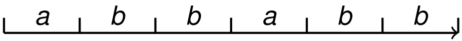
Superstring Graph in compact space

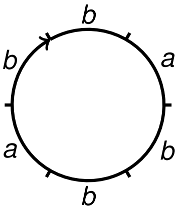
Bastien Cazaux and Eric Rivals*

* LIRMM & IBC, Montpellier

February 5, 2020 (DSB 2020)

Superstring Problems


a
b
b
a
b
b


b
a
b
b
a
b

Notation

Definition [Gusfield 1997]

Let *w* a string.

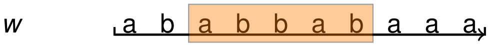
- a **substring** of *w* is a string included in *w*,
- a **prefix** of *w* is a substring which begins *w* and
- a **suffix** is a substring which ends *w*.
- an **overlap** from *w* over *v* is a suffix of *w* that is also a prefix of *v*.

w a b a b b a b a a a

Definition [Gusfield 1997]

Let *w* a string.

- a **substring** of *w* is a string included in *w*,
- a **prefix** of *w* is a substring which begins *w* and
- a **suffix** is a substring which ends *w*.
- an **overlap** from *w* over *v* is a suffix of *w* that is also a prefix of *v*.



Definition [Gusfield 1997]

Let *w* a string.

- a **substring** of *w* is a string included in *w*,
- a **prefix** of *w* is a substring which begins *w* and
- a **suffix** is a substring which ends *w*.
- an **overlap** from *w* over *v* is a suffix of *w* that is also a prefix of *v*.

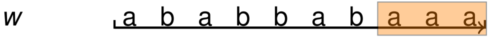
w a b a b b a b a a a

Notation

Definition [Gusfield 1997]

Let *w* a string.

- a **substring** of *w* is a string included in *w*,
- a **prefix** of *w* is a substring which begins *w* and
- a **suffix** is a substring which ends *w*.
- an **overlap** from *w* over *v* is a suffix of *w* that is also a prefix of *v*.



Notation

Definition [Gusfield 1997]

Let *w* a string.

- a **substring** of *w* is a string included in *w*,
- a **prefix** of *w* is a substring which begins *w* and
- a **suffix** is a substring which ends *w*.
- an **overlap** from *w* over *v* is a suffix of *w* that is also a prefix of *v*.

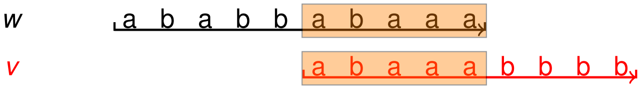
w a b a b b a b a a a

v a b a a a b b b b

Definition [Gusfield 1997]

Let *w* a string.

- a **substring** of *w* is a string included in *w*,
- a **prefix** of *w* is a substring which begins *w* and
- a **suffix** is a substring which ends *w*.
- an **overlap** from *w* over *v* is a suffix of *w* that is also a prefix of *v*.

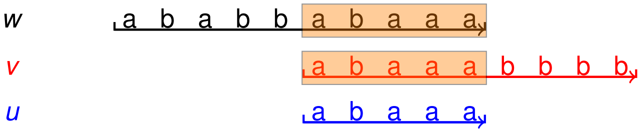


Notation

Definition [Gusfield 1997]

Let *w* a string.

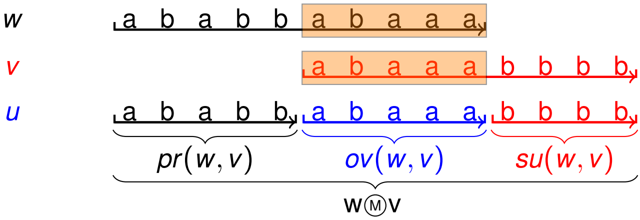
- a **substring** of *w* is a string included in *w*,
- a **prefix** of *w* is a substring which begins *w* and
- a **suffix** is a substring which ends *w*.
- an **overlap** from *w* over *v* is a suffix of *w* that is also a prefix of *v*.



Definition [Gusfield 1997]

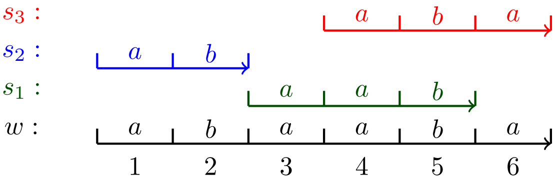
Let *w* a string.

