Text Indexing

Lecture 11: Suffix Array Construction in Distributed and External Memory

Florian Kurpicz

Recap: Suffix Array and LCP-Array

Karlsruhe Institute of Technology

Definition: Suffix Array [GBS92; MM93]

Given a text T of length n, the **suffix array** (SA) is a permutation of $[1..n]$, such that for $i \leq j \in [1..n]$

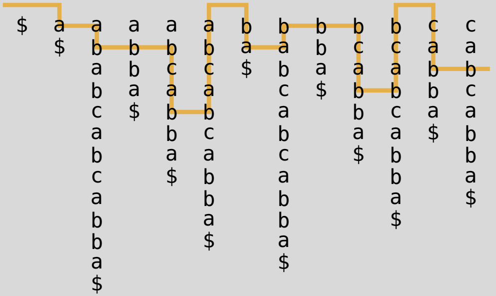
$$T[SA[i]..n] \leq T[SA[j]..n]$$

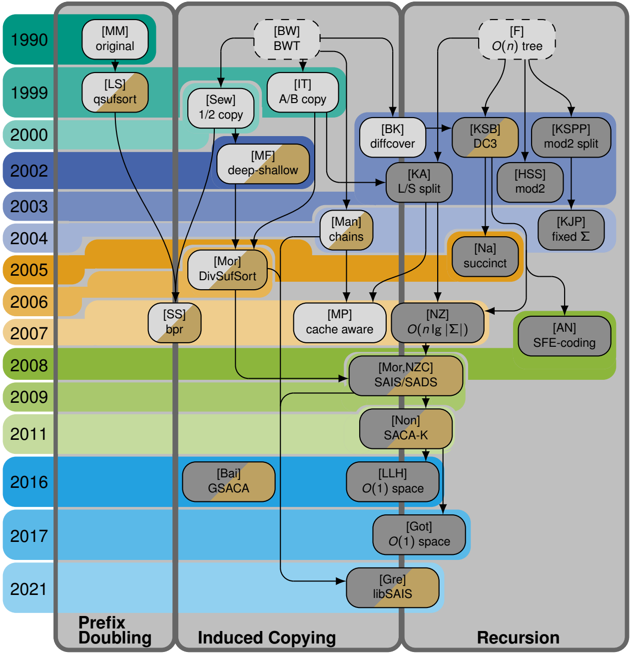
Definition: Longest Common Prefix Array

Given a text T of length n and its SA, the **LCP-array** is defined as

$$LCP[i] = \begin{cases} 0 & i = 1 \\ \max\{\ell \colon T[SA[i]..SA[i]+\ell) = \\ \qquad T[SA[i-1]..SA[i-1]+\ell)\} & i \neq 1 \end{cases}$$

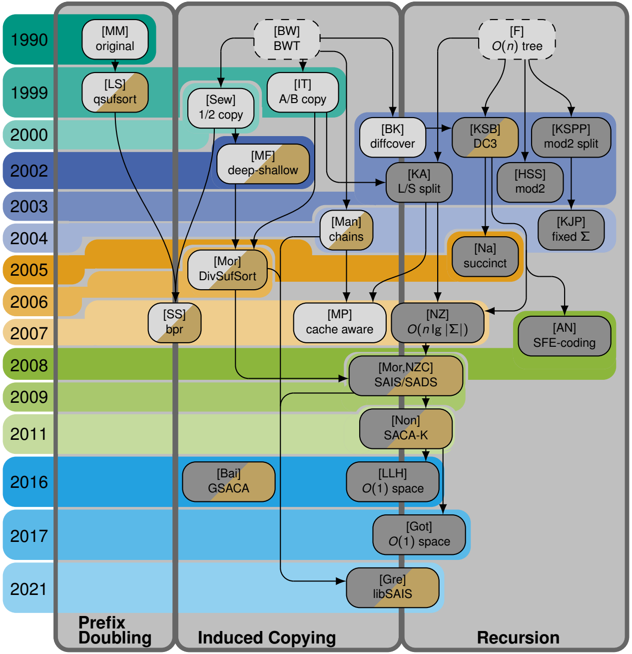
	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
LCP	0	0	1	2	2	5	0	2	1	1	4	0	3





Timeline Sequential Suffix Sorting

- based on [Bah+19; Bin18; Kur20; PST07]
- **darker grey**: linear running time
- **brown**: available implementation

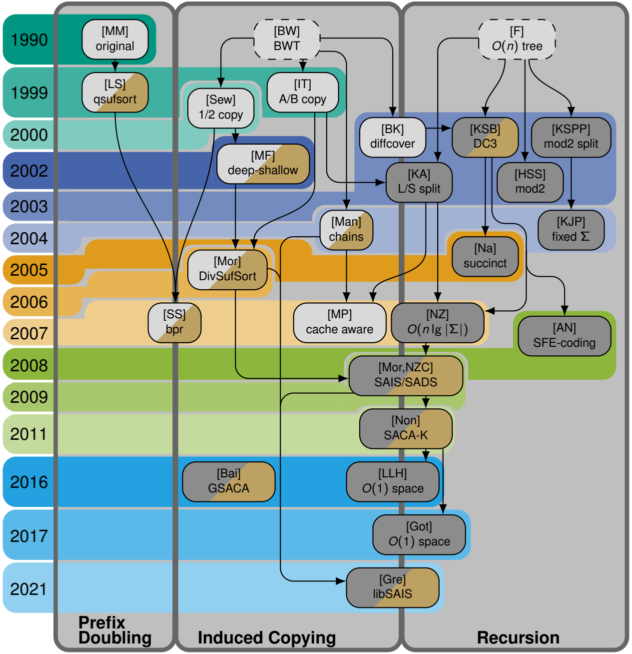


Timeline Sequential Suffix Sorting

- based on [Bah+19; Bin18; Kur20; PST07]
- **darker grey**: linear running time
- **brown**: available implementation

Special Mentions

- DC3 first $O(n)$ algorithm
- $O(n)$ running time and $O(1)$ space for integer alphabets possible

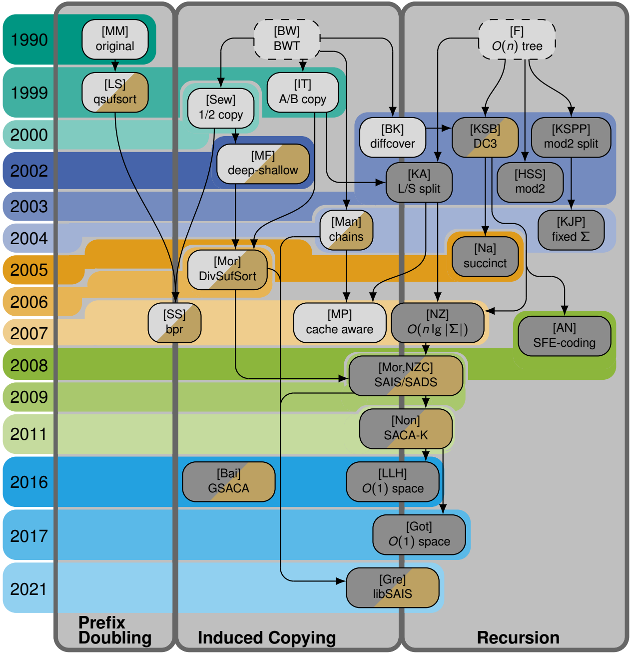


Timeline Sequential Suffix Sorting

- based on [Bah+19; Bin18; Kur20; PST07]
- **darker grey**: linear running time
- **brown**: available implementation

Special Mentions

- DC3 first $O(n)$ algorithm
- $O(n)$ running time and $O(1)$ space for integer alphabets possible
- until 2021: DivSufSort fastest in practice with $O(n \lg n)$ running time



Timeline Sequential Suffix Sorting

- based on [Bah+19; Bin18; Kur20; PST07]
- **darker grey**: linear running time
- **brown**: available implementation

