

ALGORITHMIC RESEARCH: COOPERATION AROUND ORESOUND

# Time-Space Trade-Offs for Longest Common Extensions

(To appear at CPM 2012)

Philip Bille<sup>1</sup>, Inge Li Gørtz<sup>1</sup>, Benjamin Sach<sup>2</sup>, and Hjalte Wedel Vildhøj<sup>1</sup>

<sup>1</sup>Technical University of Denmark, DTU Informatics, {phbi, ilg, hwvi}@imm.dtu.dk

<sup>2</sup>University of Warwick, Department of Computer Science, sach@dcs.warwick.ac.uk

University of Copenhagen, April 17, 2012

# The Longest Common Extension Problem

## Definition

**Problem:** Preprocess a string  $T$  of length  $n$  to support LCE queries:

- ▶  $\text{LCE}(i,j)$  = The length of the longest common prefix of the suffixes starting at position  $i$  and  $j$  in  $T$ .

## Example

$T$  =    <sup>1</sup> b   <sup>2</sup> a   <sup>3</sup> n   <sup>4</sup> a   <sup>5</sup> n   <sup>6</sup> a   <sup>7</sup> s

$\text{LCE}(2,4) = ?$

# The Longest Common Extension Problem

## Definition

**Problem:** Preprocess a string  $T$  of length  $n$  to support LCE queries:

- ▶  $\text{LCE}(i, j) =$  The length of the longest common prefix of the suffixes starting at position  $i$  and  $j$  in  $T$ .

## Example

$T =$     <sup>1</sup> <sup>2</sup> <sup>3</sup> <sup>4</sup> <sup>5</sup> <sup>6</sup> <sup>7</sup>  
  b a n a n a s

                  ↑                    ↑  
                  a n a s

                  a n a n a s

LCE(2, 4) = ?

# The Longest Common Extension Problem

## Definition

**Problem:** Preprocess a string  $T$  of length  $n$  to support LCE queries:

- ▶  $\text{LCE}(i,j)$  = The length of the longest common prefix of the suffixes starting at position  $i$  and  $j$  in  $T$ .

## Example

$T =$

	1	2	3	4	5	6	7	
	b	a	n	a	n	a	s	
		↑		↑				
				a	n	a	s	
			a	n	a	s		

$\text{LCE}(2,4) = 3$

# The Longest Common Extension Problem

## Definition

**Problem:** Preprocess a string  $T$  of length  $n$  to support LCE queries:

- ▶  $LCE(i, j) =$  The length of the longest common prefix of the suffixes starting at position  $i$  and  $j$  in  $T$ .

## Example

$T =$

1	2	3	4	5	6	7
b	a	n	a	n	a	s

↑

a	n	a	n	a	s
---	---	---	---	---	---

$LCE(2, 5) = 0$

# The Longest Common Extension Problem

## Motivation

Longest Common Extensions appear as a subproblem in many string matching problems, including

- ▶ Approximate string matching. I.e., find substrings of  $T$  such that  $T[i \dots j] \approx P$  (hamming or edit distance).
- ▶ Finding palindromes. I.e., find substrings of  $T$  such that  $T[i \dots j] = T[i \dots j]^R$ .
- ▶ Finding tandem repeats. I.e., find substrings of  $T$  such that  $T[i \dots j] = UU$  for some string  $U$ .

## Example: Palindromes

$T =$       1 2 3 4 5 6 7 8 9  
          b a b a a b a a c

# The Longest Common Extension Problem

## Motivation

Longest Common Extensions appear as a subproblem in many string matching problems, including

- ▶ Approximate string matching. I.e., find substrings of  $T$  such that  $T[i \dots j] \approx P$  (hamming or edit distance).
- ▶ Finding palindromes. I.e., find substrings of  $T$  such that  $T[i \dots j] = T[i \dots j]^R$ .
- ▶ Finding tandem repeats. I.e., find substrings of  $T$  such that  $T[i \dots j] = UU$  for some string  $U$ .

