Time-Optimal Construction of String Synchronizing Sets

Jonas Ellert and Tomasz Kociumaka

Some gems of stringology...

- exact pattern matching...

$$T = \overset{1}{\boxed{\texttt{s t a c k s a t s t a c s a t t r a c k a}}}\overset{n}{} \qquad P = \overset{1}{\boxed{\texttt{a c k}}}\overset{m}{}$$

Some gems of stringology...

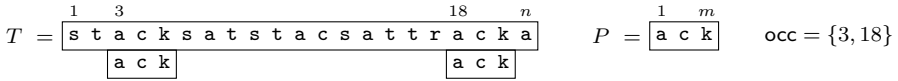
- exact pattern matching...

Some gems of stringology...

- exact pattern matching...

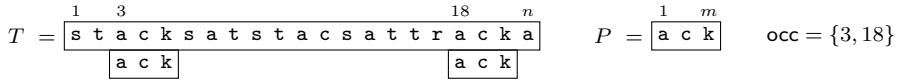
Some gems of stringology...

- exact pattern matching...



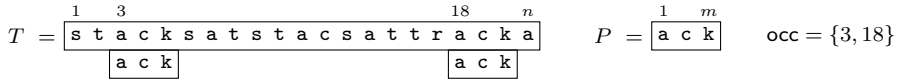
Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

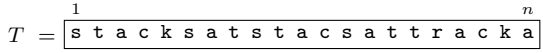


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

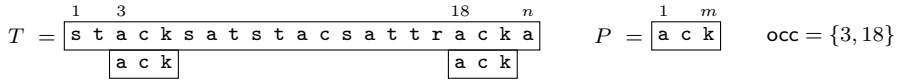


- Lempel-Ziv compression...

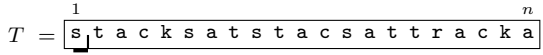


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

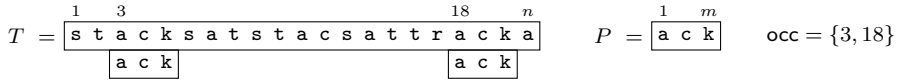


- Lempel-Ziv compression...

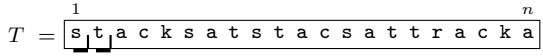


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

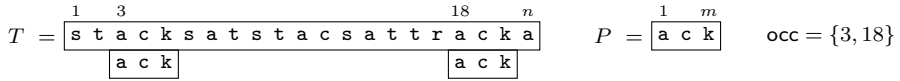


- Lempel-Ziv compression...

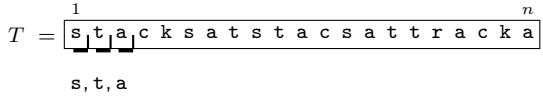


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

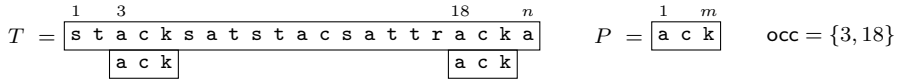


- Lempel-Ziv compression...

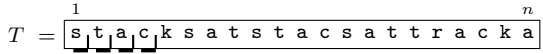


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

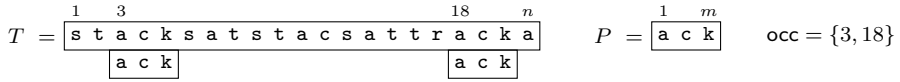


- Lempel-Ziv compression...

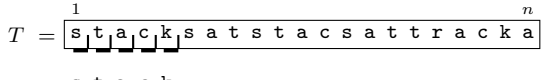


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

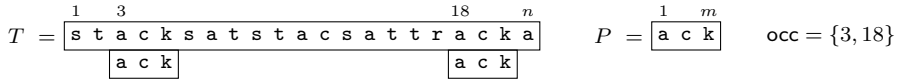


- Lempel-Ziv compression...

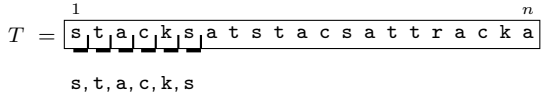


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

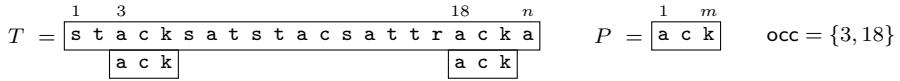


- Lempel-Ziv compression...

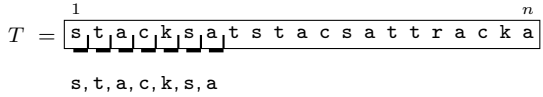


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

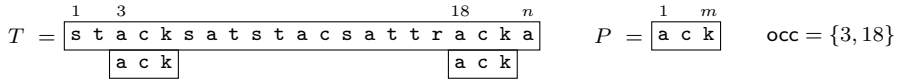


- Lempel-Ziv compression...

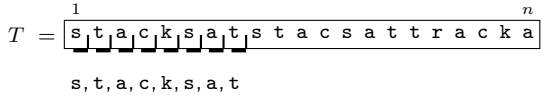


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

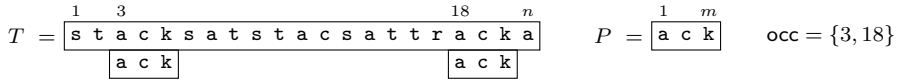


- Lempel-Ziv compression...

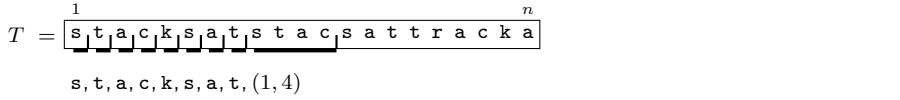


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

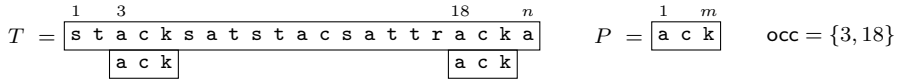


- Lempel-Ziv compression...

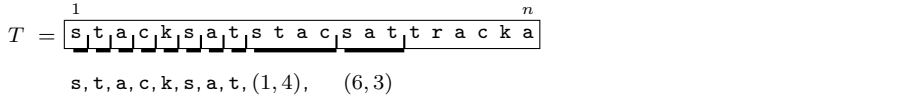


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

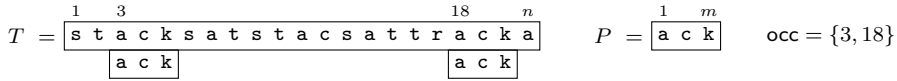


- Lempel-Ziv compression...

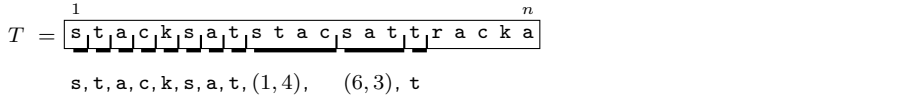


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

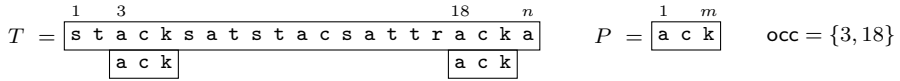


- Lempel-Ziv compression...

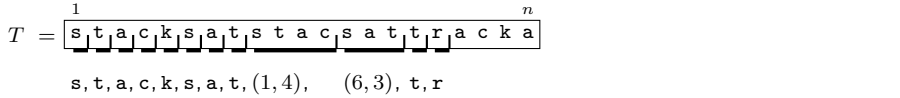


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

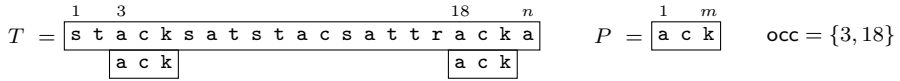


- Lempel-Ziv compression...

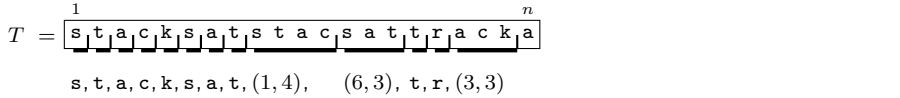


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

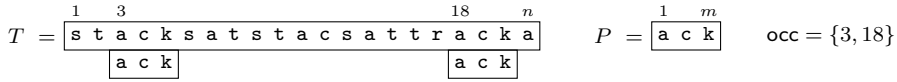


- Lempel-Ziv compression...

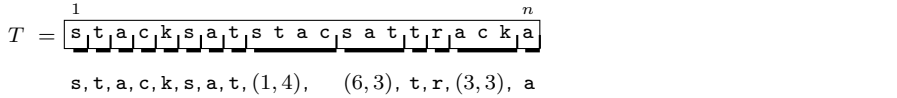


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]



- Lempel-Ziv compression...



Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

$$T = \texttt{s t a c k s a t s t a c s a t t r a c k a} \qquad P = \texttt{a c k} \qquad \text{occ} = \{3, 18\}$$

(positions marked above T: 1, 3, 18, n; above P: 1, m; occurrences of $\texttt{a c k}$ shown below T at positions 3 and 18)

- Lempel-Ziv compression... $\mathcal{O}(n)$ time [Ziv+Lempel '77; Storer+Szymanski '82]

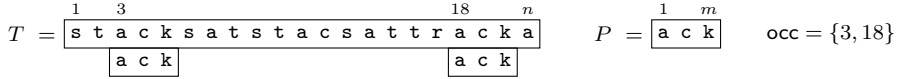
$$T = \texttt{s|t|a|c|k|s|a|t|s t a c|s a t|t|r|a c k|a}$$

(positions marked above T: 1, n)

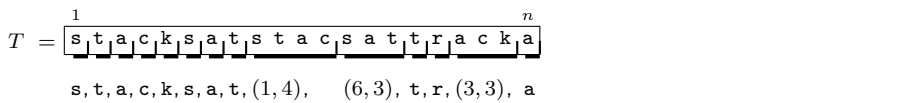
$$\texttt{s}, \texttt{t}, \texttt{a}, \texttt{c}, \texttt{k}, \texttt{s}, \texttt{a}, \texttt{t}, (1, 4), \quad (6, 3), \ \texttt{t}, \texttt{r}, (3, 3), \ \texttt{a}$$

Some gems of stringology...

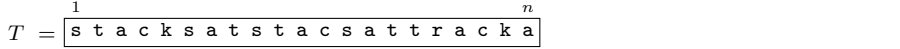
- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]



- Lempel-Ziv compression... $\mathcal{O}(n)$ time [Ziv+Lempel '77; Storer+Szymanski '82]

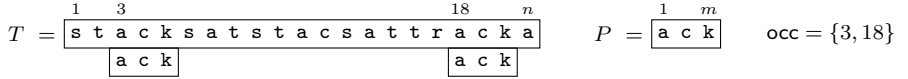


- Longest Common Extension (LCE) data structure...
  - query in $\mathcal{O}(1)$ time: given i, j, return maximal ℓ s.t. $T[i\,..\,i+\ell) = T[j\,..\,j+\ell)$

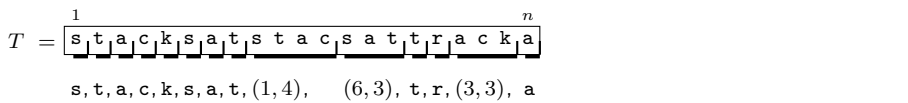


Some gems of stringology...

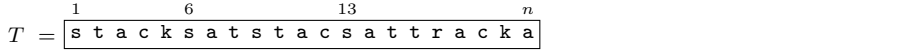
- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]



- Lempel-Ziv compression... $\mathcal{O}(n)$ time [Ziv+Lempel '77; Storer+Szymanski '82]

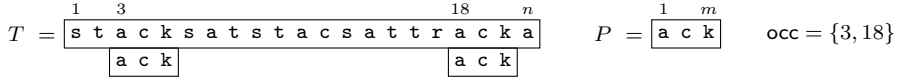


- Longest Common Extension (LCE) data structure...
  - query in $\mathcal{O}(1)$ time: given i, j, return maximal ℓ s.t. $T[i\,..\,i+\ell) = T[j\,..\,j+\ell)$

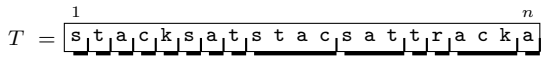


Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]

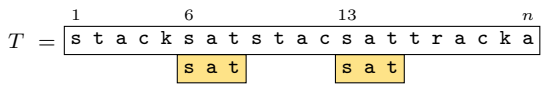


- Lempel-Ziv compression... $\mathcal{O}(n)$ time [Ziv+Lempel '77; Storer+Szymanski '82]



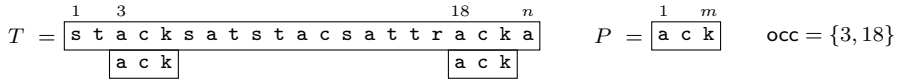
s, t, a, c, k, s, a, t, $(1,4)$, $(6,3)$, t, r, $(3,3)$, a

- Longest Common Extension (LCE) data structure...
  - query in $\mathcal{O}(1)$ time: given i, j, return maximal ℓ s.t. $T[i\,..\,i+\ell) = T[j\,..\,j+\ell)$

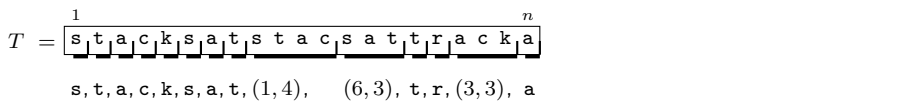


Some gems of stringology...

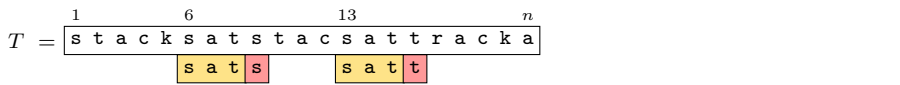
- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]



- Lempel-Ziv compression... $\mathcal{O}(n)$ time [Ziv+Lempel '77; Storer+Szymanski '82]

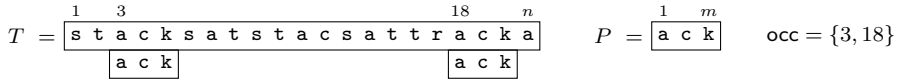


- Longest Common Extension (LCE) data structure...
  - query in $\mathcal{O}(1)$ time: given i, j, return maximal ℓ s.t. $T[i\,..\,i+\ell) = T[j\,..\,j+\ell)$

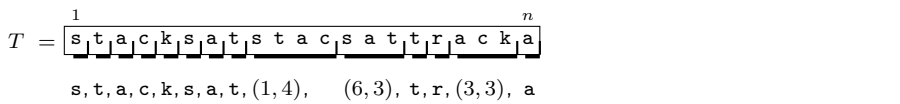


Some gems of stringology...

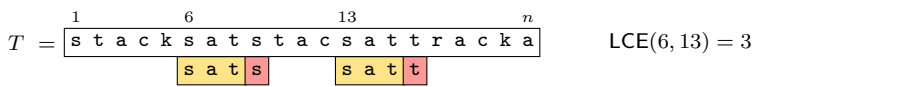
- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]



- Lempel-Ziv compression... $\mathcal{O}(n)$ time [Ziv+Lempel '77; Storer+Szymanski '82]

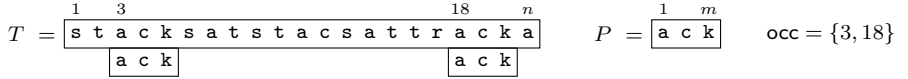


- Longest Common Extension (LCE) data structure...
  - query in $\mathcal{O}(1)$ time: given i, j, return maximal ℓ s.t. $T[i\,..\,i+\ell) = T[j\,..\,j+\ell)$

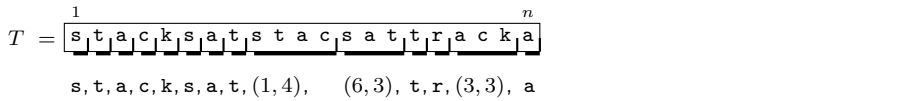


Some gems of stringology...

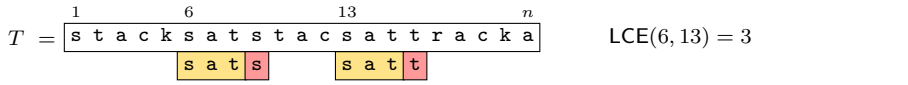
- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]



- Lempel-Ziv compression... $\mathcal{O}(n)$ time [Ziv+Lempel '77; Storer+Szymanski '82]



- Longest Common Extension (LCE) data structure... $\mathcal{O}(n)$ construction time [Farach '97]
  - query in $\mathcal{O}(1)$ time: given i, j, return maximal ℓ s.t. $T[i\,..\,i+\ell) = T[j\,..\,j+\ell)$



Some gems of stringology...

- often $\mathcal{O}(n)$ time solutions on input of length n
- ... surely that must be optimal, right?

Some gems of stringology...

- often $\mathcal{O}(n)$ time solutions on input of length n
- ... surely that must be optimal, right?

- **most algorithms in word RAM model** [Hagerup '98]

Some gems of stringology...

- often $\mathcal{O}(n)$ time solutions on input of length n
- ... surely that must be optimal, right?

- **most algorithms in word RAM model** [Hagerup '98]
- memory words and CPU registers consist of w bits
- representing integers in $[0\,..\,2^w)$

Some gems of stringology...

- often $\mathcal{O}(n)$ time solutions on input of length n
- ... surely that must be optimal, right?

- **most algorithms in word RAM model** [Hagerup '98]
- memory words and CPU registers consist of w bits
- representing integers in $[0\,..\,2^w)$

Operations:
arithmetic ($+\ -\ \times\ /$)
bit-wise logic (`NOT`,`AND`,`OR`,`XOR`)
bit-shifting ($\ll$, $\gg$)

Some gems of stringology...

- often $\mathcal{O}(n)$ time solutions on input of length n
- ... surely that must be optimal, right?