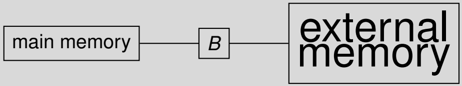
Special Mentions

- DC3 first $O(n)$ algorithm
- $O(n)$ running time and $O(1)$ space for integer alphabets possible
- until 2021: DivSufSort fastest in practice with $O(n \lg n)$ running time
- since 2021: libSAIS fastest in practice with $O(n)$ running time

External and Distributed Memory

External Memory

- internal memory of size M words
- external memory of unlimited size
- transfer of blocks of size B words



- scanning N elements: $\Theta(\frac{N}{B})$
- sorting N elements: $\Theta(\frac{N}{B} \lg_{\frac{M}{B}} \frac{N}{B})$

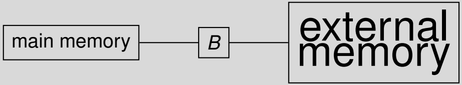
- semi-external memory 

External and Distributed Memory

Karlsruhe Institute of Technology

External Memory

- internal memory of size M words
- external memory of unlimited size
- transfer of blocks of size B words

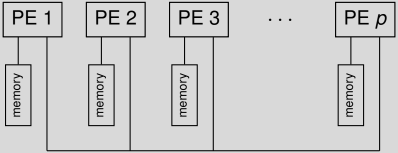


- scanning N elements: $\Theta(\frac{N}{B})$
- sorting N elements: $\Theta(\frac{N}{B} \lg_{\frac{M}{B}} \frac{N}{B})$

- semi-external memory 

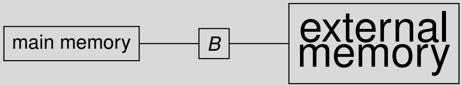
Distributed Memory

- p PEs with internal memory
- communication between PEs over network

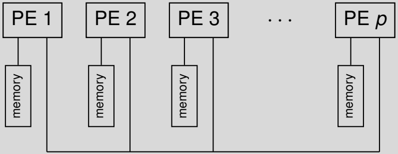


- bulk-synchronous parallel model [Val90]
- supersteps: local work, communication, synchronization

Challenges for Suffix Array Construction



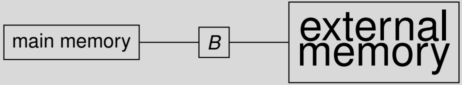
External Memory



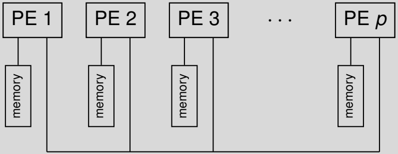
Distributed Memory

- suffixes span over whole input ℹ no locality
- comparing suffixes requires text access
  ℹ random access

Challenges for Suffix Array Construction



External Memory

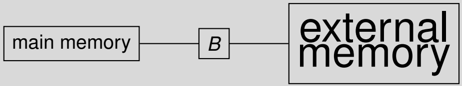


Distributed Memory

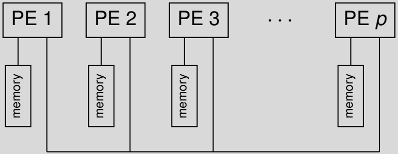
- suffixes span over whole input ℹ no locality
- comparing suffixes requires text access
  ℹ random access

- random access expensive in both models
- whole suffix not available locally in distributed memory

Challenges for Suffix Array Construction



External Memory

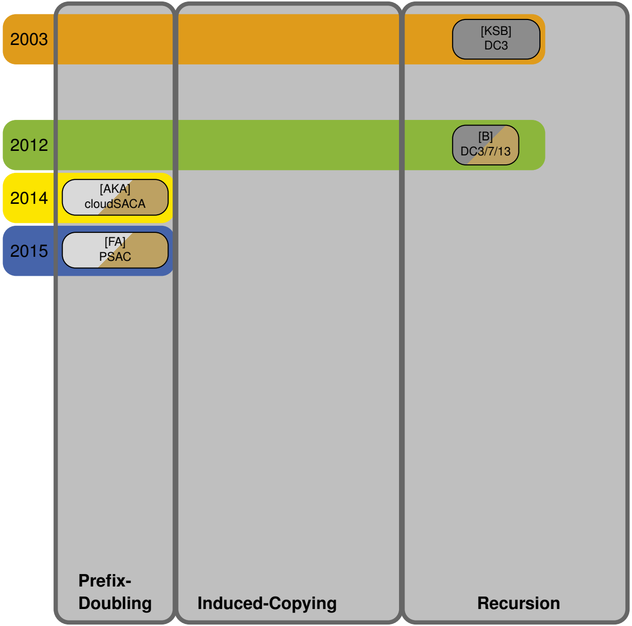


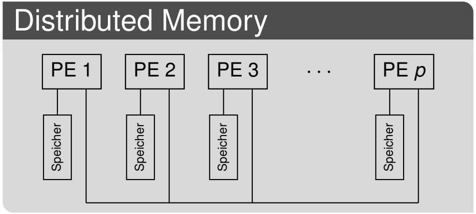
Distributed Memory

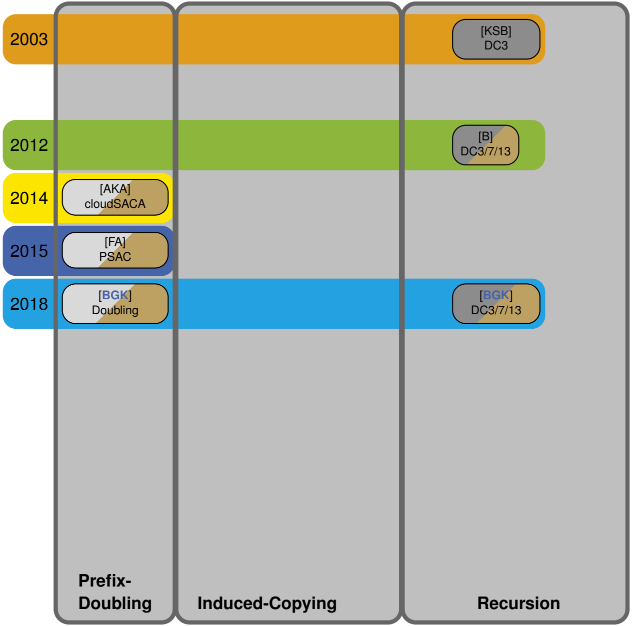
- suffixes span over whole input ℹ no locality
- comparing suffixes requires text access
  ℹ random access

- random access expensive in both models
- whole suffix not available locally in distributed memory

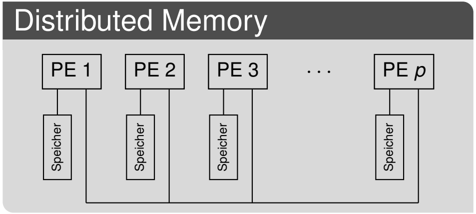
- express suffix array construction algorithm using
  - scanning
  - sorting
  - merging


2003
[KSB]
DC3
2012
[B]
DC3/7/13
2014
[AKA]
cloudSACA
2015
[FA]
PSAC
Prefix-
Doubling
Induced-Copying
Recursion

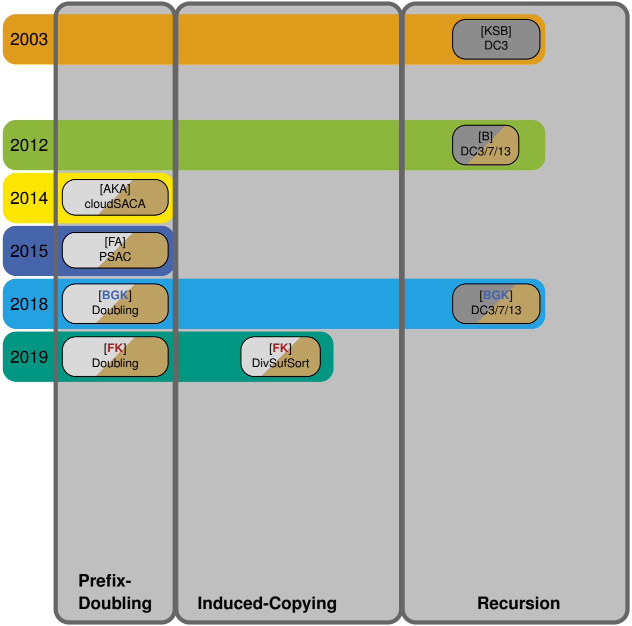

Distributed Memory
PE 1
PE 2
PE 3
...
PE p
Speicher
Speicher
Speicher
Speicher

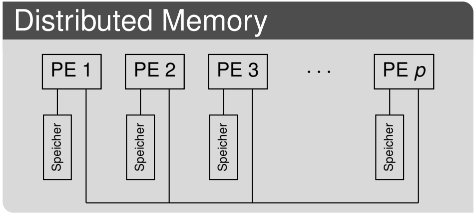


2003
[KSB]
DC3
2012
[B]
DC3/7/13
2014
[AKA]
cloudSACA
2015
[FA]
PSAC
2018
[BGK]
Doubling
[BGK]
DC3/7/13
Prefix-
Doubling
Induced-Copying
Recursion