- a **substring** of *w* is a string included in *w*,
- a **prefix** of *w* is a substring which begins *w* and
- a **suffix** is a substring which ends *w*.
- an **overlap** from *w* over *v* is a suffix of *w* that is also a prefix of *v*.

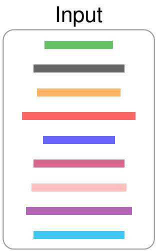


Definition

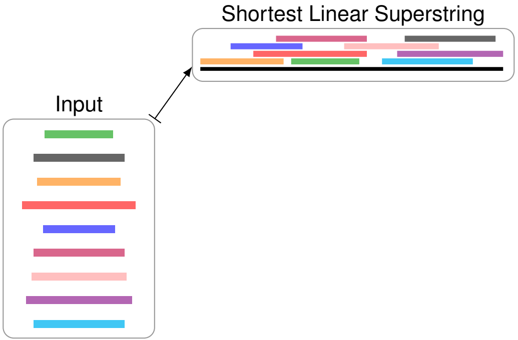
Let $P = \{s_1, s_2, \ldots, s_p\}$ be a set of strings. A *superstring* of P is a string w such that any s_i is a substring of w.



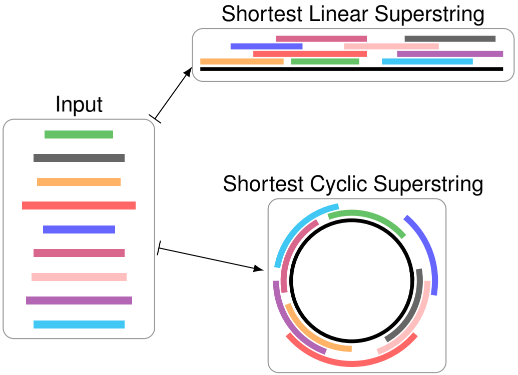
Shortest Superstrings problems



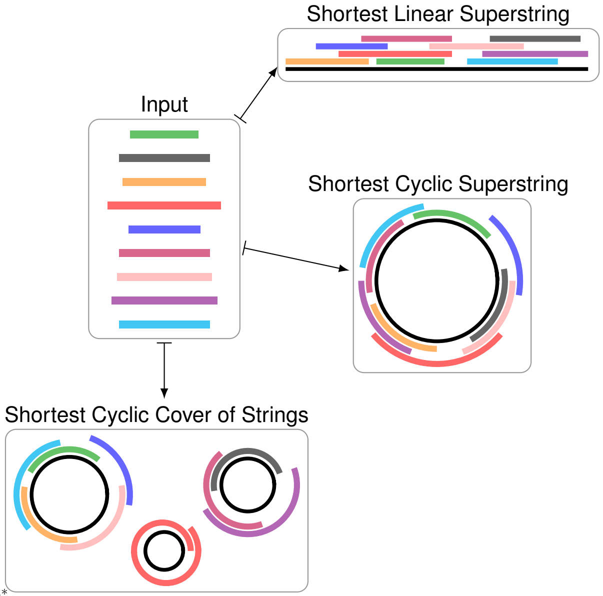
Shortest Superstrings problems



Shortest Superstrings problems



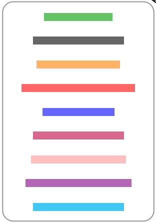
Shortest Superstrings problems



Shortest Superstrings problems

- NP-hard [Gallant 1980]
- APX-hard [Blum 1991]
- Approximation $2 + \frac{11}{30}$ [Paluch 2015]

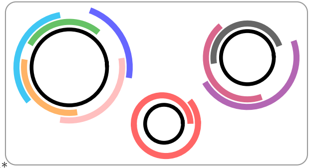
Input



Shortest Cyclic Superstring

- NP-hard [Cazaux PhD 2016]

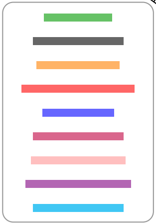
Shortest Cyclic Cover of Strings



Shortest Superstrings problems

- NP-hard [Gallant 1980]
- APX-hard [Blum 1991]
- Approximation $2 + \frac{11}{30}$ [Paluch 2015]

Input



Shortest Cyclic Superstring

- NP-hard [Cazaux PhD 2016]

Shortest Cyclic Cover of Strings

- In $O(|P|^3 + ||P||)$ time [Papadimitriou 1982]
- **Linear time in $||P||$** [Cazaux, Rivals JDA 2016]