- **most algorithms in word RAM model** [Hagerup '98]
- memory words and CPU registers consist of w bits
- representing integers in $[0\,..\,2^w)$
- assumption: $w = \Omega(\log n)$, usually $w = \Theta(\log n)$

Operations:
arithmetic ($+\ -\ \times\ /$)
bit-wise logic (`NOT`,`AND`,`OR`,`XOR`)
bit-shifting ($\ll$, $\gg$)

Some gems of stringology...

- often $\mathcal{O}(n)$ time solutions on input of length n
- ... surely that must be optimal, right?

- **most algorithms in word RAM model** [Hagerup '98]
- memory words and CPU registers consist of w bits
- representing integers in $[0\,..\,2^w)$
- assumption: $w = \Omega(\log n)$, usually $w = \Theta(\log n)$

Operations:
arithmetic ($+$ $-$ $\times$ $/$)
bit-wise logic (`NOT`,`AND`,`OR`,`XOR`)
bit-shifting ($\ll$, $\gg$)

year	data pp (MB)	w	w (SIMD)
1986	539	32	–
1993	2,866	32	–
2000	8,988	32–64	128
2007	44,716	64	128

Some gems of stringology...

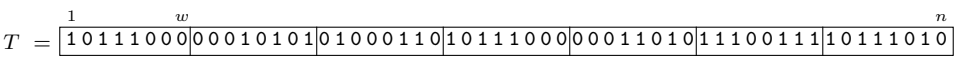
- often $\mathcal{O}(n)$ time solutions on input of length n
- ... surely that must be optimal, right?

Operations:
arithmetic ($+$ $-$ $\times$ $/$)
bit-wise logic (`NOT`,`AND`,`OR`,`XOR`)
bit-shifting ($\ll$, $\gg$)

- **most algorithms in word RAM model** [Hagerup '98]
- memory words and CPU registers consist of w bits
- representing integers in $[0\,..\,2^w)$
- assumption: $w = \Omega(\log n)$, usually $w = \Theta(\log n)$

year	data pp (MB)	w	w (SIMD)
1986	539	32	–
1993	2,866	32	–
2000	8,988	32–64	128
2007	44,716	64	128

Implications for, e.g., a binary string:



Some gems of stringology...

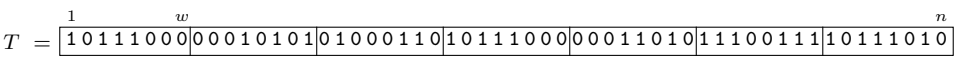
- often $\mathcal{O}(n)$ time solutions on input of length n
- ... surely that must be optimal, right?

Operations:
arithmetic ($+$ $-$ $\times$ $/$)
bit-wise logic (`NOT`,`AND`,`OR`,`XOR`)
bit-shifting ($\ll$, $\gg$)

- **most algorithms in word RAM model** [Hagerup '98]
- memory words and CPU registers consist of w bits
- representing integers in $[0\,..\,2^w)$
- assumption: $w = \Omega(\log n)$, usually $w = \Theta(\log n)$

year	data pp (MB)	w	w (SIMD)
1986	539	32	–
1993	2,866	32	–
2000	8,988	32–64	128
2007	44,716	64	128

Implications for, e.g., a binary string:



... only $\approx n/w = \Theta(n/\log n)$ words. Why not process in $\mathcal{O}(n/\log n)$ time?

Some gems of stringology...

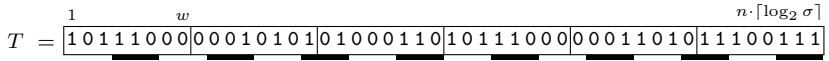
- often $\mathcal{O}(n)$ time solutions on input of length n
- ... surely that must be optimal, right?

Operations:
arithmetic ($+$ $-$ $\times$ $/$)
bit-wise logic (`NOT`,`AND`,`OR`,`XOR`)
bit-shifting ($\ll$, $\gg$)

- **most algorithms in word RAM model** [Hagerup '98]
- memory words and CPU registers consist of w bits
- representing integers in $[0\,..\,2^w)$
- assumption: $w = \Omega(\log n)$, usually $w = \Theta(\log n)$

year	data pp (MB)	w	w (SIMD)
1986	539	32	–
1993	2,866	32	–
2000	8,988	32–64	128
2007	44,716	64	128

More generally, consider $T \in [0\,..\,\sigma)^n$



Some gems of stringology...

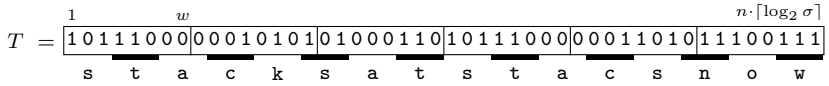
- often $\mathcal{O}(n)$ time solutions on input of length n
- ... surely that must be optimal, right?

Operations:
arithmetic ($+$ $-$ $\times$ /)
bit-wise logic (NOT,AND,OR,XOR)
bit-shifting ($\ll$, $\gg$)

- **most algorithms in word RAM model** [Hagerup '98]
- memory words and CPU registers consist of w bits
- representing integers in $[0\,..\,2^w)$
- assumption: $w = \Omega(\log n)$, usually $w = \Theta(\log n)$

year	data pp (MB)	w	w (SIMD)
1986	539	32	–
1993	2,866	32	–
2000	8,988	32–64	128
2007	44,716	64	128

More generally, consider $T \in [0\,..\,\sigma)^n$



s	bin(s)
a	000
c	001
k	010
n	011
o	100
s	101
t	110
w	111

Some gems of stringology...

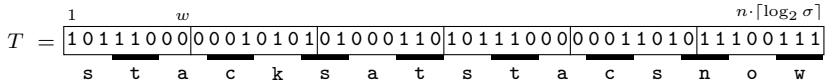
- often $\mathcal{O}(n)$ time solutions on input of length n
- ... surely that must be optimal, right?

Operations:
arithmetic ($+ \; - \; \times \; /$)
bit-wise logic (`NOT`,`AND`,`OR`,`XOR`)
bit-shifting ($\ll$, $\gg$)

- **most algorithms in word RAM model** [Hagerup '98]
- memory words and CPU registers consist of w bits
- representing integers in $[0\,..\,2^w)$
- assumption: $w = \Omega(\log n)$, usually $w = \Theta(\log n)$

year	data pp (MB)	w	w (SIMD)
1986	539	32	–
1993	2,866	32	–
2000	8,988	32–64	128
2007	44,716	64	128

More generally, consider $T \in [0\,..\,\sigma)^n$

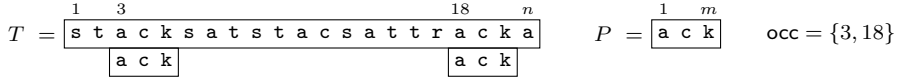


... $\approx n \cdot \lceil \log_2 \sigma \rceil / w$ words. **Truly linear time:** $\mathcal{O}(n \log \sigma / \log n) = \mathcal{O}(n / \log_\sigma n)$

s	bin(s)
a	000
c	001
k	010
n	011
o	100
s	101
t	110
w	111

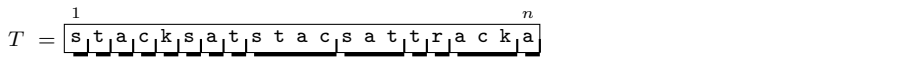
Some gems of stringology...

- exact pattern matching... $\mathcal{O}(n)$ time [Knuth et al. '77]



$T =$ s t a c k s a t s t a c s a t t r a c k a $\quad P =$ a c k $\quad \text{occ} = \{3, 18\}$

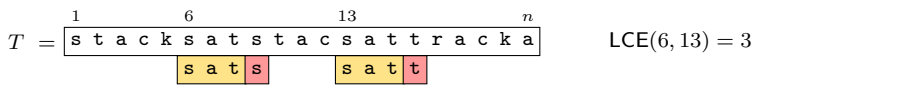
- Lempel-Ziv compression... $\mathcal{O}(n)$ time [Ziv+Lempel '77; Storer+Szymanski '82]



$T =$

s, t, a, c, k, s, a, t, $(1, 4)$, $\quad (6, 3)$, t, r, $(3, 3)$, a

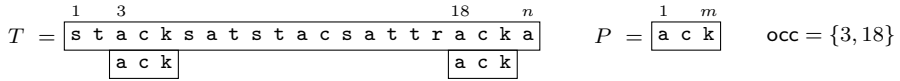
- Longest Common Extension (LCE) data structure... $\mathcal{O}(n)$ construction time [Farach '97]
  - query in $\mathcal{O}(1)$ time: given i, j, return maximal ℓ s.t. $T[i\,..\,i+\ell) = T[j\,..\,j+\ell)$



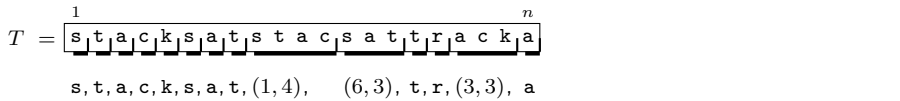
$T =$ $\quad \text{LCE}(6, 13) = 3$

Some gems of stringology...

