

StringBeginners @ AIP, 26 April 2019

# 索引構造オンライン構築 四方山話

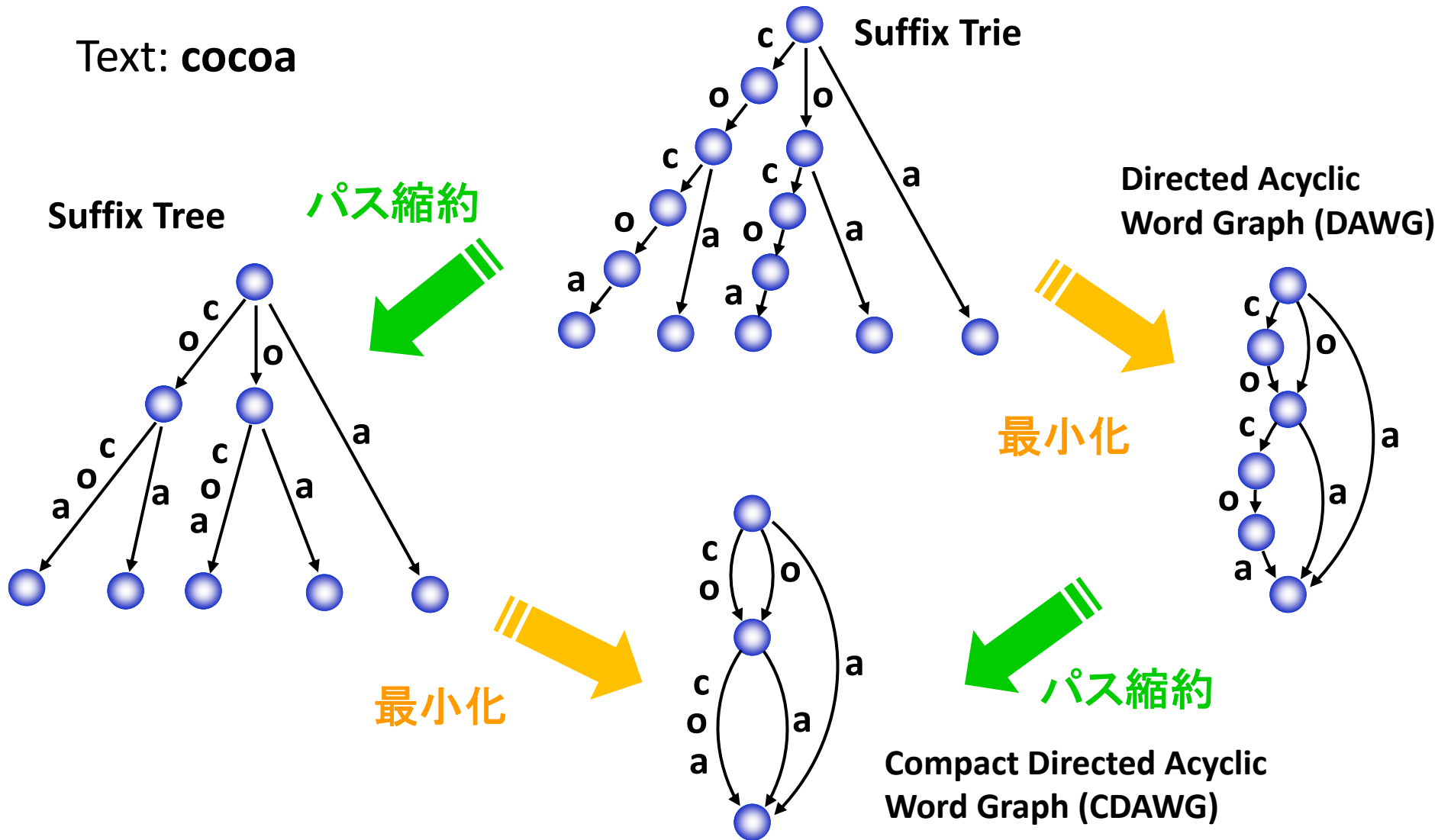
---

稲永 俊介



# 基礎的なテキスト索引

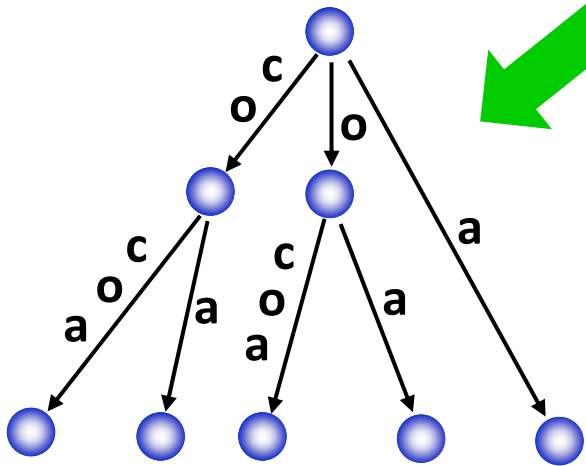
Text: cocoa



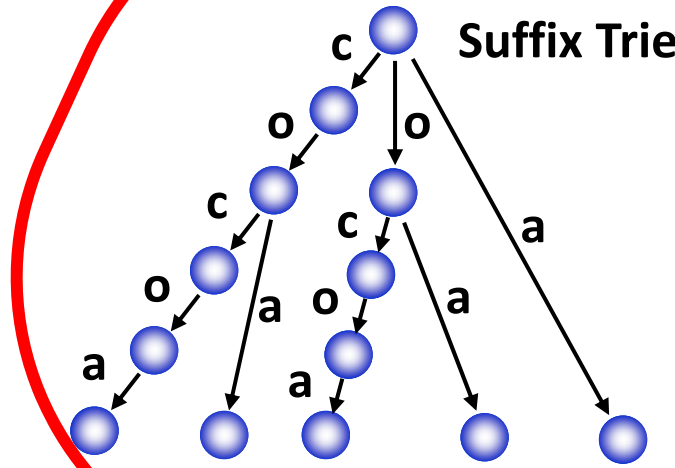
# 基礎的なテキスト索引

Text: cocoa

Suffix Tree



パス縮約

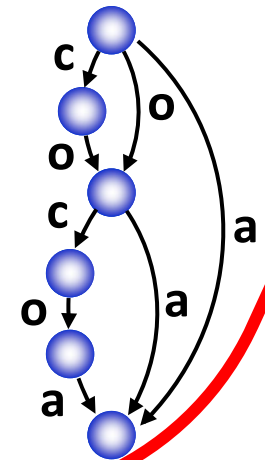


Suffix Trie

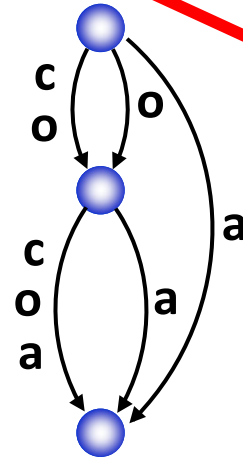
Directed Acyclic Word Graph (DAWG)



最小化

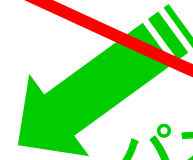


最小化

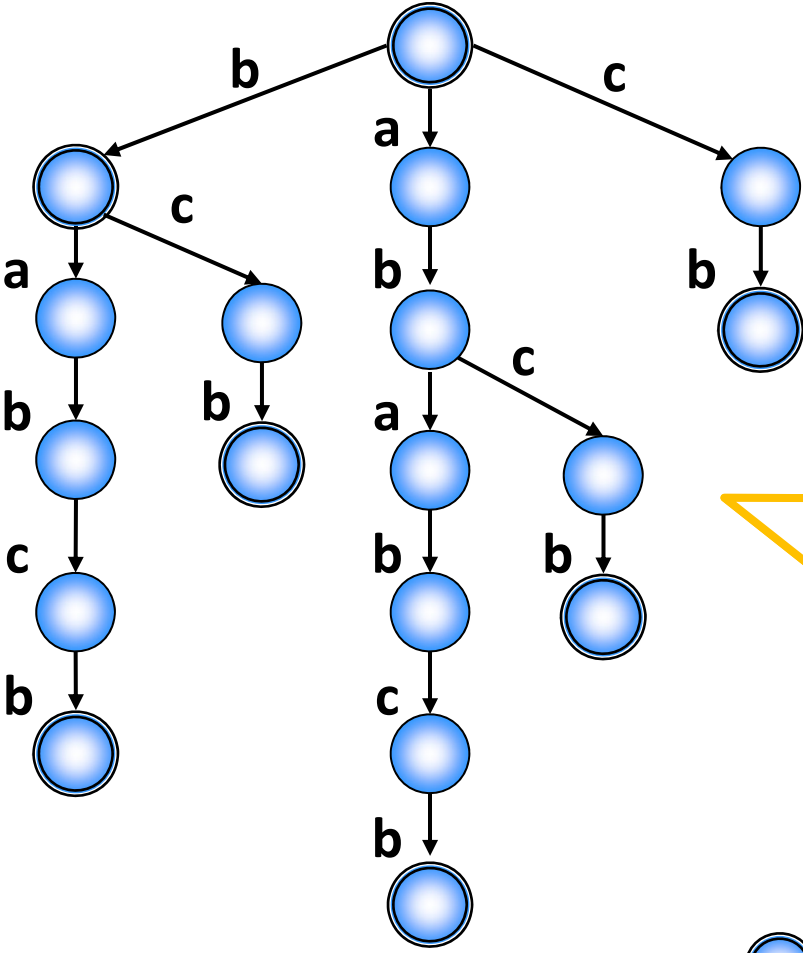


Compact Directed Acyclic Word Graph (CDAWG)