## Example: Palindromes

	1	2	3	4	5	6	7	8	9	
$T$	=	b	a	b	a	a	b	a	a	c

↑  
center

# The Longest Common Extension Problem

## Motivation

Longest Common Extensions appear as a subproblem in many string matching problems, including

- ▶ Approximate string matching. I.e., find substrings of  $T$  such that  $T[i \dots j] \approx P$  (hamming or edit distance).
- ▶ Finding palindromes. I.e., find substrings of  $T$  such that  $T[i \dots j] = T[i \dots j]^R$ .
- ▶ Finding tandem repeats. I.e., find substrings of  $T$  such that  $T[i \dots j] = UU$  for some string  $U$ .

## Example: Palindromes

$T =$ 

1	2	3	4	5	6	7	8	9
b	a	b	a	a	b	a	a	c

↑  
center



# The Longest Common Extension Problem

## Motivation

Longest Common Extensions appear as a subproblem in many string matching problems, including

- ▶ Approximate string matching. I.e., find substrings of  $T$  such that  $T[i \dots j] \approx P$  (hamming or edit distance).
- ▶ Finding palindromes. I.e., find substrings of  $T$  such that  $T[i \dots j] = T[i \dots j]^R$ .
- ▶ Finding tandem repeats. I.e., find substrings of  $T$  such that  $T[i \dots j] = UU$  for some string  $U$ .

## Example: Palindromes

$T =$       1   2   3   4   5   6   7   8   9  
          b   a   b   a   a   b   a   a   c  
                  ←———|———→  
                  ↑  
                  center

All maximal palindromes in  $P$  can be reported by performing  $2n - 1$  LCE queries (one for each possible center).

# Two Simple Solutions

#1: Store nothing

$T =$     <sup>1</sup> <sup>2</sup> <sup>3</sup> <sup>4</sup> <sup>5</sup> <sup>6</sup> <sup>7</sup>  
          **b a n a n a s**  
                  ↑       ↑  
                  *i*       *j*

$LCE(i,j) =$

# Two Simple Solutions

#1: Store nothing

$T =$     <sup>1</sup> <sup>2</sup> <sup>3</sup> <sup>4</sup> <sup>5</sup> <sup>6</sup> <sup>7</sup>  
          **b** **a** **n** **a** **n** **a** **s**  
                  ↑          ↑  
                  *i*          *j*

$$\text{LCE}(i,j) = 1$$

# Two Simple Solutions

#1: Store nothing

$T$  =    <sup>1</sup> b   <sup>2</sup> a   <sup>3</sup> n   <sup>4</sup> a   <sup>5</sup> n   <sup>6</sup> a   <sup>7</sup> s  
                  ↑            ↑  
                  *i*            *j*

$$\text{LCE}(i,j) = 2$$

# Two Simple Solutions

#1: Store nothing

$T =$     <sup>1</sup> <sup>2</sup> <sup>3</sup> <sup>4</sup> <sup>5</sup> <sup>6</sup> <sup>7</sup>  
          **b** **a** **n** **a** **n** **a** **s**  
                  ↑          ↑  
                  *i*          *j*

$$\text{LCE}(i,j) = 3$$

# Two Simple Solutions

#1: Store nothing

$T =$     <sup>1</sup> b   <sup>2</sup> a   <sup>3</sup> n   <sup>4</sup> a   <sup>5</sup> n   <sup>6</sup> a   <sup>7</sup> s  
                  ↑            ↑  
                  *i*            *j*

$$\text{LCE}(i,j) = 3$$

Time:  $O(n)$

Space:  $O(1)$

# Two Simple Solutions

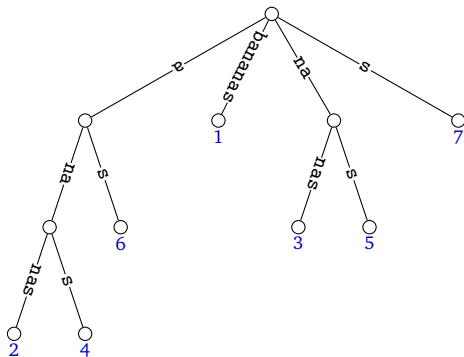
#1: Store nothing

$T =$     <sup>1</sup> b   <sup>2</sup> a   <sup>3</sup> n   <sup>4</sup> a   <sup>5</sup> n   <sup>6</sup> a   <sup>7</sup> s  
                  ↑            ↑  
                  *i*            *j*

