SPIRE 2023

Generalized minimal absent words of multiple strings

Kouta Okabe[1], Takuya Mieno[2], Yuto Nakashima[1], Shunsuke Inenaga[1], Hideo Bannai[3]

[1] Kyushu University
[2] University of Electro-Communications
[3] Tokyo Medical and Dental University

Minimal Absent Words (MAWs) [1/2]

- A string w over an alphabet Σ is called a **Minimal Absent Word (MAW)** for a string S, if:
  1. w is a character from Σ not occurring in S, or
  2. $w = aub$ ($a, b \in \Sigma$, $u \in \Sigma^*$) does not occur in S, but both au and ub occur in S.

Example

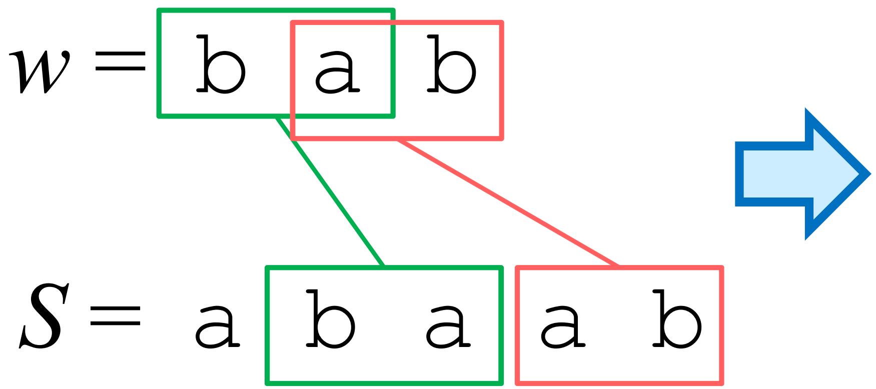
$w =$ b a b

$S =$ a b a a b

Minimal Absent Words (MAWs) [1/2]

- A string w over an alphabet Σ is called a **Minimal Absent Word (MAW)** for a string S, if:
  1. w is a character from Σ not occurring in S, or
  2. $w = aub$ ($a, b \in \Sigma$, $u \in \Sigma^*$) does not occur in S, but both au and ub occur in S.

Example



bab is a MAW for abaab

Minimal Absent Words (MAWs) [2/2]

- $\mathsf{MAW}(S)$ denotes the set of MAWs for a string S.

Example

$S =$ `abaab` $\qquad \Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$

$\mathsf{MAW}(S) = \{\mathtt{aaa}, \mathtt{aaba}, \mathtt{bab}, \mathtt{bb}, \mathtt{c}\}$

- The <u>number $|\mathsf{MAW}(S)|$ of MAWs</u> for a string S of length n over an alphabet of size σ is $O(\sigma n)$, and there is a matching lower bound [Crochemore et al. 1998].

Symmetric Difference of MAWs of Two Strings

- A string similarity measure based on the symmetric difference $\mathsf{MAW}(S_1) \triangle \mathsf{MAW}(S_2)$ of MAWs for two input strings S_1 and S_2 has been proposed [Chairungsee & Crochemore, 2012].
- Enumeration: $\mathsf{MAW}(S_1) \triangle \mathsf{MAW}(S_2)$ can be computed in $O(\sigma n)$ time and space [Charalampopoulos et al., 2018].
- Counting: The cardinality $|\mathsf{MAW}(S_1) \triangle \mathsf{MAW}(S_2)|$ can be computed in $O(n)$ time for integer alphabets [Charalampopoulos, Crochemore, Pissis, 2018].

Our Starting Point

Can we compute all elements of $\mathsf{MAW}(S_1) \triangle \mathsf{MAW}(S_2)$ in optimal $O(n + |\mathsf{MAW}(S_1) \triangle \mathsf{MAW}(S_2)|)$ time?

Our Problem

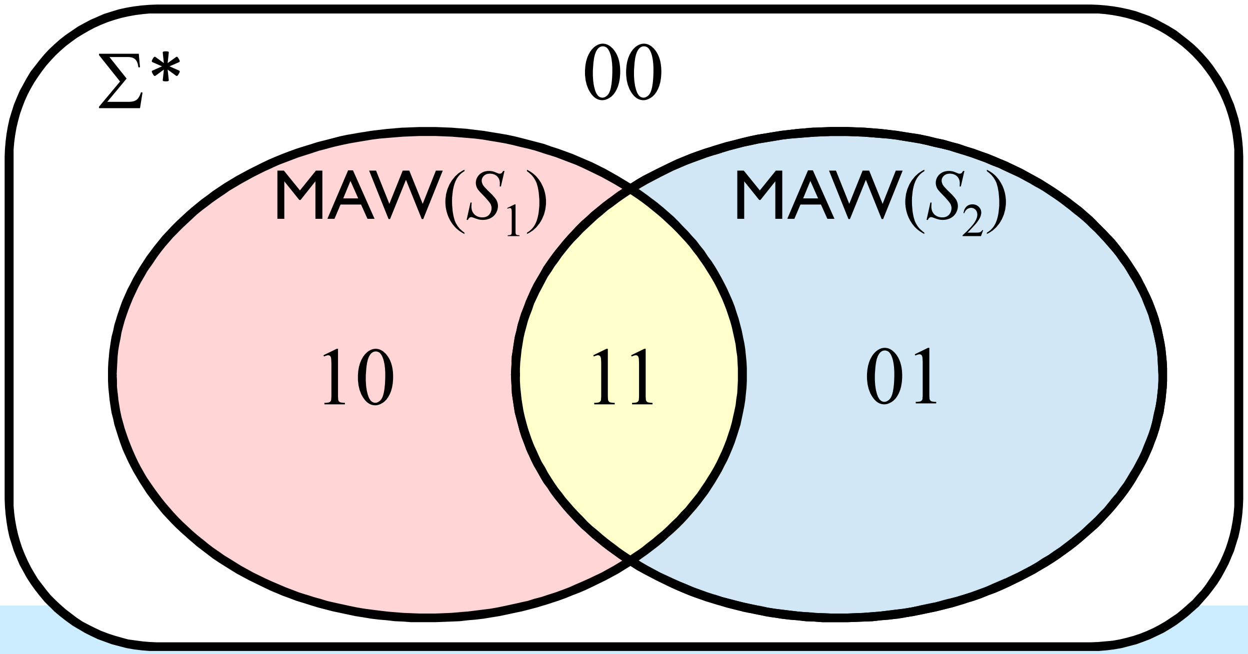
- We extend the notion of MAWs to $k \geq 2$ strings as follows:

Problem 1

Input: Set $\mathbf{S} = \{S_1, \ldots, S_k\}$ of k strings of total length n
and a bit vector $\mathbf{B}$ of length k.

Output: $\mathsf{MAW}(\mathbf{B}) = \{w \mid w$ is a MAW for string S_i iff $\mathbf{B}[i] = 1\}$.

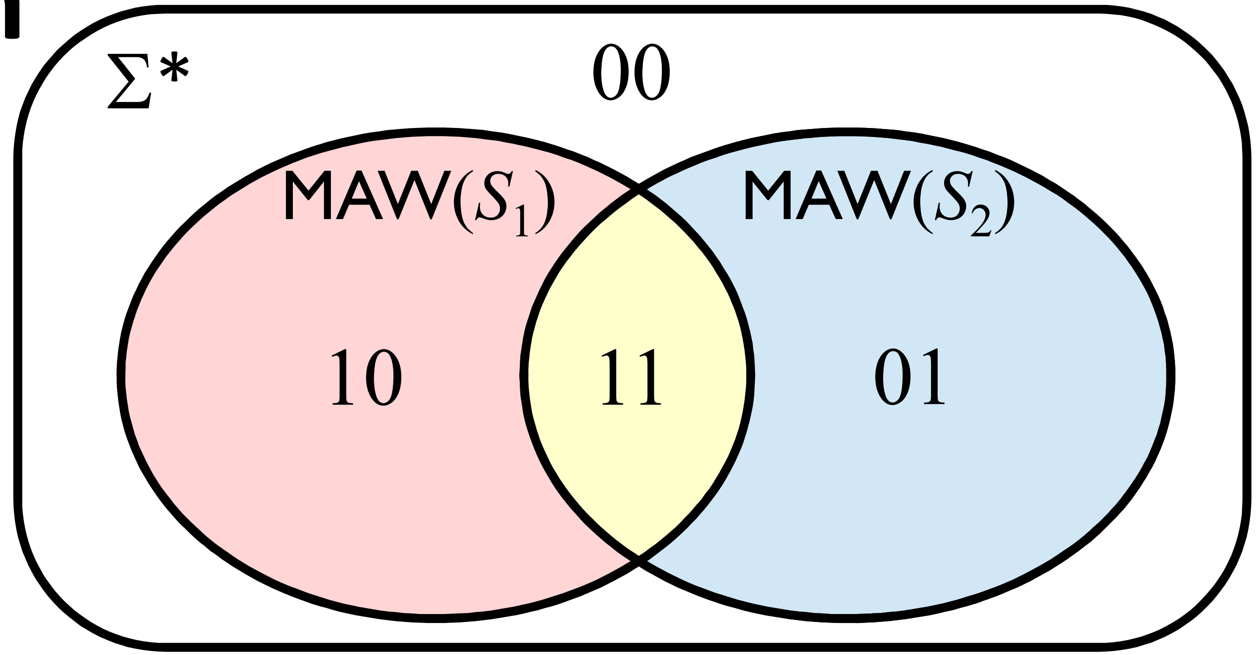
When $k = 2$



$\mathsf{MAW}(S_1) \triangle \mathsf{MAW}(S_2)$
is obtained by computing
$\mathsf{MAW}(10)$ and $\mathsf{MAW}(01)$.

Our Problem

When $k = 2$



Example

$S_1 =$ abaab

$S_2 =$ aacbba

$\Sigma = \{$a, b, c, d$\}$

MAW(10) = {aaba, bab, bb, c}

MAW(11) = {aaa, d}

MAW(01) = {ab, baa, bac, bbb, bc, ca, cba, cc}

Our Contributions

Problem 1

Input: Set $\mathbf{S} = \{S_1, \ldots, S_k\}$ of k strings of total length n
and a bit vector $\mathbf{B}$ of length k.

Output: $\mathsf{MAW}(\mathbf{B}) = \{w \mid w$ is a MAW for string S_i iff $\mathbf{B}[i] = 1\}$.

Theorem 1

For $k = 2$, we can solve Problem 1
in optimal $\mathrm{O}(n + |\mathsf{MAW}(\mathbf{B})|)$ time with $\mathrm{O}(n)$ working space.

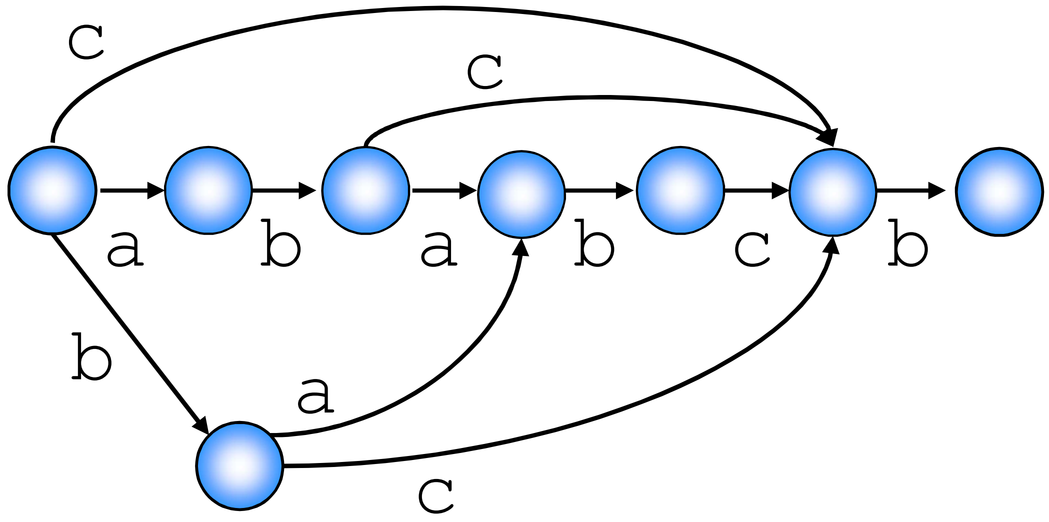
Theorem 2

For general $k > 2$, we can solve Problem 1
in $\mathrm{O}\left(n \left\lceil \frac{k}{\log n} \right\rceil + |\mathsf{MAW}(\mathbf{B})|\right)$ time with $\mathrm{O}(n)$ working space.

Computing MAWs with DAWG [1/2]

- Previous algorithms [Crochemore et al. 1998, Fujishige et al. 2016] for computing MAWs for a <u>single string</u> S use **DAWG (Directed Acyclic Word Graph)** for S, which is an $\mathrm{O}(n)$-size automaton representing all substrings of S.

E.g. $S = \mathtt{ababcb}$

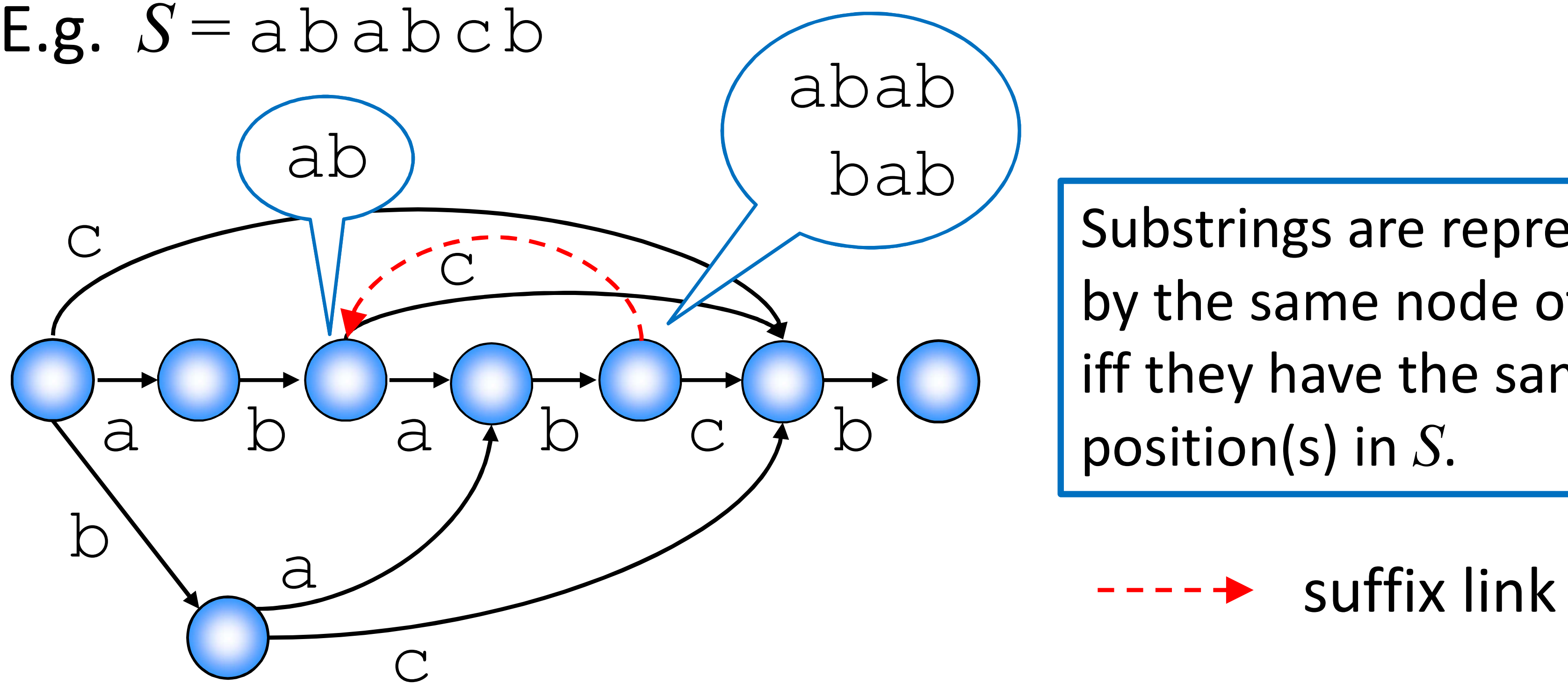


Substrings are represented by the same node of DAWG(S) iff they have the same ending position(s) in S.

