Suffix sorting, trees, arrays, selection and Burrows-Wheeler transform

Gerth Stølting Brodal

1	2	3	4	5	6	7	8	9	10	11	12
m	i	s	s	i	s	s	i	p	p	i	$

Suffixes / circular shifts

1	mississippi$
2	ississippi$m
3	ssissippi$mi
4	sissippi$mis
5	issippi$miss
6	ssippi$missi
7	sippi$missis
8	ippi$mississ
9	ppi$mississi
10	pi$mississip
11	i$mississipp
12	$mississippi

Sorted suffixes

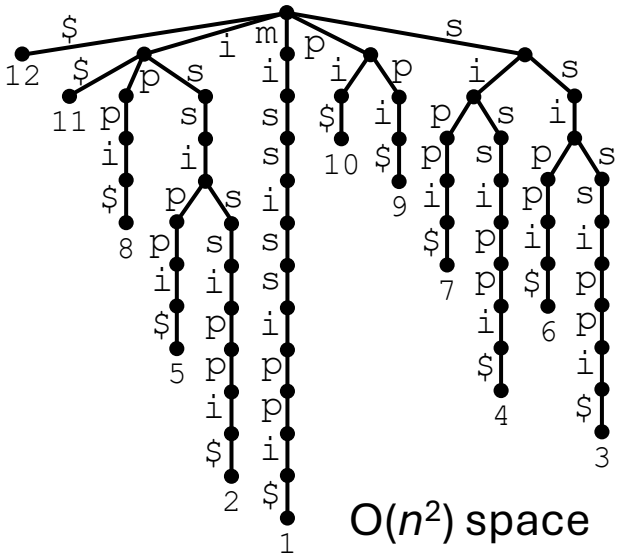
12	$mississippi
11	i$mississipp
8	ippi$mississ
5	issippi$miss
2	ississippi$m
1	mississippi$
10	pi$mississip
9	ppi$mississi
7	sippi$missis
4	sissippi$mis
6	ssippi$missi
3	ssissippi$mi

Suffix array

1	2	3	4	5	6	7	8	9	10	11	12
12	11	8	5	2	1	10	9	7	4	6	3

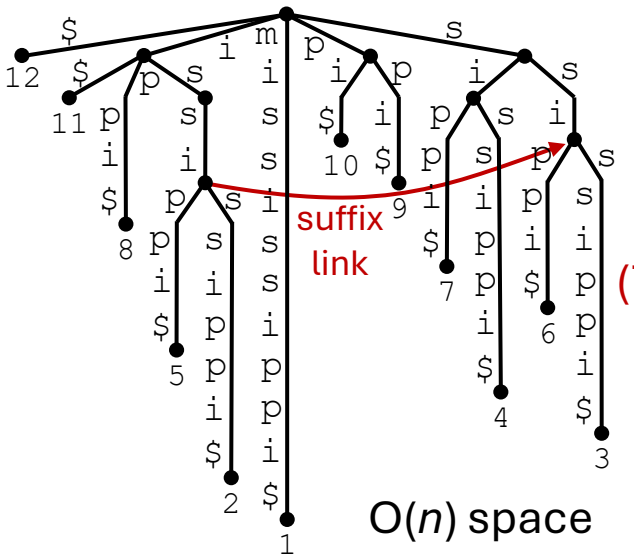
Gonet (1987), Manber & Myers (1990)

Suffix trie



$O(n^2)$ space

Suffix tree, Weiner (1973)



$O(n)$ space

- **Sorting suffixes** using e.g. MergeSort ⇒ $O(n^2 \cdot \log n)$ time ☹
- **Suffix tree** : Weiner (1973) $O(n \cdot \log n)$, $O(n)$ for $\sum$=$O(1)$
  Farach (1997) $O(\text{Sort}_\Sigma(n))$, $O(n)$ for $\sum$=$n^{O(1)}$
- **Suffix array** : Kärkkäinen & Sanders (2003) $O(\text{Sort}_\Sigma(n))$ DC3/skew $T(n) = T(\frac{2}{3} \cdot n) + \text{Sort}_\Sigma(n)$
- **Suffix selection** : Find the k^{th} lexicographically smallest suffix

Select(mississippi$, 4)

Franceschini & Muthukrishnan (2007), $O(n)$ time — cache aware, $\omega(n)$ work ↓

Franceschini, Grossi & Muthukrishnan (2009), $O(n/B)$ IOs, $M = \Omega(B^{1+\varepsilon})$

Suffix selection – cache oblivious? tall cache assumption? $O(n)$ work?

Burrows-Wheeler transform (1994)

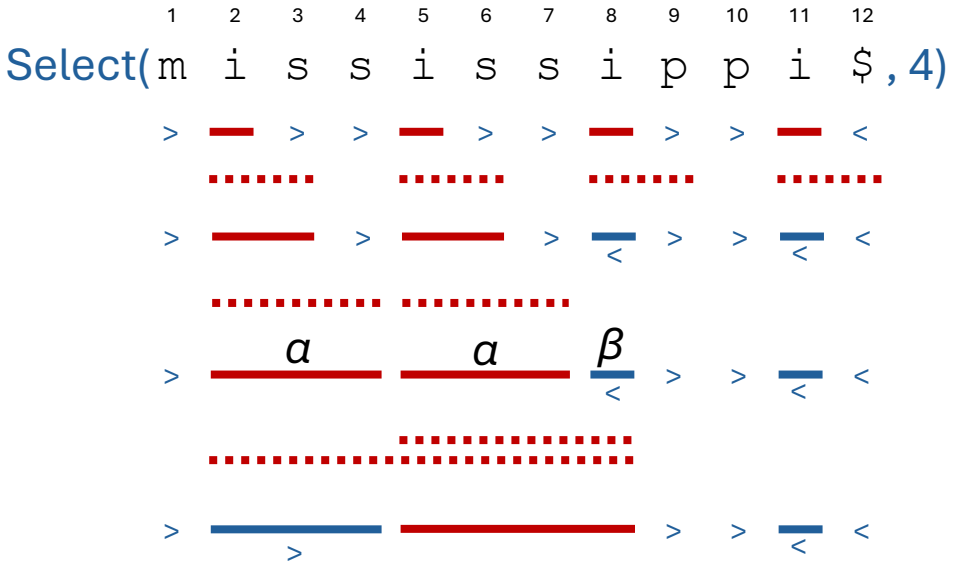
1	2	3	4	5	6	7	8	9	10	11	12
i	p	s	s	m	$	p	i	s	s	i	i

decoding

1	2	3	4	5	6	7	8	9	10	11	12
m	i	s	s	i	s	s	i	p	p	i	$

Decoding $O(\text{Sort}_\Sigma(n))$ time

Suffix selection – Franceschini, Grossi & Muthukrishnan (2009)



O(n) time selection on symbols, Blum et al. (1973) ⇒ 4 candidates
4 prospective candidates
select(3) ⇒ 2 candidates
2 prospective candidates
select(1) ⇒ 2 candidates
2 prospective candidates ($\alpha\beta$ and $\alpha^2\beta$, where we know "$\beta<\alpha$")
select(1) ⇒ 1 candidate ⇒ done

- **Analysis**
  Elements participating in selections either become *inactive* or reduce the *cover* ⇒ total O(n) time
- **Making it cache oblivious ?**
  - The algorithm is CO but not cache efficient
  - Blum et al. selection is cache efficient
  - Candidate set = sorted list of indices
  - **Bottleneck** – constructing prospective candidates
- **… or just simpler, or cache aware with O(n) internal work ?**