Distributed Memory
PE 1
PE 2
PE 3
. . .
PE p
Speicher
Speicher
Speicher
Speicher


2003
2012
2014
2015
2018
2019
[KSB]
DC3
[B]
DC3/7/13
[AKA]
cloudSACA
[FA]
PSAC
[BGK]
Doubling
[BGK]
DC3/7/13
[FK]
Doubling
[FK]
DivSufSort
Prefix-
Doubling
Induced-Copying
Recursion


Distributed Memory
PE 1
PE 2
PE 3
. . .
PE p
Speicher
Speicher
Speicher
Speicher

h-Order, h-Groups, and h-Ranks

Definition: h-Order

- ***h*-Order**:
  $T[i..n] \leq_h T[j..n] \iff T[i..i+h) \leq T[j..j+h)$
- SA_h is the suffix array of all suffixes orderd by h-order ℹ not unambigously

h-Order, h-Groups, and h-Ranks

Karlsruhe Institute of Technology

Definition: h-Order

- ***h*-Order**:
  $T[i..n] \leq_h T[j..n] \iff T[i..i+h) \leq T[j..j+h)$
- SA_h is the suffix array of all suffixes orderd by h-order ℹ not unambiguously

Definition: h-Ranks und h-Groups

- all suffixes that are equal w.r.t. an h-order are in an ***h*-group**
- ***h*-rank**: number of lexicographically smaller h-groups plus one

h-Order, *h*-Groups, and *h*-Ranks

Karlsruhe Institute of Technology

Definition: *h*-Order

- ***h*-Order**:
  $T[i..n] \leq_h T[j..n] \iff T[i..i+h) \leq T[j..j+h)$
- SA_h is the suffix array of all suffixes orderd by *h*-order ℹ not unambigously

Definition: *h*-Ranks und *h*-Groups

- all suffixes that are equal w.r.t. an h-order are in an ***h*-group**
- ***h*-rank**: number of lexicographically smaller *h*-groups plus one

m	i	s	s	i	s	s	i	p	p	i	$
i	s	s	i	s	s	i	p	p	i	$	
s	s	i	s	s	i	p	p	i	$		
s	i	s	s	i	p	p	i	$			
i	s	s	i	p	p	i	$				
s	s	i	p	p	i	$					
s	i	p	p	i	$						
i	p	p	i	$							
p	p	i	$								
p	i	$									
i	$										
$											

h-Order, *h*-Groups, and *h*-Ranks

Karlsruhe Institute of Technology

Definition: *h*-Order

- ***h*-Order**:
  $T[i..n] \leq_h T[j..n] \iff T[i..i+h) \leq T[j..j+h)$
- SA_h is the suffix array of all suffixes orderd by *h*-order ℹ not unambigously

Definition: *h*-Ranks und *h*-Groups

- all suffixes that are equal w.r.t. an h-order are in an ***h*-group**
- ***h*-rank**: number of lexicographically smaller *h*-groups plus one

m	i	s	s	i	s	s	i	p	p	i	$
i	s	s	i	s	s	i	p	p	i	$	
s	s	i	s	s	i	p	p	i	$		
s	i	s	s	i	p	p	i	$			
i	s	s	i	p	p	i	$				
s	s	i	p	p	i	$					
s	i	p	p	i	$						
i	p	p	i	$							
p	p	i	$								
p	i	$									
i	$										
$											

h-Order, h-Groups, and h-Ranks

Karlsruhe Institute of Technology

Definition: h-Order

- ***h*-Order**:
  $T[i..n] \leq_h T[j..n] \iff T[i..i+h) \leq T[j..j+h)$
- SA_h is the suffix array of all suffixes orderd by h-order ℹ not unambigously

Definition: h-Ranks und h-Groups

- all suffixes that are equal w.r.t. an h-order are in an ***h*-group**
- ***h*-rank**: number of lexicographically smaller h-groups plus one

m	i	s	s	i	s	s	i	p	p	i	$
i	s	s	i	s	s	i	p	p	i	$	
s	s	i	s	s	i	p	p	i	$		
s	i	s	s	i	p	p	i	$			
i	s	s	i	p	p	i	$				
s	s	i	p	p	i	$					
s	i	p	p	i	$						
i	p	p	i	$							
p	p	i	$								
p	i	$									
i	$										
$											

Prefix-Doubling: The Idea

- 1-rank is the first character

Prefix-Doubling: The Idea

- 1-rank is the first character
- 2-rank can be computed from first 2 characters

Prefix-Doubling: The Idea

- 1-rank is the first character
- 2-rank can be computed from first 2 characters
- 3-rank can be computed from first 3 characters

Prefix-Doubling: The Idea

- 1-rank is the first character
- 2-rank can be computed from first 2 characters
- 3-rank can be computed from first 3 characters
- 4-rank can be computed from first 4 characters

Prefix-Doubling: The Idea

- 1-rank is the first character
- 2-rank can be computed from first 2 characters
- 3-rank can be computed from first 3 characters
- 4-rank can be computed from first 4 characters
- 4-rank can be computed from two 2-ranks

Prefix-Doubling: The Idea

- 1-rank is the first character
- 2-rank can be computed from first 2 characters
- 3-rank can be computed from first 3 characters
- 4-rank can be computed from first 4 characters
- 4-rank can be computed from two 2-ranks

- compute 2^{k+1}-ranks using 2^k-ranks

Prefix-Doubling: Example

1. initial rank is $T[i]$ ℹ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

$$ISA_{2^k}[i] \;\&\; ISA_{2^k}[i + 2^k]$$

4. if all ranks are unique, break
5. compute SA from ISA

```
m i s s i s s i p p i $
i s s i s s i p p i $
s s i s s i p p i $
s i s s i p p i $
i s s i p p i $
s s i p p i $
s i p p i $
i p p i $
p p i $
p i $
i $
$
```

Prefix-Doubling: Example

1. initial rank is $T[i]$ ⓘ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

$$ISA_{2^k}[i] \;\&\; ISA_{2^k}[i+2^k]$$

4. if all ranks are unique, break
5. compute SA from ISA

m	i	s	s	i	s	s	i	p	p	i	$
i	s	s	i	s	s	i	p	p	i	$	
s	s	i	s	s	i	p	p	i	$		
s	i	s	s	i	p	p	i	$			
i	s	s	i	p	p	i	$				
s	s	i	p	p	i	$					
s	i	p	p	i	$						
i	p	p	i	$							
p	p	i	$								
p	i	$									
i	$										
$											