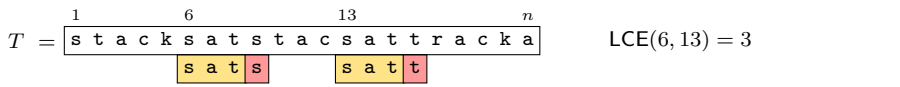
- exact pattern matching... $\mathcal{O}(n/\log_\sigma n)$ time [Bille '11]



- Lempel-Ziv compression... $\mathcal{O}(n)$ time [Ziv+Lempel '77; Storer+Szymanski '82]

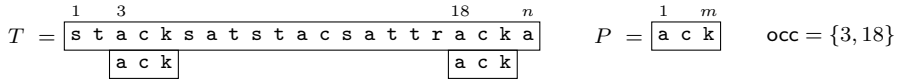


- Longest Common Extension (LCE) data structure... $\mathcal{O}(n)$ construction time [Farach '97]
  - query in $\mathcal{O}(1)$ time: given i, j, return maximal ℓ s.t. $T[i\,..\,i+\ell) = T[j\,..\,j+\ell)$

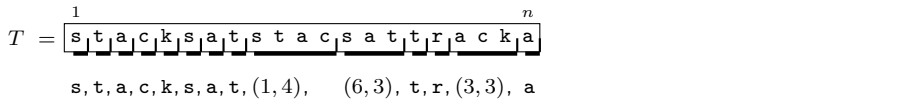


Some gems of stringology...

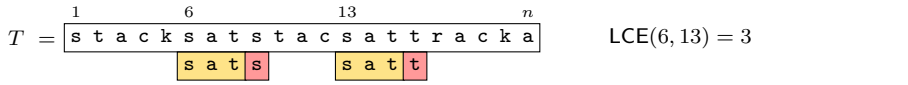
- exact pattern matching... $\mathcal{O}(n/\log_\sigma n)$ time [Bille '11]



- Lempel-Ziv compression... $\mathcal{O}(n/\log_\sigma n)$ or $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time [Ellert '23; Kempa+Kociumaka '24]

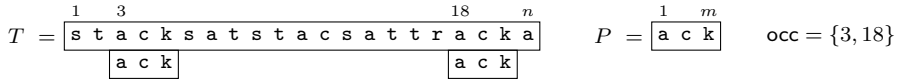


- Longest Common Extension (LCE) data structure... $\mathcal{O}(n)$ construction time [Farach '97]
  - query in $\mathcal{O}(1)$ time: given i, j, return maximal ℓ s.t. $T[i\,..\,i+\ell) = T[j\,..\,j+\ell)$

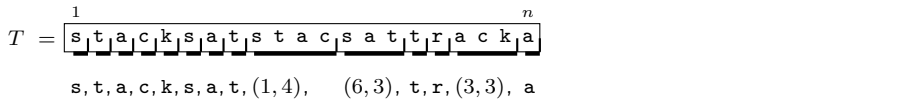


Some gems of stringology...

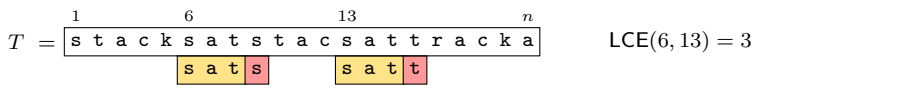
- exact pattern matching... $\mathcal{O}(n/\log_\sigma n)$ time [Bille '11]



- Lempel-Ziv compression... $\mathcal{O}(n/\log_\sigma n)$ or $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time [Ellert '23; Kempa+Kociumaka '24]



- Longest Common Extension (LCE) data structure... $\mathcal{O}(n/\log_\sigma n)$ time [Kempa+Kociumaka '19]
  - query in $\mathcal{O}(1)$ time: given i, j, return maximal ℓ s.t. $T[i\,..\,i+\ell) = T[j\,..\,j+\ell)$

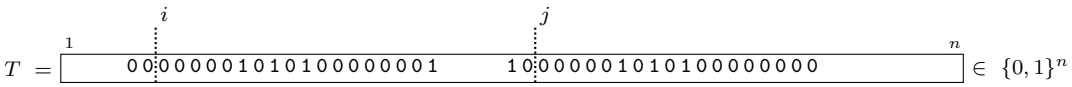


String synchronizing sets

- would like a powerful tool for designing truly linear time algorithms
- should consider words not symbols or bits

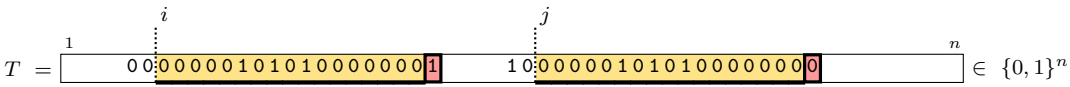
String synchronizing sets

- would like a powerful tool for designing truly linear time algorithms
- should consider words not symbols or bits
- example for LCE data structure (for binary alphabet)



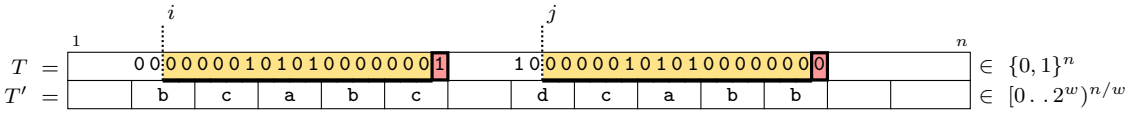
String synchronizing sets

- would like a powerful tool for designing truly linear time algorithms
- should consider words not symbols or bits
- example for LCE data structure (for binary alphabet)



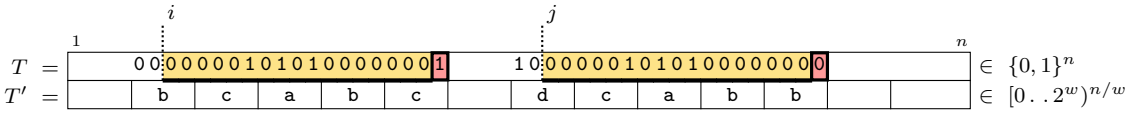
String synchronizing sets

- would like a powerful tool for designing truly linear time algorithms
- should consider words not symbols or bits
- example for LCE data structure (for binary alphabet)



String synchronizing sets

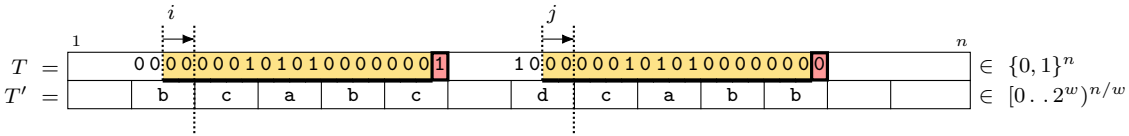
- would like a powerful tool for designing truly linear time algorithms
- should consider words not symbols or bits
- example for LCE data structure (for binary alphabet)



- compute LCE data structure for T' in $\mathcal{O}(n/w) = \mathcal{O}(n/\log n)$ time

String synchronizing sets

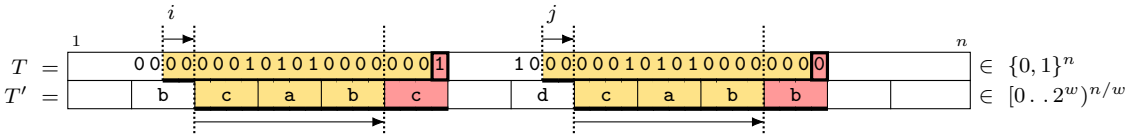
- would like a powerful tool for designing truly linear time algorithms
- should consider words not symbols or bits
- example for LCE data structure (for binary alphabet)



- compute LCE data structure for T' in $\mathcal{O}(n/w) = \mathcal{O}(n/\log n)$ time

String synchronizing sets

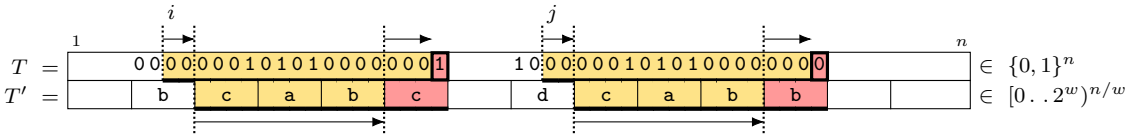
- would like a powerful tool for designing truly linear time algorithms
- should consider words not symbols or bits
- example for LCE data structure (for binary alphabet)



- compute LCE data structure for T' in $\mathcal{O}(n/w) = \mathcal{O}(n/\log n)$ time

String synchronizing sets

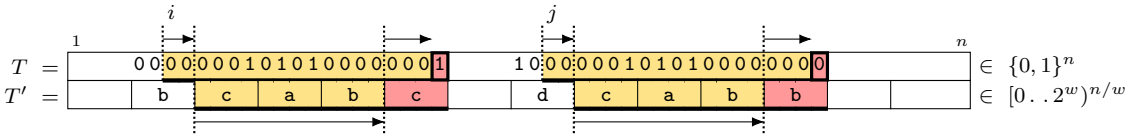
- would like a powerful tool for designing truly linear time algorithms
- should consider words not symbols or bits
- example for LCE data structure (for binary alphabet)



- compute LCE data structure for T' in $\mathcal{O}(n/w) = \mathcal{O}(n/\log n)$ time

String synchronizing sets

- would like a powerful tool for designing truly linear time algorithms
- should consider words not symbols or bits
- example for LCE data structure (for binary alphabet)



