Computing the LCP Array of a Labeled Graph

Jarno N. Alanko[1], Davide Cenzato[2], Nicola Cotumaccio[1],
Sung-Hwan Kim[2], Giovanni Manzini[3] and Nicola Prezza[2]

[1]University of Helsinki, Finland
[2]Ca' Foscari University of Venice, Italy
[3]University of Pisa, Italy

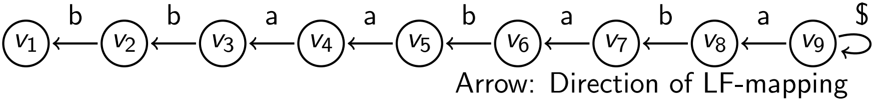
CPM 2024

SA and LCP for $T = bbaababa\$$

j	1	2	3	4	5	6	7	8	9
$T[j]$	b	b	a	a	b	a	b	a	$

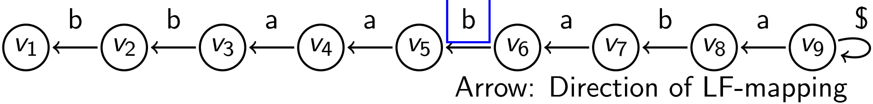
SA and LCP for $T = bbaababa\$$

j	1	2	3	4	5	6	7	8	9
$T[j]$	b	b	a	a	b	a	b	a	\$



SA and LCP for $T = bbaababa\$$

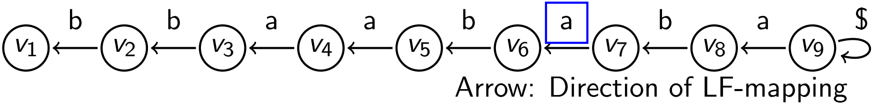
j	1	2	3	4	5	6	7	8	9
$T[j]$	b	b	a	a	b	a	b	a	\$



v_5: b

SA and LCP for $T = bbaababa\$$

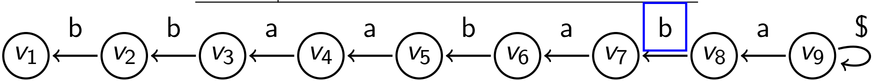
j	1	2	3	4	5	6	7	8	9
$T[j]$	b	b	a	a	b	a	b	a	\$



v_5: ba

SA and LCP for $T = bbaababa\$$

j	1	2	3	4	5	6	7	8	9
$T[j]$	b	b	a	a	b	a	b	a	\$

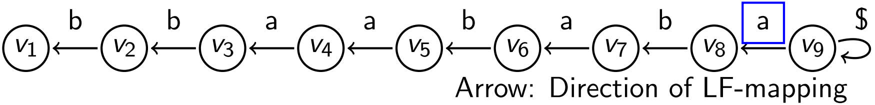


Arrow: Direction of LF-mapping

v_5: bab

SA and LCP for $T = bbaababa\$$

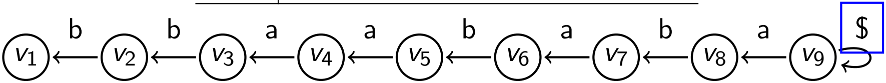
j	1	2	3	4	5	6	7	8	9
$T[j]$	b	b	a	a	b	a	b	a	\$



v_5: baba

SA and LCP for $T = bbaababa\$$

j	1	2	3	4	5	6	7	8	9
$T[j]$	b	b	a	a	b	a	b	a	$

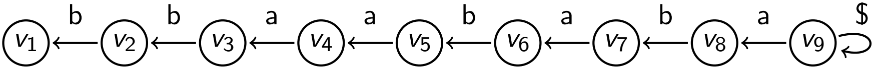


Arrow: Direction of LF-mapping

v_5: baba$

SA and LCP for $T = bbaababa\$$

j	1	2	3	4	5	6	7	8	9
$T[j]$	b	b	a	a	b	a	b	a	\$

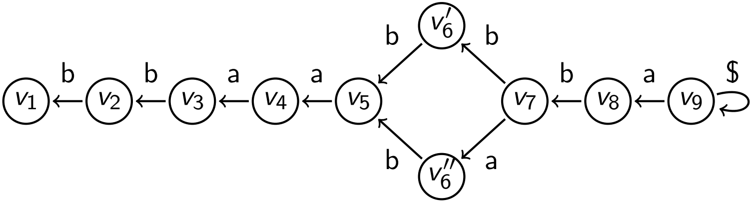


Arrow: Direction of LF-mapping

LCP	SA	Suffix
	9	\$
0	8	a\$
1	3	aababa\$
1	6	aba\$
3	4	ababa\$
0	7	ba\$
2	2	baababa\$
2	5	baba\$
1	1	bbaababa\$

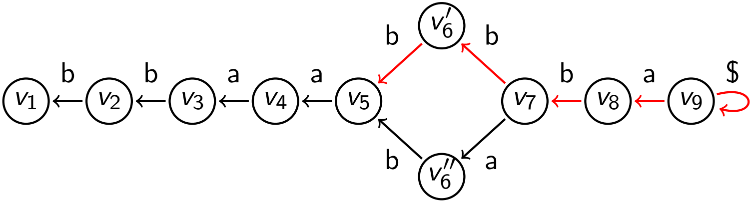
General graph?