パス縮約




# From Suffix Trie to DAWG



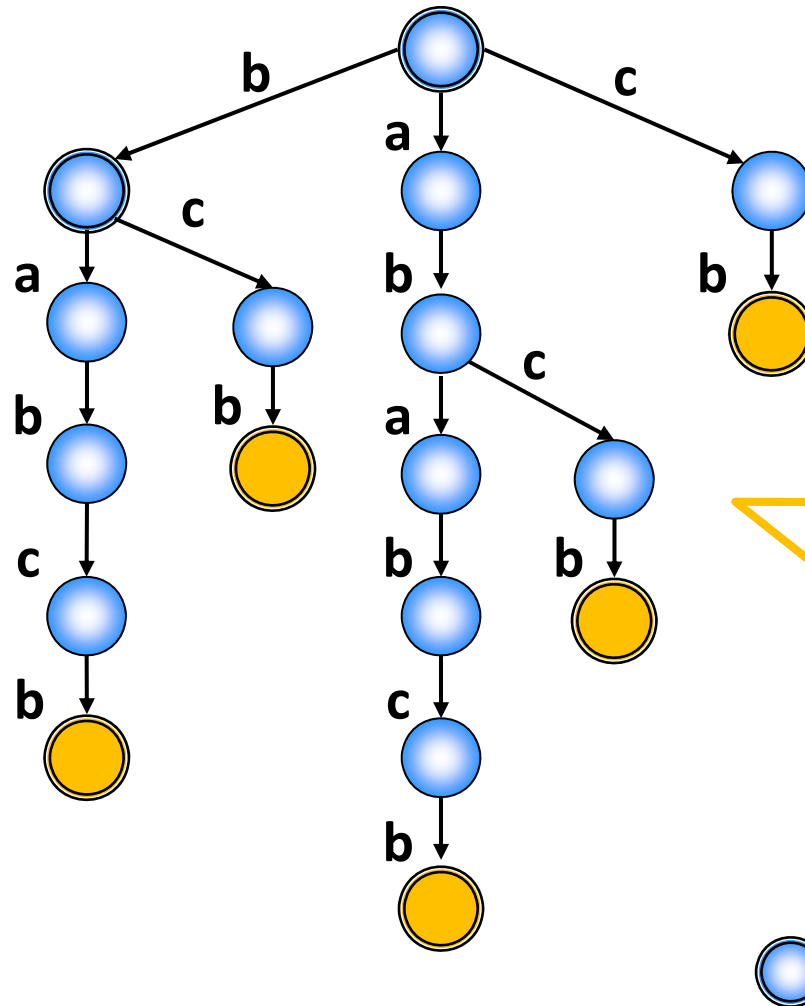
ababcb

同型な木(グラフ)を  
ボトムアップに  
マージしていく

 接尾辞を受理するノード

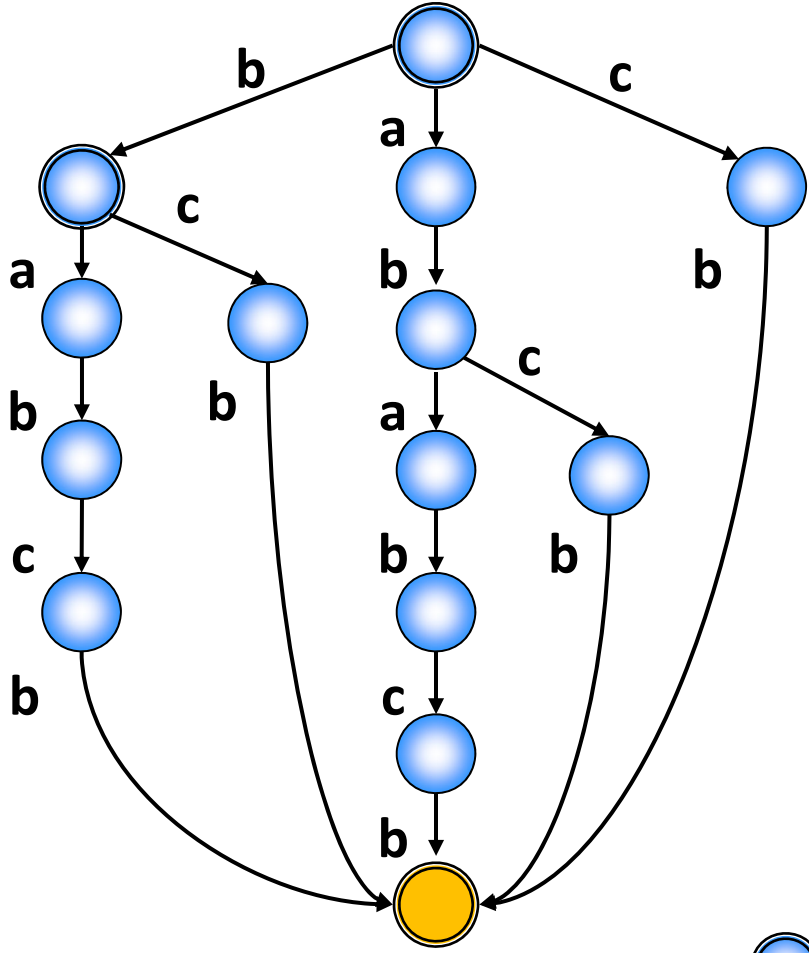


# From Suffix Trie to DAWG




同型な木(グラフ)を  
ボトムアップに  
マージしていく

# From Suffix Trie to DAWG

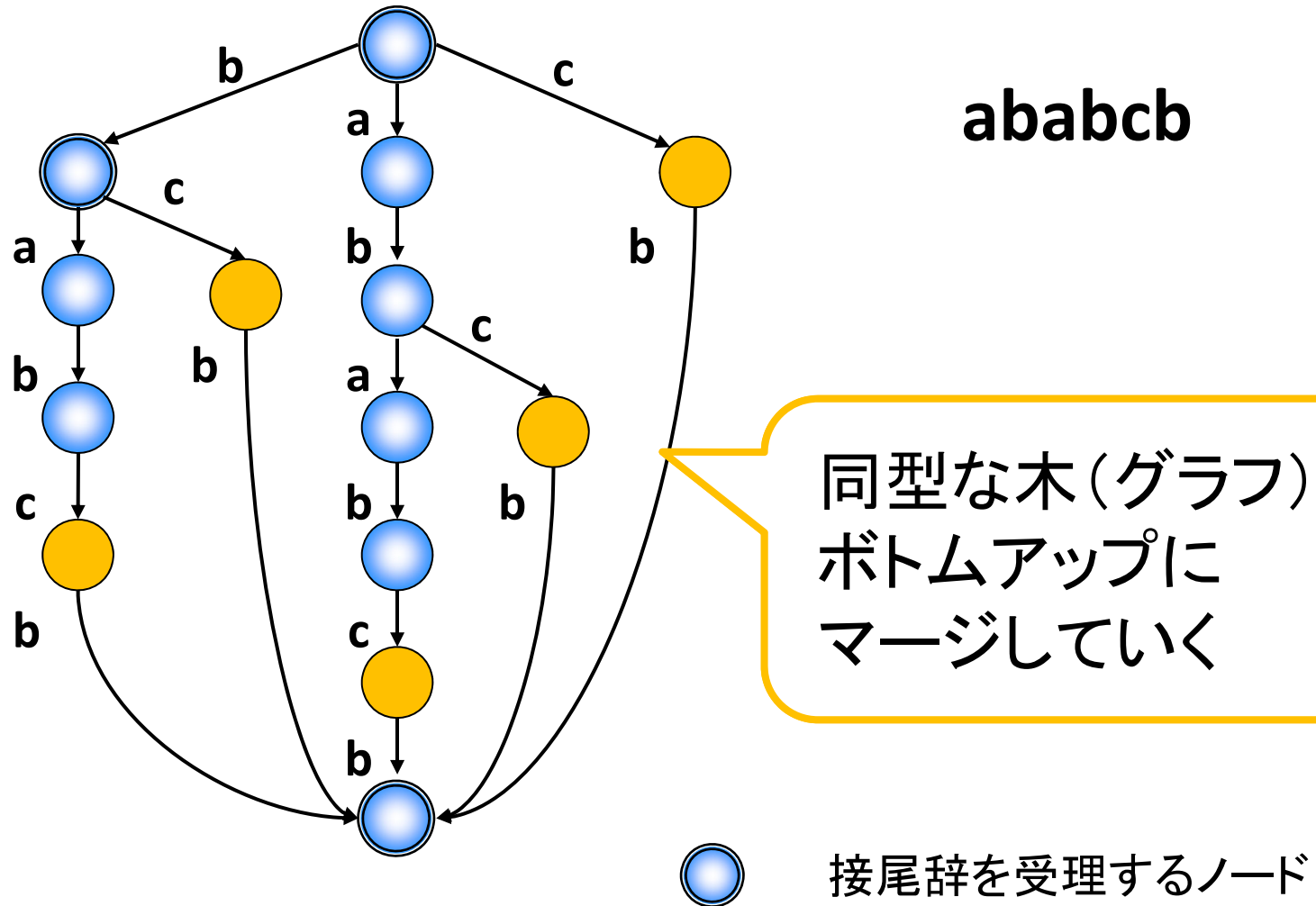


ababcb

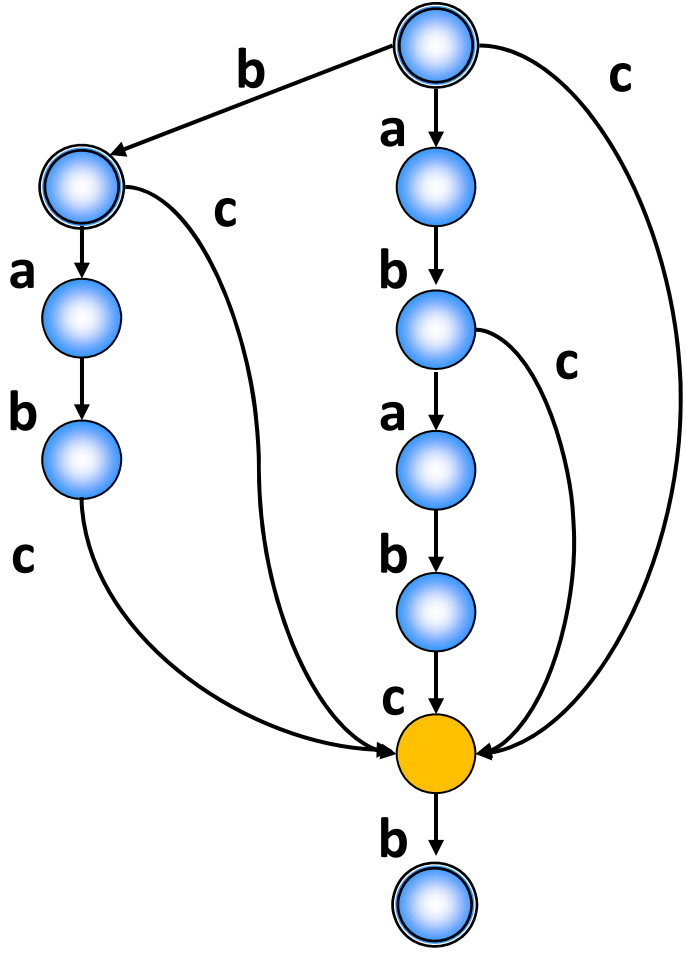
 接尾辞を受理するノード




# From Suffix Trie to DAWG

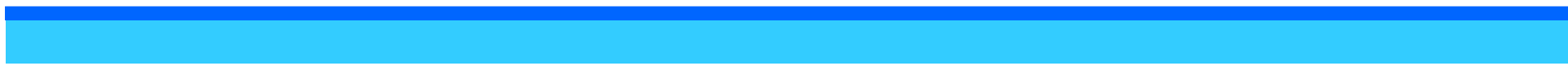


# From Suffix Trie to DAWG



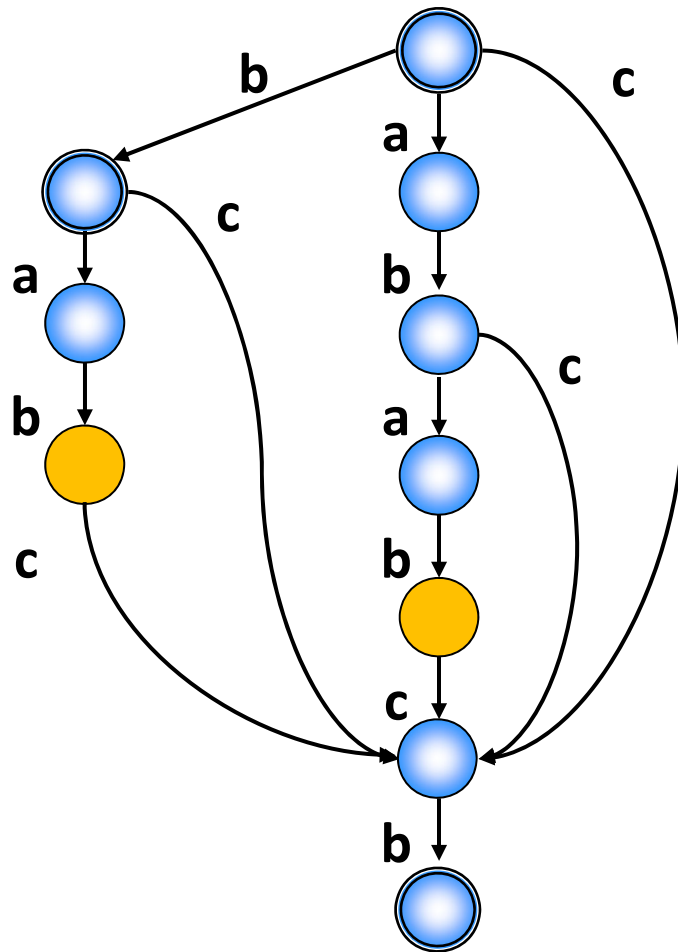