Computing MAWs with DAWG [1/2]

- Previous algorithms [Crochemore et al. 1998, Fujishige et al. 2016] for computing MAWs for a single string S use **DAWG (Directed Acyclic Word Graph)** for S, which is an $\mathrm{O}(n)$-size automaton representing all substrings of S.

E.g. $S = \mathtt{ababcb}$

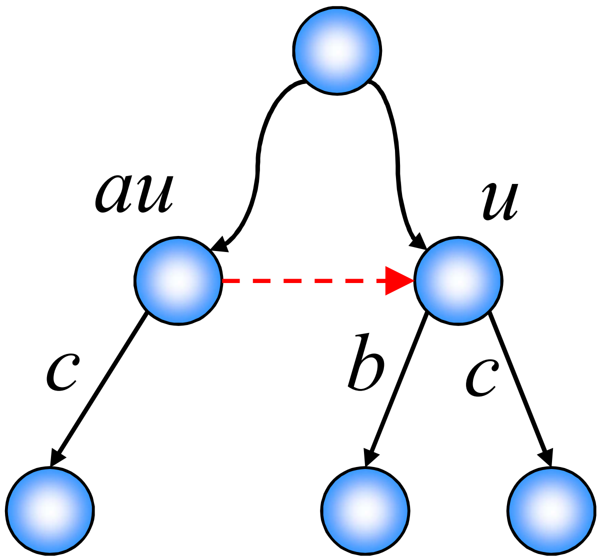


Substrings are represented by the same node of DAWG(S) iff they have the same ending position(s) in S.

Computing MAWs with DAWG [1/2]

- If the edges of DAWG are sorted, then one can compute MAW(S) in $O(n + |\mathrm{MAW}(S)|)$ time [Fujishige et al. 2016].

DAWG for string S



- Consider each pair of nodes au and u which are connected by a suffix link, where a is a character and u is a string.
- Compare the labels of the out-edges of nodes au and u in sorted order.
  - For b: au has no out-edge with b, but u has an out-edge with b.
    → aub is a MAW for the input string S.
  - For c: both au and u have out-edges with c
    → auc is not a MAW for the input string S, but this cost of character comparisons can be charged to this out-edge of au labeled c.

Building DAWG for Multiple Strings

- The best known algorithm for building the DAWG for multiple strings takes $O(n \log \sigma)$ time [Blumer et al. 1985].

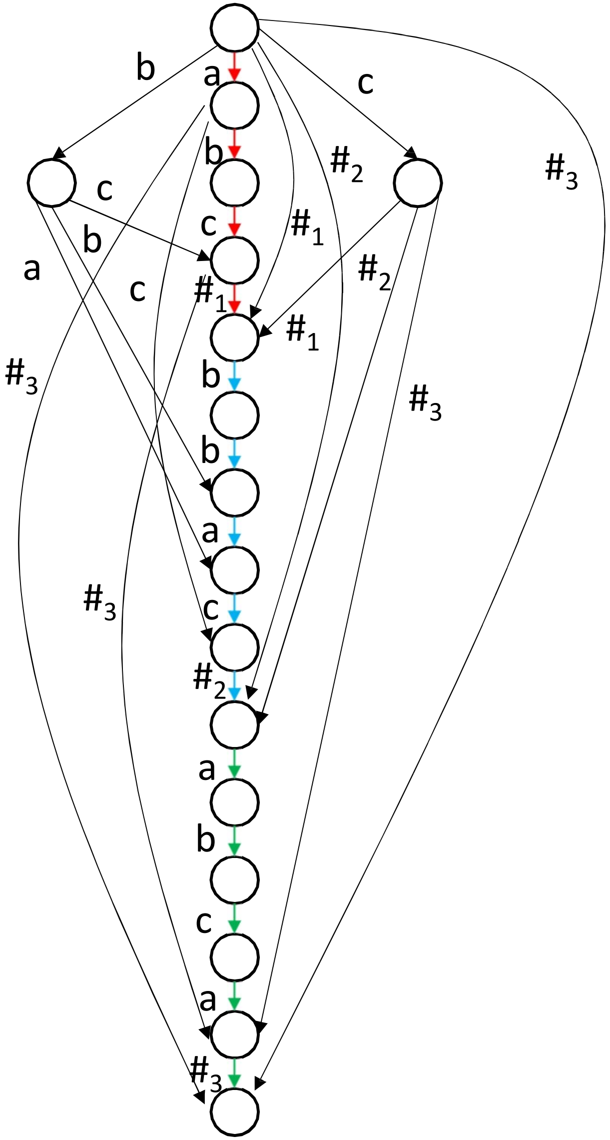
Lemma 1

The DAWG for a set $\mathbf{S} = \{S_1\#_1, \ldots, S_k\#_k\}$ of k strings of total length n can be built in $O(n)$ time for integer alphabets.

1. We build the DAWG for the concatenated string $T = S_1\#_1 \cdots S_k\#_k$ in $O(n)$ time by the DAWG-construction algorithm of Fujishige et al. (2016) for a single string.
2. We convert the DAWG for T to the DAWG for $\mathbf{S}$ in $O(n)$ time.

Building DAWG for Multiple Strings [2/2]

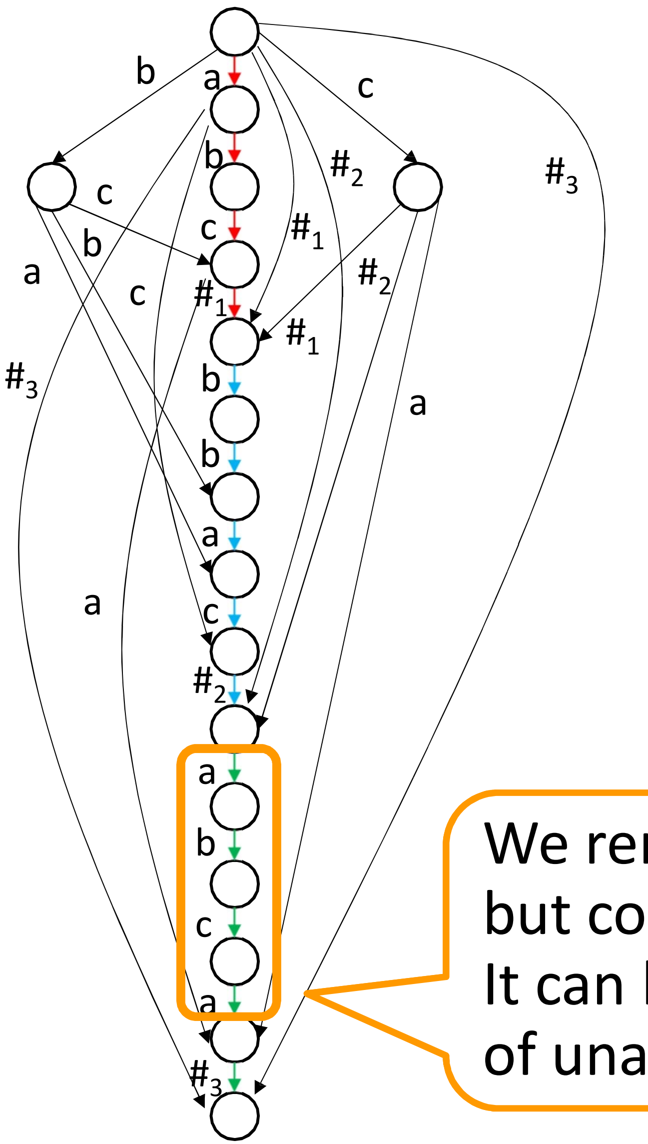
$T =$ abc$\#_1$ bbac$\#_2$ abca$\#_3$



We first build the DAWG for the concatenated string T.

Building DAWG for Multiple Strings [2/2]

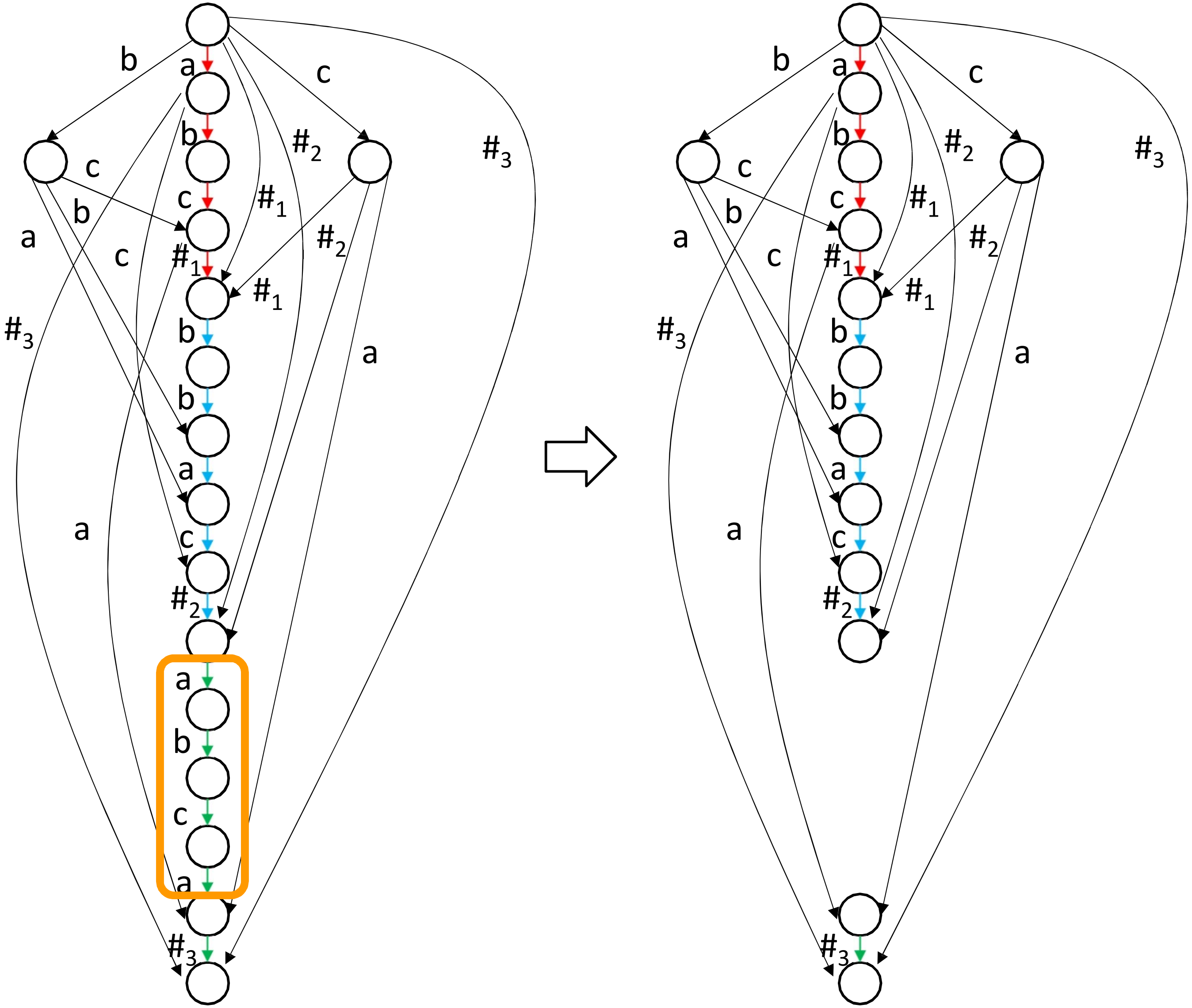
$T =$ **abc**$\#_1$ **bbac**$\#_2$ **abca**$\#_3$



We remove the paths that lead to $\#_3$ but contain $\#_1$ and/or $\#_2$ inside.
It can be done by deleting this chain of unary nodes from the "spine".

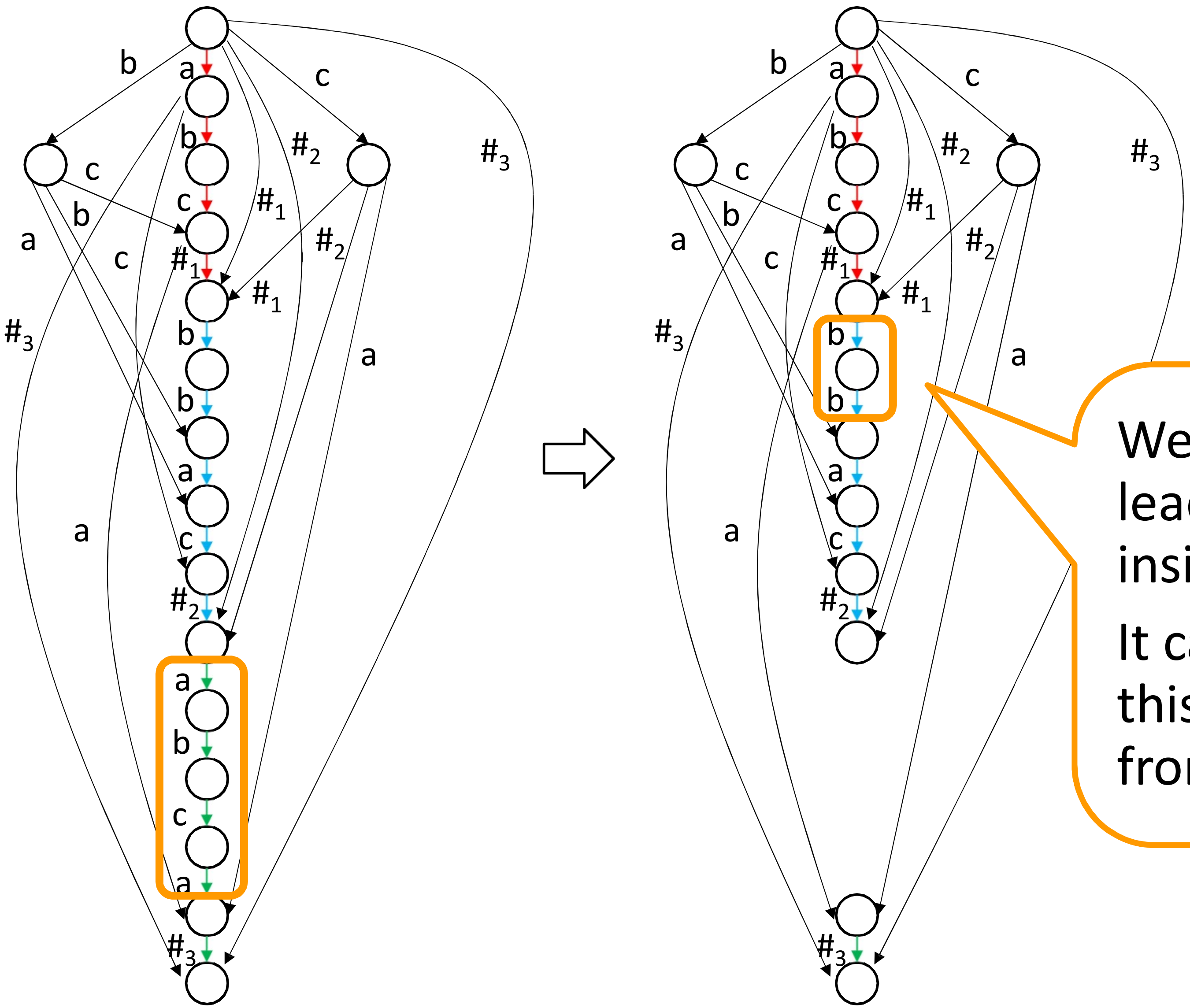
Building DAWG for Multiple Strings [2/2]

$T = \text{abc}\#_1\ \text{bbac}\#_2\ \text{abca}\#_3$



Building DAWG for Multiple Strings [2/2]