- There are possibly many strings associated with a node.



General graph?

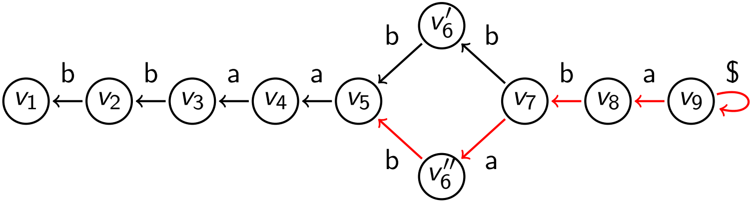
- There are possibly many strings associated with a node.



Strings associated with v_5: bbba$

General graph?

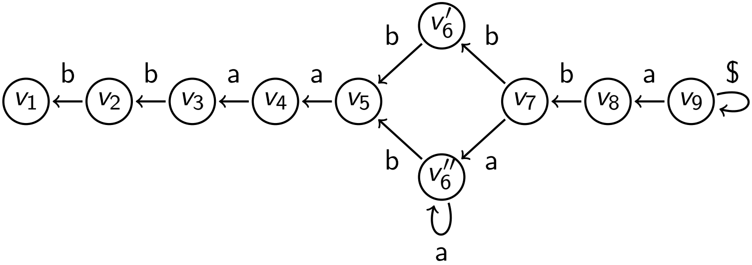
- There are possibly many strings associated with a node.



Strings associated with v_5: bbba$, baba$

General graph?

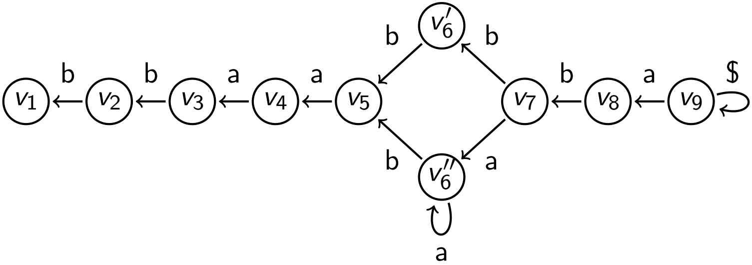
- There are possibly many strings associated with a node.



Strings associated with v_5: bbba\$, baba\$
baaba\$, baaaba\$, baaaaba\$, $\cdots$

General graph?

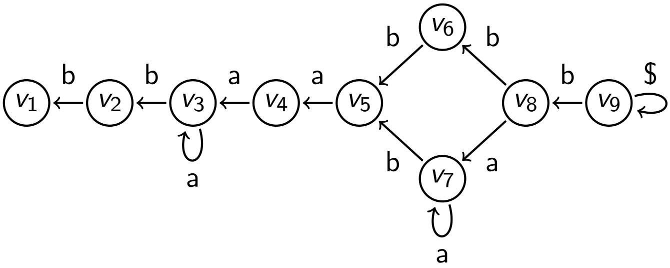
- There are possibly many strings associated with a node.



Strings associated with v_5: bbba\$, baba\$
baaba\$, baaaba\$, baaaaba\$, $\cdots$

- <u>Question: which strings should be considered?</u>
  - Answer: the smallest and the largest string.
    [Conte et al., DCC 2023], [Cotumaccio et al., SPIRE 2023]
  - e.g.) for v_5: baaaaa$\cdots$ and bbba\$

Infima and Suprema Strings



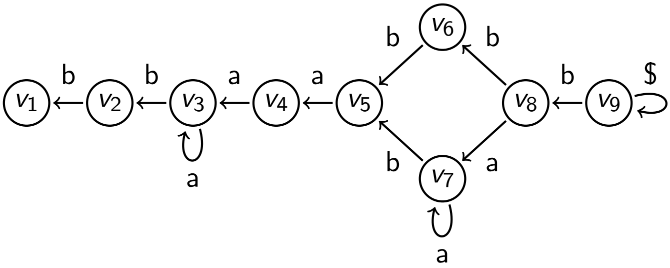
Node	inf	sup	Node	inf	sup
v_1	bbaaa$\cdots$	bbaabbb\$	v_6	bb\$	bb\$
v_2	baaa$\cdots$	baabbb\$	v_7	aaa$\cdots$	ab\$
v_3	aaa$\cdots$	aabbb\$	v_8	b\$	b\$
v_4	abaaa$\cdots$	abbb\$	v_9	\$	\$
v_5	baaa$\cdots$	bbb\$			