ababcb

 接尾辞を受理するノード





# From Suffix Trie to DAWG



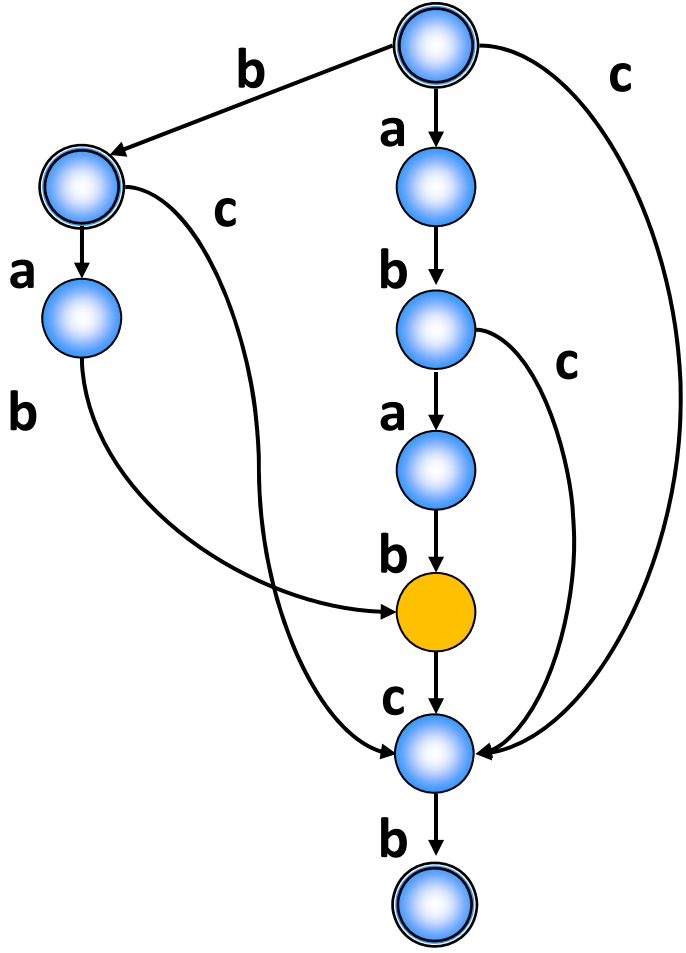
ababcb

同型な木(グラフ)を  
ボトムアップに  
マージしていく




接尾辞を受理するノード

# From Suffix Trie to DAWG

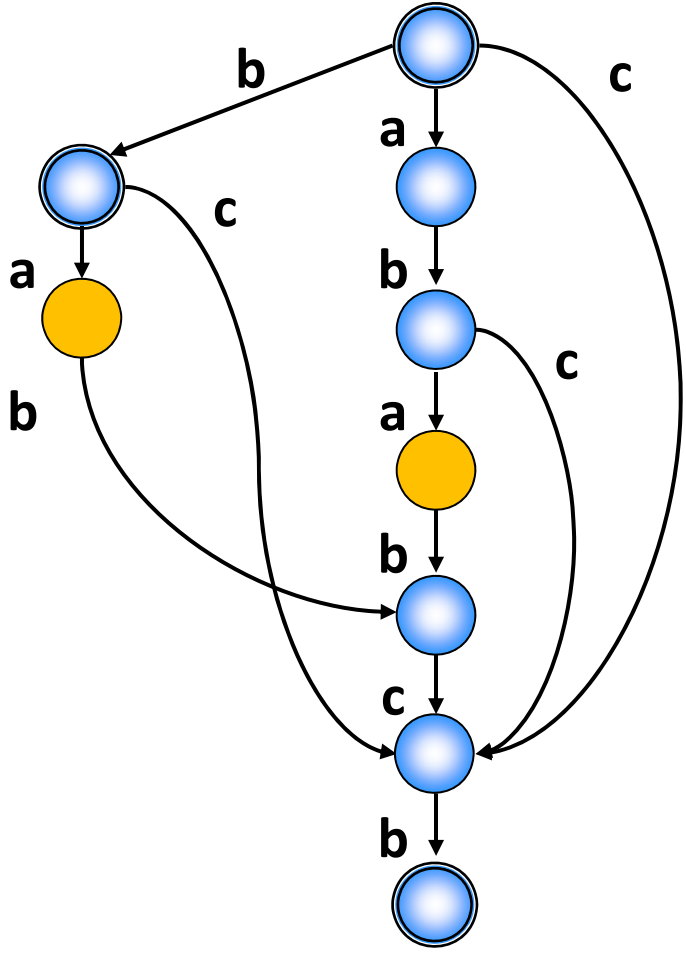


ababcb

 接尾辞を受理するノード




# From Suffix Trie to DAWG



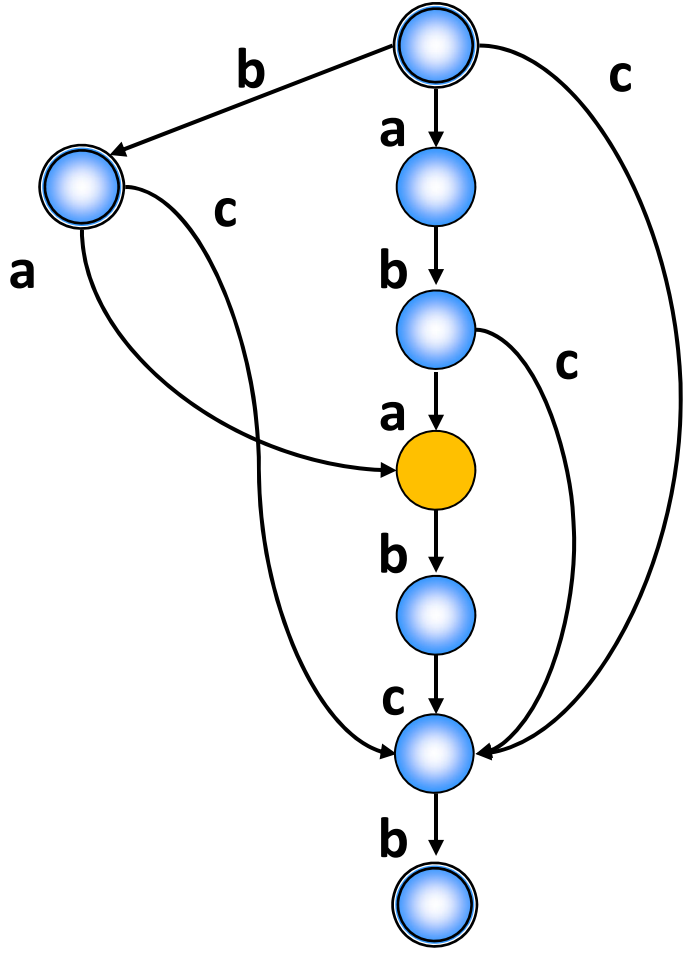
ababcb

同型な木(グラフ)を  
ボトムアップに  
マージしていく


 接尾辞を受理するノード



# From Suffix Trie to DAWG

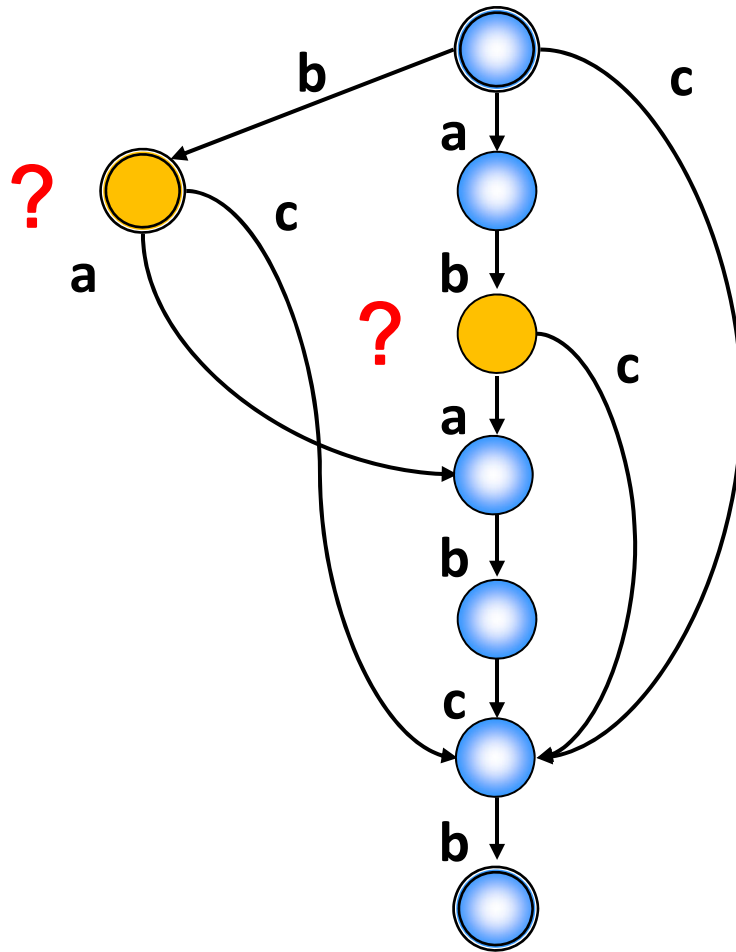


ababcb