$T =$ **abc**$\#_1$ **bbac**$\#_2$ **abca**$\#_3$



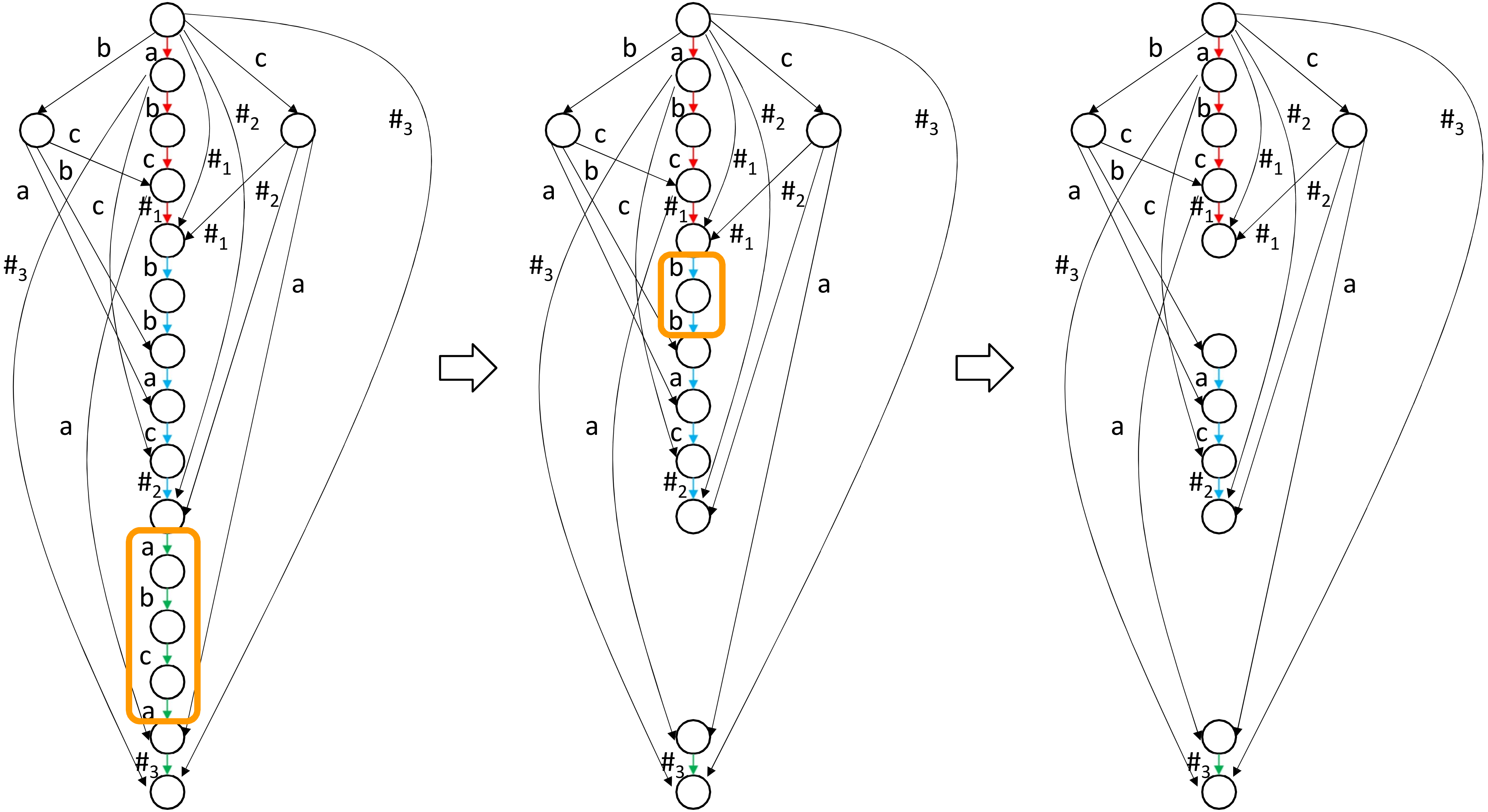
We remove the paths that lead to $\#_2$ but contain $\#_1$ inside.

It can be done by deleting this chain of unary nodes from the "spine".

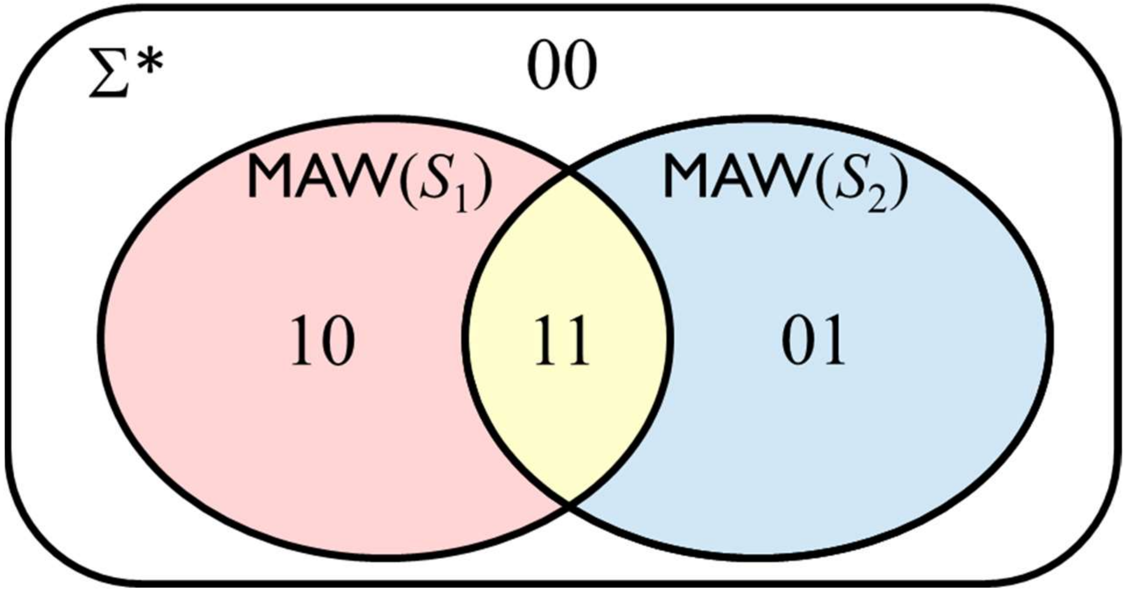
Building DAWG for Multiple Strings [2/2]

$T =$ **abc**$\#_1$ **bbac**$\#_2$ **abca**$\#_3$

DAWG for $\{$**abc**$\#_1$, **bbac**$\#_2$, **abca**$\#_3\}$

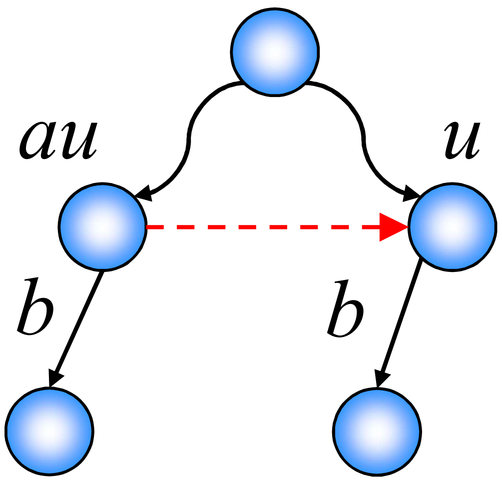


Computing MAWs for $k = 2$



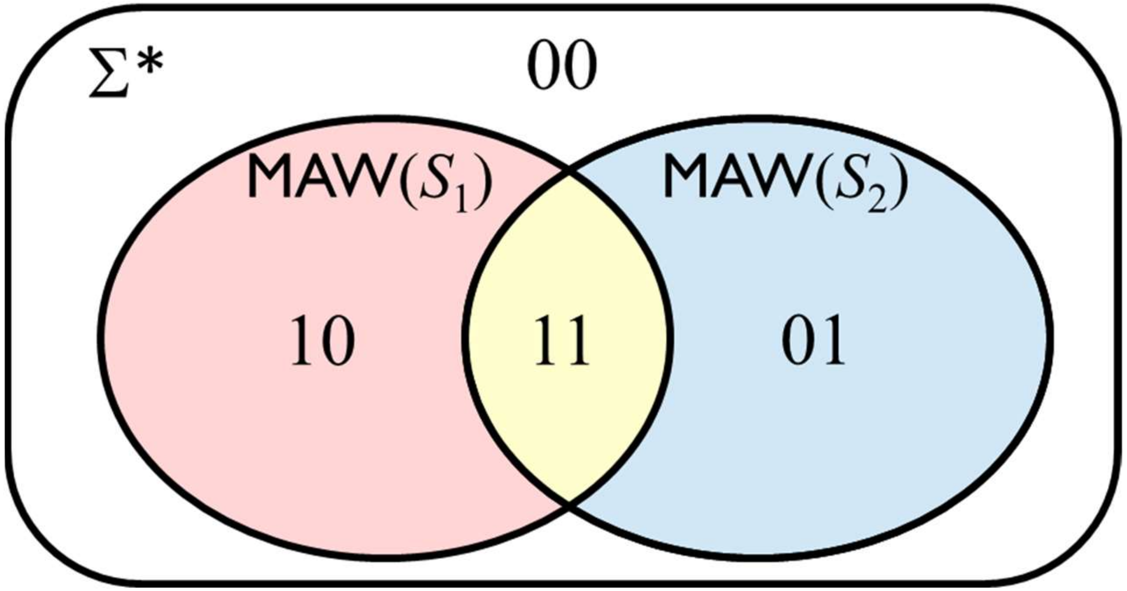
Node au is labeled $\#_1\#_2$ iff au is a substring of S_1 and S_2, and so on.

DAWG for two strings $S_1\#_1$ and $S_2\#_2$

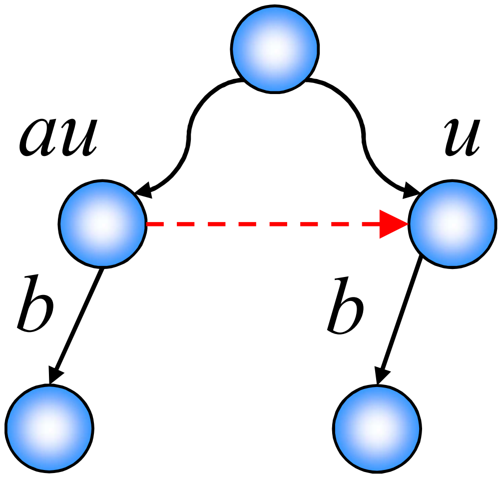


All possible combinations of node labels		au		
		$\#_1\#_2$	$\#_1$	$\#_2$
aub	ub	**B**		
$\#_1\#_2$	$\#_1\#_2$	00	-	-
$\#_1$	$\#_1$	00	00	-
	$\#_1\#_2$	01	00	-
$\#_2$	$\#_2$	00	-	00
	$\#_1\#_2$	10	-	00
absent	$\#_1$	10	10	00
	$\#_2$	01	00	01
	$\#_1\#_2$	11	10	01

Case where $\mathbf{B} = 10$



DAWG for two strings $S_1\#_1$ and $S_2\#_2$

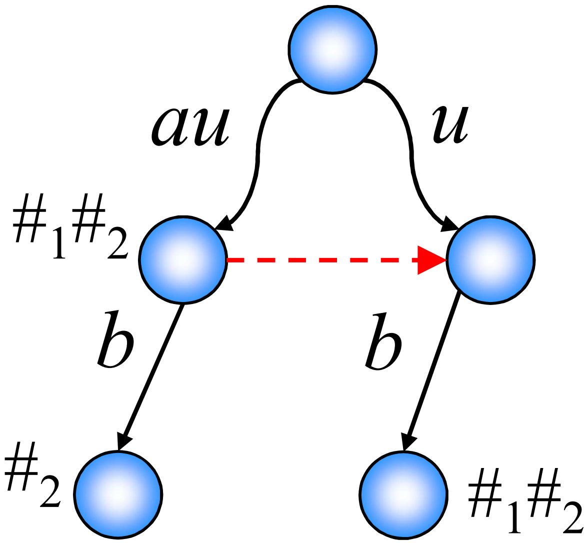


Node au is labeled $\#_1\#_2$ iff au is a substring of S_1 and S_2, and so on.

All possible combinations of node labels		au		
		$\#_1\#_2$	$\#_1$	$\#_2$
aub	ub	$\mathbf{B}$		
$\#_1\#_2$	$\#_1\#_2$	00	-	-
$\#_1$	$\#_1$	00	00	-
	$\#_1\#_2$	01	00	-
$\#_2$	$\#_2$	00	-	00
	$\#_1\#_2$	10	-	00
absent	$\#_1$	10	10	00
	$\#_2$	01	00	01
	$\#_1\#_2$	11	10	01

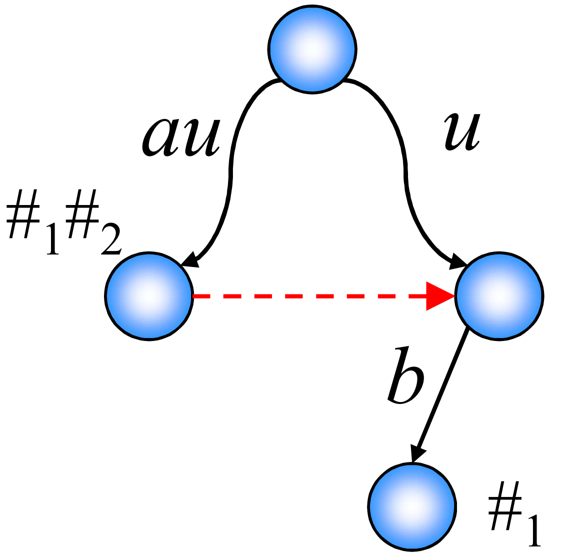
Case where $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11



- au is a substring of S_1 and S_2
- aub is not a substring of S_1
- aub is a substring of S_2
- ub is a substring of S_1 and S_2

➔ $aub \in$ MAW(10)

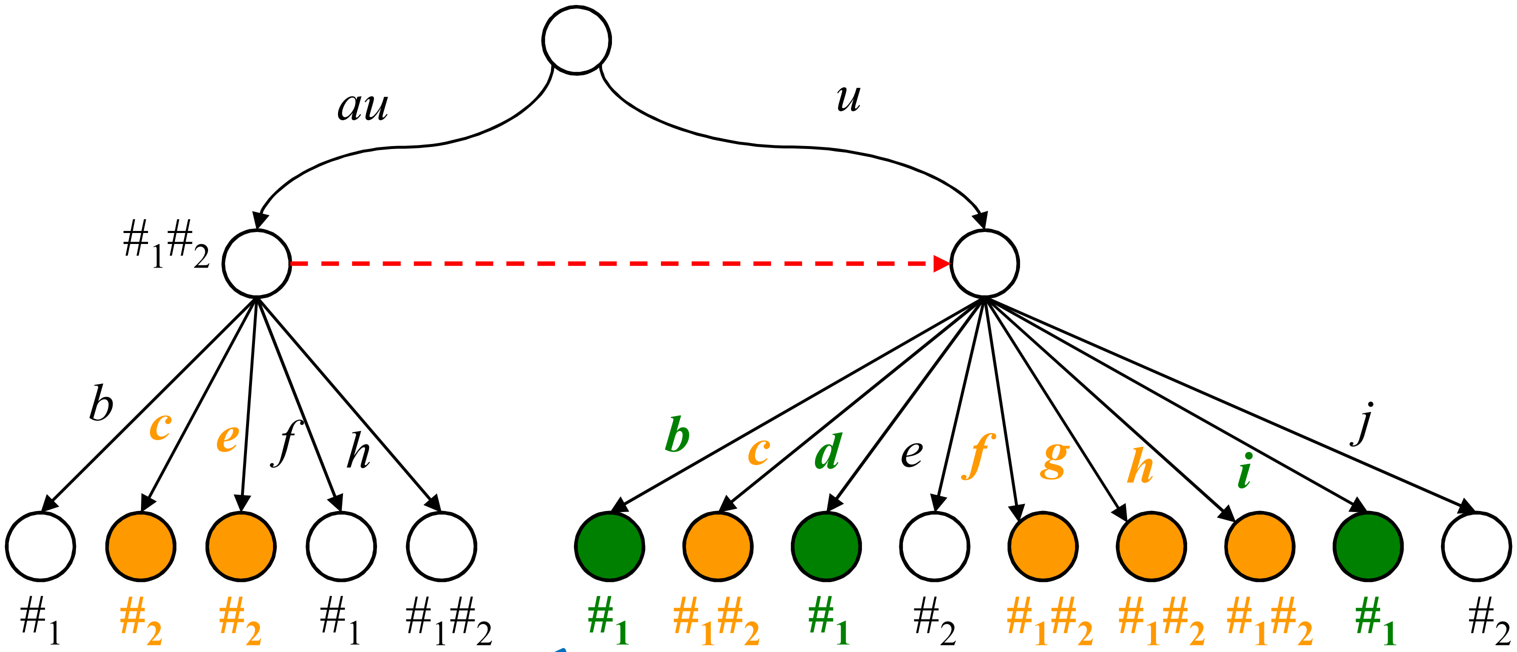


- au is a substring of S_1 and S_2
- aub is not a substring of S_1
- aub is not a substring of S_2
- ub is a substring of S_1
- ub is not a substring of S_2