Motivation

- LCP array is useful for several problems related to string matching.
  - E.g., faster string matching, suffix tree functionalities.
- Recent work: Also useful for graph cases!
  - Navigation of de Bruijn graph [Boucher et al., DCC 2015]
  - Matching statistics [Conte et al., DCC 2023]

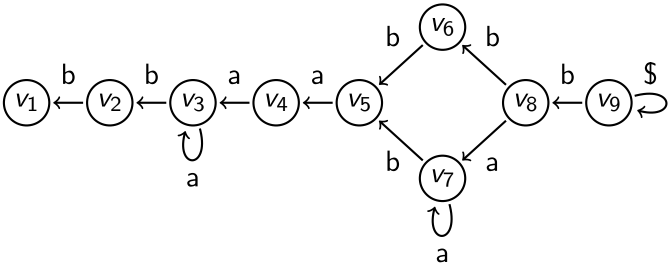
  - Computation algorithm for general graphs has not been proposed.

Infima and Suprema Strings



Node	inf	sup	Node	inf	sup
v_1	bbaaa···	bbaabbb$	v_6	bb$	bb$
v_2	baaa···	baabbb$	v_7	aaa···	ab$
v_3	aaa···	aabbb$	v_8	b$	b$
v_4	abaaa···	abbb$	v_9	$	$
v_5	baaa···	bbb$			

Infima and Suprema Strings



Node	inf	sup	Node	inf	sup
v_1	bbaaa··· (11)	bbaabbb$ (12)	v_6	bb$ (10)	bb$ (10)
v_2	baaa··· (8)	baabbb$ (9)	v_7	aaa··· (2)	ab$ (4)
v_3	aaa··· (2)	aabbb$ (3)	v_8	b$ (7)	b$ (7)
v_4	abaaa··· (5)	abbb$ (6)	v_9	$ (1)	$ (1)
v_5	baaa··· (8)	bbb$ (13)			

Sorted Infima and Suprema Strings

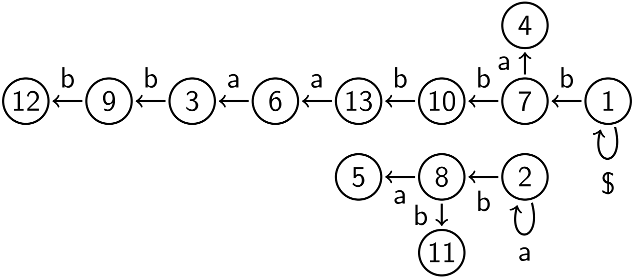
rank	string	nodes
1	\$	inf(9), sup(9)
2	aaa$\cdots$	inf(3), inf(7)
3	aabbb\$	sup(3)
4	ab\$	sup(7)
5	abaaa$\cdots$	inf(4)
6	abbb\$	sup(4)
7	b\$	inf(8), sup(8)
8	baaa$\cdots$	inf(2), inf(5)
9	baabbb\$	sup(2)
10	bb\$	inf(6), sup(6)
11	bbaaa$\cdots$	inf(1)
12	bbaabbb\$	sup(1)
13	bbb\$	sup(5)

LCP of Infima and Suprema Strings

rank	LCP	string	nodes
1		\$	inf(9), sup(9)
2	0	aaa···	inf(3), inf(7)
3	2	aabbb\$	sup(3)
4	1	ab\$	sup(7)
5	2	abaaa···	inf(4)
6	2	abbb\$	sup(4)
7	0	b\$	inf(8), sup(8)
8	1	baaa···	inf(2), inf(5)
9	3	baabbb\$	sup(2)
10	1	bb\$	inf(6), sup(6)
11	2	bbaaa···	inf(1)
12	4	bbaabbb\$	sup(1)
13	2	bbb\$	sup(5)

Graph encoding of Inf/sup strings

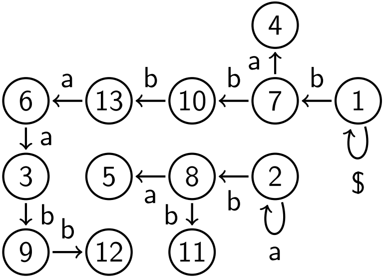
rank	string
1	\$
2	aaa···
3	aabbb\$
4	ab\$
5	abaaa···
6	abbb\$
7	b\$
8	baaa···
9	baabbb\$
10	bb\$
11	bbaaa···
12	bbaabbb\$
13	bbb\$