 接尾辞を受理するノード



# From Suffix Trie to DAWG



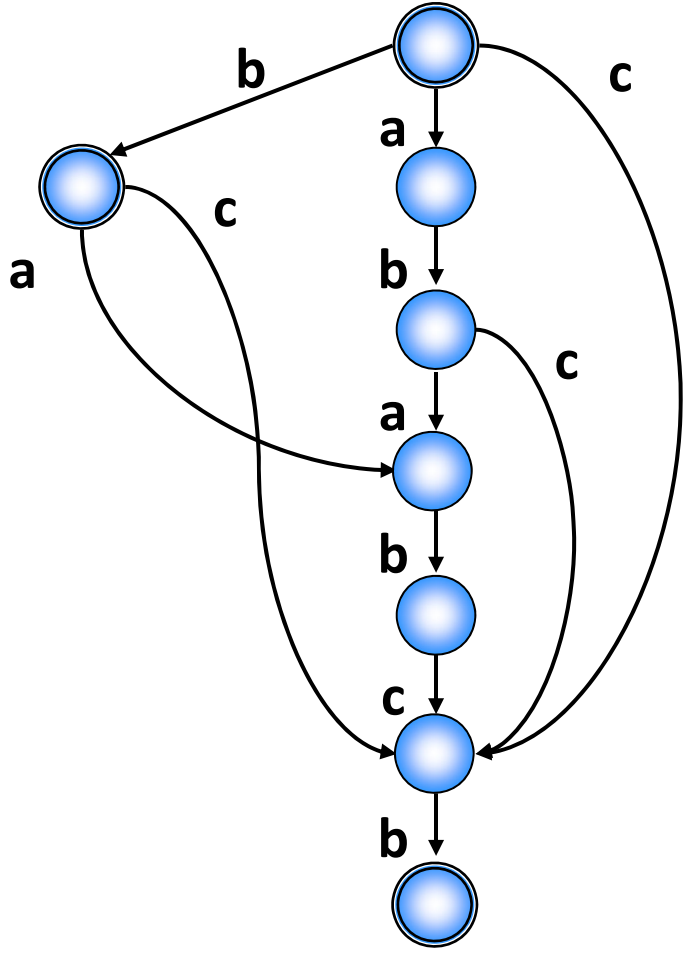
ababcb

同型に見えるが、  
一方は受理ノードで  
他方は違うので  
マージしない




接尾辞を受理するノード

# From Suffix Trie to DAWG



**ababcb**

DAWG of string  
**ababcb**

 接尾辞を受理するノード



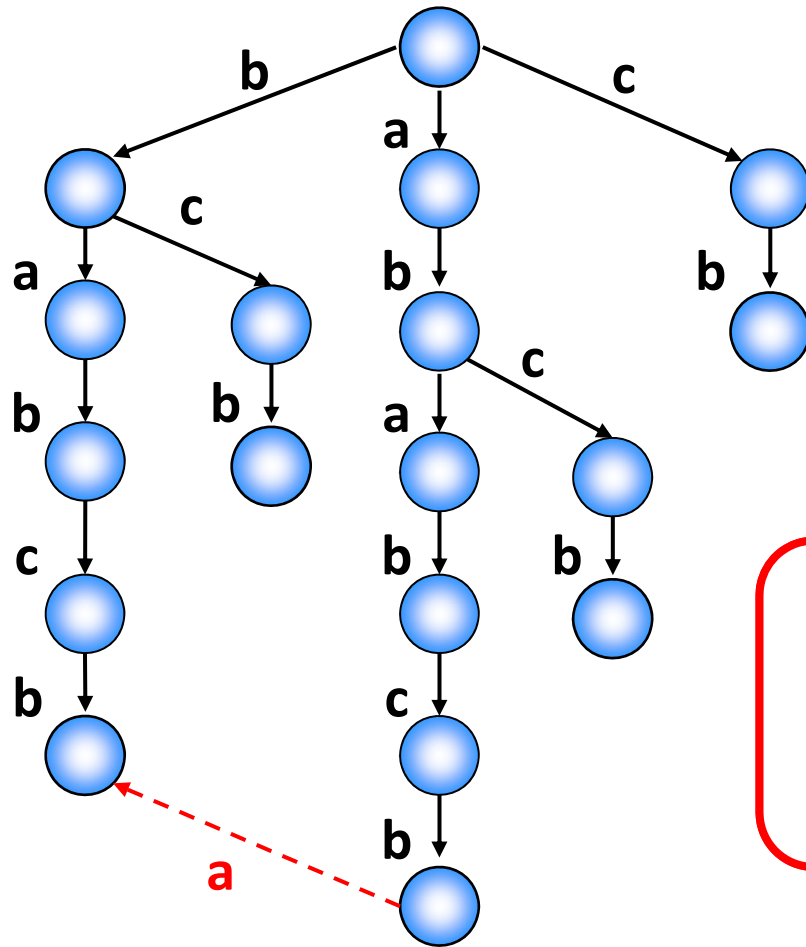
# DAWG の最小性

常識1 [A. Blumer et al. 1985]

文字列  $w$  の DAWG は  $w$  の接尾辞を受理する最小のオートマトンである.

- 明らかに suffix trie は  $w$  の接尾辞を受理する. よって DAWG も接尾辞を受理する.
- マージできるところはすべてマージしたので, DAWG が  $w$  の接尾辞を受理する最小のオートマトンである.