Superstring Graph

One graph to rule them all

Greedy Algorithm for SCCS

$a\ a\ b$

$a\ b\ b\ a$

$a\ b\ a\ a$

$a\ b\ a\ b\ b$

Greedy Algorithm for SCCS

$$|ov(ababb, abba)| = |abb| = 3$$

a a b

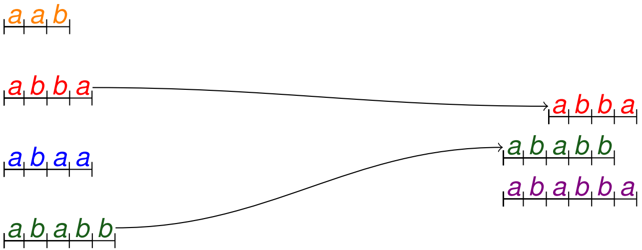
a b b a

a b a a

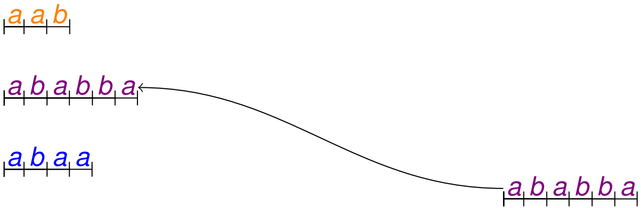
a b a b b

Greedy Algorithm for SCCS

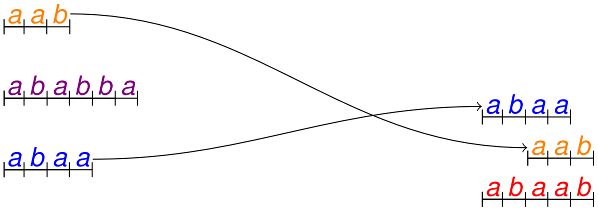
$$|ov(ababb, abba)| = |abb| = 3$$



$|ov(ababb, abba)| = |abb| = 3$

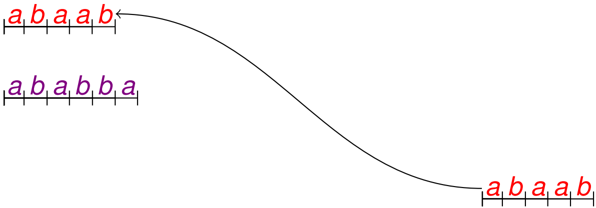


$$|ov(abaa, aab)| = |aa| = 2$$



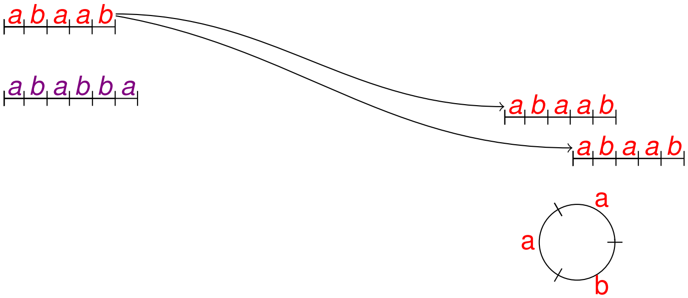
Greedy Algorithm for SCCS

$$|ov(abaa, aab)| = |aa| = 2$$

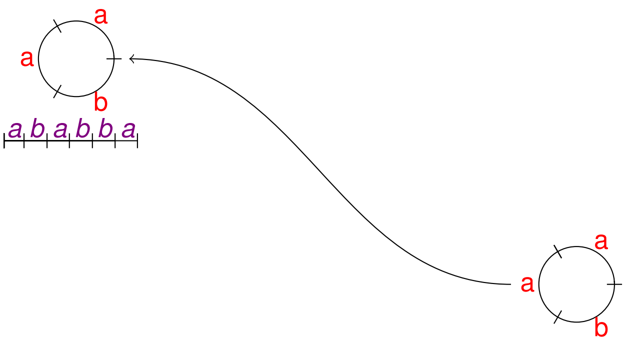


Greedy Algorithm for SCCS