- Computed in $O(\min\{m \log n, n^2\})$ time.
  - Becker et al., ESA 2023
  - Cotumaccio, ISAAC 2023
  - Wheeler DFA: $O(m)$ time.
- Wheeler Pseudo-forest
  - Exactly one in-edge per node
  - Totally ordered (Wheeler graph)

LF-mapping works!

rank	string	out	
1	$	$	b
2	aaa· · ·	a	b
3	aabbb$		b
4	ab$		
5	abaaa· · ·		
6	abbb$	a	
7	b$	a	b
8	baaa· · ·	a	b
9	baabbb$		b
10	bb$		b
11	bbaaa· · ·		
12	bbaabbb$		
13	bbb$	a	

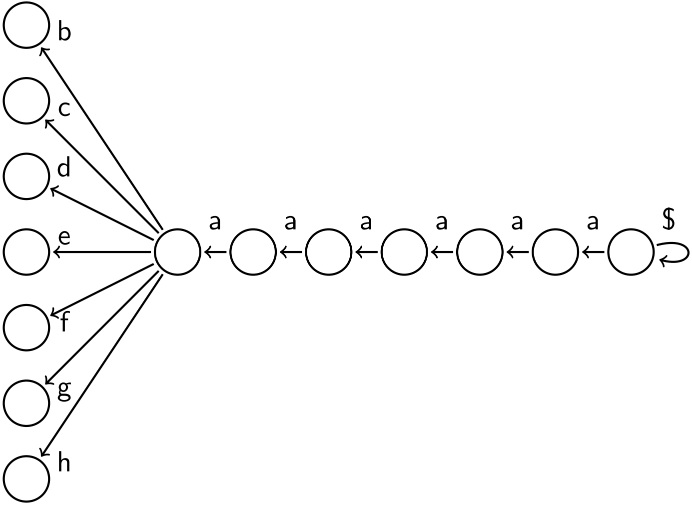


- Maybe LCP computation algorithm for strings work?

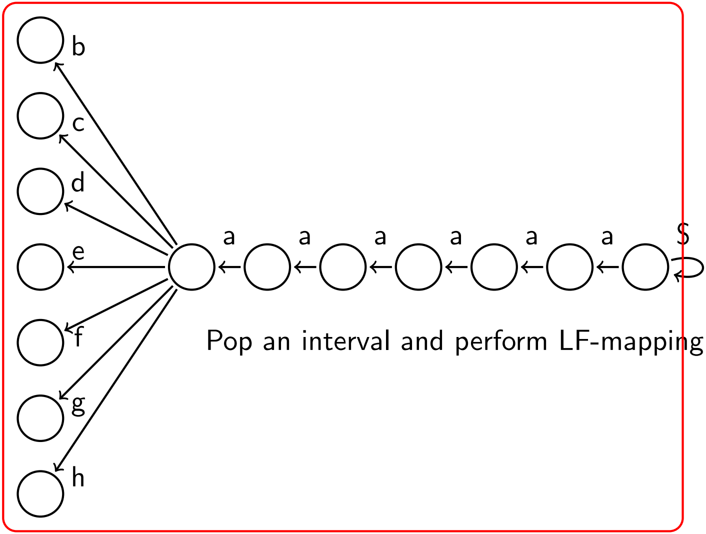
Main contribution

Algorithm	String	W. Pseudo-forest	Space (bits)
Manber and Myers	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Beller et al.	$O(n \log \sigma)$	$O(n\sigma \log n)$	$n \log \sigma + O(n)$
Ours	$O(n \log \sigma)$	$O(n \log \sigma)$	$O(n \log \sigma)$