- compute LCE data structure for T' in $\mathcal{O}(n/w) = \mathcal{O}(n/\log n)$ time
- can answer LCE on T if query positions are $i \equiv j \bmod w$

String synchronizing sets

- fix parameter τ, typically $\tau = \Theta(w)$
- synchronizing function sync : $\{0,1\}^{2\tau} \to \{\text{true}, \text{false}\}$

String synchronizing sets

- fix parameter τ, typically $\tau = \Theta(w)$
- synchronizing function sync : $\{0,1\}^{2\tau} \to \{\text{true}, \text{false}\}$
- position i is synchronizing if sync$(T[i\,..\,i+2\tau))$

String synchronizing sets

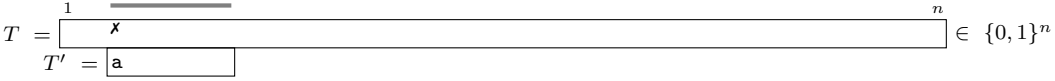
- fix parameter τ, typically $\tau = \Theta(w)$
- synchronizing function sync : $\{0,1\}^{2\tau} \to \{\text{true}, \text{false}\}$
- position i is synchronizing if sync$(T[i\,..\,i+2\tau))$

$T =$ 

$\in \{0,1\}^n$

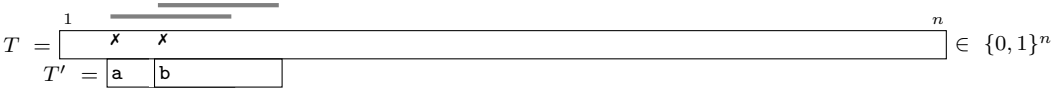
String synchronizing sets

- fix parameter τ, typically $\tau = \Theta(w)$
- synchronizing function sync : $\{0,1\}^{2\tau} \to \{\text{true}, \text{false}\}$
- position i is synchronizing if sync$(T[i\,..\,i+2\tau))$



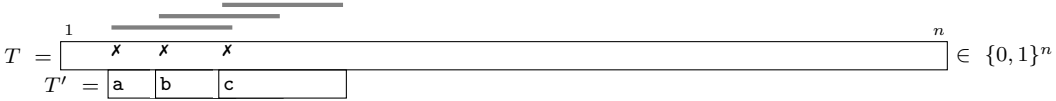
String synchronizing sets

- fix parameter τ, typically $\tau = \Theta(w)$
- synchronizing function sync : $\{0,1\}^{2\tau} \to \{\text{true}, \text{false}\}$
- position i is synchronizing if sync$(T[i\,..\,i+2\tau])$



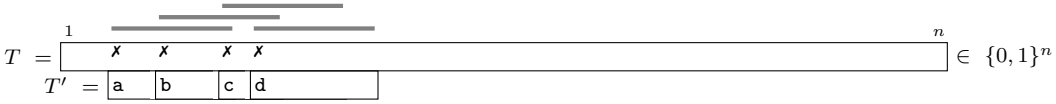
String synchronizing sets

- fix parameter τ, typically $\tau = \Theta(w)$
- synchronizing function sync : $\{0,1\}^{2\tau} \to \{\text{true}, \text{false}\}$
- position i is synchronizing if sync$(T[i\,..\,i+2\tau])$



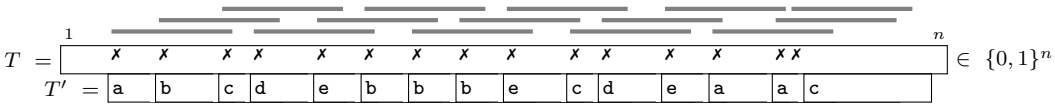
String synchronizing sets

- fix parameter τ, typically $\tau = \Theta(w)$
- synchronizing function sync : $\{0,1\}^{2\tau} \to \{\text{true}, \text{false}\}$
- position i is synchronizing if sync$(T[i\,..\,i+2\tau])$



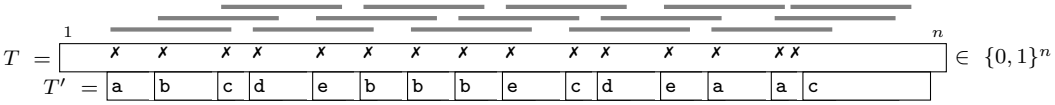
String synchronizing sets

- fix parameter τ, typically $\tau = \Theta(w)$
- synchronizing function sync : $\{0,1\}^{2\tau} \to \{\text{true}, \text{false}\}$
- position i is synchronizing if sync$(T[i\,..\,i+2\tau))$



String synchronizing sets

- fix parameter τ, typically $\tau = \Theta(w)$
- synchronizing function sync : $\{0,1\}^{2\tau} \to \{\text{true}, \text{false}\}$
- position i is synchronizing if sync$(T[i\,..\,i+2\tau))$

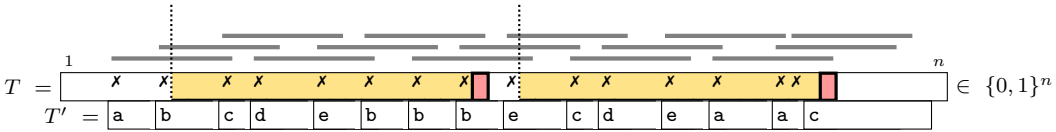


τ-synchronizing set of T:

- only $\mathcal{O}(n/\tau)$ synchronizing positions, so $|T'| = \mathcal{O}(n/\tau) = \mathcal{O}(n/\log n)$
- all of T overlapped by synchronizing fragments

String synchronizing sets

- fix parameter τ, typically $\tau = \Theta(w)$
- synchronizing function sync : $\{0,1\}^{2\tau} \to \{\text{true}, \text{false}\}$
- position i is synchronizing if sync$(T[i\,..\,i+2\tau))$

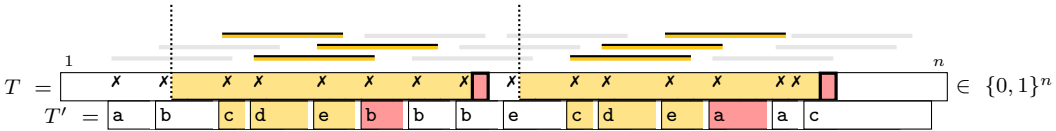


τ-synchronizing set of T:

- only $\mathcal{O}(n/\tau)$ synchronizing positions, so $|T'| = \mathcal{O}(n/\tau) = \mathcal{O}(n/\log n)$
- all of T overlapped by synchronizing fragments

String synchronizing sets

- fix parameter τ, typically $\tau = \Theta(w)$
- synchronizing function sync : $\{0,1\}^{2\tau} \to \{\text{true}, \text{false}\}$
- position i is synchronizing if sync$(T[i\,..\,i+2\tau))$

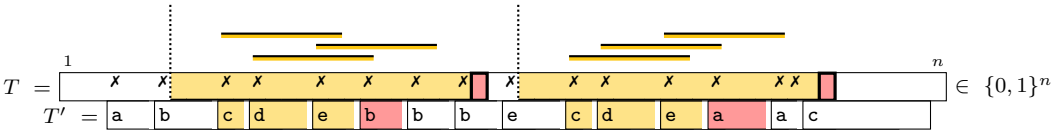


τ-synchronizing set of T:

- only $\mathcal{O}(n/\tau)$ synchronizing positions, so $|T'| = \mathcal{O}(n/\tau) = \mathcal{O}(n/\log n)$
- all of T overlapped by synchronizing fragments

String synchronizing sets

- fix parameter τ, typically $\tau = \Theta(w)$
- synchronizing function sync : $\{0,1\}^{2\tau} \to \{\text{true}, \text{false}\}$
- position i is synchronizing if sync$(T[i\,..\,i+2\tau))$

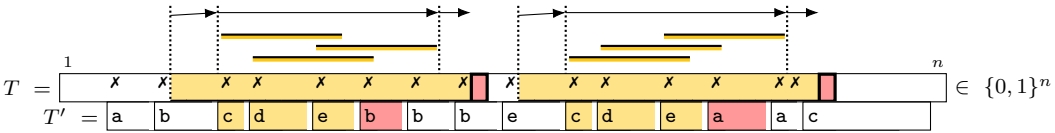


τ-synchronizing set of T:

- only $\mathcal{O}(n/\tau)$ synchronizing positions, so $|T'| = \mathcal{O}(n/\tau) = \mathcal{O}(n/\log n)$
- all of T overlapped by synchronizing fragments

String synchronizing sets

- fix parameter τ, typically $\tau = \Theta(w)$
- synchronizing function sync : $\{0,1\}^{2\tau} \to \{\text{true}, \text{false}\}$
- position i is synchronizing if sync$(T[i\,..\,i+2\tau))$

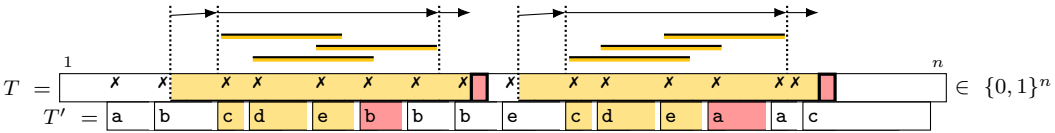


τ-synchronizing set of T:

- only $\mathcal{O}(n/\tau)$ synchronizing positions, so $|T'| = \mathcal{O}(n/\tau) = \mathcal{O}(n/\log n)$
- all of T overlapped by synchronizing fragments

String synchronizing sets

- fix parameter τ, typically $\tau = \Theta(w)$
- synchronizing function sync : $\{0,1\}^{2\tau} \to \{\text{true}, \text{false}\}$
- position i is synchronizing if sync$(T[i\,..\,i+2\tau))$

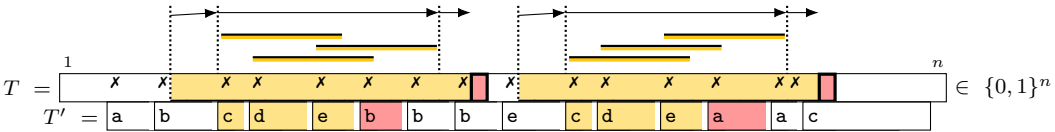


τ-synchronizing set of T:

- only $\mathcal{O}(n/\tau)$ synchronizing positions, so $|T'| = \mathcal{O}(n/\tau) = \mathcal{O}(n/\log n)$
- all of T overlapped by synchronizing fragments

String synchronizing sets

- fix parameter τ, typically $\tau = \Theta(w)$
- synchronizing function sync : $\{0,1\}^{2\tau} \to \{\text{true}, \text{false}\}$
- position i is synchronizing if sync$(T[i\,..\,i+2\tau))$



τ-synchronizing set of T:

- only $\mathcal{O}(n/\tau)$ synchronizing positions, so $|T'| = \mathcal{O}(n/\tau) = \mathcal{O}(n/\log n)$
- all of T overlapped by synchronizing fragments
- some extra conditions for periodic fragments apply

String synchronizing sets

	preprocessing (independent of τ)	construction (depends on τ)	
[Kempa+Kociumaka '19]	none	$\mathcal{O}(n/\log_\sigma n)$	only for $\tau < \frac{1}{5}\log_\sigma n$

String synchronizing sets

	preprocessing (independent of τ)	construction (depends on τ)	
[Kempa+Kociumaka '19]	none	$\mathcal{O}(n/\log_\sigma n)$	only for $\tau < \frac{1}{5}\log_\sigma n$
[Kociumaka et al. '24]	$\mathcal{O}(n)$	$\mathcal{O}(n/\tau)$	

String synchronizing sets

	preprocessing (independent of τ)	construction (depends on τ)	
[Kempa+Kociumaka '19]	none	$\mathcal{O}(n/\log_\sigma n)$	only for $\tau < \frac{1}{5}\log_\sigma n$
[Kociumaka et al. '24]	$\mathcal{O}(n)$	$\mathcal{O}(n/\tau)$	
this paper	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n/\tau)$	simple representation

String synchronizing sets

	preprocessing (independent of τ)	construction (depends on τ)	
[Kempa+Kociumaka '19]	none	$\mathcal{O}(n/\log_\sigma n)$	only for $\tau < \frac{1}{5}\log_\sigma n$
[Kociumaka et al. '24]	$\mathcal{O}(n)$	$\mathcal{O}(n/\tau)$	
this paper	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n/\tau)$	simple representation
	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n\log\tau/(\tau\log n))$	compressed representation

String synchronizing sets

	preprocessing (independent of τ)	construction (depends on τ)	
[Kempa+Kociumaka '19]	none	$\mathcal{O}(n/\log_\sigma n)$	only for $\tau < \frac{1}{5}\log_\sigma n$
[Kociumaka et al. '24]	$\mathcal{O}(n)$	$\mathcal{O}(n/\tau)$	
this paper	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n/\tau)$	simple representation
	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n\log\tau/(\tau\log n))$	compressed representation

- most applications use $\tau = \Theta(\log_\sigma n)$
- some applications can benefit from arbitrary τ

String synchronizing sets

	preprocessing (independent of τ)	construction (depends on τ)	
[Kempa+Kociumaka '19]	none	$\mathcal{O}(n/\log_\sigma n)$	only for $\tau < \frac{1}{5}\log_\sigma n$
[Kociumaka et al. '24]	$\mathcal{O}(n)$	$\mathcal{O}(n/\tau)$	
this paper	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n/\tau)$	simple representation
	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n\log\tau/(\tau\log n))$	compressed representation

- most applications use $\tau = \Theta(\log_\sigma n)$
- some applications can benefit from arbitrary τ

Example: Space-time trade-off for LCEs:

- LCE reduces to scanning $\mathcal{O}(\tau)$ symbols and computing one LCE in T'

String synchronizing sets

	preprocessing (independent of τ)	construction (depends on τ)	
[Kempa+Kociumaka '19]	none	$\mathcal{O}(n/\log_\sigma n)$	only for $\tau < \frac{1}{5}\log_\sigma n$
[Kociumaka et al. '24]	$\mathcal{O}(n)$	$\mathcal{O}(n/\tau)$	
this paper	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n/\tau)$	simple representation
	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n\log\tau/(\tau\log n))$	compressed representation

- most applications use $\tau = \Theta(\log_\sigma n)$
- some applications can benefit from arbitrary τ

Example: Space-time trade-off for LCEs:

- LCE reduces to scanning $\mathcal{O}(\tau)$ symbols and computing one LCE in T'
- scan takes $\mathcal{O}(1 + \tau/\log_\sigma n)$ time in a word-wise manner

String synchronizing sets

	preprocessing (independent of τ)	construction (depends on τ)	
[Kempa+Kociumaka '19]	none	$\mathcal{O}(n/\log_\sigma n)$	only for $\tau < \frac{1}{5}\log_\sigma n$
[Kociumaka et al. '24]	$\mathcal{O}(n)$	$\mathcal{O}(n/\tau)$	
this paper	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n/\tau)$	simple representation
	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n\log\tau/(\tau\log n))$	compressed representation

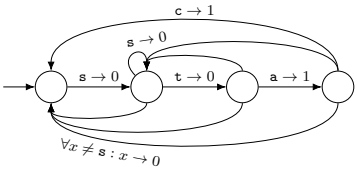
- most applications use $\tau = \Theta(\log_\sigma n)$
- some applications can benefit from arbitrary τ

Example: Space-time trade-off for LCEs:

- LCE reduces to scanning $\mathcal{O}(\tau)$ symbols and computing one LCE in T'
- scan takes $\mathcal{O}(1 + \tau/\log_\sigma n)$ time in a word-wise manner
- LCE data structure of T' is of size $\mathcal{O}(n/\tau)$ words

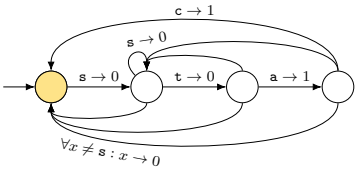
Some new tools we developed

$$T = \boxed{\texttt{s t a r t s t a c k s a t s t a c s}}$$



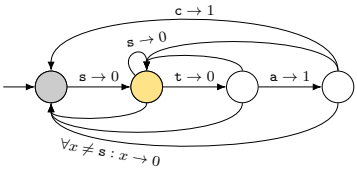
Some new tools we developed

$$T = \boxed{\texttt{s t a r t s t a c k s a t s t a c s}}$$



Some new tools we developed

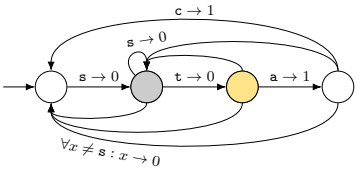
$$T = \underset{0}{\boxed{\texttt{s}\ \texttt{t}\ \texttt{a}\ \texttt{r}\ \texttt{t}\ \texttt{s}\ \texttt{t}\ \texttt{a}\ \texttt{c}\ \texttt{k}\ \texttt{s}\ \texttt{a}\ \texttt{t}\ \texttt{s}\ \texttt{t}\ \texttt{a}\ \texttt{c}\ \texttt{s}}}$$



Some new tools we developed

$$T = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{r} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{k} & \mathtt{s} & \mathtt{a} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{s} \\ \hline \end{array}$$

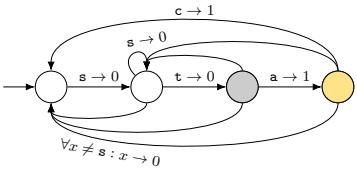
$0 \quad 0$



Some new tools we developed

$$T = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{r} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{k} & \mathtt{s} & \mathtt{a} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{s} \\ \hline \end{array}$$

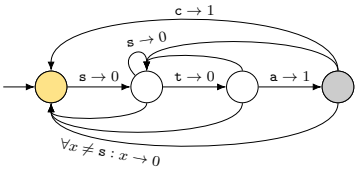
$$0 \quad 0 \quad 0$$



Some new tools we developed

$$T = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline \texttt{s} & \texttt{t} & \texttt{a} & \texttt{r} & \texttt{t} & \texttt{s} & \texttt{t} & \texttt{a} & \texttt{c} & \texttt{k} & \texttt{s} & \texttt{a} & \texttt{t} & \texttt{s} & \texttt{t} & \texttt{a} & \texttt{c} & \texttt{s} \\ \hline \end{array}$$