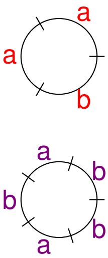
$|ov(abaab, abaab)| = |ab| = 2$

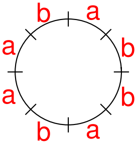


$|ov(abaab, abaab)| = |ab| = 2$



An other solution

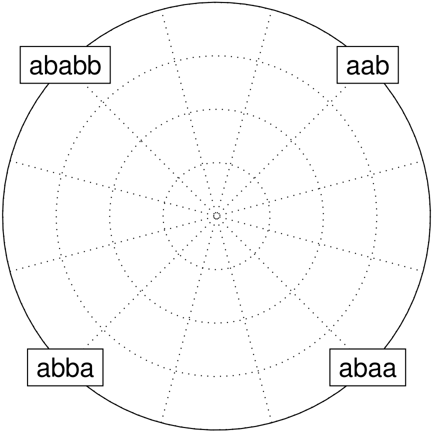




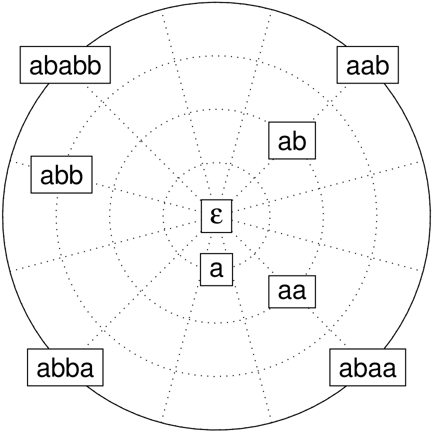
Theorem [Cazaux et al. 2014]

The greedy algorithm solves exactly the Shortest Cyclic Cover of Strings problem.

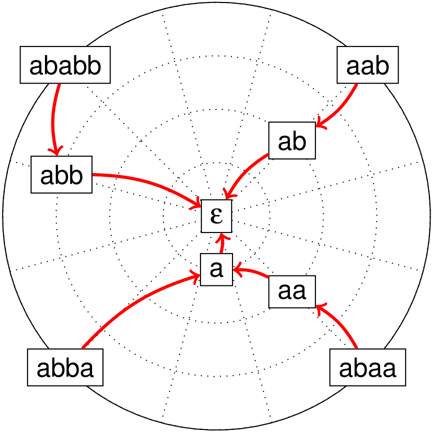
Extended Hierarchical Overlap Graph (EHOG)



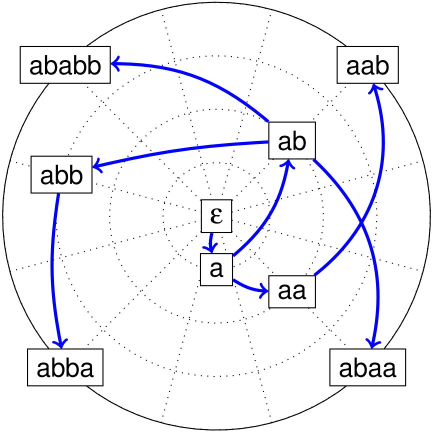
Extended Hierarchical Overlap Graph (EHOG)



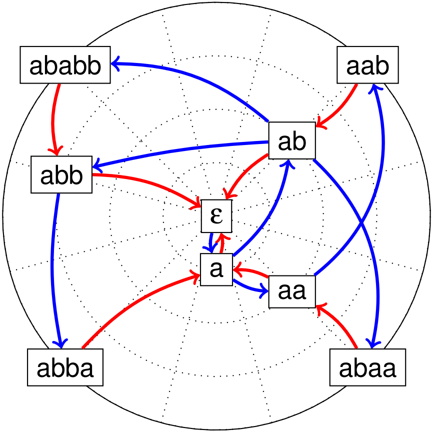
Extended Hierarchical Overlap Graph (EHOG)

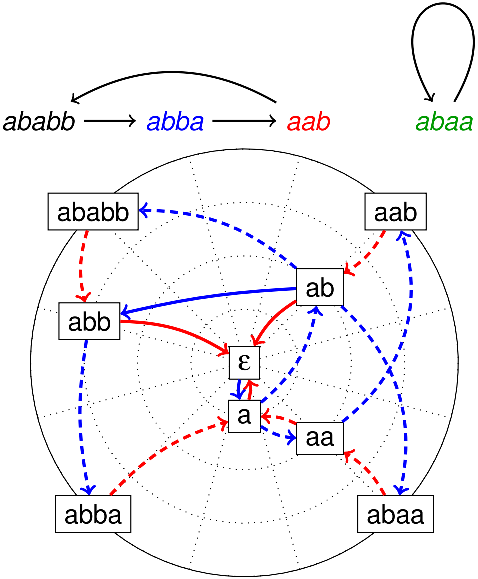


Extended Hierarchical Overlap Graph (EHOG)