# Suffix Links of Suffix Trie

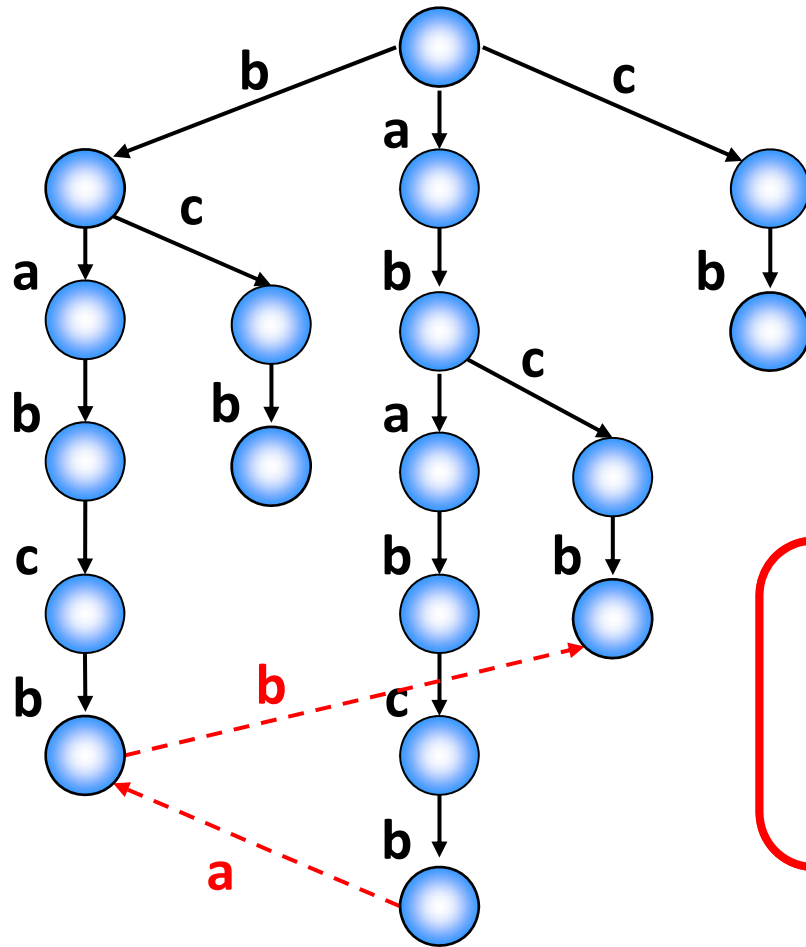


ノード  $ax$  の suffix link の  
文字ラベルは  $a$  で、  
その行き先は  $x$  である。

$$a \in \Sigma, x \in \Sigma^*$$



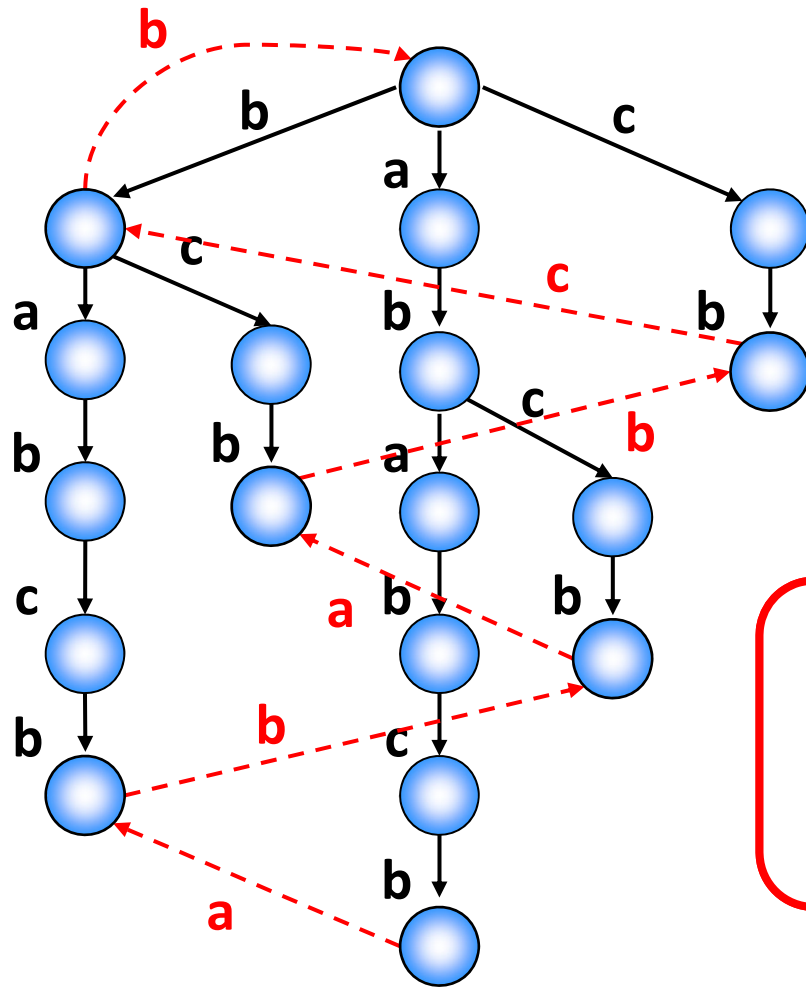
# Suffix Links of Suffix Trie



ノード  $ax$  の suffix link の  
文字ラベルは  $a$  で、  
その行き先は  $x$  である。

$$a \in \Sigma, x \in \Sigma^*$$

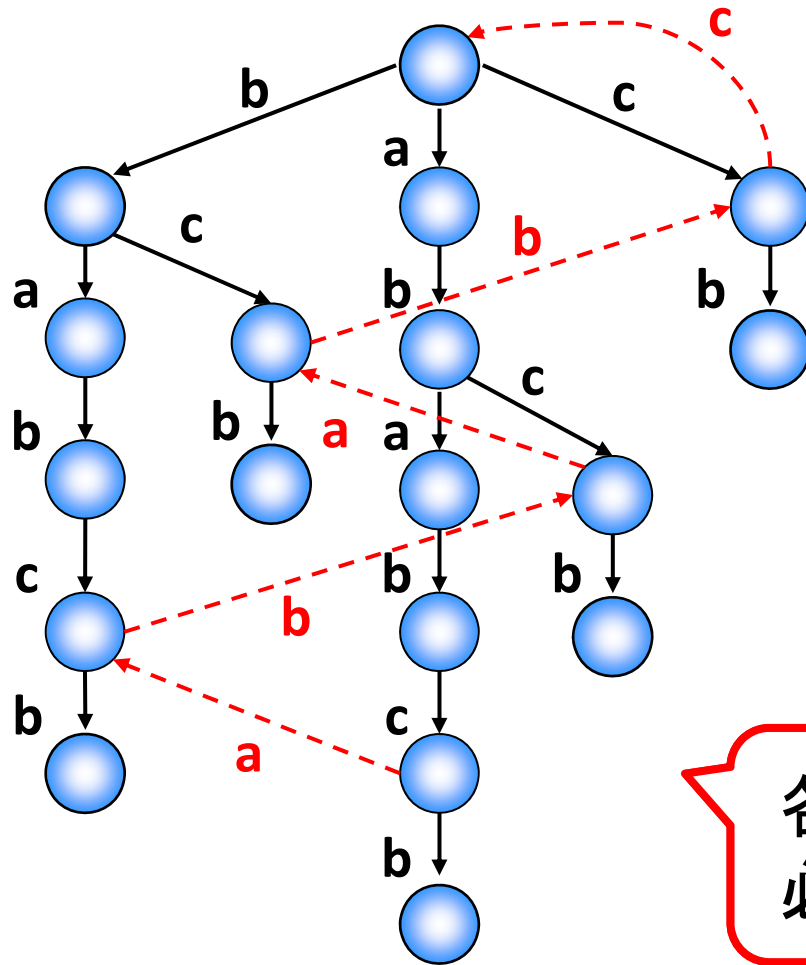
# Suffix Links of Suffix Trie



ノード  $ax$  の suffix link の  
文字ラベルは  $a$  で、  
その行き先は  $x$  である。

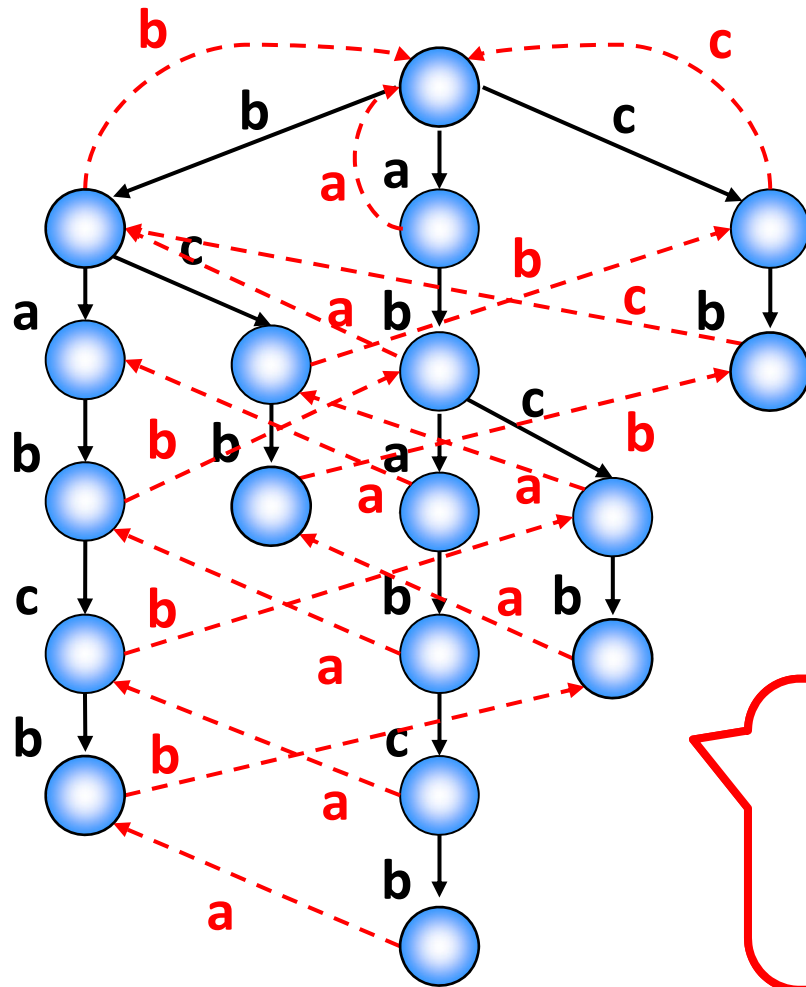
$$a \in \Sigma, x \in \Sigma^*$$

# Suffix Links of Suffix Trie



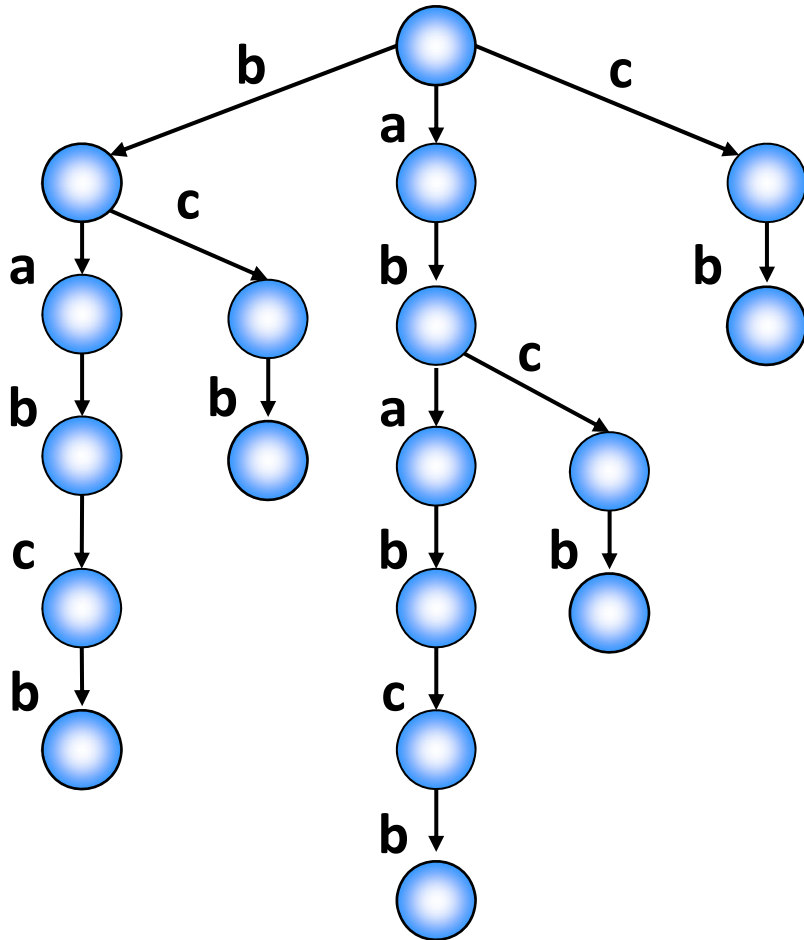
各 suffix link パスは必ず根に到達する.