$$\text{LCE}(i, j) = 3$$

Time:  $O(n)$   
Space:  $O(1)$

#2: Store the suffix tree



# Two Simple Solutions

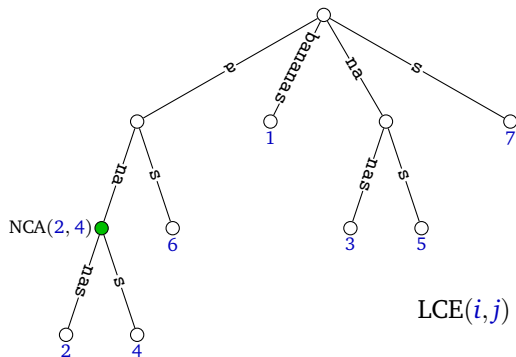
#1: Store nothing

$T =$     <sup>1</sup> b   <sup>2</sup> a   <sup>3</sup> n   <sup>4</sup> a   <sup>5</sup> n   <sup>6</sup> a   <sup>7</sup> s  
                  ↑            ↑  
                  *i*            *j*

$$\text{LCE}(i,j) = 3$$

Time:  $O(n)$   
Space:  $O(1)$

#2: Store the suffix tree



$$\text{LCE}(i,j) = |\text{NCA}(i,j)| = 3$$



# Two Simple Solutions

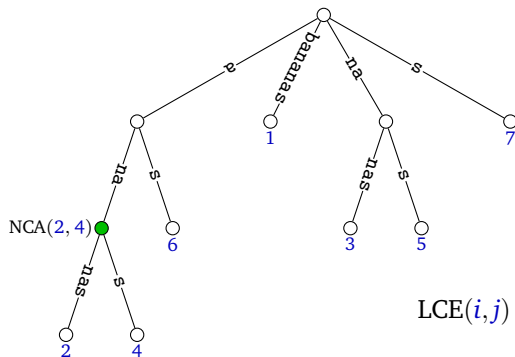
#1: Store nothing

$T =$     <sup>1</sup> b   <sup>2</sup> a   <sup>3</sup> n   <sup>4</sup> a   <sup>5</sup> n   <sup>6</sup> a   <sup>7</sup> s  
                  ↑            ↑  
                  *i*            *j*

$$\text{LCE}(i,j) = 3$$

Time:  $O(n)$   
Space:  $O(1)$

#2: Store the suffix tree



Time:  $O(1)$   
Space:  $O(n)$

$$\text{LCE}(i,j) = |\text{NCA}(i,j)| = 3$$

# Two Simple Solutions

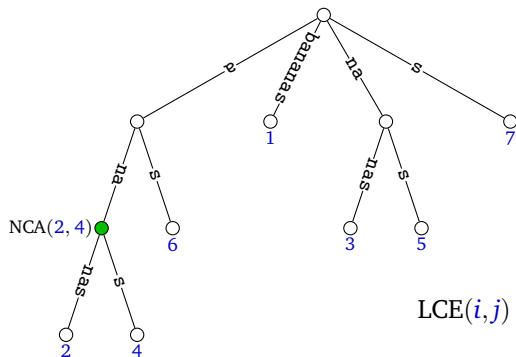
#1: Store nothing

$T =$     <sup>1</sup> b   <sup>2</sup> a   <sup>3</sup> n   <sup>4</sup> a   <sup>5</sup> n   <sup>6</sup> a   <sup>7</sup> s  
                  ↑            ↑  
                  *i*            *j*

$$\text{LCE}(i,j) = 3$$

Time:  $O(n)$   
Space:  $O(1)$

#2: Store the suffix tree



$$\text{LCE}(i,j) = |\text{NCA}(i,j)| = 3$$

Trade-off?

Time:  $O(1)$   
Space:  $O(n)$

# Our Results

Store nothing

Time:  $O(n)$   
Space:  $O(1)$

Trade-off?

Time:  $O(1)$   
Space:  $O(n)$

Store suffix tree



# Our Results

Trade-off parameter  $\tau$ ,  $1 \leq \tau \leq n$

Store nothing

Time:  $O(n)$   
Space:  $O(1)$

Trade-off?