Extended Hierarchical Overlap Graph (EHOG)




ababb
abba
aab
abaa
ababb
aab
ab
abb
ε
a
aa
abba
abaa

Definition

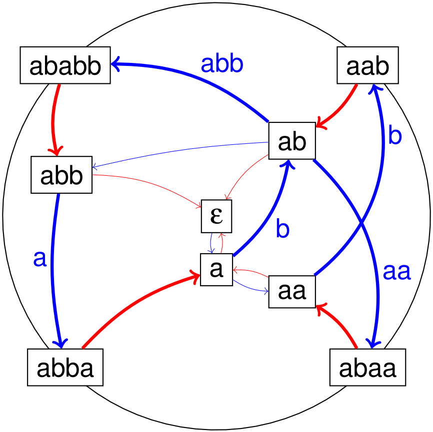
All the solutions of the greedy algorithm for SCCS give the same graph on the EHOG, and it is called Superstring Graph.

Definition

All the solutions of the greedy algorithm for SCCS give the same graph on the EHOG, and it is called Superstring Graph.

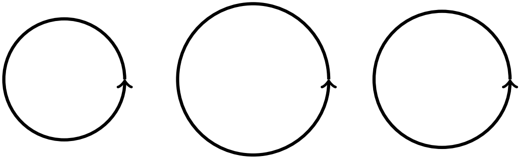
Propositions [Cazaux et al. 2015]

- The size of the Superstring Graph is linear in the size of the input.
- We can build the Superstring Graph in liner time in the size of the input.
- A labeled eulerian cycle of the Superstring Graph is a solution of the greedy algorithm for the Shortest Cyclic Cover of Strings problem.

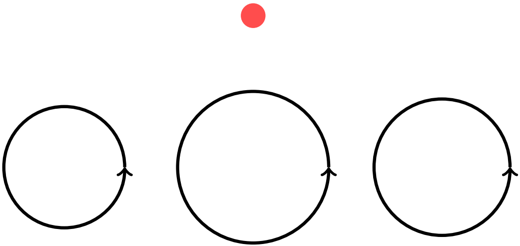

ababb
abb
aab
b
ab
abb
ε
a
b
a
aa
aa
abba
abaa

Why do we want to compute the Superstring Graph?

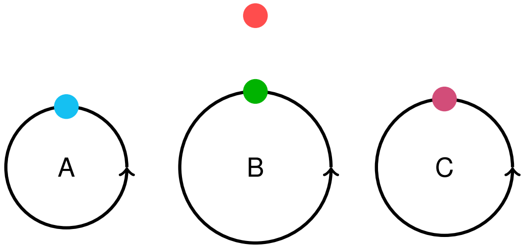
Application 1: Compute bounds for optimal solution of SLS



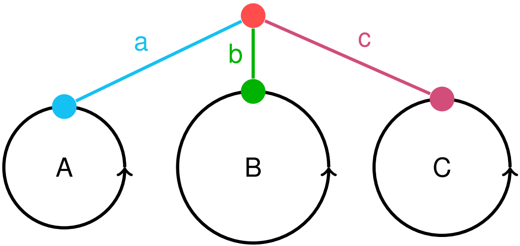
Application 1: Compute bounds for optimal solution of SLS



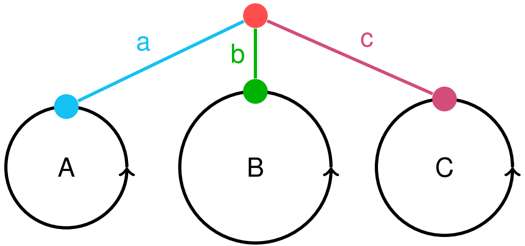
Application 1: Compute bounds for optimal solution of SLS