Prefix-Doubling: Example

1. initial rank is $T[i]$ ⓘ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

$$ISA_{2^k}[i] \;\&\; ISA_{2^k}[i + 2^k]$$

4. if all ranks are unique, break
5. compute SA from ISA

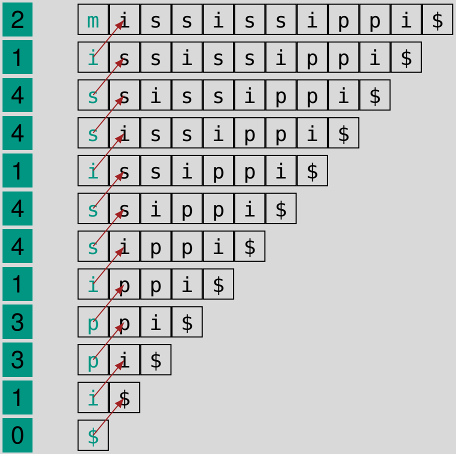
2	m i s s i s s i p p i $
1	i s s i s s i p p i $
4	s s i s s i p p i $
4	s i s s i p p i $
1	i s s i p p i $
4	s s i p p i $
4	s i p p i $
1	i p p i $
3	p p i $
3	p i $
1	i $
0	$

Prefix-Doubling: Example

1. initial rank is $T[i]$ ℹ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

$$ISA_{2^k}[i] \;\&\; ISA_{2^k}[i + 2^k]$$

4. if all ranks are unique, break
5. compute SA from ISA



Prefix-Doubling: Example

1. initial rank is $T[i]$ ℹ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

$$ISA_{2^k}[i] \;\&\; ISA_{2^k}[i + 2^k]$$

4. if all ranks are unique, break
5. compute SA from ISA

2	m i s s i s s i p p i $
1	i s s i s s i p p i $
4	s s i s s i p p i $
4	s i s s i p p i $
1	i s s i p p i $
4	s s i p p i $
4	s i p p i $
1	i p p i $
3	p p i $
3	p i $
1	i $
0	$

Prefix-Doubling: Example

1. initial rank is $T[i]$ ⓘ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

$$ISA_{2^k}[i] \text{ \& } ISA_{2^k}[i + 2^k]$$

4. if all ranks are unique, break
5. compute SA from ISA

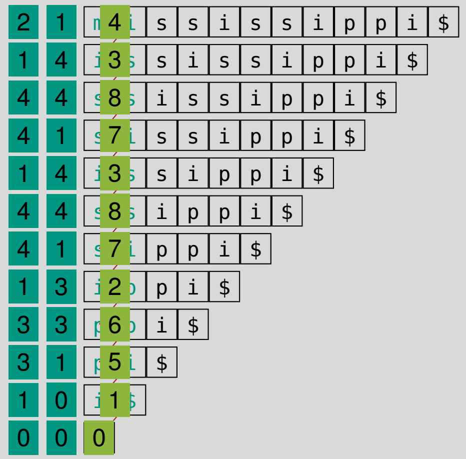
2	1	m	i	s	s	i	s	s	i	p	p	i	$
1	4	i	s	s	i	s	s	i	p	p	i	$	
4	4	s	s	i	s	s	i	p	p	i	$		
4	1	s	i	s	s	i	p	p	i	$			
1	4	i	s	s	i	p	p	i	$				
4	4	s	s	i	p	p	i	$					
4	1	s	i	p	p	i	$						
1	3	i	p	p	i	$							
3	3	p	p	i	$								
3	1	p	i	$									
1	0	i	$										
0	0	$											

Prefix-Doubling: Example

1. initial rank is $T[i]$ ⓘ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

$$ISA_{2^k}[i] \; \& \; ISA_{2^k}[i + 2^k]$$

4. if all ranks are unique, break
5. compute SA from ISA



Prefix-Doubling: Example

1. initial rank is $T[i]$ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

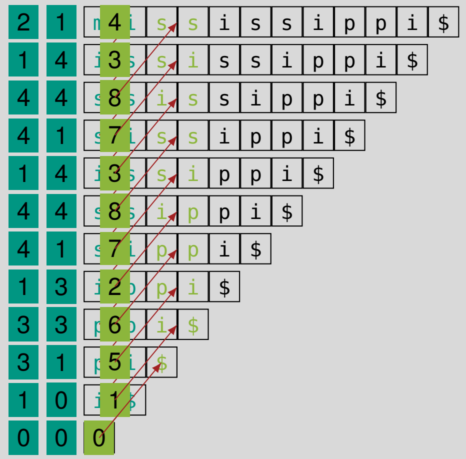
$$ISA_{2^k}[i] \;\&\; ISA_{2^k}[i + 2^k]$$

4. if all ranks are unique, break
5. compute SA from ISA

Prefix-Doubling: Example

Karlsruhe Institute of Technology

1. initial rank is $T[i]$ ℹ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3.   new 2^{k+1}-ranks based on
   $$ISA_{2^k}[i] \;\&\; ISA_{2^k}[i + 2^k]$$
4.   if all ranks are unique, break
5. compute SA from ISA



Prefix-Doubling: Example

1. initial rank is $T[i]$ ⓘ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

$$ISA_{2^k}[i] \;\&\; ISA_{2^k}[i + 2^k]$$

4. if all ranks are unique, break
5. compute SA from ISA

Prefix-Doubling: Example

1. initial rank is $T[i]$ ℹ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

$$ISA_{2^k}[i] \ \& \ ISA_{2^k}[i + 2^k]$$

4. if all ranks are unique, break
5. compute SA from ISA

Prefix-Doubling: Example

1. initial rank is $T[i]$ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

$$ISA_{2^k}[i] \;\&\; ISA_{2^k}[i+2^k]$$

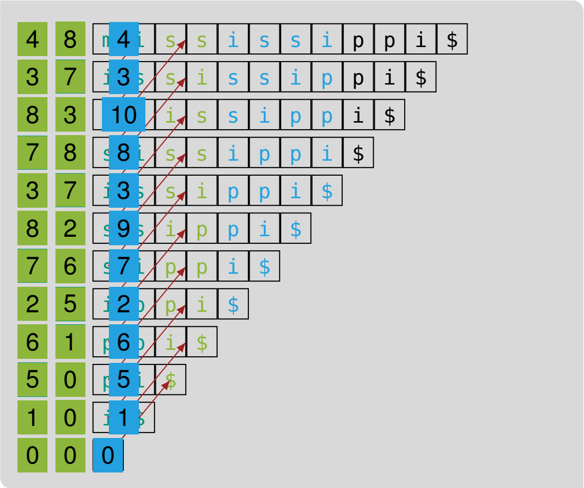
4. if all ranks are unique, break
5. compute SA from ISA

Prefix-Doubling: Example

1. initial rank is $T[i]$ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

$$ISA_{2^k}[i] \; \& \; ISA_{2^k}[i + 2^k]$$

4. if all ranks are unique, break
5. compute SA from ISA



Prefix-Doubling: Example

1. initial rank is $T[i]$ ℹ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

$$ISA_{2^k}[i] \;\&\; ISA_{2^k}[i + 2^k]$$

4. if all ranks are unique, break
5. compute SA from ISA

Prefix-Doubling: Example

1. initial rank is $T[i]$ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

$$ISA_{2^k}[i] \;\&\; ISA_{2^k}[i + 2^k]$$

4. if all ranks are unique, break
5. compute SA from ISA

Prefix-Doubling: Example

1. initial rank is $T[i]$ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

$$ISA_{2^k}[i] \;\&\; ISA_{2^k}[i + 2^k]$$

4. if all ranks are unique, break
5. compute SA from ISA

Prefix-Doubling: Example

1. initial rank is $T[i]$ ℹ 1-rank
2. for $k = 0$ to $\lceil \lg n \rceil$
3. new 2^{k+1}-ranks based on

$$ISA_{2^k}[i] \;\&\; ISA_{2^k}[i + 2^k]$$

4. if all ranks are unique, break
5. compute SA from ISA

Simple Algorithm

- N. Jesper Larsson and Kunihiko Sadakane. “Faster Suffix Sorting”. In: *Theor. Comput. Sci.* 387.3 (2007), pages 258–272. DOI: `10.1016/j.tcs.2007.07.017`

Prefix-Doubling: Practical Approaches

Use ISA_h [FA15]

- use ISA_{2^k} to compute rank tuples
- for position i use rank $ISA_{2^k}[i + 2^k]$
- if $i + 2^k > n$, second rank is 0
- example on the board

Prefix-Doubling: Practical Approaches

Use ISA_h [FA15]