➔ $aub \in$ MAW(10)

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

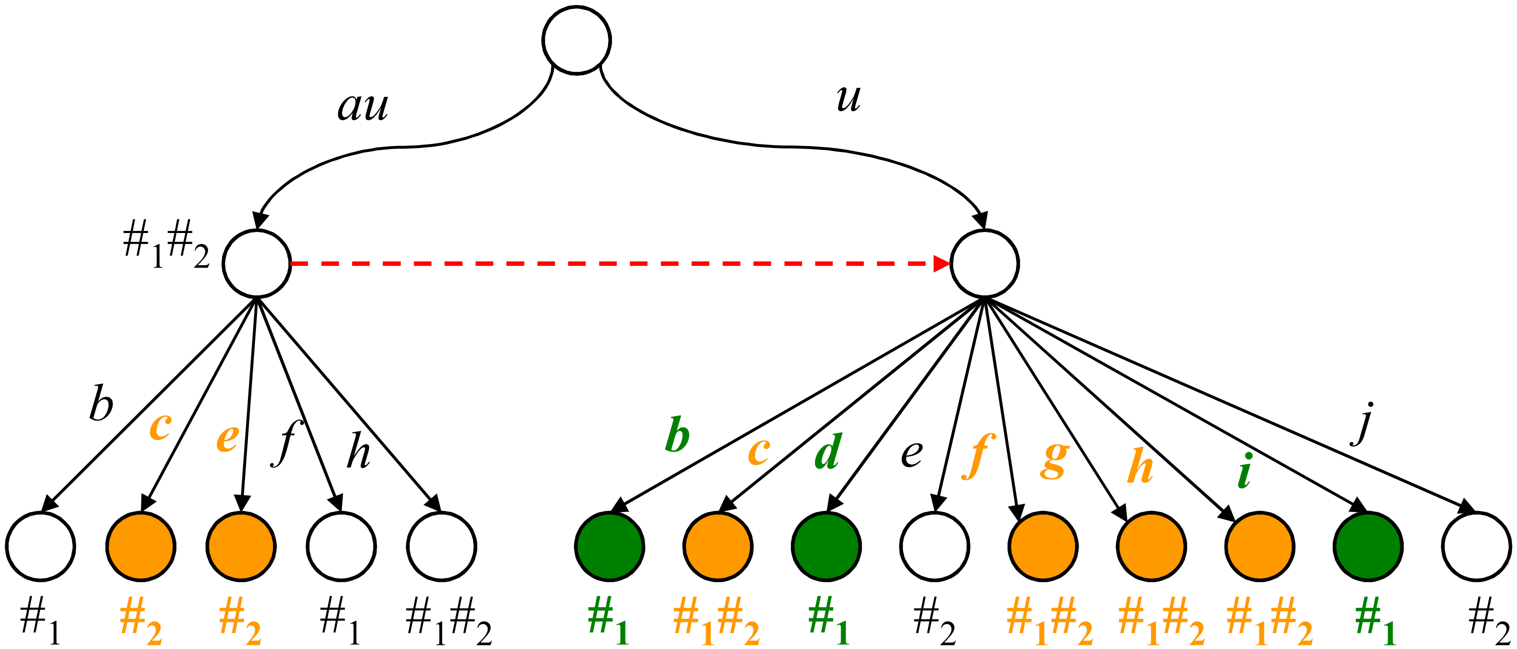
		au
		$\#_1\#_2$
aub	ub	**B**
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11



Here we highlight the out-edges of nodes au and u which meet <u>necessary conditions</u> for the orange/green cases shown in the table.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

		au
		$\#_1\#_2$
aub	ub	**B**
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11

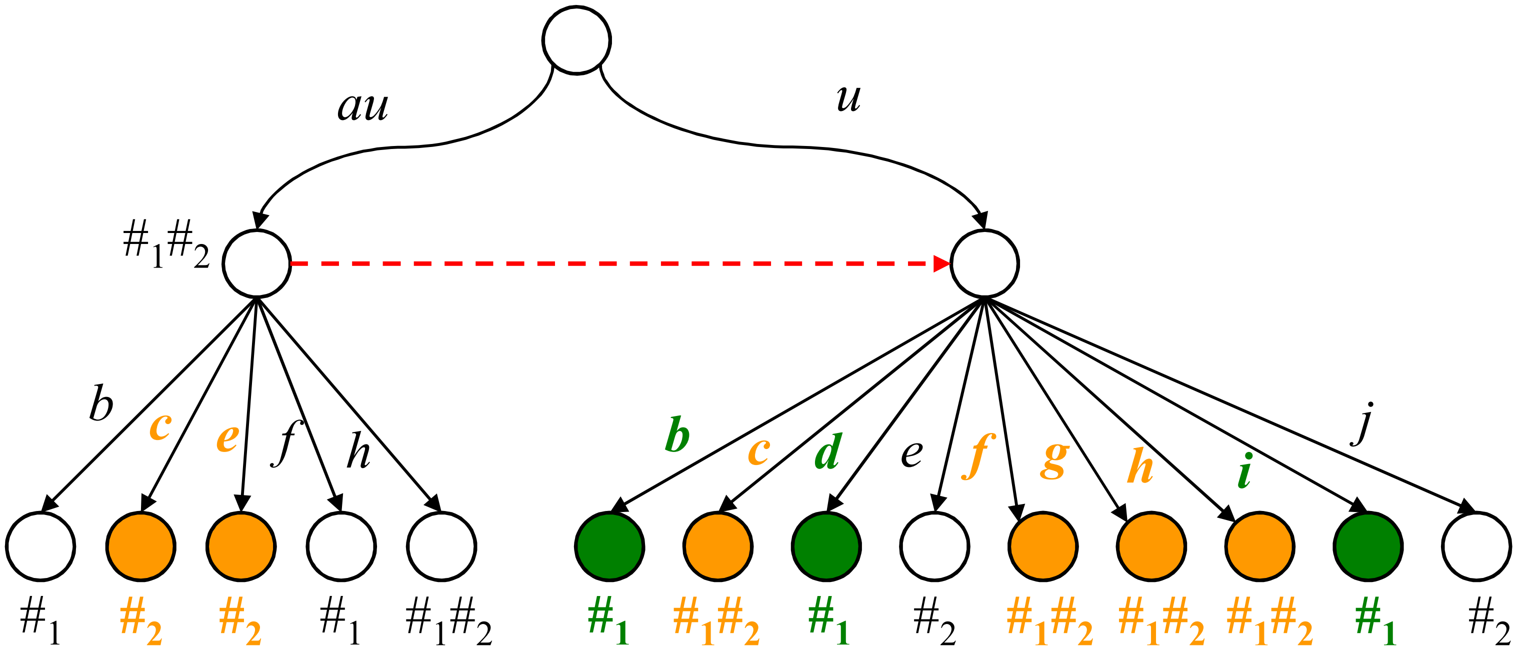


Out-edge of au	b	c		e	f		h			\$
Out-edge of u	b	c	d	e	f	g	h	i	j	\$

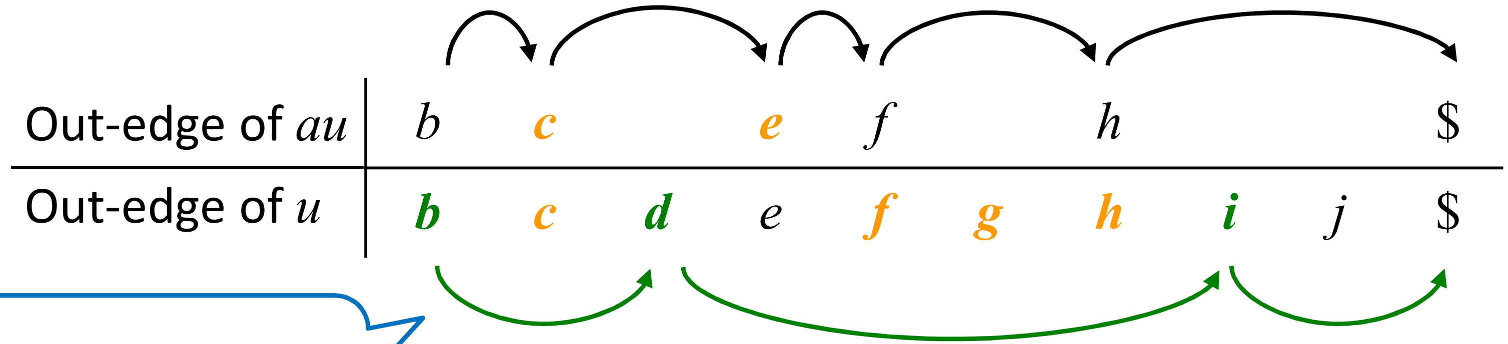
Sorted List of out-edges of nodes au and u.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11



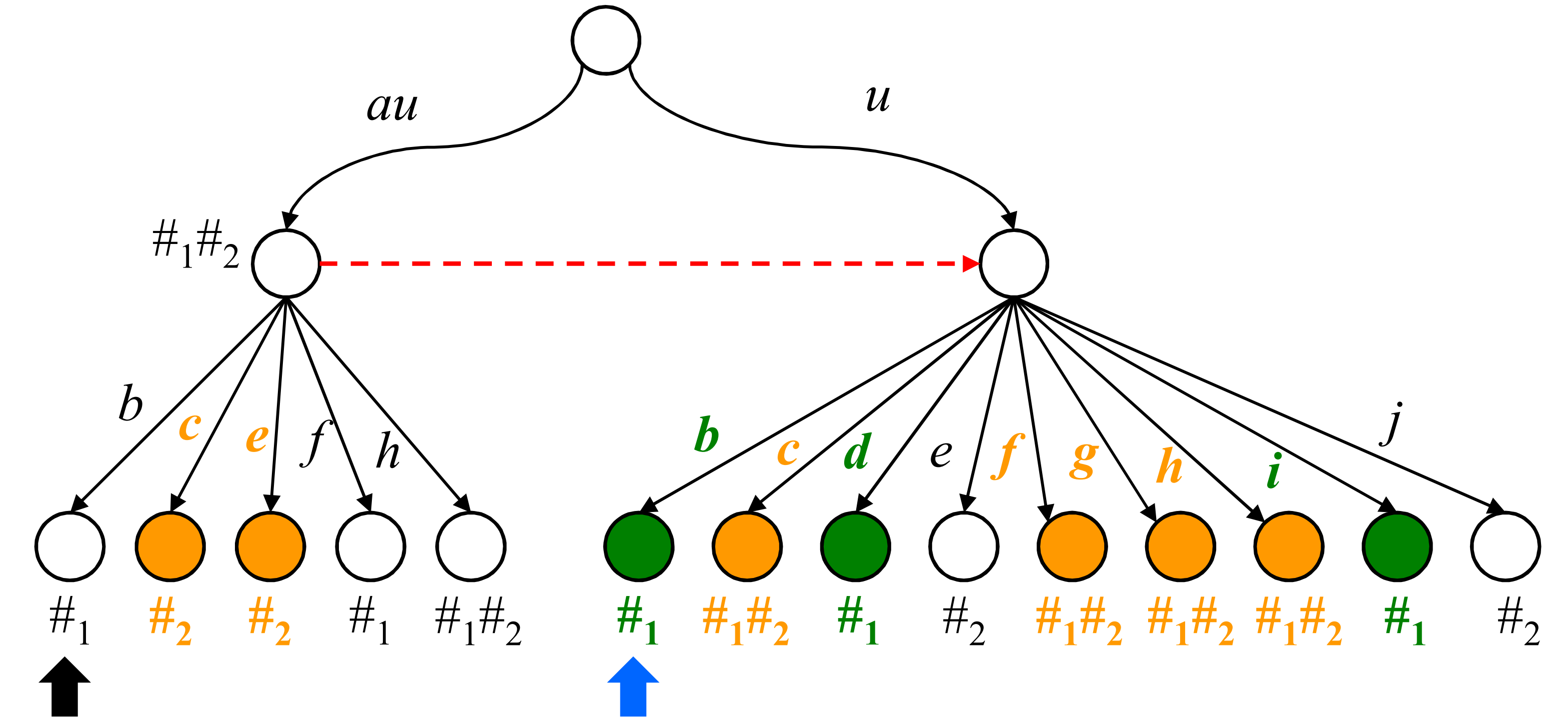
Link of all out-edges of au.

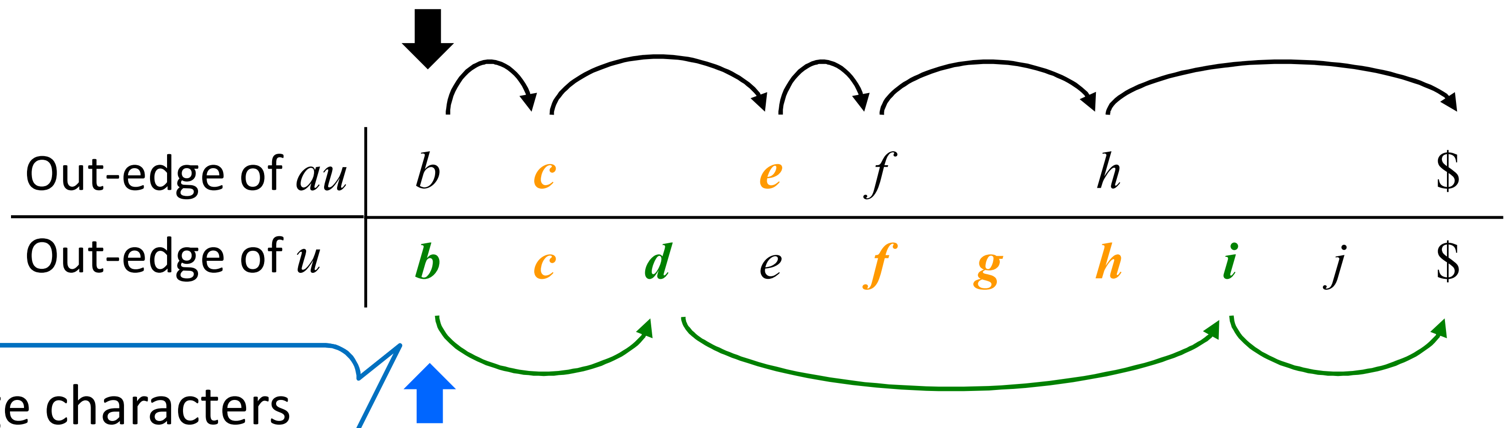


Link of the out-edges of node u that are labeled $\#_1$.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
$\#_1$	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
$\#_2$	$\#_1\#_2$	10
absent	$\#_1$	10
absent	$\#_2$	01
absent	$\#_1\#_2$	11