Application 1: Compute bounds for optimal solution of SLS

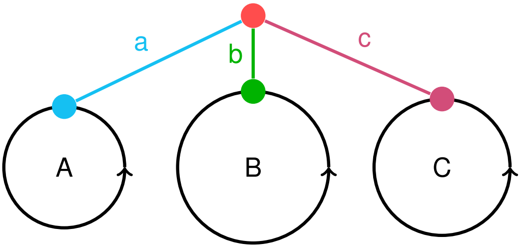


Application 1: Compute bounds for optimal solution of SLS



With a $\geq$ b and a $\geq$ c,

A + a + B + C $\leq$ | Optimal solution of SLS | $\leq$ A + a + B + b + C + c



With a $\geq$ b and a $\geq$ c,

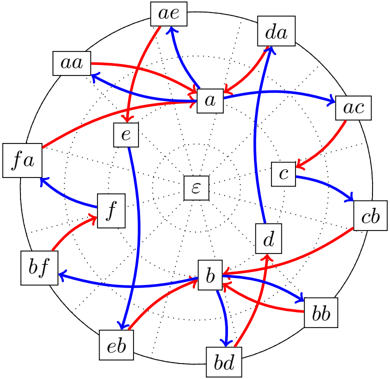
A + a + B + C $\leq$ | Optimal solution of SLS | $\leq$ A + a + B + b + C + c

Results on real data [Cazaux et al. 2018]

Input: *E. coli* genome of 50x:
4 503 422 reads (454 845 622 symbols)

Result: length of optimal solutions
between 187 250 434 and 187 250 672
A difference of 710 symbols (**0,00038**%)

Application 2: Use SG as a genome assembly graph



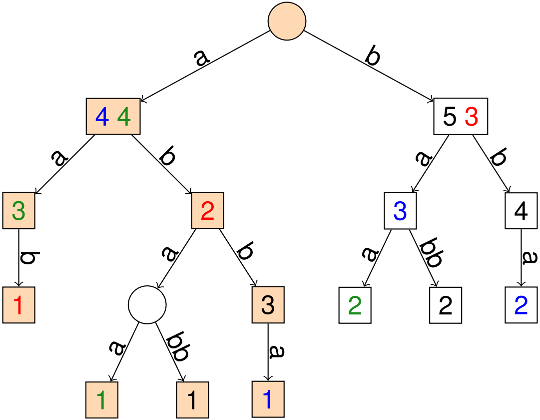
Results [Cazaux et al. 2016]

We can build a mixed cover that includes unitigs of the dBG (or Variable order dBG) in time in $O(||P||)$, and in linear space in the size of the de Bruijn Graph.

How can we **store** the Superstring Graph in compact space?

Difficulty

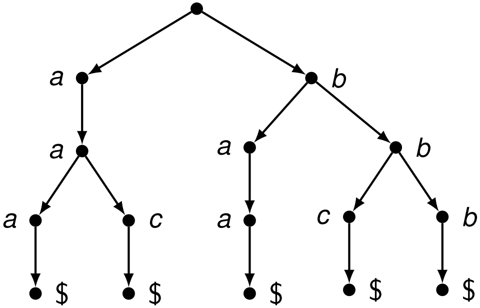
The Superstring Graph is a graph with integer value (each value in $\log ||P||$ for a set of strings P)



Propositions

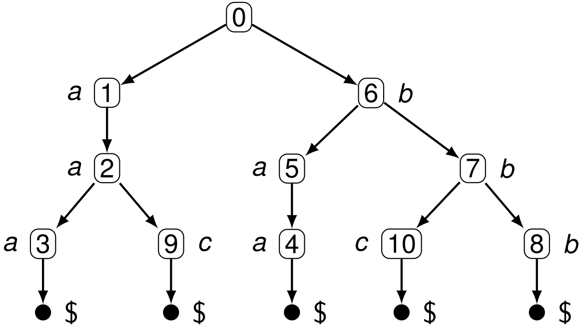
Each node of *EHOG*(P) is an explicit node of the Suffix Tree of P.

Use the BWT-AC decomposition



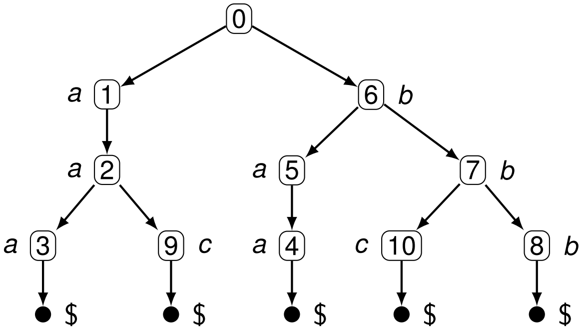
$$S = \{aaa\$, aac\$, baa\$, bbc\$, bbb\$\}$$