- use ISA_{2^k} to compute rank tuples
- for position i use rank $ISA_{2^k}[i + 2^k]$
- if $i + 2^k > n$, second rank is 0
- example on the board

Sort by Text Positions [Dem+08; FK19]

- especially good if access to ISA_h is expensive
- sort tuples (Textposition i, Rang r)
- using $(i, r) \leq (j, r')$ iff

$$(i \bmod 2^k, \lfloor i/2^k \rfloor) < (j \bmod 2^k, \lfloor j/2^k \rfloor)$$

- example on the board

Prefix-Doubling: Running Time

- running time: $O(n \lg n)$
- memory requirements: $8n(+n)$ words ⓘ for texts $\leq 4\,\text{GiB}$
- worst-case input: $T = a^{n-1}\$$

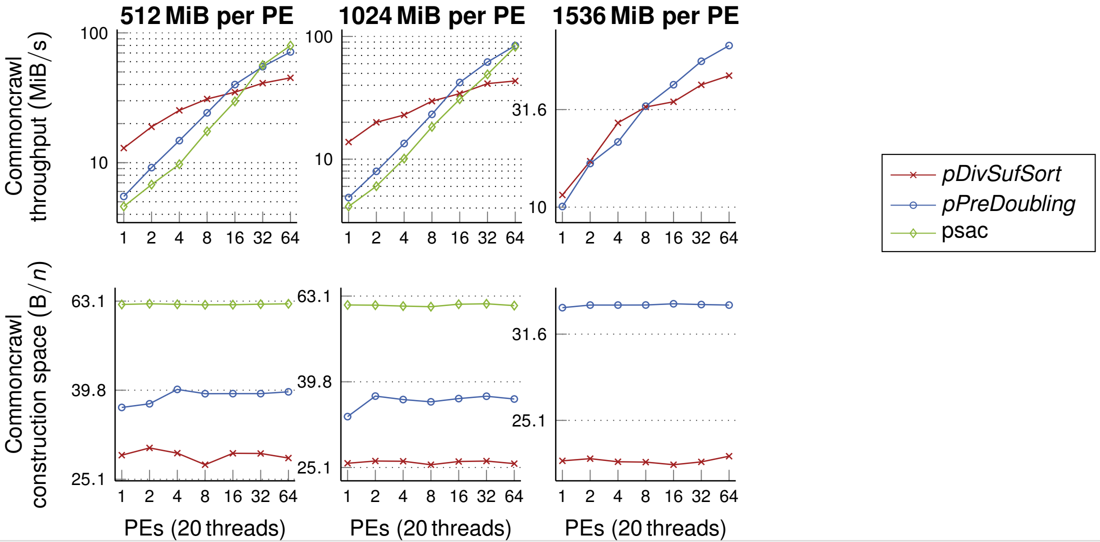
Prefix-Doubling: Running Time

- running time: $O(n \lg n)$
- memory requirements: $8n(+n)$ words for texts ≤ 4 GiB
- worst-case input: $T = a^{n-1}\$$

Generalization

- more than doubling is possible
- compute α^{k+1}-ranks using α α^k-ranks
- can save I/Os in EM $\alpha = 4$ requires 30 % less I/Os than $\alpha = 2$ [Dem+08]

Prefix Doubling: Experimental Results [Kur20]



Recap: SAIS

The Idea: Inducing

Given a text T of length n and two positions $i, j \in [1..n]$ with $T[i] = T[j]$, then

$$T[i..n] < T[j..n] \iff T[i+1..n] < T[j+1..n]$$

Recap: SAIS

The Idea: Inducing

Given a text T of length n and two positions $i, j \in [1..n]$ with $T[i] = T[j]$, then

$$T[i..n] < T[j..n] \iff T[i+1..n] < T[j+1..n]$$

a	α

a	β

Recap: SAIS

The Idea: Inducing

Given a text T of length n and two positions $i, j \in [1..n]$ with $T[i] = T[j]$, then

$$T[i..n] < T[j..n] \iff T[i+1..n] < T[j+1..n]$$

a	α

a	β

The Algorithm: SAIS

- using inducing for everything
- described in [NZC11]

Recap: SAIS

The Idea: Inducing

Given a text T of length n and two positions $i, j \in [1..n]$ with $T[i] = T[j]$, then

$$T[i..n] < T[j..n] \iff T[i+1..n] < T[j+1..n]$$

a	α
a	β

The Algorithm: SAIS

- using inducing for everything
- described in [NZC11]

Suffix Array Construction in 3 Phases

- classification
- sort special substrings/suffixes recursively
- induce all non-sorted suffixes

Recap: SAIS

The Idea: Inducing

Given a text T of length n and two positions $i, j \in [1..n]$ with $T[i] = T[j]$, then

$$T[i..n] < T[j..n] \iff T[i+1..n] < T[j+1..n]$$

a	α

a	β

The Algorithm: SAIS

- using inducing for everything
- described in [NZC11]

Suffix Array Construction in 3 Phases

- classification
- sort special substrings/suffixes recursively
- induce all non-sorted suffixes

- classification helps identifying special suffixes
- everything in linear time

SAIS in External Memory [BFO16; Kär+17]

Classification

- simple scan of the text
- works well in external memory

Sort Special Substrings

- recursion
- works well in external memory if rest works well

Inducing

- keep buffer for each α-interval of suffix array
- scan text and induce characters by writing them in buffer

- separate text during classification
- blockwise preinducing
- heavily relies on external memory priority queue

Jack of all Trades: DC3

- first direct linear time suffix array construction algorithm: DC3
- suffix tree construction algorithm with similar idea [Far97]
- Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. “Linear work suffix array construction”. In: *J. ACM* 53.6 (2006), pages 918–936. DOI: `10.1145/1217856.1217858`
- based on Difference Cover

Difference Cover

Karlsruhe Institute of Technology

Definition: Difference Cover

The set $D \subseteq [0, \nu)$ is a **difference cover** modulo ν, if

$$\{(i - j) \mod \nu \colon i, j \in D\} = [0, \nu)$$

- $\{0, 1\}$ is difference cover modulo 3
- $\{0, 1, 3\}$ is difference cover modulo 7
- $\{0, 1, 3, 9\}$ is difference cover modulo 13

Difference Cover

Karlsruhe Institute of Technology

Definition: Difference Cover

The set $D \subseteq [0, \nu)$ is a **difference cover** modulo ν, if

$$\{(i - j) \mod \nu \colon i, j \in D\} = [0, \nu)$$

- $\{0, 1\}$ is difference cover modulo 3
- $\{0, 1, 3\}$ is difference cover modulo 7
- $\{0, 1, 3, 9\}$ is difference cover modulo 13

- $0 \equiv 0 - 0 \pmod 3$
- $1 \equiv 1 - 0 \pmod 3$
- $2 \equiv 0 - 1 \pmod 3$

Difference Cover

Definition: Difference Cover

The set $D \subseteq [0, \nu)$ is a **difference cover** modulo ν, if

$$\{(i - j) \mod \nu \colon i, j \in D\} = [0, \nu)$$

- $\{0, 1\}$ is difference cover modulo 3
- $\{0, 1, 3\}$ is difference cover modulo 7
- $\{0, 1, 3, 9\}$ is difference cover modulo 13

- $0 \equiv 0 - 0 \pmod 3$
- $1 \equiv 1 - 0 \pmod 3$
- $2 \equiv 0 - 1 \pmod 3$

- $0 \equiv 0 - 0 \pmod 7$
- $1 \equiv 1 - 0 \pmod 7$
- $2 \equiv 3 - 1 \pmod 7$
- $3 \equiv 3 - 0 \pmod 7$
- $4 \equiv 0 - 3 \pmod 7$
- $5 \equiv 1 - 3 \pmod 7$
- $6 \equiv 0 - 1 \pmod 7$

Suffix Array Construction with DC3 (1/6)

1. Sample Suffixes

- for $i \in \{0,1,2\}$ let be

$$B_k = \{i \in [0,n) \colon i \mod 3 = k\}$$

- $C = B_0 \cdot B_1$
  - $\{0,1\}$ is difference cover modulo 3

0	1	2	3	4	5	6	7	8	9	10	11
m	i	s	s	i	s	s	i	p	p	i	$

Suffix Array Construction with DC3 (1/6)