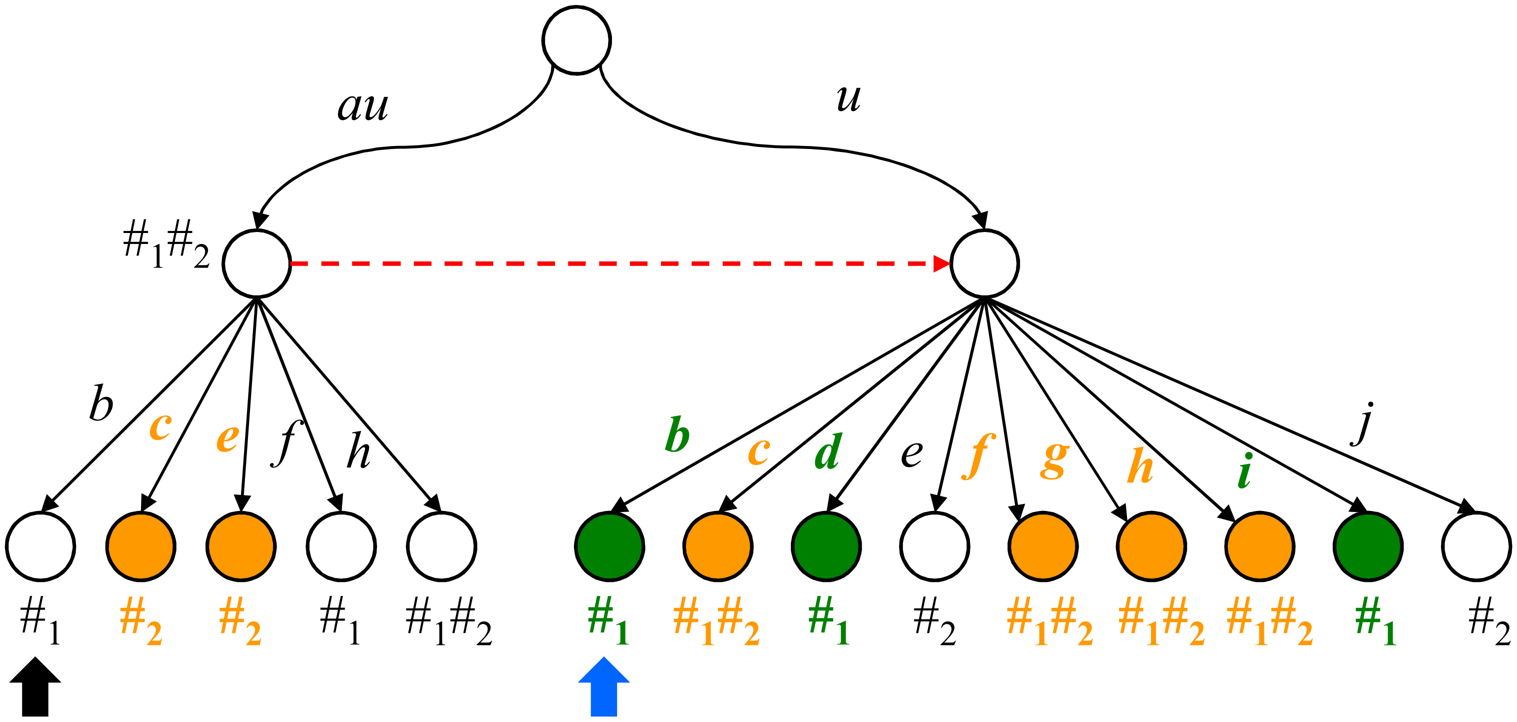


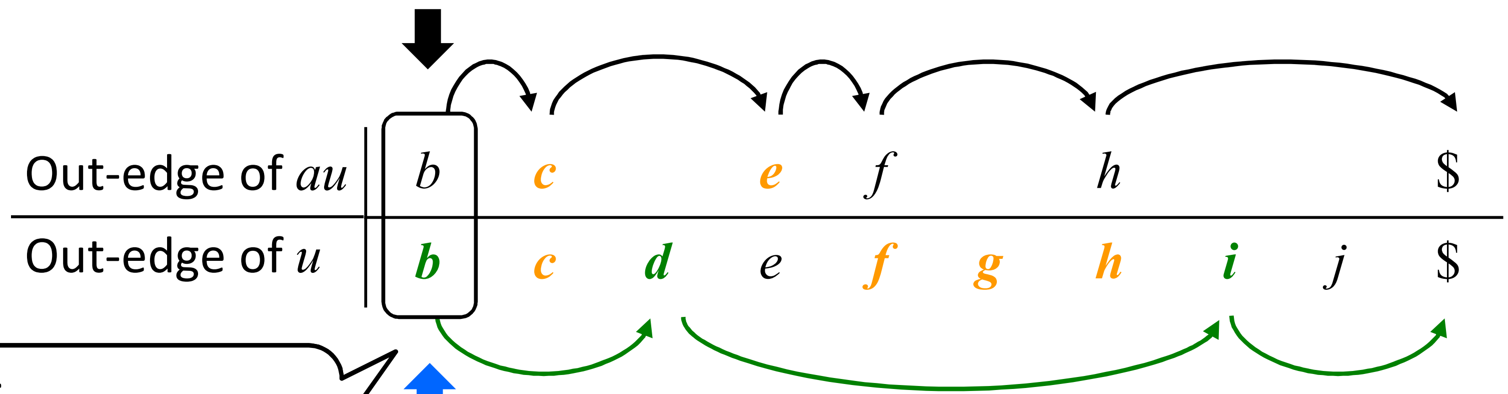


Compare the out-edge characters of au and u by following these links.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11

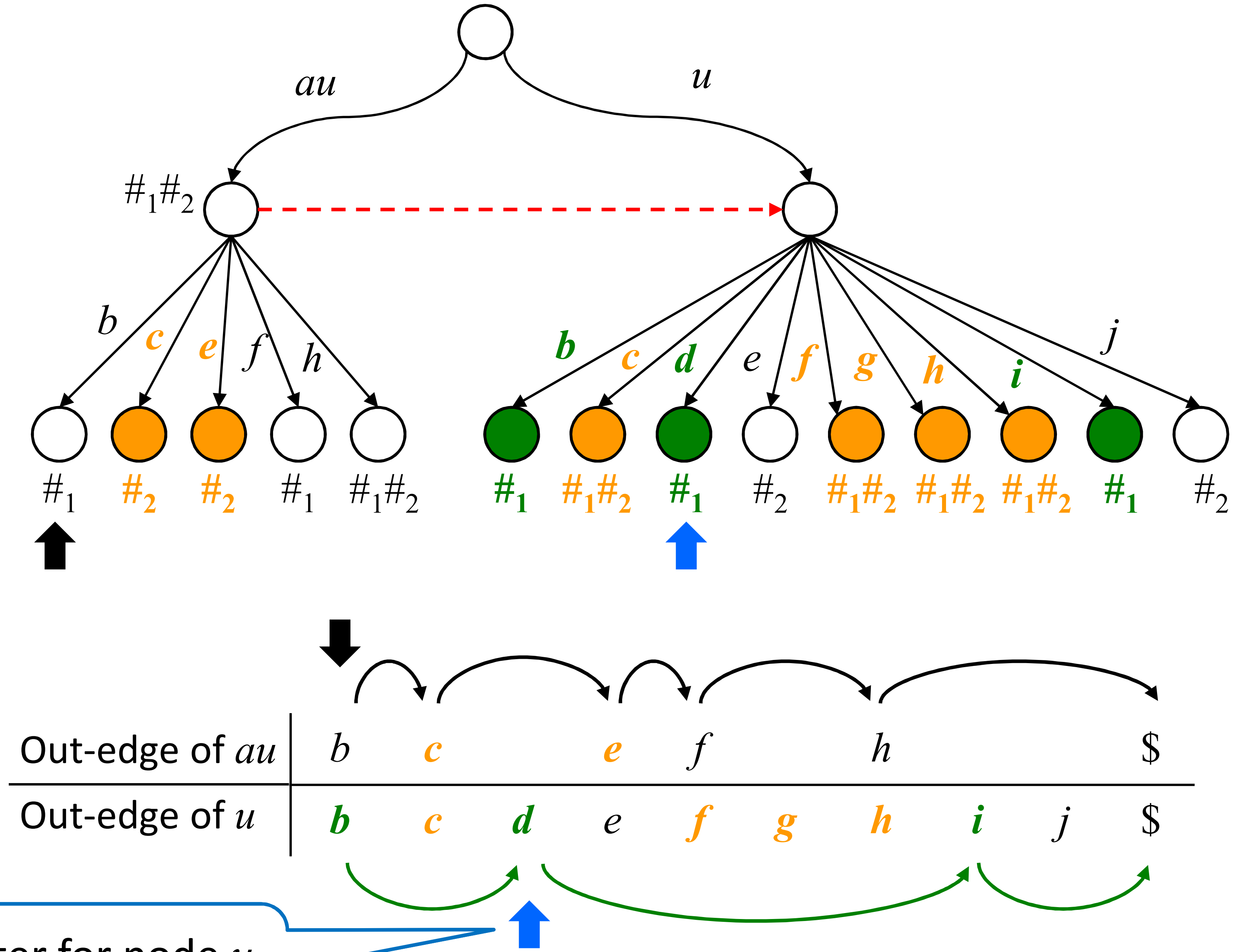




aub must be absent for green case
→ *aub* is not a MAW to output.

Algorithm for $\mathrm{B} = 10$ and au is labeled $\#_1\#_2$

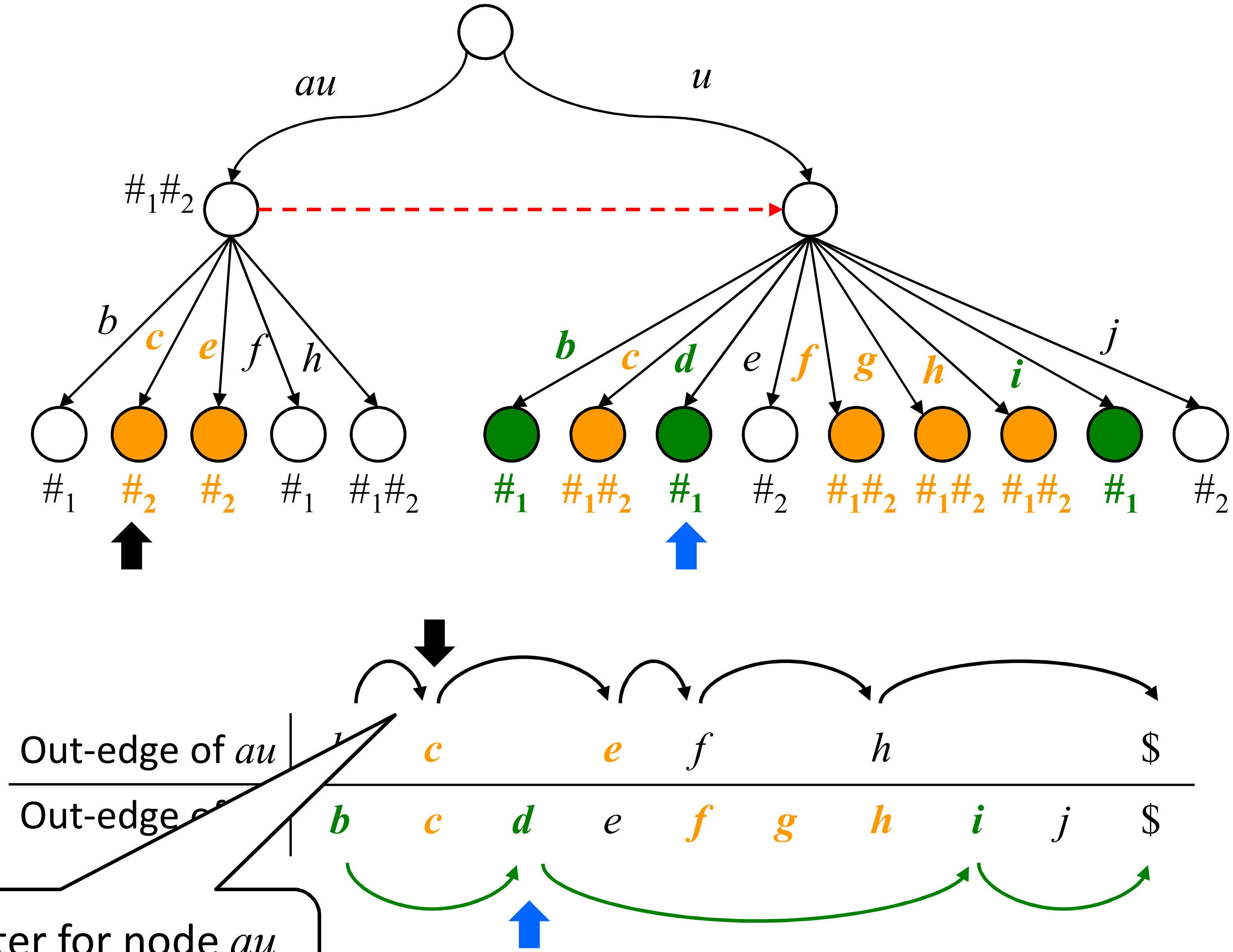
		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11



Shift the pointer for node u by following the link.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

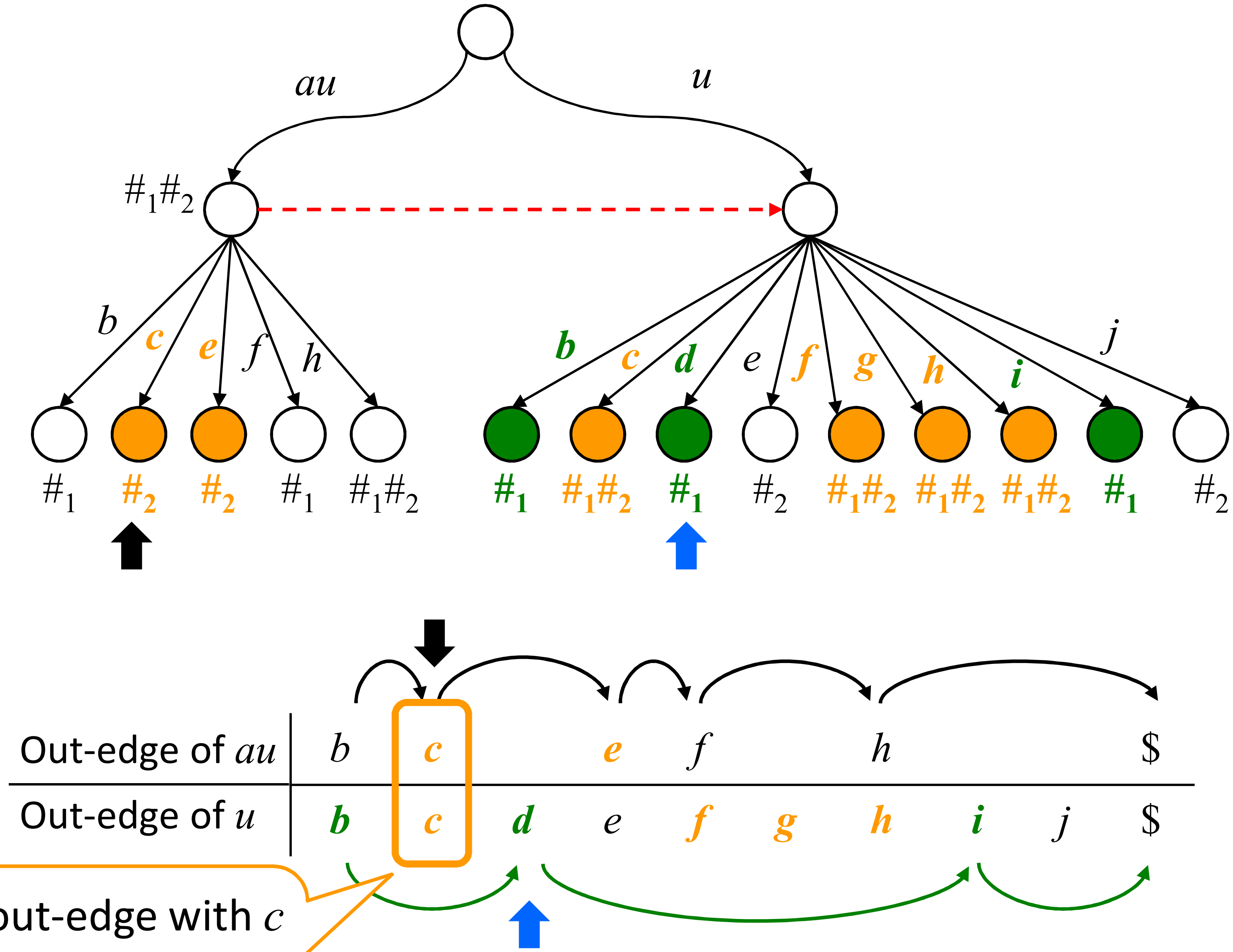
		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11



Shift the pointer for node au
by following the link.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