Time:  $O(1)$   
Space:  $O(n)$

Store suffix tree

Randomized

Time:  $O\left(\tau \log\left(\frac{\text{LCE}(i,j)}{\tau}\right)\right)$   
Space:  $O\left(\frac{n}{\tau}\right)$

Time:  $O(\tau)$   
Space:  $O\left(\frac{n}{\sqrt{\tau}}\right)$

Deterministic

Less space

Faster

## A Deterministic Solution

**Idea:** Store a subset of the  $n$  suffixes in a compacted trie.

$T =$     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16  
d b c a a b c a b c a a b c a c  
          •       •       •       •       •       •

# A Deterministic Solution

**Idea:** Store a subset of the  $n$  suffixes in a compacted trie.

$T =$     1   2   3   4   5   6   7   8   9   10   11   12   13   14   15   16  
d   b   c   a   a   b   c   a   b   c   a   a   b   c   a   c  
          •          •      •          •          •  
                          ↑                          ↑  
                          *i*                          *j*

# A Deterministic Solution

**Idea:** Store a subset of the  $n$  suffixes in a compacted trie.

$T =$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	d	b	c	a	a	b	c	a	b	c	a	a	b	c	a	c
				•		•	•		•			•	•			
		↑							↑							
		$i$							$j$							

# A Deterministic Solution

**Idea:** Store a subset of the  $n$  suffixes in a compacted trie.

$T =$     1   2   3   4   5   6   7   8   9   10   11   12   13   14   15   16  
d   b   c   a   a   b   c   a   b   c   a   a   b   c   a   c  
          •          •          •          •          •  
          ↑                          ↑  
          *i*                          *j*



# A Deterministic Solution

**Idea:** Store a subset of the  $n$  suffixes in a compacted trie.

$T =$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	d	b	c	a	a	b	c	a	b	c	a	a	b	c	a	c
				•		•	•		•			•	•			
		↑							↑							
		$i$							$j$							

# A Deterministic Solution

**Idea:** Store a subset of the  $n$  suffixes in a compacted trie.

$T =$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	d	b	c	a	a	b	c	a	b	c	a	a	b	c	a	c
				•		•	•		•			•	•			
		↑							↑							
		$i$							$j$							

# A Deterministic Solution

**Idea:** Store a subset of the  $n$  suffixes in a compacted trie.

$T =$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	d	b	c	a	a	b	c	a	b	c	a	a	b	c	a	c
				•		•	•		•			•	•			
		↑							↑							
		$i$							$j$							

# A Deterministic Solution

**Idea:** Store a subset of the  $n$  suffixes in a compacted trie.

$T =$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	d	b	c	a	a	b	c	a	b	c	a	a	b	c	a	c
				•		•	•		•			•	•			
		↑							↑							
		$i$							$j$							

# A Deterministic Solution

**Idea:** Store a subset of the  $n$  suffixes in a compacted trie.

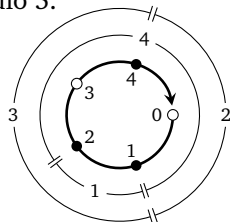
$T =$       1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16  
d b c a a b c a b c a a b c a c

## Difference Covers

A *difference cover modulo  $\tau$*  is a set of integers  $D \subseteq \{0, 1, \dots, \tau - 1\}$  such that for any distance  $d \in \{0, 1, \dots, \tau - 1\}$ ,  $D$  contains two elements separated by distance  $d$  modulo  $\tau$ .

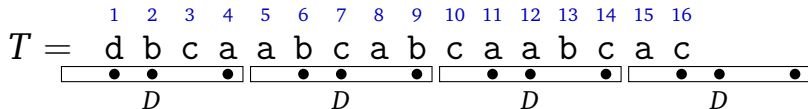
Ex: The set  $D = \{1, 2, 4\}$  is a difference cover modulo 5.

$d$	0	1	2	3	4
$i, j$	1, 1	2, 1	1, 4	4, 1	1, 2



## A Deterministic Solution

**Idea:** Store a subset of the  $n$  suffixes in a compacted trie.

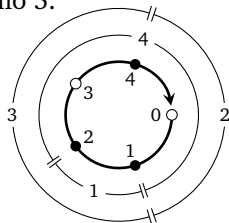


## Difference Covers

A *difference cover modulo  $\tau$*  is a set of integers  $D \subseteq \{0, 1, \dots, \tau - 1\}$  such that for any distance  $d \in \{0, 1, \dots, \tau - 1\}$ ,  $D$  contains two elements separated by distance  $d$  modulo  $\tau$ .