1. Sample Suffixes

- for $i \in \{0,1,2\}$ let be

$$B_k = \{i \in [0,n) \colon i \mod 3 = k\}$$

- $C = B_0 \cdot B_1$
  - {0, 1} is difference cover modulo 3

0	1	2	3	4	5	6	7	8	9	10	11
m	i	s	s	i	s	s	i	p	p	i	$

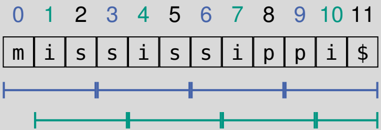
Suffix Array Construction with DC3 (1/6)

1. Sample Suffixes

- for $i \in \{0, 1, 2\}$ let be

$$B_k = \{i \in [0, n) \colon i \mod 3 = k\}$$

- $C = B_0 \cdot B_1$
  - $\{0, 1\}$ is difference cover modulo 3



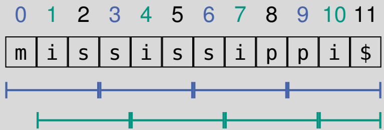
Suffix Array Construction with DC3 (1/6)

1. Sample Suffixes

- for $i \in \{0, 1, 2\}$ let be

$$B_k = \{i \in [0, n) \colon i \mod 3 = k\}$$

- $C = B_0 \cdot B_1$
  - $\{0, 1\}$ is difference cover modulo 3



- $C = \{0, 3, 6, 9, 1, 4, 7, 10\}$

Suffix Array Construction with DC3 (2/6)

2. Sort Sampled Suffixes

- for $k = 0, 1$ let be

$$R_k = [T[k]T[k+1]T[k+2]][T[k+3]T[k+4]T[k+5]]\ldots[T[\max B_k]T[\max B_k+1]T[\max B_k+2]]$$

- $R = R_0 \cdot R_1$
- sort R with Radix Sort in $O(n)$ time
- all characters unique: ranks of sampled suffixes are known
- otherwise: recursively execute algorithm on R

Suffix Array Construction with DC3 (2/6)

2. Sort Sampled Suffixes

- for $k = 0, 1$ let be

$$R_k = [T[k]T[k+1]T[k+2]][T[k+3]T[k+4]T[k+5]] \dots [T[\max B_k]T[\max B_k + 1]T[\max B_k + 2]]$$

- $R = R_0 \cdot R_1$
- sort R with Radix Sort in $O(n)$ time
- all characters unique: ranks of sampled suffixes are known
- otherwise: recursively execute algorithm on R

0	1	2	3	4	5	6	7
$[mis]$	$[sis]$	$[sip]$	$[pi\$]$	$[iss]$	$[iss]$	$[ipp]$	$[i\$\$]$
3	6	5	4	2	2	1	0

Suffix Array Construction with DC3 (2/6)

2. Sort Sampled Suffixes

- for $k = 0, 1$ let be

$$R_k = [T[k]T[k+1]T[k+2]][T[k+3]T[k+4]T[k+5]]\dots[T[\max B_k]T[\max B_k+1]T[\max B_k+2]]$$

- $R = R_0 \cdot R_1$
- sort R with Radix Sort in $O(n)$ time
- all characters unique: ranks of sampled suffixes are known
- otherwise: recursively execute algorithm on R

0	1	2	3	4	5	6	7
[*mis*]	[*sis*]	[*sip*]	[*pi*\$]	[*iss*]	[*iss*]	[*ipp*]	[*i*\$\$]
3	6	5	4	2	2	1	0

Suffix Array Construction with DC3 (3/6)

Recursion: Step 1

0	1	2	3	4	5	6	7
3	6	5	4	2	2	1	0

Suffix Array Construction with DC3 (3/6)

Recursion: Step 1

0	1	2	3	4	5	6	7
3	6	5	4	2	2	1	0

Suffix Array Construction with DC3 (3/6)

Recursion: Step 1

0	1	2	3	4	5	6	7
3	6	5	4	2	2	1	0

Suffix Array Construction with DC3 (3/6)

Recursion: Step 1

0	1	2	3	4	5	6	7
3	6	5	4	2	2	1	0

- $C = \{0, 3, 6, 1, 4, 7\}$

Suffix Array Construction with DC3 (3/6)

Recursion: Step 1

0	1	2	3	4	5	6	7
3	6	5	4	2	2	1	0

- $C = \{0, 3, 6, 1, 4, 7\}$

Recursion: Step 2

0	1	2	3	4	5
[365]	[422]	[100]	[654]	[221]	[000]
3	4	1	5	2	0

Suffix Array Construction with DC3 (4/6)

Karlsruhe Institute of Technology

3. Sort Non-Sampled Suffixes

- let $i, j \in B_2$, then

$$S_i \leq S_j \iff (T[i], Rang(S_{i+1})) \leq (T[j], Rang(S_{j+1}))$$

- ranks of next two suffixes is known
- sort tuples (in B_2) using Radix Sort
- $O(n)$ time

Suffix Array Construction with DC3 (4/6)

3. Sort Non-Sampled Suffixes

- let $i, j \in B_2$, then

$$S_i \leq S_j \iff (T[i], Rang(S_{i+1})) \leq (T[j], Rang(S_{j+1}))$$

- ranks of next two suffixes is known
- sort tuples (in B_2) using Radix Sort
- $O(n)$ time

	0	1	2	3	4	5	6	7
	3	6	5	4	2	2	1	0
ranks	3	5	$\perp$	4	2	$\perp$	1	0

Suffix Array Construction with DC3 (4/6)

3. Sort Non-Sampled Suffixes

- let $i, j \in B_2$, then

$$S_i \leq S_j \iff (T[i], Rang(S_{i+1})) \leq (T[j], Rang(S_{j+1}))$$

- ranks of next two suffixes is known
- sort tuples (in B_2) using Radix Sort
- $O(n)$ time

	0	1	2	3	4	5	6	7
	3	6	5	4	2	2	1	0
ranks	3	5	$\perp$	4	2	$\perp$	1	0

Suffix Array Construction with DC3 (4/6)

3. Sort Non-Sampled Suffixes

- let $i, j \in B_2$, then

$$S_i \leq S_j \iff (T[i], Rang(S_{i+1})) \leq (T[j], Rang(S_{j+1}))$$

- ranks of next two suffixes is known
- sort tuples (in B_2) using Radix Sort
- $O(n)$ time

	0	1	2	3	4	5	6	7
	3	6	5	4	2	2	1	0
ranks	3	5	$\perp$	4	2	$\perp$	1	0

- $\underbrace{(2, 1)}_{S_2} \leq \underbrace{(5, 4)}_{S_5}$

Suffix Array Construction with DC3 (5/6)

4. Merge Suffixes

- let $i \in C$ and $j \in B_2$, then
  - if $i \in B_0$, then
    $S_i \leq S_j \iff$
    $(T[i], Rang(S_{i+1})) \leq (T[j], Rang(S_{j+1}))$
  - if $i \in B_1$, then
    $S_i \leq S_j \iff$
    $(T[i], T[i+1], Rang(S_{i+2})) \leq$
    $(T[j], T[j+1], Rang(S_{j+2}))$

Suffix Array Construction with DC3 (5/6)

4. Merge Suffixes

- let $i \in C$ and $j \in B_2$, then
  - if $i \in B_0$, then
    $S_i \le S_j \iff$
    $(T[i], Rang(S_{i+1})) \le (T[j], Rang(S_{j+1}))$
  - if $i \in B_1$, then
    $S_i \le S_j \iff$
    $(T[i], T[i+1], Rang(S_{i+2})) \le$
    $(T[j], T[j+1], Rang(S_{j+2}))$

	0	1	2	3	4	5	6	7
	3	6	5	4	2	2	1	0
ranks	3	5	$\perp$	4	2	$\perp$	1	0

Suffix Array Construction with DC3 (5/6)