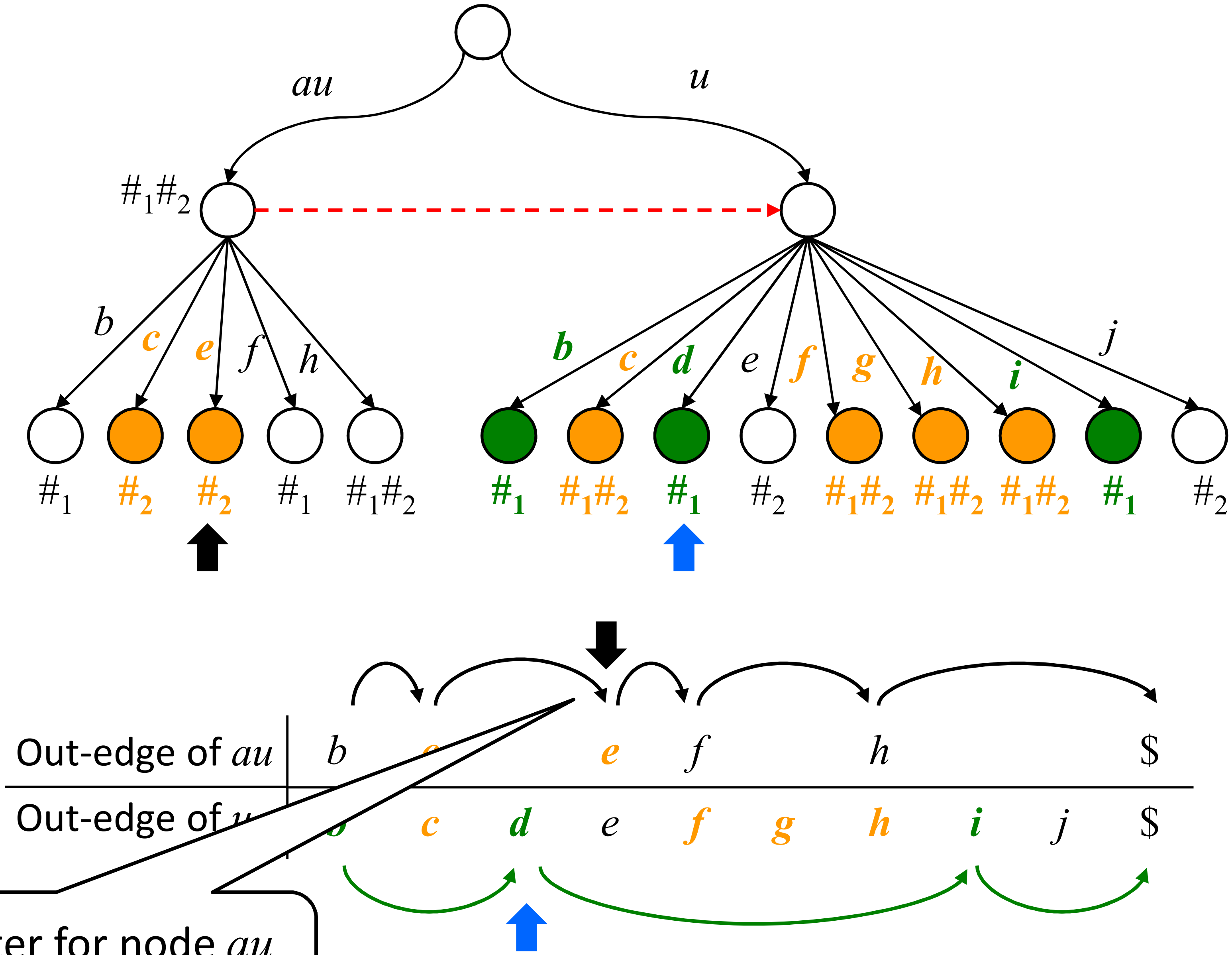
		au
		$\#_1\#_2$
aub	ub	**B**
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11



Both au and u have out-edge with c with the condition for orange case
→ auc is a MAW to output.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

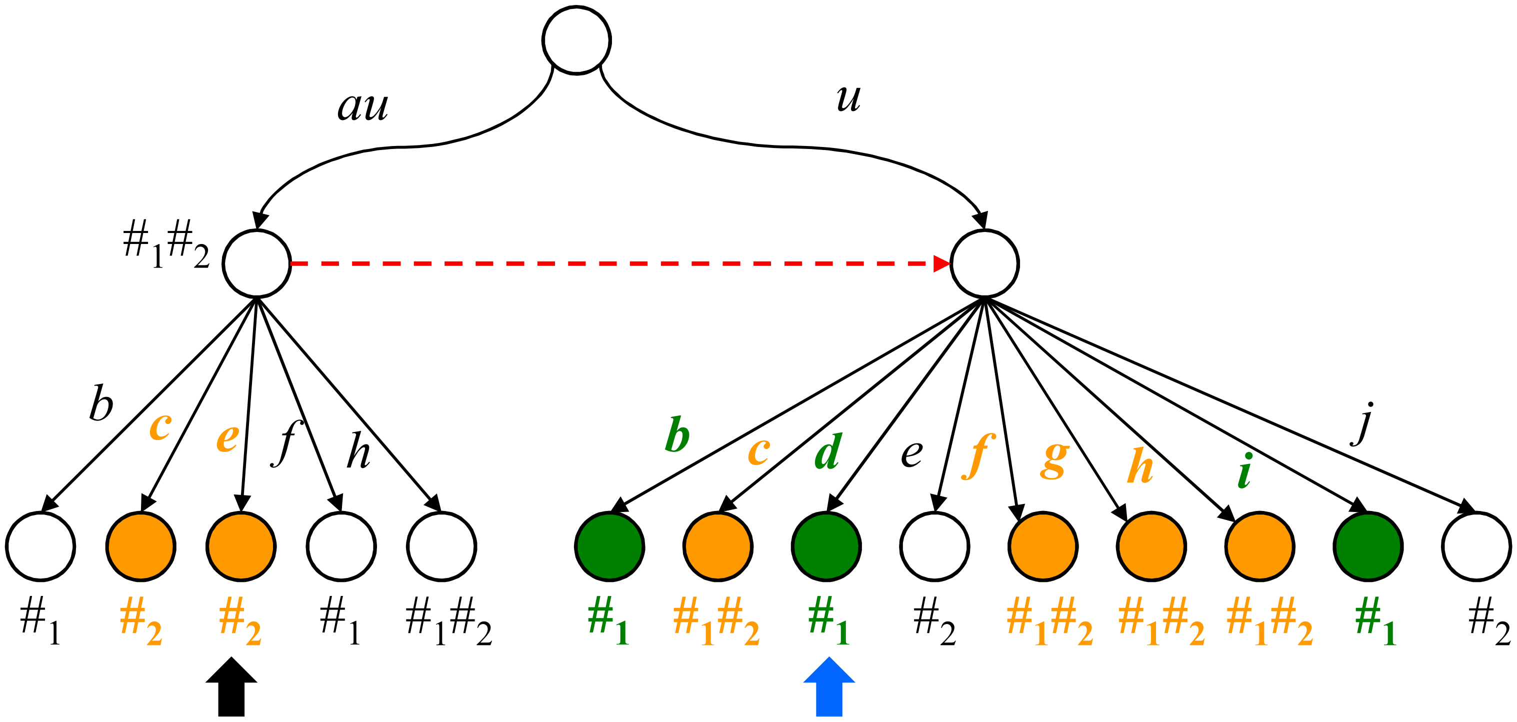
		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11

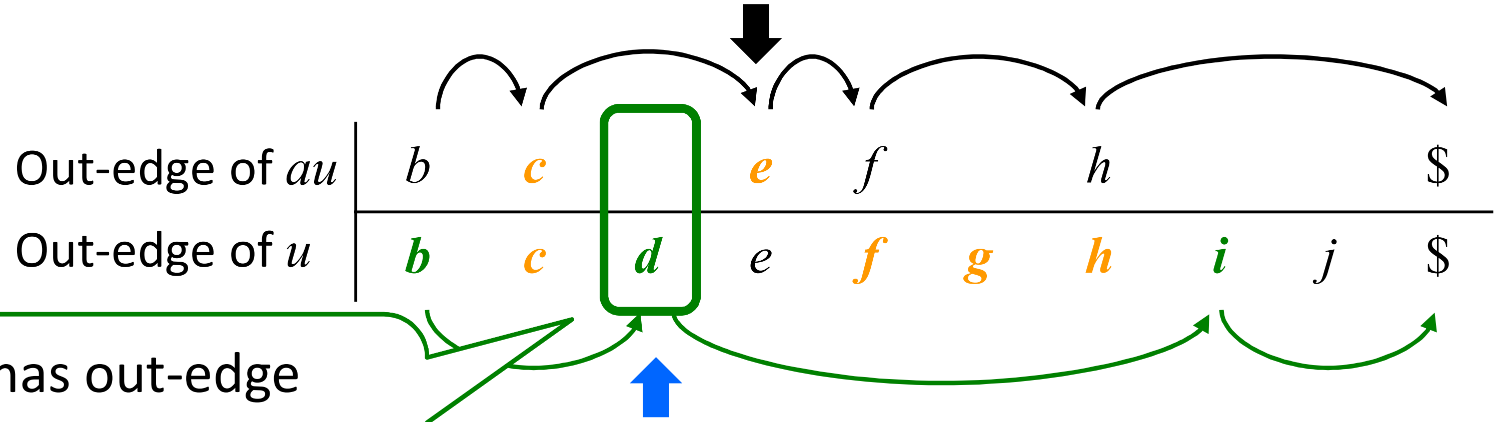


Shift the pointer for node *au* by following the link.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

		au
		$\#_1\#_2$
aub	ub	**B**
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11

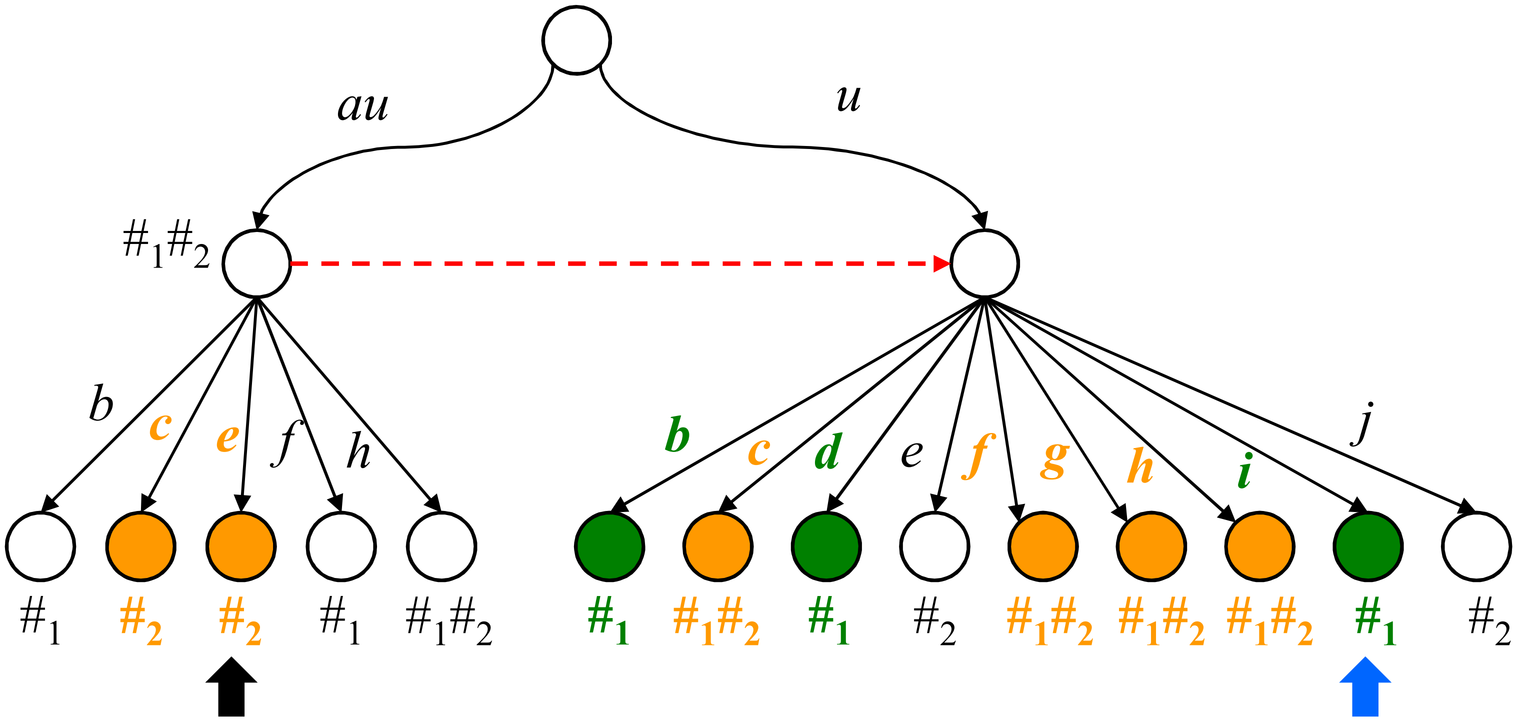




aud is absent and u has out-edge with d with condition for green case → aud is a MAW to output.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11

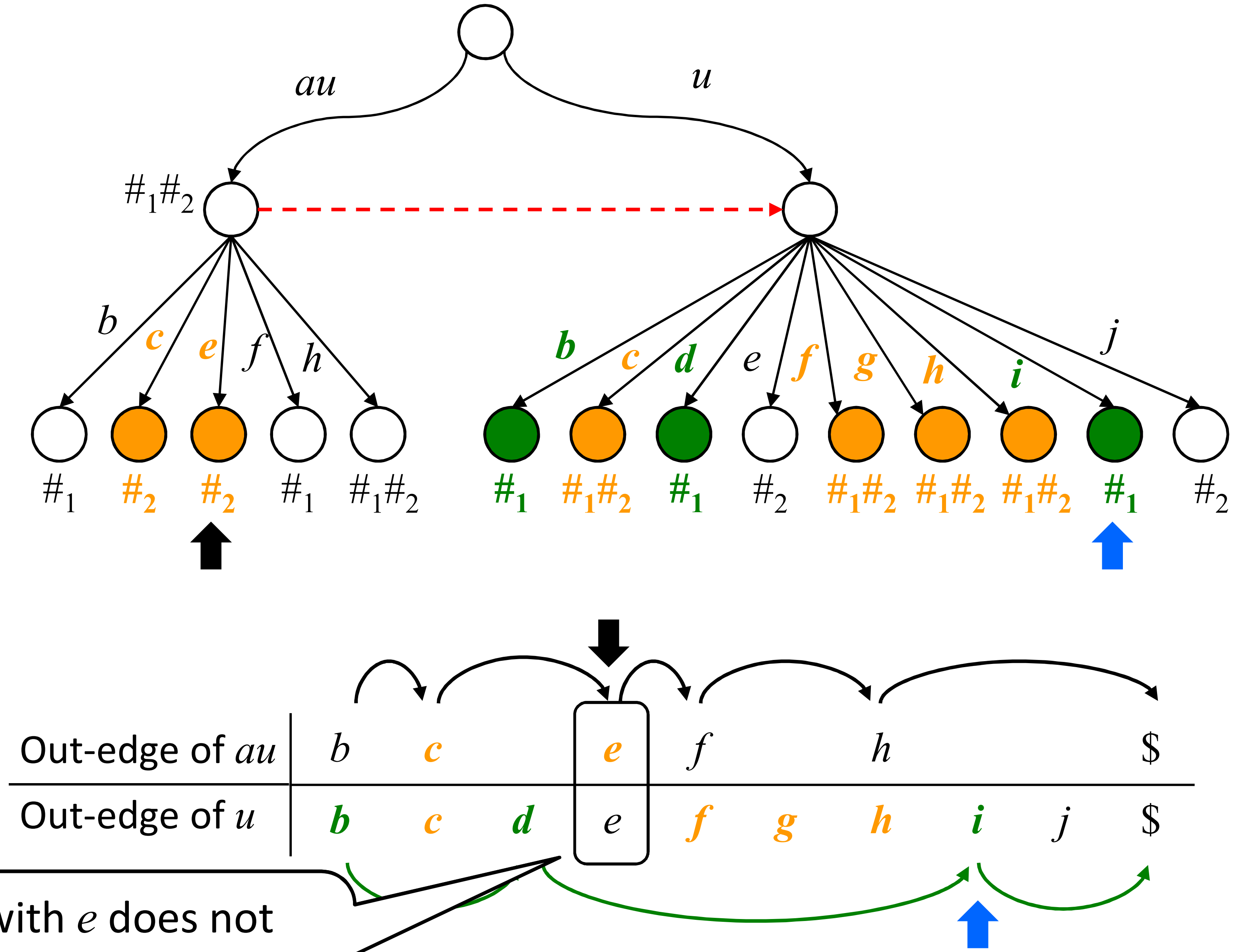


Out-edge of au	b	c		e	f		h			\$
Out-edge of u	b	c	d	e	f	g	h	i	j	\$

Shift the pointer for node u by following the link.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