Use the BWT-AC decomposition



$$S = \{aaa\$, aac\$, baa\$, bbc\$, bbb\$\}$$

Use the BWT-AC decomposition

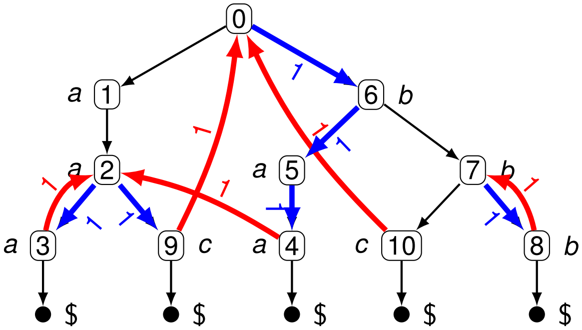


$$S = \{aaa\$, aac\$, baa\$, bbc\$, bbb\$\}$$

$$m_P = aaa\$caa\$aab\$bbb\$cbb\$$$

	0	1	2	3	4	5	6	7	8	9	10
$BWT(m_P)$	b a b a b	a a	c a	$	$	a	b a b	c b	$	$	$

Use the BWT-AC decomposition

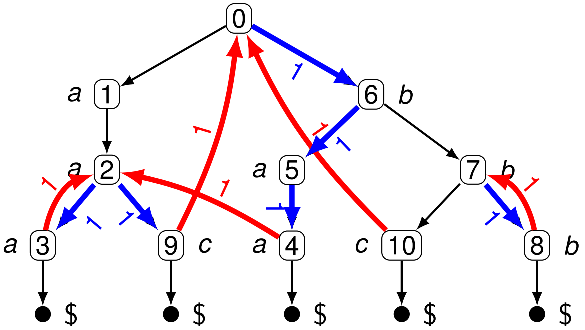


$$S = \{aaa\$, aac\$, baa\$, bbc\$, bbb\$\}$$

$$m_P = aaa\$caa\$aab\$bbb\$cbb\$$$

	0	1	2	3	4	5	6	7	8	9	10
$BWT(m_P)$	b a b a b	a a	c a	$	$	a	b a b	c b	$	$	$

Use the BWT-AC decomposition



$$S = \{aaa\$, aac\$, baa\$, bbc\$, bbb\$\}$$

$$m_P = aaa\$caa\$aab\$bbb\$cbb\$$$

	0	1	2	3	4	5	6	7	8	9	10
$BWT(m_P)$	*b a b a b*	*a a*	*c a*	$	$	*a*	*b a b*	*c b*	$	$	$
$Blue(m_P)$	1	1	1	01	01	01	01	1	01	01	1
$Red(m_P)$	1	1	1	01	01	1	1	1	01	01	01

Result

For a set of strings P,the size of the tables *Blue* et *Red* are bounded by $||P||$ $(=\sum_{s\in P}|s|)$.

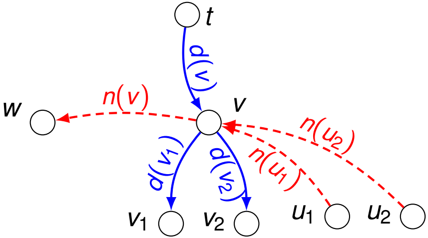
$\rightarrow$ By using BWT and BP, we can store the Superstring Graph in $O(n\log\sigma)$ bits.

How can we **build** the Superstring Graph in compact space?

Difficulty

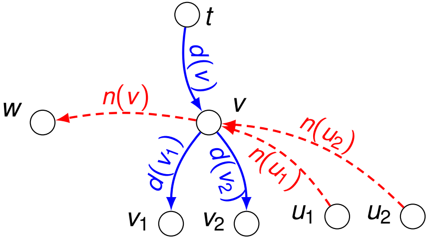
If we want just to compute the tables *Blue* and *Red*.

If we want just to compute the tables *Blue* and *Red*.



For now, algorihtm in $O(||P||)$ linear time and in $O(||P|| \log ||P||)$ bits space.

If we want just to compute the tables *Blue* and *Red*.



For now, algorihtm in $O(||P||)$ linear time and in $O(||P|| \log ||P||)$ bits space.

Can we find an algorithm in $O(||P||)$ linear time and in $O(||P||)$ bits space?

Thank you for your attention