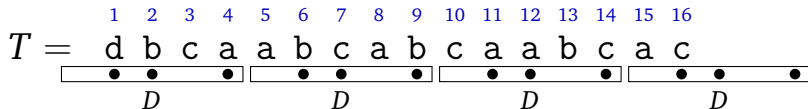
Ex: The set  $D = \{1, 2, 4\}$  is a difference cover modulo 5.

$d$	0	1	2	3	4
$i, j$	1, 1	2, 1	1, 4	4, 1	1, 2



# A Deterministic Solution

**Idea:** Store a subset of the  $n$  suffixes in a compacted trie.



## Lemma (Colbourn and Ling<sup>1</sup>)

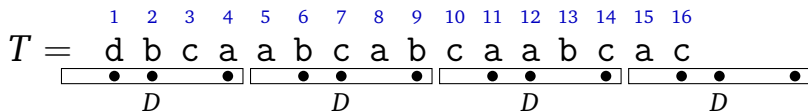
For any  $\tau$ , a difference cover modulo  $\tau$  of size at most  $\sqrt{1.5\tau} + 6$  can be computed in  $O(\sqrt{\tau})$  time.

---

<sup>1</sup>C. J. Colbourn and A. C. Ling. Quorums from difference covers. Inf. Process. Lett. 75(1-2):9-12, 2000

# A Deterministic Solution

**Idea:** Store a subset of the  $n$  suffixes in a compacted trie.



## Lemma (Colbourn and Ling<sup>1</sup>)

For any  $\tau$ , a difference cover modulo  $\tau$  of size at most  $\sqrt{1.5\tau} + 6$  can be computed in  $O(\sqrt{\tau})$  time.

## Analysis

**Time:**  $O(\tau)$

**Space:**  $O(\#\text{stored suffixes}) = O\left(\frac{n}{\tau}|D|\right) = O\left(\frac{n}{\sqrt{\tau}}\right)$

---

<sup>1</sup>C. J. Colbourn and A. C. Ling. Quorums from difference covers. Inf. Process. Lett. 75(1-2):9-12, 2000



# A Randomized Solution

## Rabin-Karp Fingerprints

Let  $p$  be a sufficiently large prime and choose  $b \in \mathbb{Z}_p$  uniformly at random.

$$\phi(S) = \sum_{k=1}^{|S|} S[k]b^k \pmod{p}.$$

$$\begin{array}{rcccccccccccccccc} T & = & \overset{1}{d} & \overset{2}{b} & \overset{3}{c} & \overset{4}{a} & \overset{5}{a} & \overset{6}{b} & \overset{7}{c} & \overset{8}{a} & \overset{9}{b} & \overset{10}{c} & \overset{11}{a} & \overset{12}{a} & \overset{13}{b} & \overset{14}{c} & \overset{15}{a} & \overset{16}{c} \\ & = & 3 & 1 & 2 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 0 & 1 & 2 & 0 & 2 \end{array}$$

$\phi(T[2 \dots 7]) = 120012 \pmod{31} = 11$

**Crucial property:** With high probability  $\phi$  is collision-free on substrings of  $T$ , i.e.,  $\phi(S_1) = \phi(S_2)$  iff  $S_1 = S_2$ .

**Also important:**  $\phi(T[i \dots j + 1])$  can be computed from  $\phi(T[i \dots j])$  in  $O(1)$  time.

# A Randomized Solution

How to answer a query

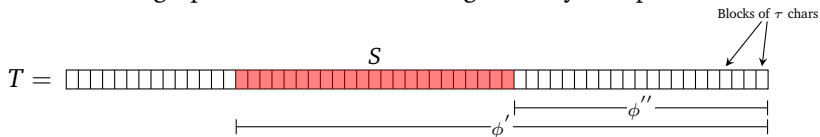
**Idea:** Store fingerprints of suffixes starting at every  $\tau$ 'th position in  $T$ .



# A Randomized Solution

How to answer a query

**Idea:** Store fingerprints of suffixes starting at every  $\tau$ 'th position in  $T$ .

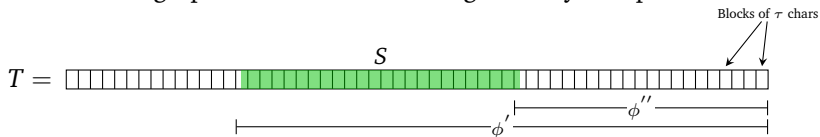


**Observation:** If  $S$  is block aligned we can compute  $\phi(S)$  in  $O(1)$  time. Otherwise, the time needed is  $O(\tau)$ .