Beller et al.'s Algorithm

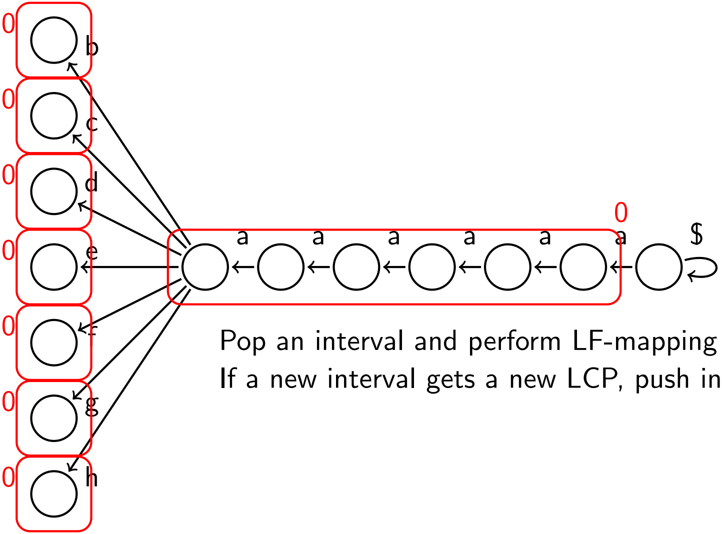


Beller et al.'s Algorithm



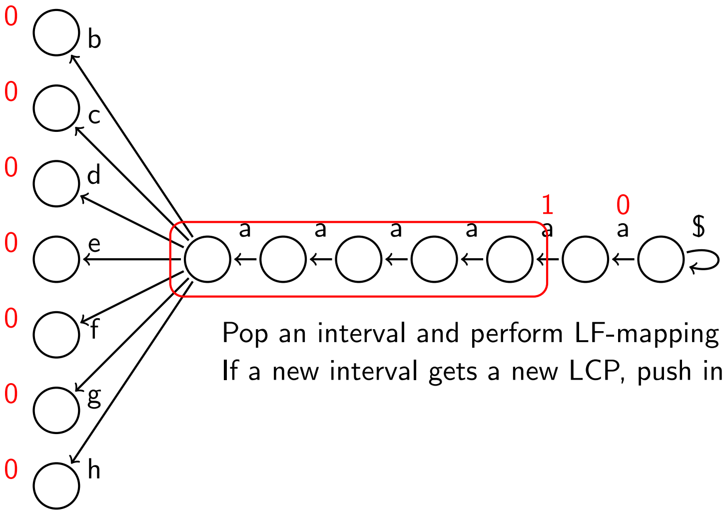
Pop an interval and perform LF-mapping for each char.

Beller et al.'s Algorithm



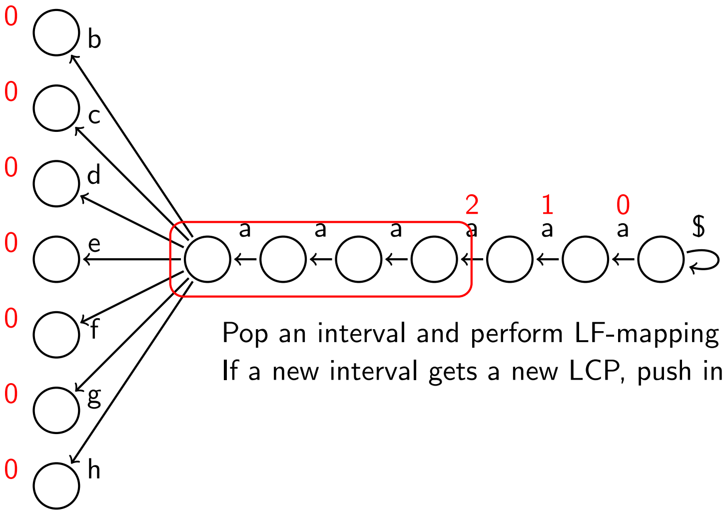
Pop an interval and perform LF-mapping for each char.
If a new interval gets a new LCP, push into the queue.

Beller et al.'s Algorithm



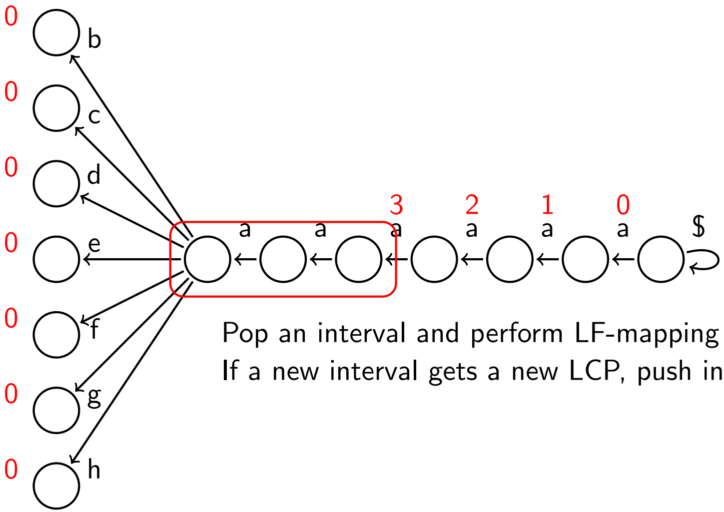
Pop an interval and perform LF-mapping for each char.
If a new interval gets a new LCP, push into the queue.