# Suffix Link Tree of Suffix Trie

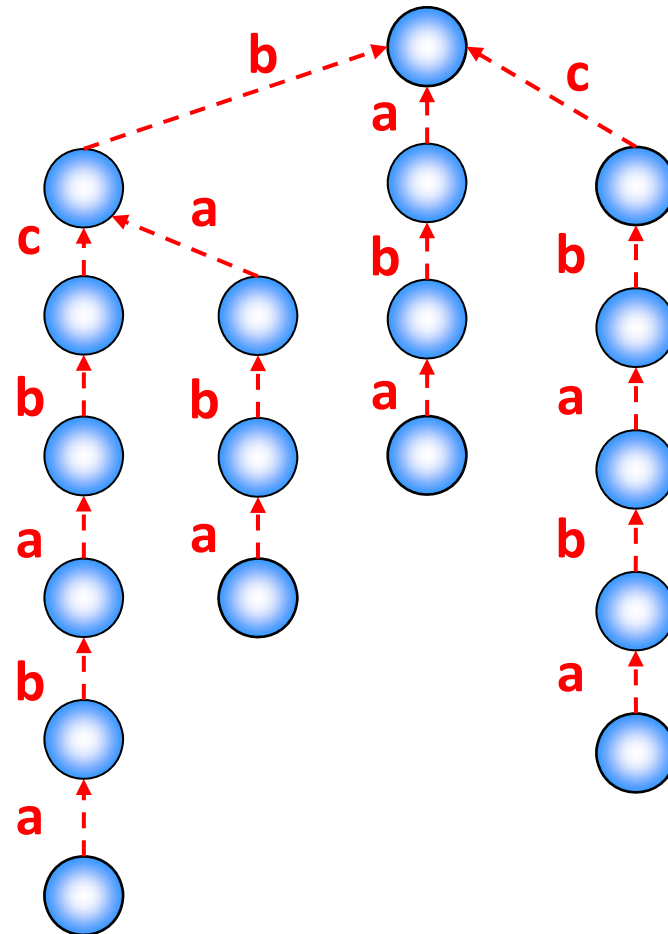


よって suffix link は  
(辺が逆向きの)木を成す  
(suffix link tree).

