
- $|$Factors($\mathcal{F}$)$|$ in $\mathcal{O}(n + m)$ time.

For a general package representation $\mathcal{F}$ consisting of m packages and a string of length n we can compute:

Our Results

For a special package representation $\mathcal{F}$ consisting of m packages and a string of length n we can compute:

- Factors($\mathcal{F}$) in $\mathcal{O}(n + m + |\text{output}|)$ time,
- $|$Factors($\mathcal{F}$)$|$ in $\mathcal{O}(n + m)$ time.

For a general package representation $\mathcal{F}$ consisting of m packages and a string of length n we can compute:

- Factors($\mathcal{F}$) in $\mathcal{O}(n \log^2 n + m \log n + |\text{output}|)$ time,
- $|$Factors($\mathcal{F}$)$|$ in $\mathcal{O}(n \log^2 n + m \log n)$ time.

Thank you for your attention!

Questions?