4. Merge Suffixes

- let $i \in C$ and $j \in B_2$, then
  - if $i \in B_0$, then
    $S_i \leq S_j \iff$
    $(T[i], Rang(S_{i+1})) \leq (T[j], Rang(S_{j+1}))$
  - if $i \in B_1$, then
    $S_i \leq S_j \iff$
    $(T[i], T[i+1], Rang(S_{i+2})) \leq$
    $(T[j], T[j+1], Rang(S_{j+2}))$

	0	1	2	3	4	5	6	7
	3	6	5	4	2	2	1	0
ranks	3	5	$\perp$	4	2	$\perp$	1	0

Suffix Array Construction with DC3 (5/6)

4. Merge Suffixes

- let $i \in C$ and $j \in B_2$, then
  - if $i \in B_0$, then
    $S_i \leq S_j \iff$
    $(T[i], Rang(S_{i+1})) \leq (T[j], Rang(S_{j+1}))$
  - if $i \in B_1$, then
    $S_i \leq S_j \iff$
    $(T[i], T[i+1], Rang(S_{i+2})) \leq$
    $(T[j], T[j+1], Rang(S_{j+2}))$

	0	1	2	3	4	5	6	7
	3	6	5	4	2	2	1	0
ranks	3	5	$\perp$	4	2	$\perp$	1	0

- $\underbrace{(2,1)}_{S_2} \leq \underbrace{(5,4)}_{S_5}$

Suffix Array Construction with DC3 (5/6)

4. Merge Suffixes

- let $i \in C$ and $j \in B_2$, then
  - if $i \in B_0$, then
    $S_i \leq S_j \iff$
    $(T[i], Rang(S_{i+1})) \leq (T[j], Rang(S_{j+1}))$
  - if $i \in B_1$, then
    $S_i \leq S_j \iff$
    $(T[i], T[i+1], Rang(S_{i+2})) \leq$
    $(T[j], T[j+1], Rang(S_{j+2}))$

	0	1	2	3	4	5	6	7
	3	6	5	4	2	2	1	0
ranks	3	5	$\perp$	4	2	$\perp$	1	0

- $\underbrace{(2,1)}_{S_2} \leq \underbrace{(5,4)}_{S_5}$

- $(0,0,0) \leq (2,0,0)$

Suffix Array Construction with DC3 (5/6)

4. Merge Suffixes

- let $i \in C$ and $j \in B_2$, then
  - if $i \in B_0$, then
    $S_i \leq S_j \iff$
    $(T[i], Rang(S_{i+1})) \leq (T[j], Rang(S_{j+1}))$
  - if $i \in B_1$, then
    $S_i \leq S_j \iff$
    $(T[i], T[i+1], Rang(S_{i+2})) \leq$
    $(T[j], T[j+1], Rang(S_{j+2}))$

	0	1	2	3	4	5	6	7
	3	6	5	4	2	2	1	0
ranks	3	5	$\perp$	4	2	$\perp$	1	0

- $\underbrace{(2,1)}_{S_2} \leq \underbrace{(5,4)}_{S_5}$

- $(0,0,0) \leq (2,0,0)$
- $(1,0) \leq (2,1)$

Suffix Array Construction with DC3 (5/6)

4. Merge Suffixes

- let $i \in C$ and $j \in B_2$, then
  - if $i \in B_0$, then
    $S_i \leq S_j \iff$
    $(T[i], Rang(S_{i+1})) \leq (T[j], Rang(S_{j+1}))$
  - if $i \in B_1$, then
    $S_i \leq S_j \iff$
    $(T[i], T[i+1], Rang(S_{i+2})) \leq$
    $(T[j], T[j+1], Rang(S_{j+2}))$

	0	1	2	3	4	5	6	7
	3	6	5	4	2	2	1	0
ranks	3	5	$\perp$	4	2	$\perp$	1	0

- $\underbrace{(2, 1)}_{S_2} \leq \underbrace{(5, 4)}_{S_5}$

- $(0, 0, 0) \leq (2, 0, 0)$
- $(1, 0) \leq (2, 1)$
- $(2, 1, 0) \leq (2, 2, 1)$
- ...

Suffix Array Construction with DC3 (5/6)

Karlsruhe Institute of Technology

4. Merge Suffixes

- let $i \in C$ and $j \in B_2$, then
  - if $i \in B_0$, then
    $S_i \leq S_j \iff$
    $(T[i], Rang(S_{i+1})) \leq (T[j], Rang(S_{j+1}))$
  - if $i \in B_1$, then
    $S_i \leq S_j \iff$
    $(T[i], T[i+1], Rang(S_{i+2})) \leq$
    $(T[j], T[j+1], Rang(S_{j+2}))$

	0	1	2	3	4	5	6	7
	3	6	5	4	2	2	1	0
ranks	3	5	$\perp$	4	2	$\perp$	1	0

- $\underbrace{(2,1)}_{S_2} \leq \underbrace{(5,4)}_{S_5}$

- $(0,0,0) \leq (2,0,0)$
- $(1,0) \leq (2,1)$
- $(2,1,0) \leq (2,2,1)$
- ...
- ranks: 4 7 6 5 3 2 1 0

Suffix Array Construction with DC3 (6/6)

Finish Recursion

0	1	2	3	4	5	6	7
[*mis*]	[*sis*]	[*sip*]	[*pi*$]	[*iss*]	[*iss*]	[*ipp*]	[*i*$$]
4	7	6	5	3	2	1	0

Suffix Array Construction with DC3 (6/6)

Finish Recursion

0	1	2	3	4	5	6	7
[*mis*]	[*sis*]	[*sip*]	[*pi$*]	[*iss*]	[*iss*]	[*ipp*]	[*i$$*]
4	7	6	5	3	2	1	0

	0	1	2	3	4	5	6	7	8	9	10	11
	m	i	s	s	i	s	s	i	p	p	i	$
ranks	4	3	$\perp$	7	2	$\perp$	6	1	$\perp$	5	0	$\perp$

Suffix Array Construction with DC3 (6/6)

Finish Recursion

0	1	2	3	4	5	6	7
[mis]	*[sis]*	*[sip]*	*[pi$]*	*[iss]*	*[iss]*	*[ipp]*	*[i$$]*
4	7	6	5	3	2	1	0

	0	1	2	3	4	5	6	7	8	9	10	11
	m	i	s	s	i	s	s	i	p	p	i	$
ranks	4	3	$\perp$	7	2	$\perp$	6	1	$\perp$	5	0	$\perp$

- rest can be used as exercise solution: 11 10 7 4 1 0 9 8 6 3 5 2

DC3: Running Times

- everything but recursion obviously in $O(n)$ time
- only sorting tuples of size ≤ 3
- Radix Sort in $O(n)$ time

DC3: Running Times

- everything but recursion obviously in $O(n)$ time
- only sorting tuples of size ≤ 3
- Radix Sort in $O(n)$ time

- recursion on texts of size $\lceil 2n/3 \rceil$
- $T(n) = T(2n/3) + O(n) = O(n)$

DC3: Running Times

- everything but recursion obviously in $O(n)$ time
- only sorting tuples of size ≤ 3
- Radix Sort in $O(n)$ time

- recursion on texts of size $\lceil 2n/3 \rceil$
- $T(n) = T(2n/3) + O(n) = O(n)$

Generalization

- works with every difference cover
- sorting somewhat more complicated
- running time: $O(\nu n)$

DC3: Running Times

- everything but recursion obviously in $O(n)$ time
- only sorting tuples of size ≤ 3
- Radix Sort in $O(n)$ time

- recursion on texts of size $\lceil 2n/3 \rceil$
- $T(n) = T(2n/3) + O(n) = O(n)$