# Suffix Link Tree of Suffix Trie

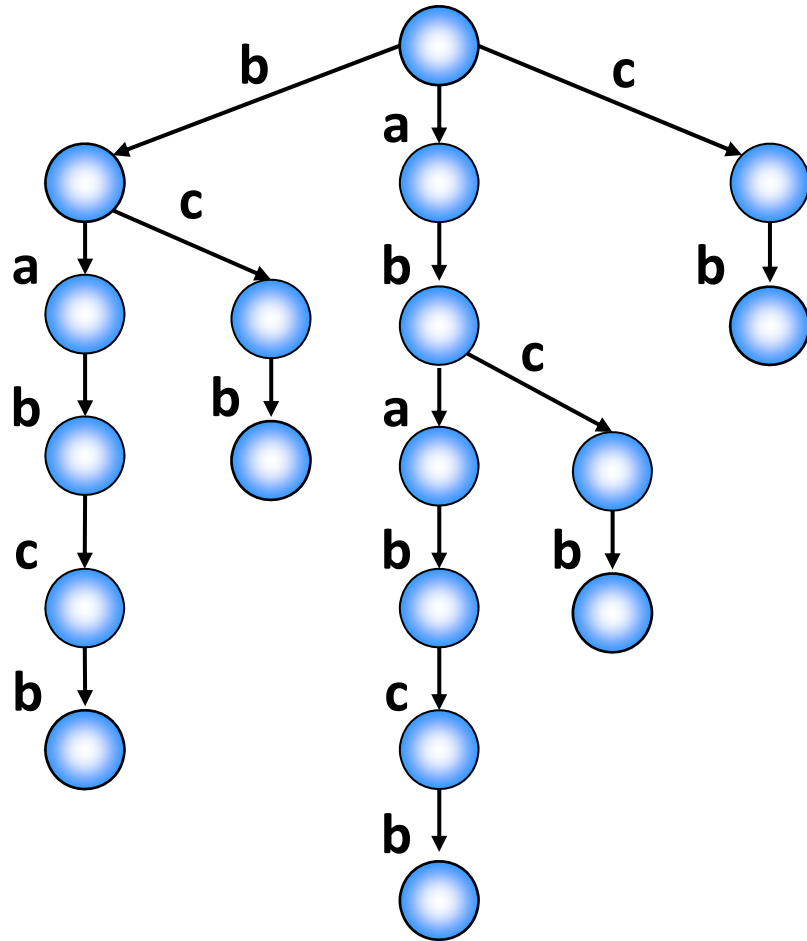


Suffix Trie of **ababcb**

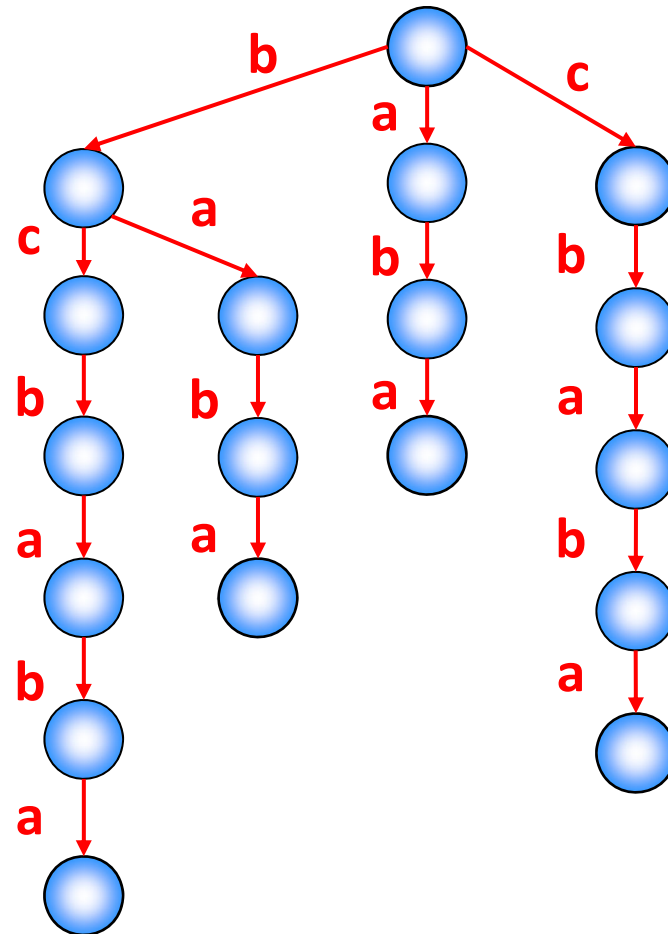


Suffix Link Tree of **ababcb**

# Suffix Link Tree = 反転文字列の Suffix Trie



Suffix Trie of **ababcb**



Suffix Trie of **bcbaba**



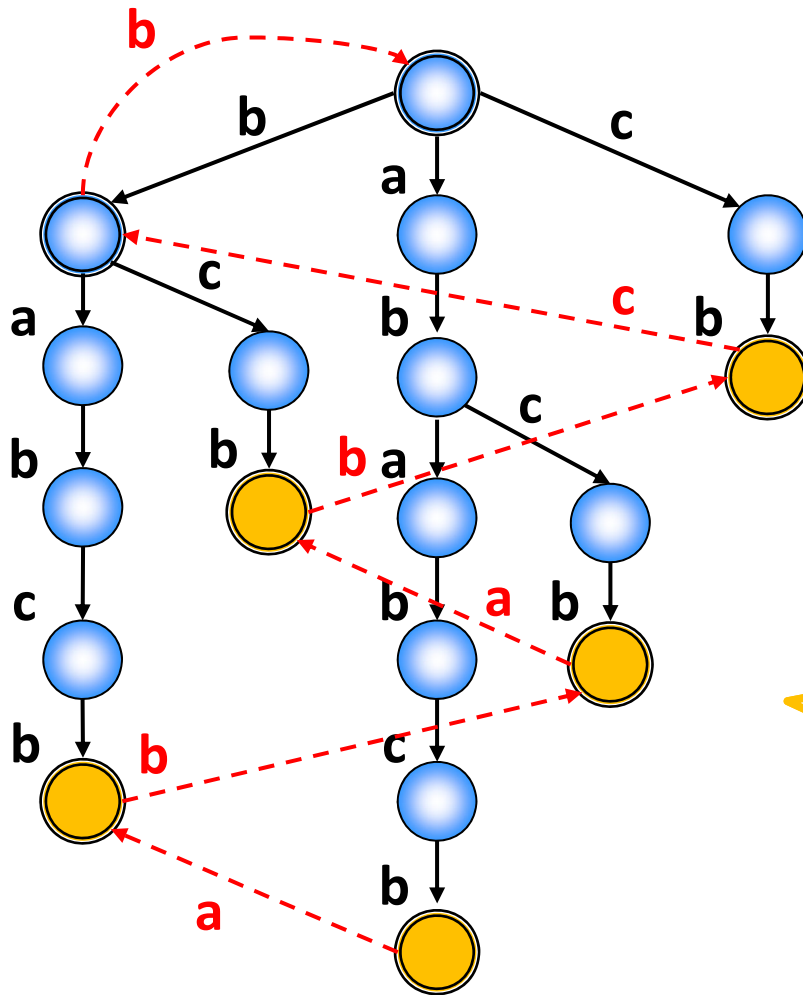
# Suffix Link Tree = 反転文字列の Suffix Trie

## 常識2

文字列  $w$  の suffix trie の suffix link tree は  