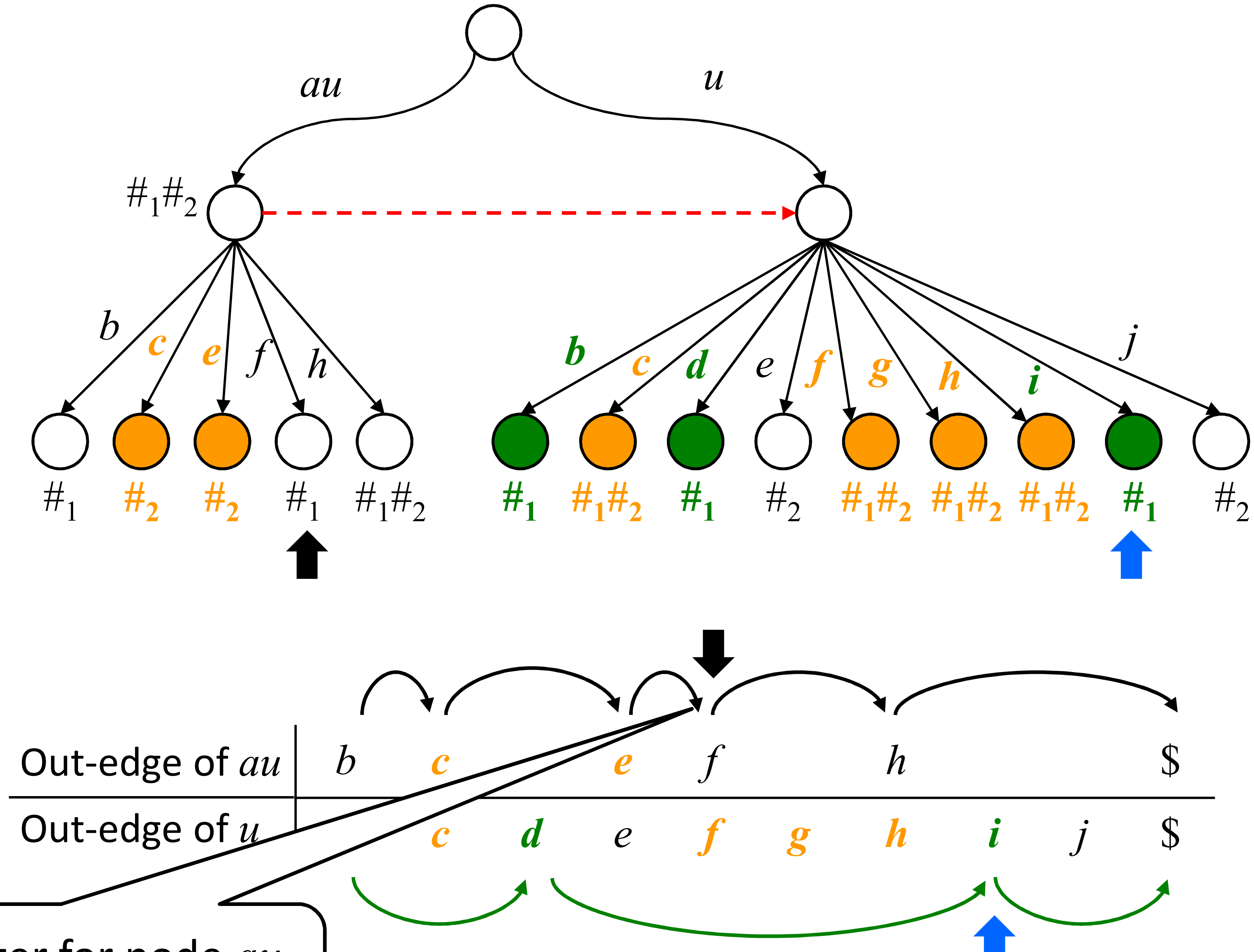
		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11



The out-edge of u with e does not meet the condition for orange case
→ aue is not a MAW to output.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

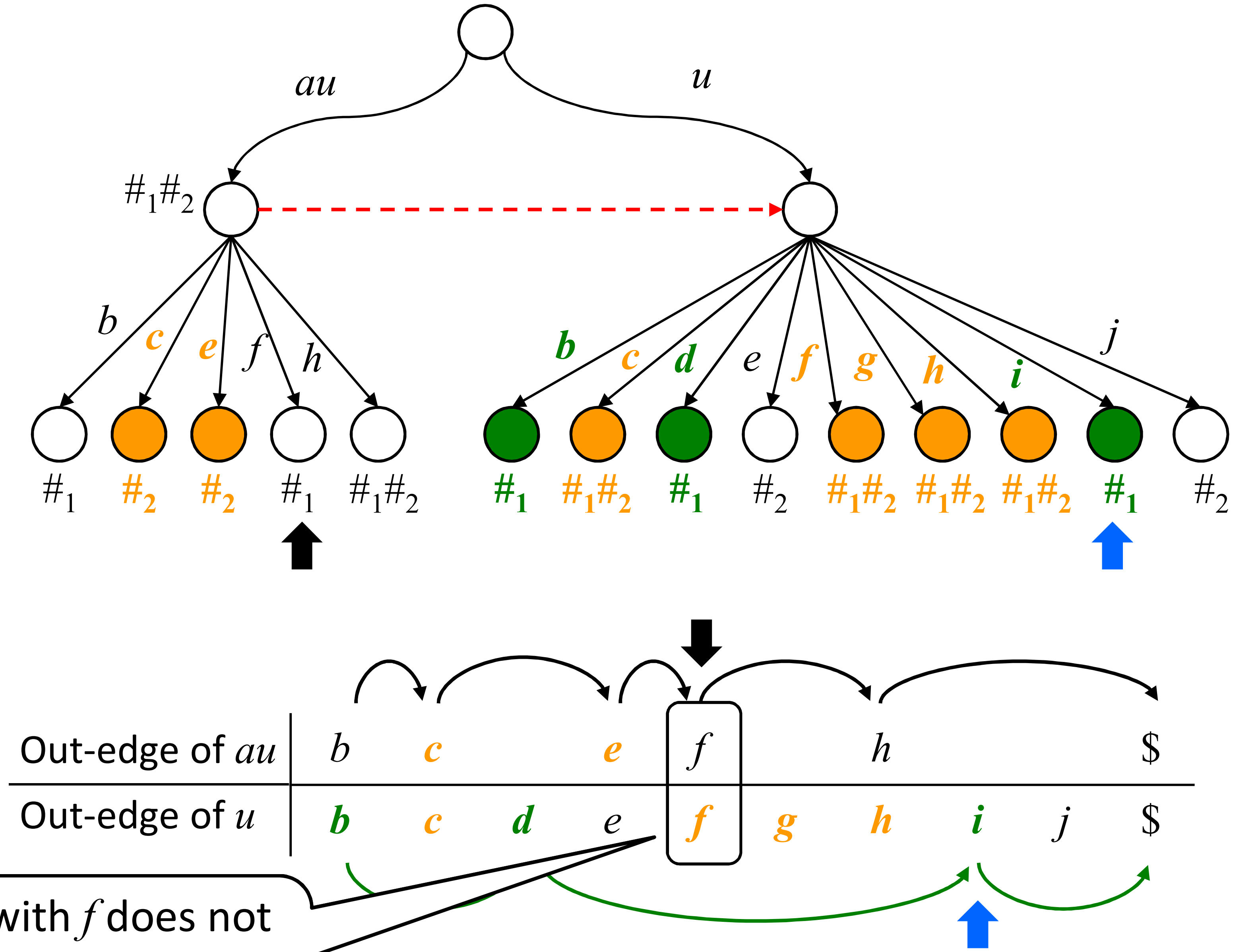
		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11



Shift the pointer for node au by following the link.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

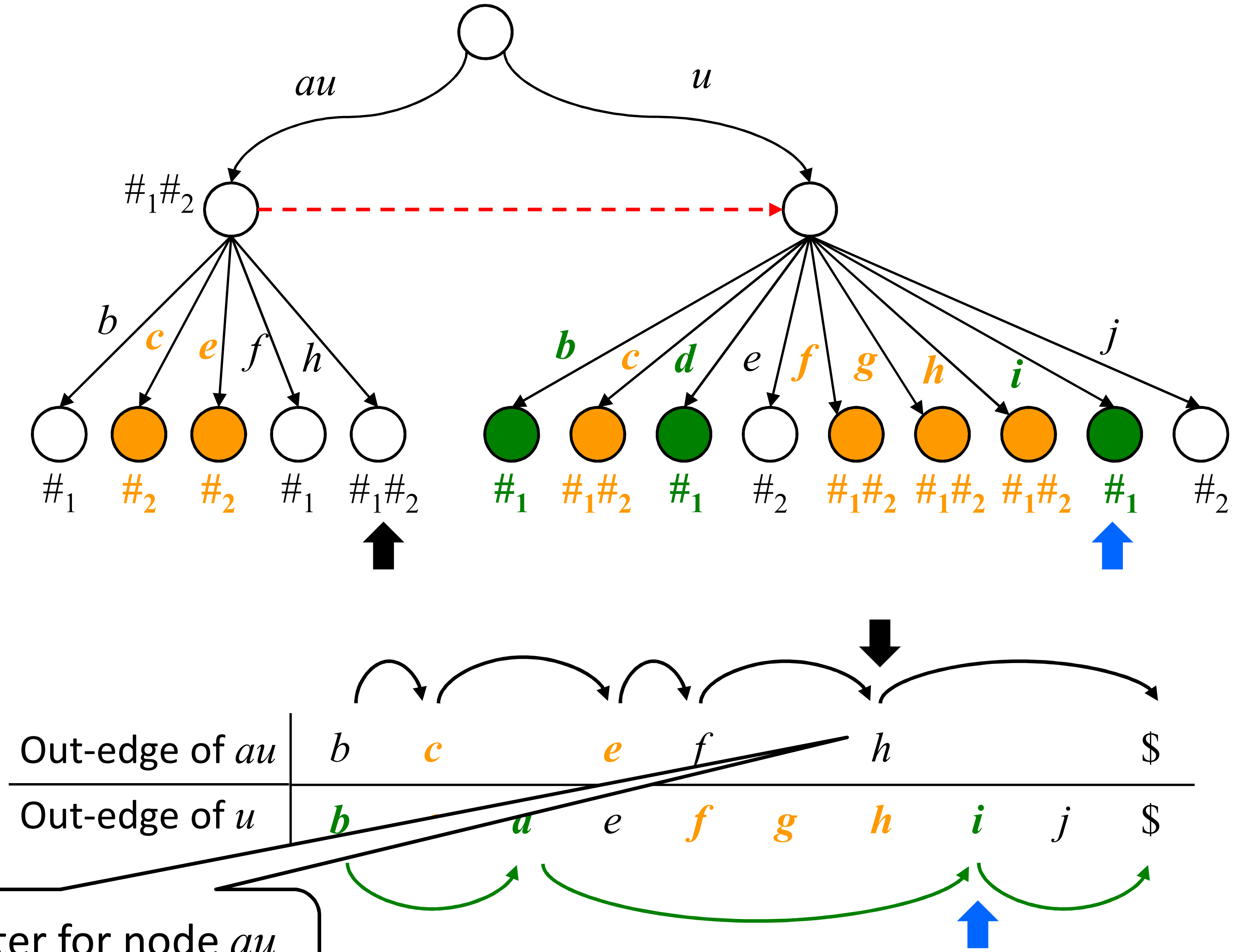
		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11



The out-edge of au with f does not meet the condition for orange case
→ auf is not a MAW to output.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

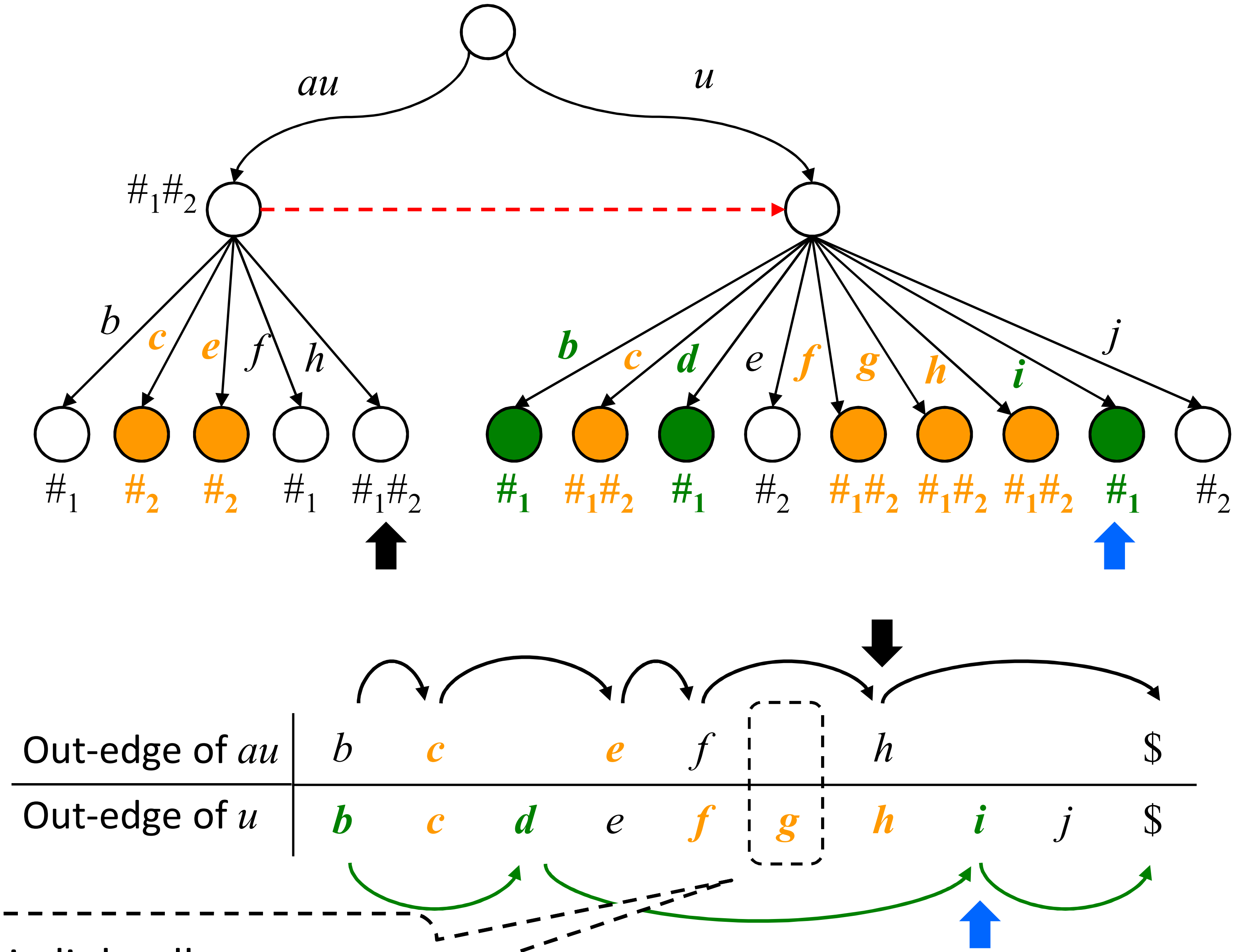
		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11



Shift the pointer for node au by following the link.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