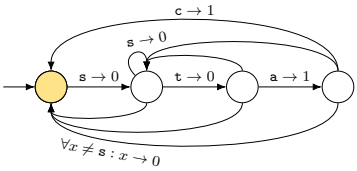
$$\begin{array}{cccc} 0 & 0 & 0 & 0 \end{array}$$



Some new tools we developed

$$T = \boxed{\texttt{s t a r t s t a c k s a t s t a c s}}$$

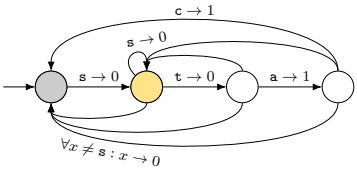
0 0 0 0 0



Some new tools we developed

$$T = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline \texttt{s} & \texttt{t} & \texttt{a} & \texttt{r} & \texttt{t} & \texttt{s} & \texttt{t} & \texttt{a} & \texttt{c} & \texttt{k} & \texttt{s} & \texttt{a} & \texttt{t} & \texttt{s} & \texttt{t} & \texttt{a} & \texttt{c} & \texttt{s} \\ \hline \end{array}$$

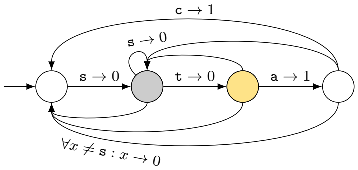
$$\begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$



Some new tools we developed

$T =$ s t a r t s **t** a c k s a t s t a c s

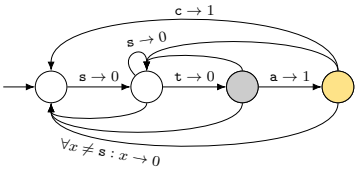
0 0 0 0 0 0 0



Some new tools we developed

$$T = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline \texttt{s} & \texttt{t} & \texttt{a} & \texttt{r} & \texttt{t} & \texttt{s} & \texttt{t} & \texttt{a} & \texttt{c} & \texttt{k} & \texttt{s} & \texttt{a} & \texttt{t} & \texttt{s} & \texttt{t} & \texttt{a} & \texttt{c} & \texttt{s} \\ \hline \end{array}$$

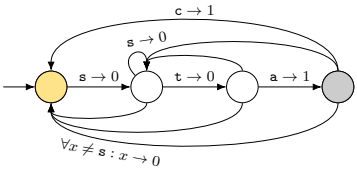
$$\begin{array}{cccccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$



Some new tools we developed

$$T = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline \texttt{s} & \texttt{t} & \texttt{a} & \texttt{r} & \texttt{t} & \texttt{s} & \texttt{t} & \texttt{a} & \texttt{c} & \texttt{k} & \texttt{s} & \texttt{a} & \texttt{t} & \texttt{s} & \texttt{t} & \texttt{a} & \texttt{c} & \texttt{s} \\ \hline \end{array}$$

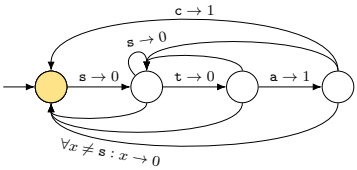
$$\begin{array}{cccccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \end{array}$$



Some new tools we developed

$$T = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{r} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{k} & \mathtt{s} & \mathtt{a} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{s} \\ \hline \end{array}$$

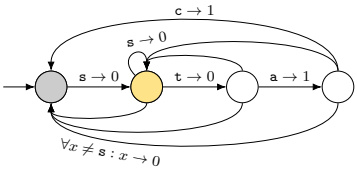
$$\begin{array}{cccccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \end{array}$$



Some new tools we developed

$$T = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline \texttt{s} & \texttt{t} & \texttt{a} & \texttt{r} & \texttt{t} & \texttt{s} & \texttt{t} & \texttt{a} & \texttt{c} & \texttt{k} & \texttt{s} & \texttt{a} & \texttt{t} & \texttt{s} & \texttt{t} & \texttt{a} & \texttt{c} & \texttt{s} \\ \hline \end{array}$$

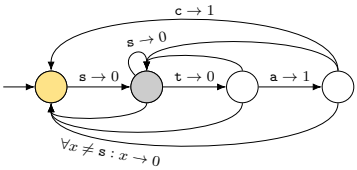
$$\begin{array}{cccccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \end{array}$$



Some new tools we developed

$$T = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{r} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{k} & \mathtt{s} & \mathtt{a} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{s} \\ \hline \end{array}$$

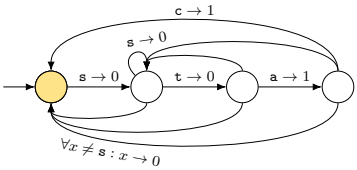
$$0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \mathbf{1}\ 0\ 0\ 0$$



Some new tools we developed

$$T = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline \texttt{s} & \texttt{t} & \texttt{a} & \texttt{r} & \texttt{t} & \texttt{s} & \texttt{t} & \texttt{a} & \texttt{c} & \texttt{k} & \texttt{s} & \texttt{a} & \texttt{t} & \texttt{s} & \texttt{t} & \texttt{a} & \texttt{c} & \texttt{s} \\ \hline \end{array}$$

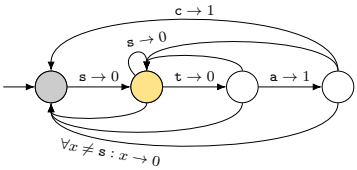
$$\begin{array}{cccccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \end{array}$$



Some new tools we developed

$$T = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline \texttt{s} & \texttt{t} & \texttt{a} & \texttt{r} & \texttt{t} & \texttt{s} & \texttt{t} & \texttt{a} & \texttt{c} & \texttt{k} & \texttt{s} & \texttt{a} & \texttt{t} & \texttt{s} & \texttt{t} & \texttt{a} & \texttt{c} & \texttt{s} \\ \hline \end{array}$$

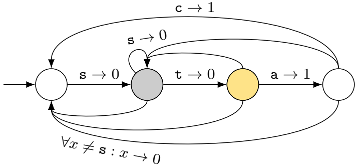
$$\begin{array}{cccccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \end{array}$$



Some new tools we developed

T = s t a r t s t a c k s a t s **t** a c s

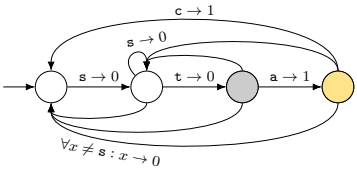
0 0 0 0 0 0 0 0 **1** 0 0 0 0 0 0



Some new tools we developed

$$T = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{r} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{k} & \mathtt{s} & \mathtt{a} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{s} \\ \hline \end{array}$$

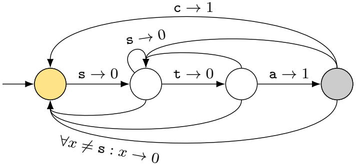
$$0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \mathbf{1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0$$



Some new tools we developed

$T =$ `s t a r t s t a c k s a t s t a c s`

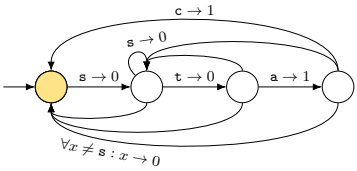
0 0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 **1**



Some new tools we developed

$$T = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{r} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{k} & \mathtt{s} & \mathtt{a} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{s} \\ \hline \end{array}$$

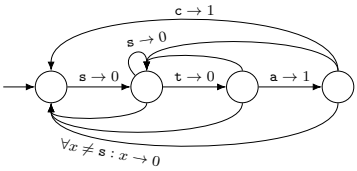
$$\begin{array}{cccccccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \end{array}$$



Some new tools we developed

$$T = \boxed{\texttt{s t a r t s t a c k s a t s t a c s}}$$

$$0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \mathbf{1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \mathbf{1}\ 0$$

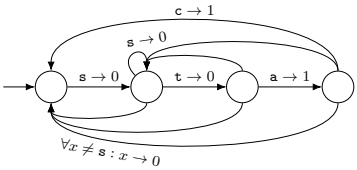


- preprocess q-state transducer in $\mathcal{O}(qn^{\varepsilon})$ time

Some new tools we developed

$$T = \boxed{\texttt{s t a r t s t a c k s a t s t a c s}}$$

$$0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \mathbf{1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \mathbf{1}\ 0$$

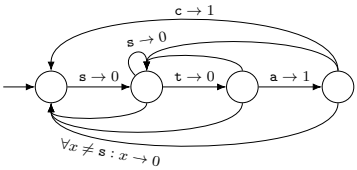


- preprocess q-state transducer in $\mathcal{O}(qn^{\varepsilon})$ time
- run transducer in $\mathcal{O}(n/\log_{\sigma} n)$ time

Some new tools we developed

$$T = \begin{array}{|cccccccccccccccc|}\hline \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{r} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{k} & \mathtt{s} & \mathtt{a} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{s} \\ \hline \end{array}$$

$$\begin{array}{cccccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \end{array}$$