# A Randomized Solution

How to answer a query

**Idea:** Store fingerprints of suffixes starting at every  $\tau$ 'th position in  $T$ .



**Observation:** If  $S$  is block aligned we can compute  $\phi(S)$  in  $O(1)$  time. Otherwise, the time needed is  $O(\tau)$ .

# A Randomized Solution

How to answer a query

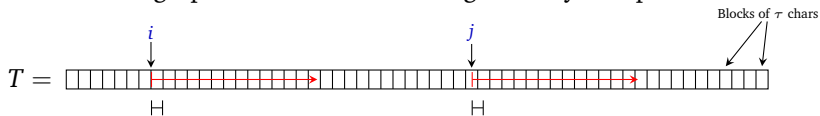
**Idea:** Store fingerprints of suffixes starting at every  $\tau$ 'th position in  $T$ .



# A Randomized Solution

How to answer a query

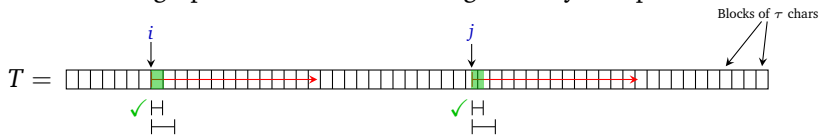
**Idea:** Store fingerprints of suffixes starting at every  $\tau$ 'th position in  $T$ .



# A Randomized Solution

How to answer a query

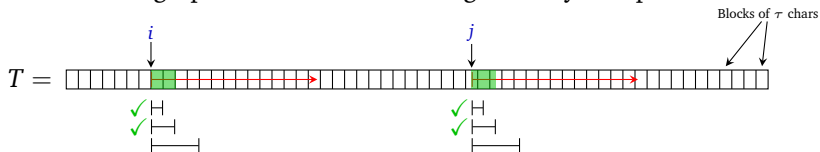
**Idea:** Store fingerprints of suffixes starting at every  $\tau$ 'th position in  $T$ .



# A Randomized Solution

How to answer a query

**Idea:** Store fingerprints of suffixes starting at every  $\tau$ 'th position in  $T$ .

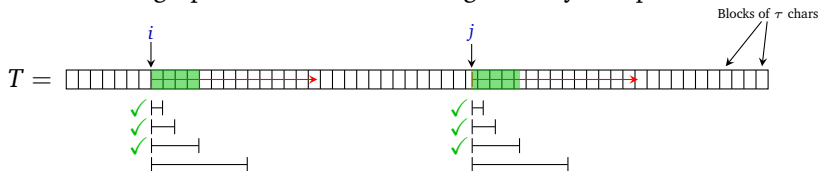




# A Randomized Solution

How to answer a query

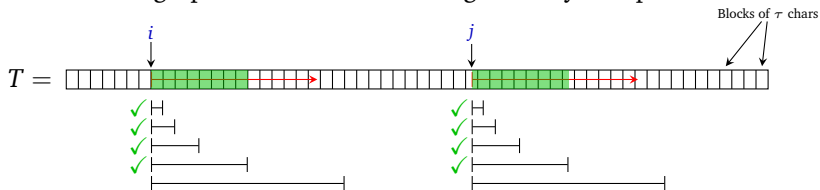
**Idea:** Store fingerprints of suffixes starting at every  $\tau$ 'th position in  $T$ .



# A Randomized Solution

How to answer a query

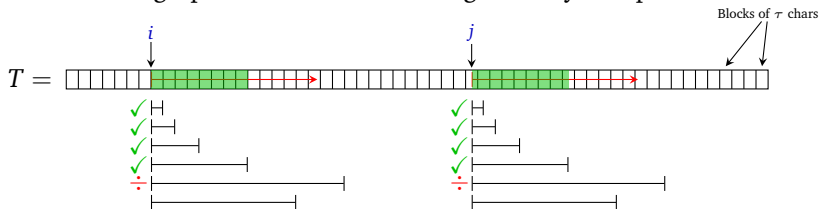
**Idea:** Store fingerprints of suffixes starting at every  $\tau$ 'th position in  $T$ .