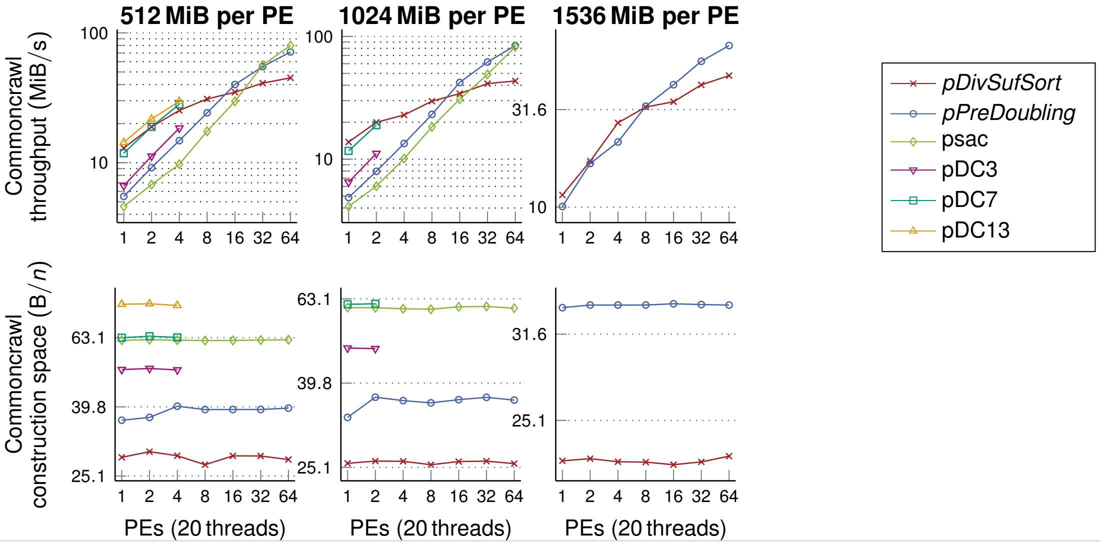
Generalization

- works with every difference cover
- sorting somewhat more complicated
- running time: $O(\nu n)$

In Other Models of Computation

- external memory: $O(\frac{n}{DB} \lg_{\frac{M}{B}} \frac{n}{B})$ using D disks
- BSP: $O(\frac{n \lg n}{P} + L \lg^2 P + g \frac{n \lg n}{P \lg(n/P)})$ using P PEs
- EREW-PRAM: $O(\lg^2 n)$ time and $O(n \lg n)$ work

Prefix Doubling: Experimental Results [Kur20]

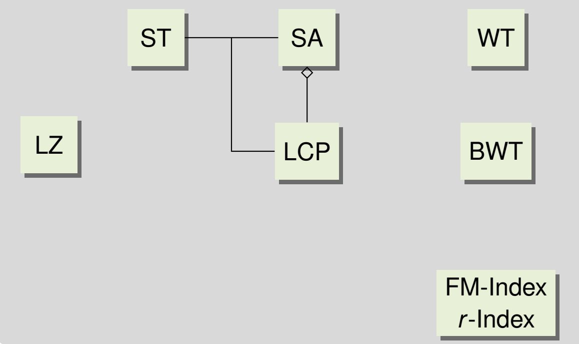


Conclusion and Outlook

This Lecture

- distributed and external memory suffix sorting
- more suffix sorting techniques

Linear Time Construction



Conclusion and Outlook

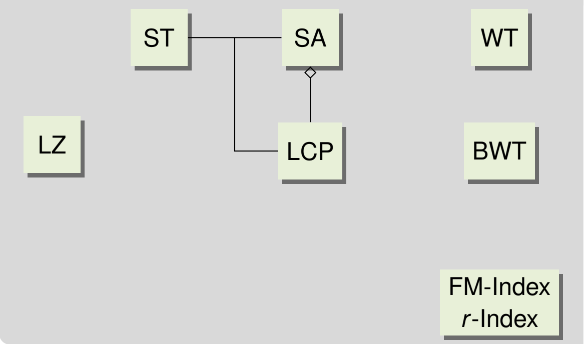
This Lecture

- distributed and external memory suffix sorting
- more suffix sorting techniques

Next Lecture

- move data structure (???)

Linear Time Construction



Evaluation

https://onlineumfrage.kit.edu/evasys/online.php?p=K2FFL

Bibliography I

[Bah+19] Johannes Bahne, Nico Bertram, Marvin Böcker, Jonas Bode, Johannes Fischer, Hermann Foot, Florian Grieskamp, Florian Kurpicz, Marvin Löbel, Oliver Magiera, Rosa Pink, David Piper, and Christopher Poeplau. “SACABench: Benchmarking Suffix Array Construction”. In: *SPIRE*. Volume 11811. Lecture Notes in Computer Science. Springer, 2019, pages 407–416. DOI: 10.1007/978-3-030-32686-9_29.

[BFO16] Timo Bingmann, Johannes Fischer, and Vitaly Osipov. “Inducing Suffix and LCP Arrays in External Memory”. In: *ACM J. Exp. Algorithmics* 21.1 (2016), 2.3:1–2.3:27. DOI: 10.1145/2975593.

[Bin18] Timo Bingmann. “Scalable String and Suffix Sorting: Algorithms, Techniques, and Tools”. PhD thesis. Karlsruhe Institute of Technology, Germany, 2018. DOI: 10.5445/IR/1000085031.

[Dem+08] Roman Dementiev, Juha Kärkkäinen, Jens Mehnert, and Peter Sanders. “Better External Memory Suffix Array Construction”. In: *ACM J. Exp. Algorithmics* 12 (2008), 3.4:1–3.4:24. DOI: 10.1145/1227161.1402296.

Bibliography II

[FA15] Patrick Flick and Srinivas Aluru. "Parallel Distributed Memory Construction of Suffix and Longest Common Prefix Arrays". In: *SC*. ACM, 2015, 16:1–16:10. DOI: `10.1145/2807591.2807609`.

[Far97] Martin Farach. "Optimal Suffix Tree Construction with Large Alphabets". In: *FOCS*. IEEE Computer Society, 1997, pages 137–143. DOI: `10.1109/SFCS.1997.646102`.

[FK19] Johannes Fischer and Florian Kurpicz. "Lightweight Distributed Suffix Array Construction". In: *ALENEX*. SIAM, 2019, pages 27–38. DOI: `10.1137/1.9781611975499.3`.

[GBS92] Gaston H. Gonnet, Ricardo A. Baeza-Yates, and Tim Snider. "New Indices for Text: Pat Trees and Pat Arrays". In: *Information Retrieval: Data Structures & Algorithms*. Prentice-Hall, 1992, pages 66–82.

[Kär+17] Juha Kärkkäinen, Dominik Kempa, Simon J. Puglisi, and Bella Zhukova. "Engineering External Memory Induced Suffix Sorting". In: *ALENEX*. SIAM, 2017, pages 98–108. DOI: `10.1137/1.9781611974768.8`.

Bibliography III

[KSB06] Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. "Linear work suffix array construction". In: *J. ACM* 53.6 (2006), pages 918–936. DOI: 10.1145/1217856.1217858.

[Kur20] Florian Kurpicz. "Parallel Text Index Construction". PhD thesis. Technical University of Dortmund, Germany, 2020. DOI: 10.17877/DE290R-21114.

[LS07] N. Jesper Larsson and Kunihiko Sadakane. "Faster Suffix Sorting". In: *Theor. Comput. Sci.* 387.3 (2007), pages 258–272. DOI: 10.1016/j.tcs.2007.07.017.

[MM93] Udi Manber and Eugene W. Myers. "Suffix Arrays: A New Method for On-Line String Searches". In: *SIAM J. Comput.* 22.5 (1993), pages 935–948. DOI: 10.1137/0222058.

[NZC11] Ge Nong, Sen Zhang, and Wai Hong Chan. "Two Efficient Algorithms for Linear Time Suffix Array Construction". In: *IEEE Trans. Computers* 60.10 (2011), pages 1471–1484. DOI: 10.1109/TC.2010.188.

[PST07] Simon J. Puglisi, William F. Smyth, and Andrew Turpin. "A Taxonomy of Suffix Array Construction Algorithms". In: *ACM Comput. Surv.* 39.2 (2007), page 4. DOI: 10.1145/1242471.1242472.

Bibliography IV

[Val90] Leslie G. Valiant. "A Bridging Model for Parallel Computation". In: *Commun. ACM* 33.8 (1990), pages 103–111. DOI: 10.1145/79173.79181.