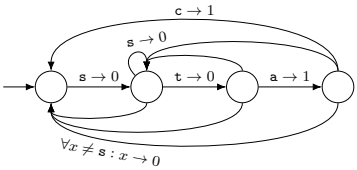


- preprocess q-state transducer in $\mathcal{O}(qn^{\varepsilon})$ time
- run transducer in $\mathcal{O}(n/\log_{\sigma} n)$ time
- also for multiple input/output streams

Some new tools we developed

$$T = \begin{array}{|cccccccccccccccc|}\hline \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{r} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{k} & \mathtt{s} & \mathtt{a} & \mathtt{t} & \mathtt{s} & \mathtt{t} & \mathtt{a} & \mathtt{c} & \mathtt{s} \\ \hline \end{array}$$

$$\begin{array}{cccccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \end{array}$$

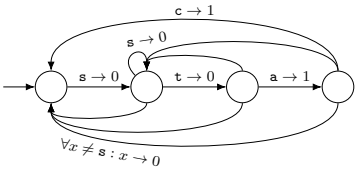


- preprocess q-state transducer in $\mathcal{O}(qn^{\varepsilon})$ time
- run transducer in $\mathcal{O}(n/\log_{\sigma} n)$ time
- also for multiple input/output streams

Custom sparse encoding:

Some new tools we developed

$$T = \boxed{\texttt{s t a r t s t a c k s a t s t a c s}}$$
$$0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \mathbf{1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \mathbf{1}\ 0$$



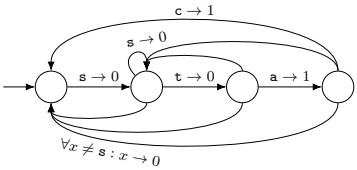
- preprocess q-state transducer in $\mathcal{O}(qn^{\varepsilon})$ time
- run transducer in $\mathcal{O}(n/\log_{\sigma} n)$ time
- also for multiple input/output streams

Custom sparse encoding:

$$T = \boxed{\texttt{0 0 0 0 2 0 0 6 0 0 0 1 0 0 0 0 0 9 0 0 4 4}}$$

Some new tools we developed

$$T = \boxed{\texttt{s t a r t s t a c k s a t s t a c s}}$$
$$0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \mathbf{1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \mathbf{1}\ 0$$



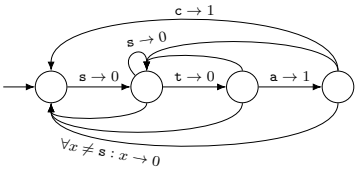
- preprocess q-state transducer in $\mathcal{O}(qn^{\varepsilon})$ time
- run transducer in $\mathcal{O}(n/\log_{\sigma} n)$ time
- also for multiple input/output streams

Custom sparse encoding:

$$T = \boxed{\texttt{0 0 0 0 2 0 0 6 0 0 0 1 0 0 0 0 0 9 0 0 4 4}}$$
$$\underbrace{}_{4}\ \underbrace{}_{2}\ \underbrace{}_{3}\ \underbrace{}_{5}\ \underbrace{}_{2}$$

Some new tools we developed

$$T = \boxed{\texttt{s t a r t s t a c k s a t s t a c s}}$$
$$\texttt{0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0}$$



- preprocess q-state transducer in $\mathcal{O}(qn^{\epsilon})$ time
- run transducer in $\mathcal{O}(n/\log_{\sigma} n)$ time
- also for multiple input/output streams

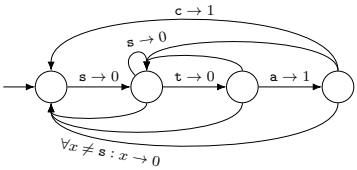
Custom sparse encoding:

$$T = \underbrace{\texttt{0 0 0 0}}_{4}\,\underbrace{\texttt{2 0 0}}_{2}\,\underbrace{\texttt{6 0 0 0}}_{3}\,\underbrace{\texttt{1 0 0 0 0 0}}_{5}\,\underbrace{\texttt{9 0 0}}_{2}\,\texttt{4 4}$$

$$\text{enc}(T) = \texttt{0}\cdot\delta(4)\cdot\texttt{1}\cdot\delta(2)\cdot\texttt{0}\cdot\delta(2)\cdot\texttt{1}\cdot\delta(6)\cdot\texttt{0}\cdot\delta(3)\cdot\texttt{1}\cdot\delta(1)\cdot\texttt{0}\cdot\delta(5)\cdot\texttt{1}\cdot\delta(9)\cdot\texttt{0}\cdot\delta(2)\cdot\texttt{1}\cdot\delta(4)\cdot\texttt{1}\cdot\delta(4)$$

Some new tools we developed

$$T = \boxed{\texttt{s t a r t s t a c k s a t s t a c s}}$$
$$\texttt{0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0}$$



- preprocess q-state transducer in $\mathcal{O}(qn^{\epsilon})$ time
- run transducer in $\mathcal{O}(n/\log_{\sigma} n)$ time or $\mathcal{O}(|\text{enc}(T)|/\log n)$
- also for multiple input/output streams

Custom sparse encoding:

$$T = \boxed{\underbrace{\texttt{0 0 0 0}}_{4}\ \underbrace{\texttt{2 0 0}}_{2}\ \underbrace{\texttt{6 0 0 0}}_{3}\ \underbrace{\texttt{1 0 0 0 0 0}}_{5}\ \underbrace{\texttt{9 0 0}}_{2}\ \texttt{4 4}}$$

$$\text{enc}(T) = \texttt{0}\cdot\delta(4)\cdot\texttt{1}\cdot\delta(2)\cdot\texttt{0}\cdot\delta(2)\cdot\texttt{1}\cdot\delta(6)\cdot\texttt{0}\cdot\delta(3)\cdot\texttt{1}\cdot\delta(1)\cdot\texttt{0}\cdot\delta(5)\cdot\texttt{1}\cdot\delta(9)\cdot\texttt{0}\cdot\delta(2)\cdot\texttt{1}\cdot\delta(4)\cdot\texttt{1}\cdot\delta(4)$$

Conclusion

- string synchronizing sets in $\mathcal{O}(n/\log_\sigma n)$ time for all τ
- truly linear time transducers
- sparse encoding for even faster transducers

Conclusion

- string synchronizing sets in $\mathcal{O}(n/\log_\sigma n)$ time for all τ
- truly linear time transducers
- sparse encoding for even faster transducers

Open questions:

- complexity gap $\mathcal{O}(n/\log_\sigma n)$ vs. $\mathcal{O}(n \log \sigma / \sqrt{\log n})$, see [Kempa+Kociumaka '25]
- faster algorithms for $w \gg \log n$, can we get $\mathcal{O}(n \log \sigma / w)$ time?

References


Bille, Philip (2011). "Fast searching in packed strings". In: *J. Discrete Algorithms* 9.1, pp. 49–56. DOI: 10.1016/J.JDA.2010.09.003.


Ellert, Jonas (2023). "Sublinear Time Lempel-Ziv (LZ77) Factorization". In: *String Processing and Information Retrieval - 30th International Symposium, SPIRE 2023, Pisa, Italy, September 26-28, 2023, Proceedings*. Ed. by Franco Maria Nardini, Nadia Pisanti, and Rossano Venturini. Vol. 14240. Lecture Notes in Computer Science. Springer, pp. 171–187. DOI: 10.1007/978-3-031-43980-3_14.


Farach, Martin (1997). "Optimal Suffix Tree Construction with Large Alphabets". In: *38th Annual Symposium on Foundations of Computer Science, FOCS 1997*. IEEE Computer Society, pp. 137–143. DOI: 10.1109/SFCS.1997.646102.


Hagerup, Torben (1998). "Sorting and Searching on the Word RAM". In: *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1998)*. Vol. 1373. Lecture Notes in Computer Science. Springer, pp. 366–398. DOI: 10.1007/BFb0028578.


Kempa, Dominik and Tomasz Kociumaka (2019). "String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*. Ed. by Moses Charikar and Edith Cohen. ACM, pp. 756–767. DOI: 10.1145/3313276.3316368.


— (2024). "Lempel-Ziv (LZ77) Factorization in Sublinear Time". In: *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*. IEEE, pp. 2045–2055. DOI: 10.1109/FOCS61266.2024.00122.

References


Kempa, Dominik and Tomasz Kociumaka (2025). "On the Hardness Hierarchy for the $O(n\sqrt{\log n})$ Complexity in the Word RAM". In: *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*. Ed. by Michal Koucký and Nikhil Bansal. ACM, pp. 290–300. DOI: 10.1145/3717823.3718291.


Knuth, Donald E., James H. Morris Jr., and Vaughan R. Pratt (1977). "Fast Pattern Matching in Strings". In: *SIAM Journal on Computing* 6.2, pp. 323–350. DOI: 10.1137/0206024.


Kociumaka, Tomasz et al. (2024). "Internal Pattern Matching Queries in a Text and Applications". In: *SIAM J. Comput.* 53.5, pp. 1524–1577. DOI: 10.1137/23M1567618.


Storer, James A. and Thomas G. Szymanski (1982). "Data compression via textual substitution". In: *Journal of the ACM* 29.4, pp. 928–951. DOI: 10.1145/322344.322346.


Ziv, Jacob and Abraham Lempel (1977). "A universal algorithm for sequential data compression". In: *IEEE Transactions on Information Theory* 23.3, pp. 337–343. DOI: 10.1109/TIT.1977.1055714.