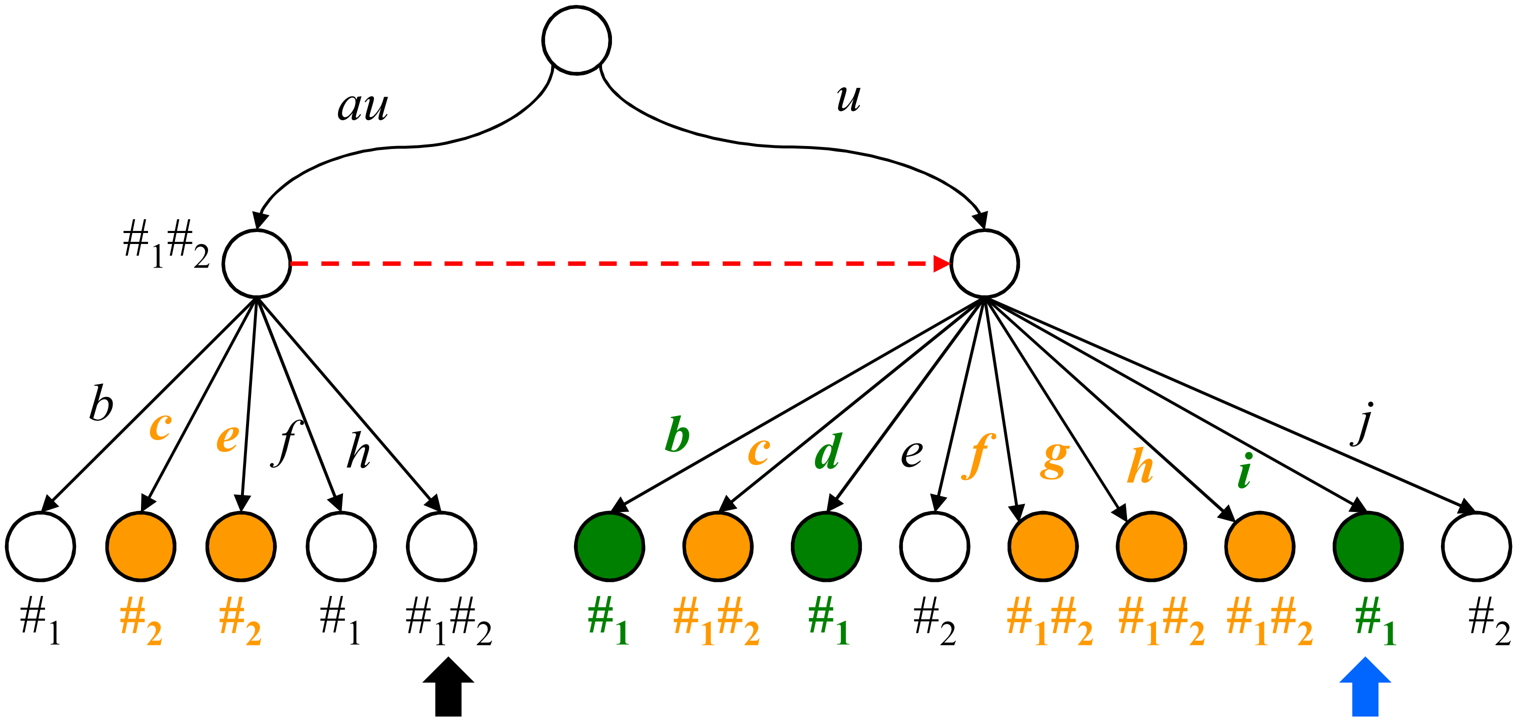
		au
		$\#_1\#_2$
aub	ub	**B**
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11



Our skip links allow us not to consider this label g.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11

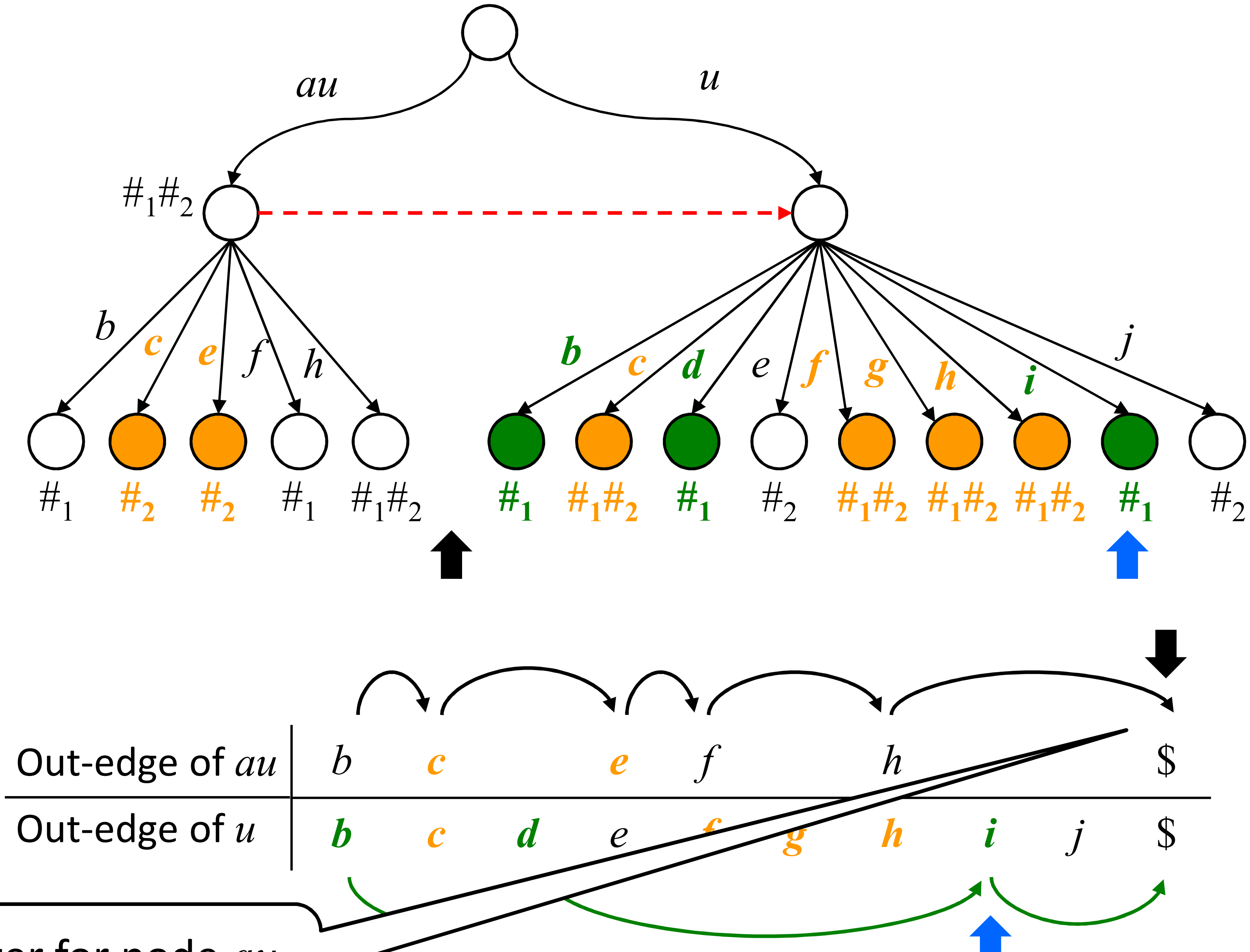


Out-edge of au	b	c		e	f		h			$\$$
Out-edge of u	b	c	d	e	f	g	h	i	j	$\$$

The out-edge of au with h does not meet the condition for orange case → auh is not a MAW to output.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

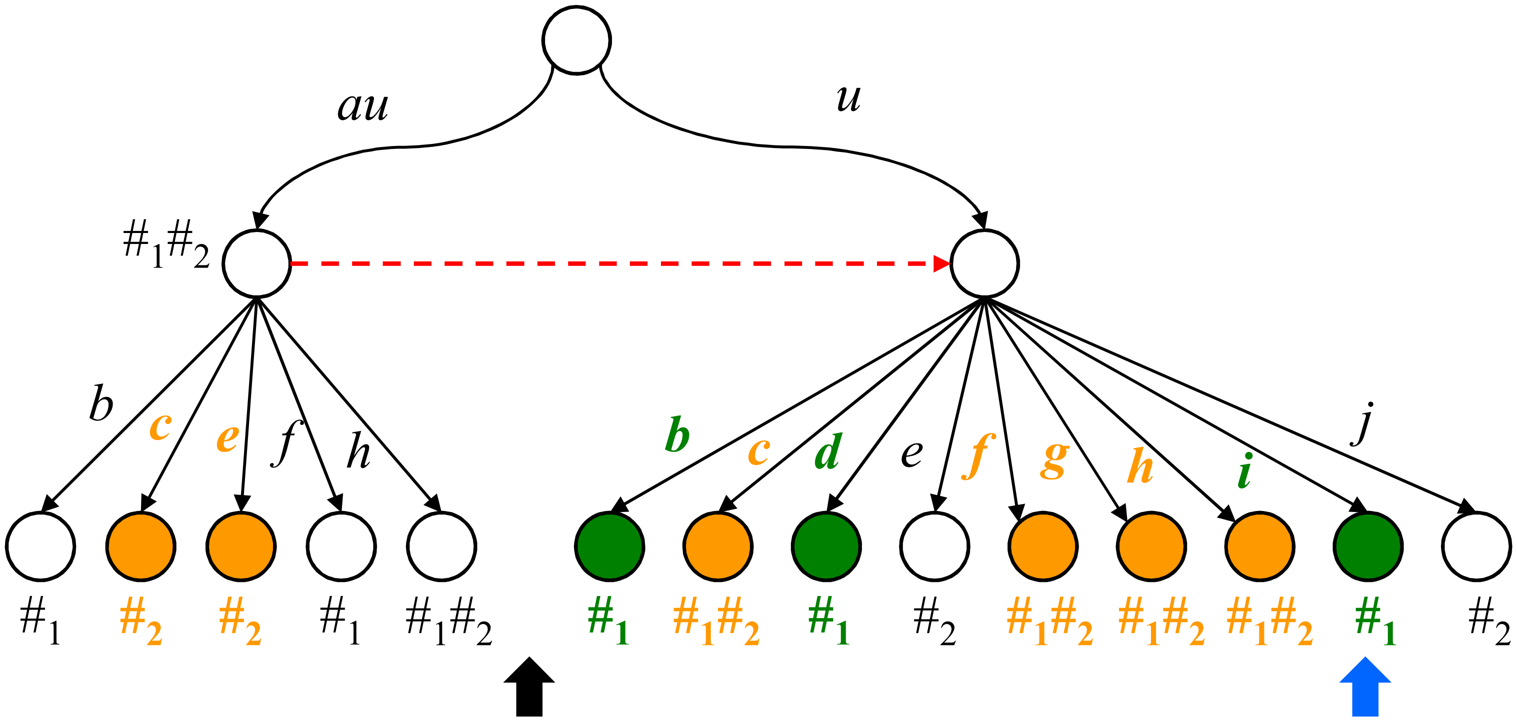
		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11



Shift the pointer for node au by following the link.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11

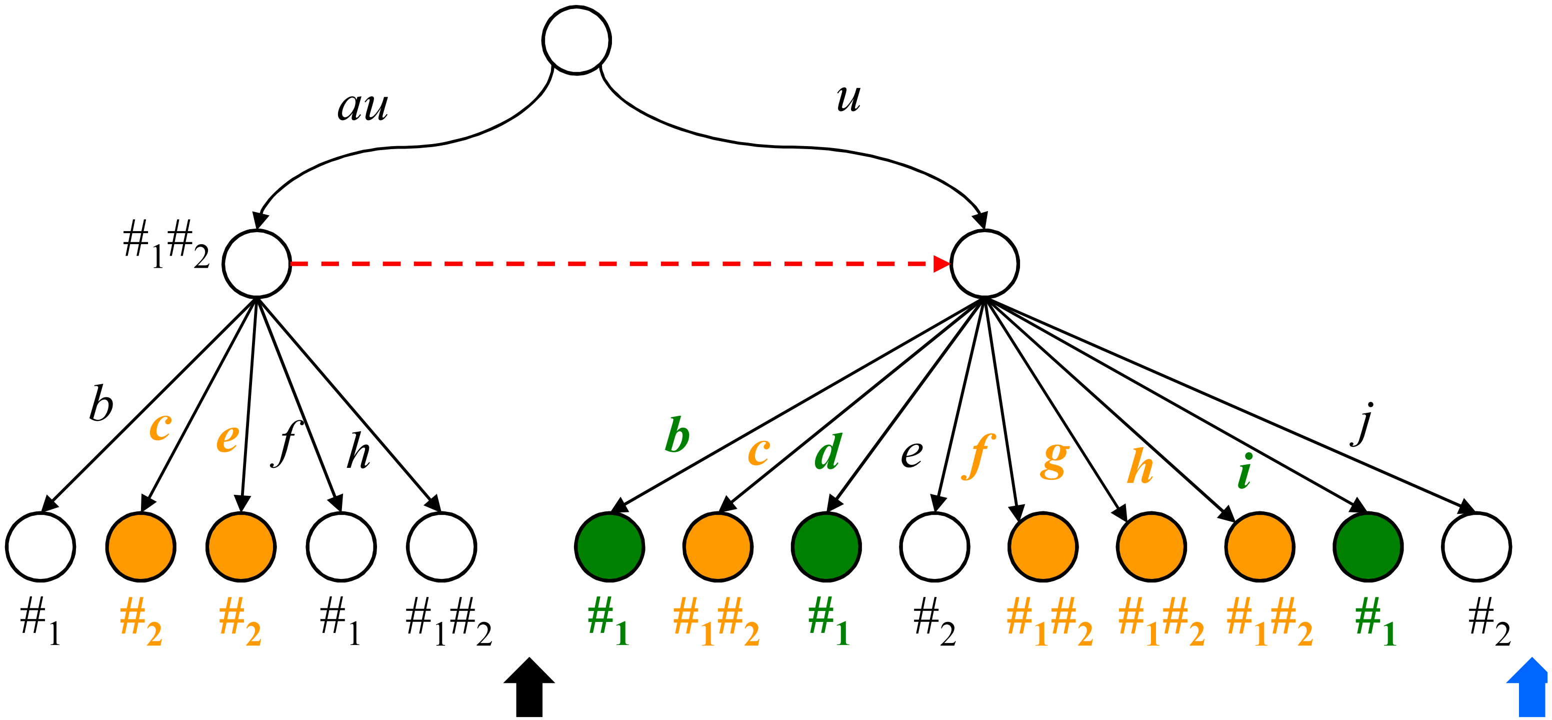


Out-edge of au	b	c		e	f		h			\$
Out-edge of u	b	c	d	e	f	g	h	i	j	\$

aui is absent and u has out-edge with i with condition for green case
→ aui is a MAW to output.

Algorithm for $\mathbf{B} = 10$ and au is labeled $\#_1\#_2$

		au
		$\#_1\#_2$
aub	ub	$\mathbf{B}$
$\#_1\#_2$	$\#_1\#_2$	00
$\#_1$	$\#_1$	00
	$\#_1\#_2$	01
$\#_2$	$\#_2$	00
	$\#_1\#_2$	10
absent	$\#_1$	10
	$\#_2$	01
	$\#_1\#_2$	11



Out-edge of au	b	c		e	f		h			\$
Out-edge of u	b	c	d	e	f	g	h	i	j	\$

Shift the pointer for node u by following the link.

Time Analysis

Out-edge of au	b	c		e	f		h			\$
Out-edge of u	b	c	d	e	f	g	h	i	j	\$

Charged to the out-edges of node au → $O(n)$ in total

Charged to output MAWs → $O(|\mathsf{MAW}(01)|)$ in total

Charged to output MAWs → $O(|\mathsf{MAW}(01)|)$ in total

Skipped comparisons → Free

Theorem 1

For $k = 2$, we can solve Problem 1 in optimal $O(n + |\mathsf{MAW}(\mathbf{B})|)$ time with $O(n)$ working space.

Final Remarks

- Beal et al. (2003) considered a different version of MAWs for a set $\mathbf{S} = \{S_1, \ldots, S_k\}$ of k strings, where aub is a MAW for $\mathbf{S}$ iff aub does not occur in $\mathbf{S}$, and both au and ub occur in $\mathbf{S}$. They presented an $\mathrm{O}(\sigma n)$-time algorithm.

- This version of MAWs can be computed in $\mathrm{O}(n + |\mathrm{output}|)$ time independently of k, by running our algorithm without skip links.

- Beal & Crochemore (2023) considered $\mathbf{T}$-specific strings w.r.t. $\mathbf{S}$, for string sets $\mathbf{T}$ and $\mathbf{S}$: a string w is a $\mathbf{T}$-specific string w.r.t. $\mathbf{S}$ iff w is a substring of $\mathbf{T}$ and w is a MAW for $\mathbf{S}$. They presented an $\mathrm{O}(\sigma n)$-time algorithm.

- The $\mathbf{T}$-specific strings w.r.t. $\mathbf{S}$ can be computed in $\mathrm{O}(n + |\mathrm{output}|)$ time by slightly modifying our algorithm for $k = 2$.