Beller et al.'s Algorithm



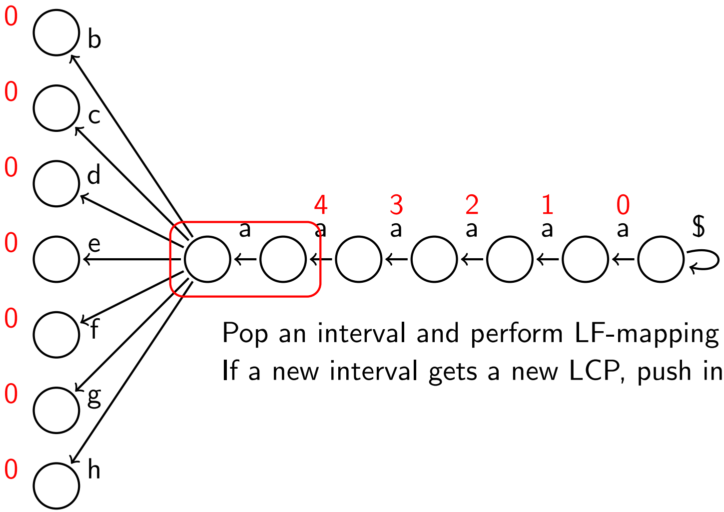
Pop an interval and perform LF-mapping for each char.
If a new interval gets a new LCP, push into the queue.

Beller et al.'s Algorithm



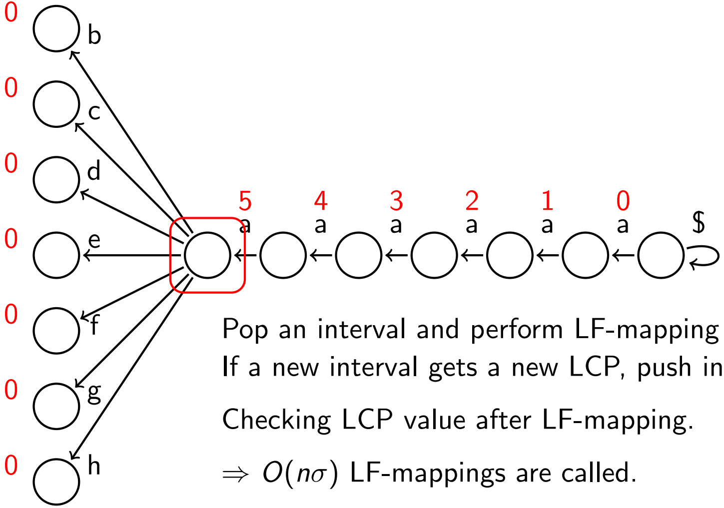
Pop an interval and perform LF-mapping for each char.
If a new interval gets a new LCP, push into the queue.

Beller et al.'s Algorithm



Pop an interval and perform LF-mapping for each char.
If a new interval gets a new LCP, push into the queue.

Beller et al.'s Algorithm



Pop an interval and perform LF-mapping for each char.
If a new interval gets a new LCP, push into the queue.

Checking LCP value after LF-mapping.

$\Rightarrow O(n\sigma)$ LF-mappings are called.

LCP Propagation

rank	string	out		LCP
1	$	$	b	0
2	aaa···	a	b	2
3	aabbb$		b	1
4	ab$			2
5	abaaa···			2
6	abbb$	a		0
7	b$	a	b	1
8	baaa···	a	b	3
9	baabbb$		b	1
10	bb$		b	2
11	bbaaa···			4
12	bbaabbb$			2
13	bbb$	a		

LCP Propagation

rank	string	out		LCP
1	$	$	b	
2	aaa···	a	b	0
3	aabbb$		b	2
4	ab$			1
5	abaaa···			2
6	abbb$	a		2
7	b$	a	b	0
8	baaa···	a	b	1
9	baabbb$		b	3
10	bb$		b	1
11	bbaaa···			2
12	bbaabbb$			4
13	bbb$	a		2

LCP Propagation

rank	string	out		LCP
1	\$	\$	b	
2	aaa· · ·	a	b	0
3	aabbb\$		b	2
4	ab\$			1
5	abaaa· · ·			2
6	abbb\$	a		2
7	b\$	a	b	0
8	baaa· · ·	a	b	1
9	baabbb\$		b	3
10	bb\$		b	1
11	bbaaa· · ·			2
12	bbaabbb\$			4
13	bbb\$	a		2

LCP Propagation

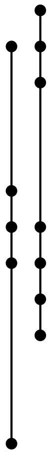
rank	string	out		LCP
1	$	$	b	
2	aaa···	a	b	0
3	aabbb$		b	2
4	ab$			1
5	abaaa···			2
6	abbb$	a		2
7	b$	a	b	0
8	baaa···	a	b	1
9	baabbb$		b	3
10	bb$		b	1
11	bbaaa···			2
12	bbaabbb$			4
13	bbb$	a		2