# A Randomized Solution

How to answer a query

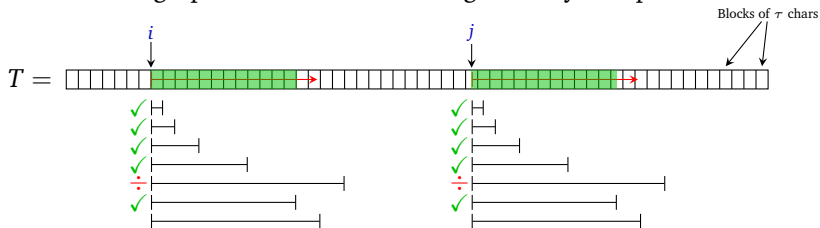
**Idea:** Store fingerprints of suffixes starting at every  $\tau$ 'th position in  $T$ .



# A Randomized Solution

How to answer a query

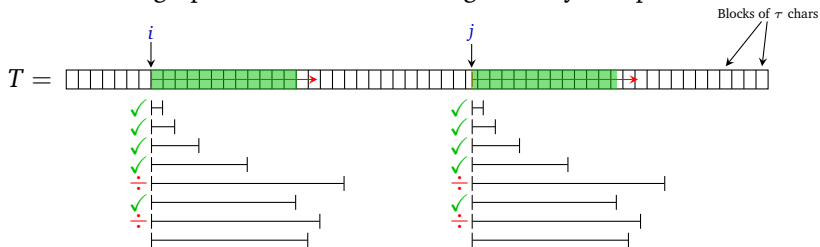
**Idea:** Store fingerprints of suffixes starting at every  $\tau$ 'th position in  $T$ .



# A Randomized Solution

How to answer a query

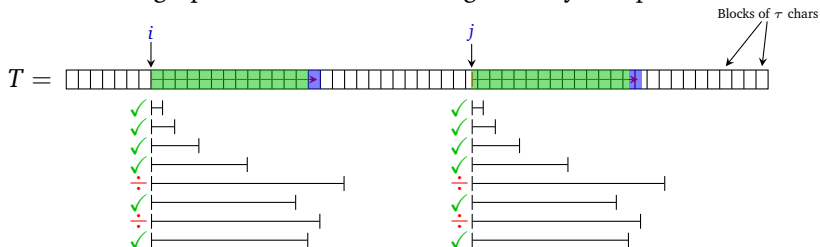
**Idea:** Store fingerprints of suffixes starting at every  $\tau$ 'th position in  $T$ .



# A Randomized Solution

How to answer a query

**Idea:** Store fingerprints of suffixes starting at every  $\tau$ 'th position in  $T$ .



## Analysis

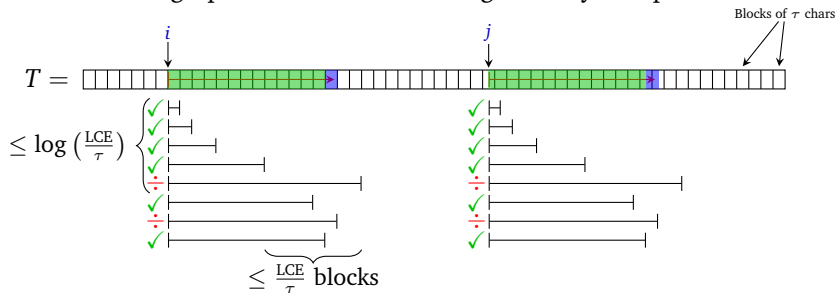
**Time:** At most  $2 \log\left(\frac{\text{LCE}}{\tau}\right)$  fingerprint comparisons each taking time  $O(\tau)$ . Hence query time  $O\left(\tau \log\left(\frac{\text{LCE}}{\tau}\right)\right)$ .

**Space:**  $O\left(\frac{n}{\tau}\right)$ .

# A Randomized Solution

How to answer a query

**Idea:** Store fingerprints of suffixes starting at every  $\tau$ 'th position in  $T$ .



## Analysis

**Time:** At most  $2 \log\left(\frac{\text{LCE}}{\tau}\right)$  fingerprint comparisons each taking time  $O(\tau)$ . Hence query time  $O\left(\tau \log\left(\frac{\text{LCE}}{\tau}\right)\right)$ .

**Space:**  $O\left(\frac{n}{\tau}\right)$ .