LCP Propagation

rank	string	out		LCP
1	$	$	b	
2	aaa···	a	b	0
3	aabbb$		b	2
4	ab$			1
5	abaaa···			2
6	abbb$	a		2
7	b$	a	b	0
8	baaa···	a	b	1
9	baabbb$		b	3
10	bb$		b	1
11	bbaaa···			2
12	bbaabbb$			4
13	bbb$	a		2

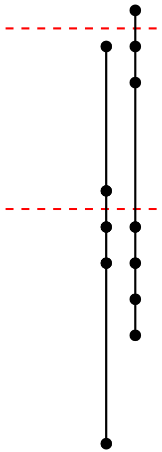
LCP Propagation with Interval Stabbing

rank	string	out			lcp
1	\$	\$		b	
2	aaa···		a	b	
3	aabbb\$			b	
4	ab\$				
5	abaaa···				
6	abbb\$		a		
7	b\$		a	b	
8	baaa···		a	b	
9	baabbb\$			b	
10	bb\$			b	
11	bbaaa···				
12	bbaabbb\$				
13	bbb\$		a		



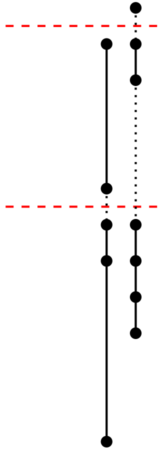
LCP Propagation with Interval Stabbing

rank	string	out	lcp
1	$	$ b	
2	aaa···	a b	0
3	aabbb$	b	
4	ab$		
5	abaaa···		
6	abbb$	a	
7	b$	a b	0
8	baaa···	a b	
9	baabbb$	b	
10	bb$	b	
11	bbaaa···		
12	bbaabbb$		
13	bbb$	a	



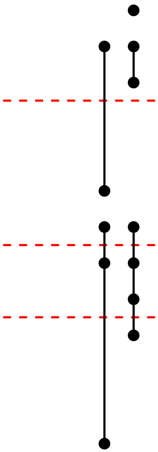
LCP Propagation with Interval Stabbing

rank	string	out		lcp
1	$	$	b	
2	aaa···	a	b	0
3	aabbb$		b	
4	ab$			1
5	abaaa···			
6	abbb$	a		
7	b$	a	b	0
8	baaa···	a	b	1
9	baabbb$		b	
10	bb$		b	1
11	bbaaa···			
12	bbaabbb$			
13	bbb$	a		



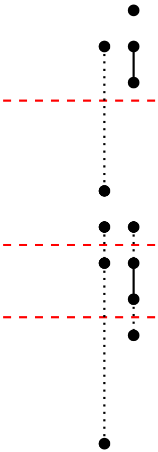
LCP Propagation with Interval Stabbing

rank	string	out	lcp
1	$	$ b	0
2	aaa···	a b	
3	aabbb$	b	1
4	ab$		
5	abaaa···		
6	abbb$	a	0
7	b$	a b	1
8	baaa···	a b	
9	baabbb$	b	1
10	bb$	b	
11	bbaaa···		
12	bbaabbb$		
13	bbb$	a	



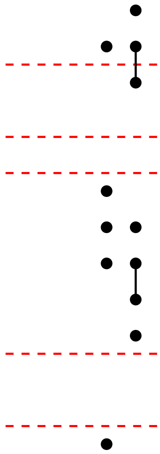
LCP Propagation with Interval Stabbing

rank	string	out		lcp
1	$	$	b	
2	aaa···	a	b	0
3	aabbb$		b	2
4	ab$			1
5	abaaa···			2
6	abbb$	a		2
7	b$	a	b	0
8	baaa···	a	b	1
9	baabbb$		b	
10	bb$		b	1
11	bbaaa···			2
12	bbaabbb$			
13	bbb$	a		2



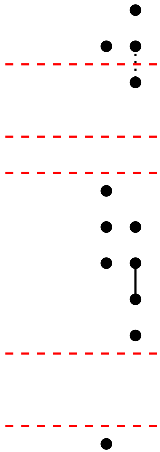
LCP Propagation with Interval Stabbing

rank	string	out		lcp
1	$	$	b	
2	aaa···	a	b	0
3	aabbb$		b	2
4	ab$			1
5	abaaa···			2
6	abbb$	a		2
7	b$	a	b	0
8	baaa···	a	b	1
9	baabbb$		b	
10	bb$		b	1
11	bbaaa···			2
12	bbaabbb$			
13	bbb$	a		2



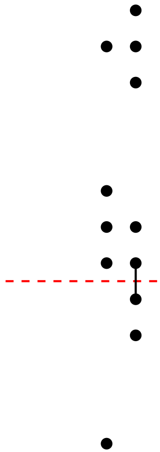
LCP Propagation with Interval Stabbing

rank	string	out	lcp
1	$	$ b	
2	aaa···	a b	0
3	aabbb$	b	2
4	ab$		1
5	abaaa···		2
6	abbb$	a	2
7	b$	a b	0
8	baaa···	a b	1
9	baabbb$	b	3
10	bb$	b	1
11	bbaaa···		2
12	bbaabbb$		
13	bbb$	a	2



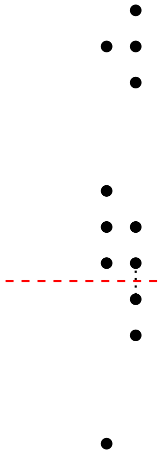
LCP Propagation with Interval Stabbing

rank	string	out		lcp
1	$	$	b	
2	aaa···	a	b	0
3	aabbb$		b	2
4	ab$			1
5	abaaa···			2
6	abbb$	a		2
7	b$	a	b	0
8	baaa···	a	b	1
9	baabbb$		b	3
10	bb$		b	1
11	bbaaa···			2
12	bbaabbb$			
13	bbb$	a		2



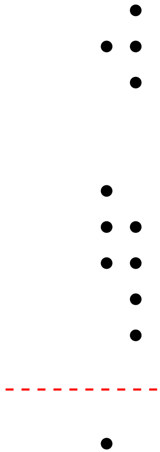
LCP Propagation with Interval Stabbing

rank	string	out		lcp
1	$	$	b	
2	aaa···	a	b	0
3	aabbb$		b	2
4	ab$			1
5	abaaa···			2
6	abbb$	a		2
7	b$	a	b	0
8	baaa···	a	b	1
9	baabbb$		b	3
10	bb$		b	1
11	bbaaa···			2
12	bbaabbb$			4
13	bbb$	a		2

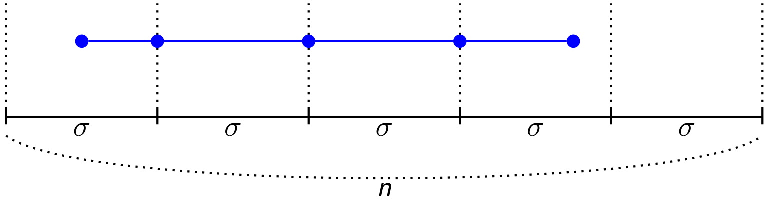


LCP Propagation with Interval Stabbing

rank	string	out	lcp
1	$	$ b	
2	aaa···	a b	0
3	aabbb$	b	2
4	ab$		1
5	abaaa···		2
6	abbb$	a	2
7	b$	a b	0
8	baaa···	a b	1
9	baabbb$	b	3
10	bb$	b	1
11	bbaaa···		2
12	bbaabbb$		4
13	bbb$	a	2



Partitioned Interval Tree



- Split the universe into n/σ blocks.
- Cut the intervals crossing block boundaries.
- Each block is indexed with an interval tree.

Partitioned Interval Tree

- The total number of (segmented) intervals: $O(n)$
  - Intervals contained in boundaries: $O(n)$
  - Segments of intervals cut by boundaries: $O(\frac{n}{\sigma} \cdot \sigma) = O(n)$.
- Each word in an interval tree uses $O(\log \sigma)$ bits.
  - In each block, there are at most σ^2 intervals.
  - Block size is σ.
- Total space: $O(n \log \sigma)$ bits

- Time for each stab-and-remove query: $O(\log \sigma + k)$ where k is the number of segments to be removed.
- Total time amortizes to $O(n \log \sigma)$.

- Construction (line sweeping): $O(n \log \sigma)$ time and space (bits).

Conclusions

Algorithm	String	W. Pseudo-forest	Space (bits)
Manber and Myers	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Beller et al.	$O(n \log \sigma)$	$O(n\sigma \log n)$	$n \log \sigma + O(n)$
Ours	$O(n \log \sigma)$	$O(n \log \sigma)$	$O(n \log \sigma)$

- Future Work:
  - Efficient computation of inf/sup graphs: $O(m)$ time?
  - $O(n)$ time or $n \log \sigma$ bits space?

Thank You!

Jarno Alanko, Nicola Cotumaccio: Funded by the Helsinki Institute for Information Technology (HIIT). *Davide Cenzato, Sung-Hwan Kim, Nicola Prezza*: Funded by the European Union (ERC, REGINDEX, 101039208). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. *Giovanni Manzini*: Funded by the Italian Ministry of Health, POS 2014-2020, project ID T4-AN-07, CUP I53C22001300001, by INdAM-GNCS Project CUP E53C23001670001 and by PNRR ECS00000017 Tuscany Health Ecosystem, Spoke 6 CUP I53C22000780001.