反転文字列  $w^R$  の suffix trie に等しい。

- 文字列  $x$  を表すノードの suffix link path は  
 $x$  の文字を逆順に読みながら根に到達するから。

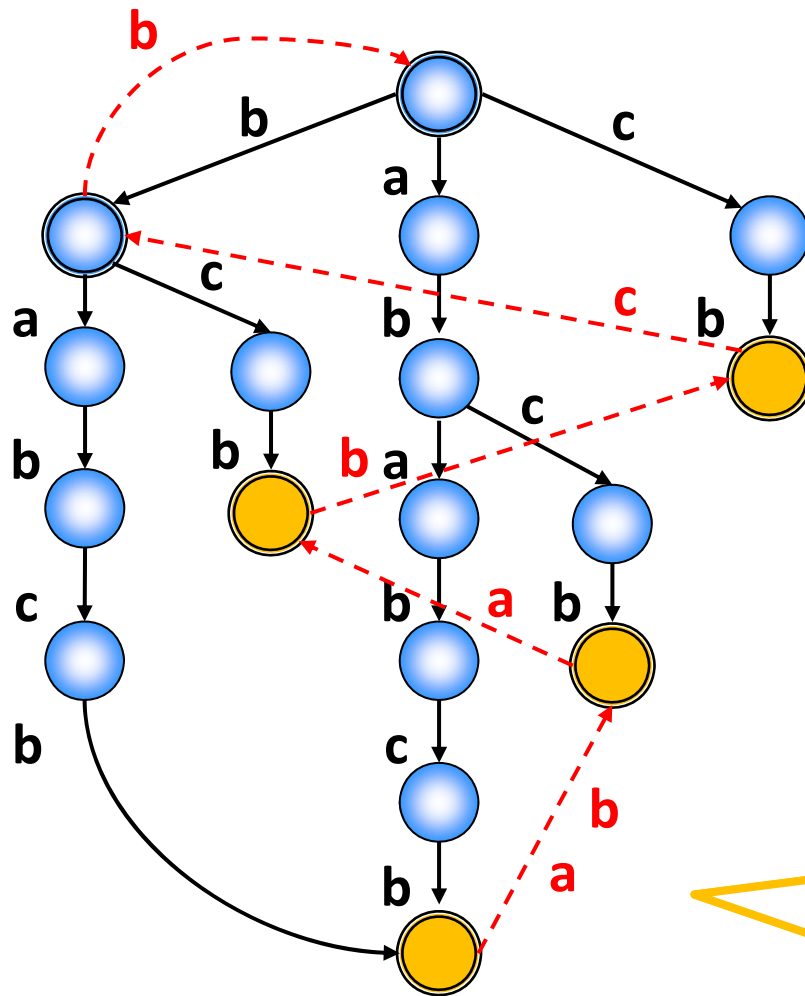
# ノードのマージと Suffix Link の関係



マージされるノードは  
suffix link パスで  
繋がっている

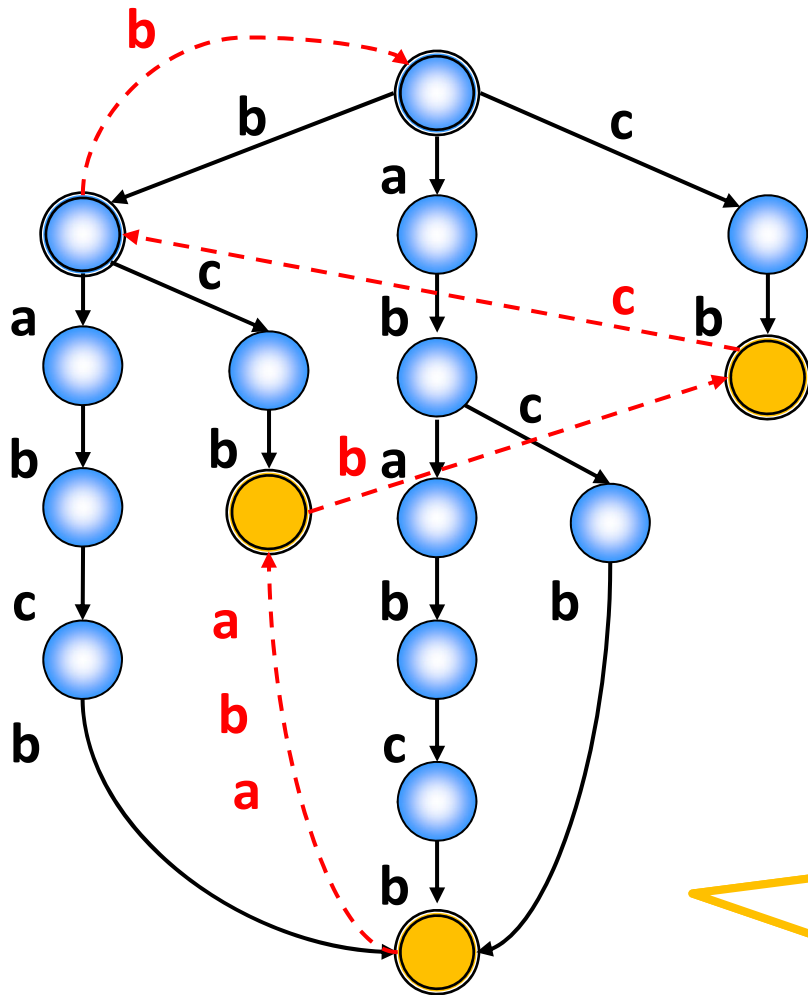


# ノードのマージと Suffix Link の関係



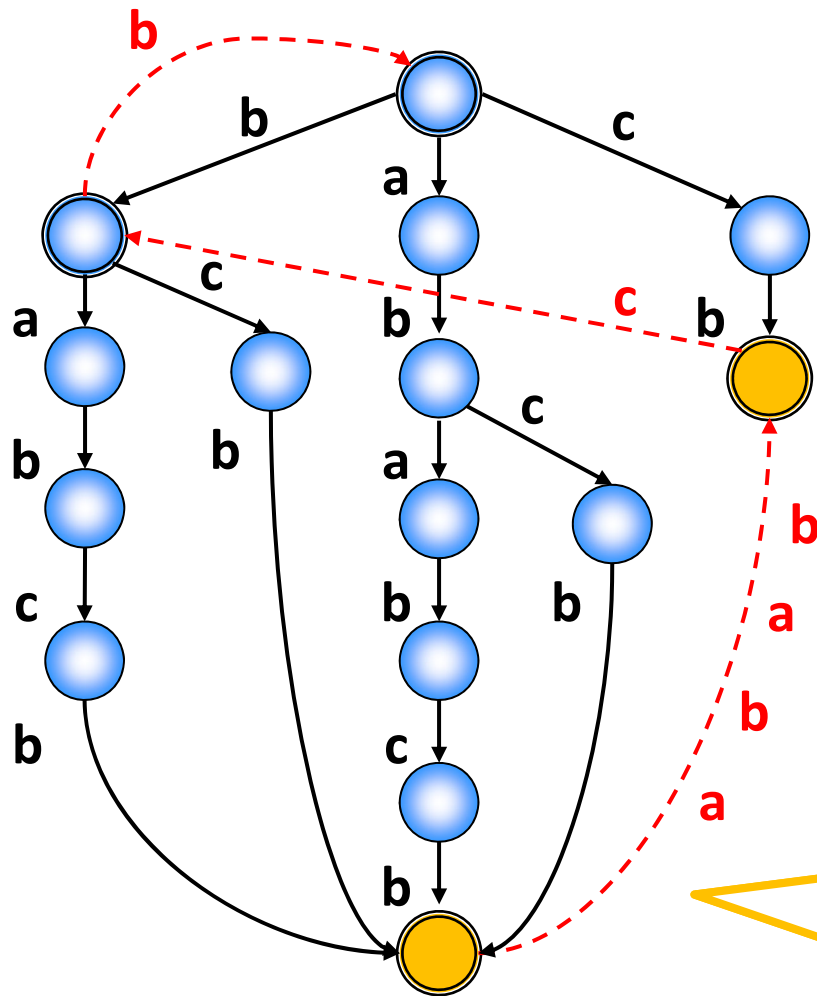
ノードをマージするとき、  
suffix link を同時に  
パス縮約する

# ノードのマージと Suffix Link の関係



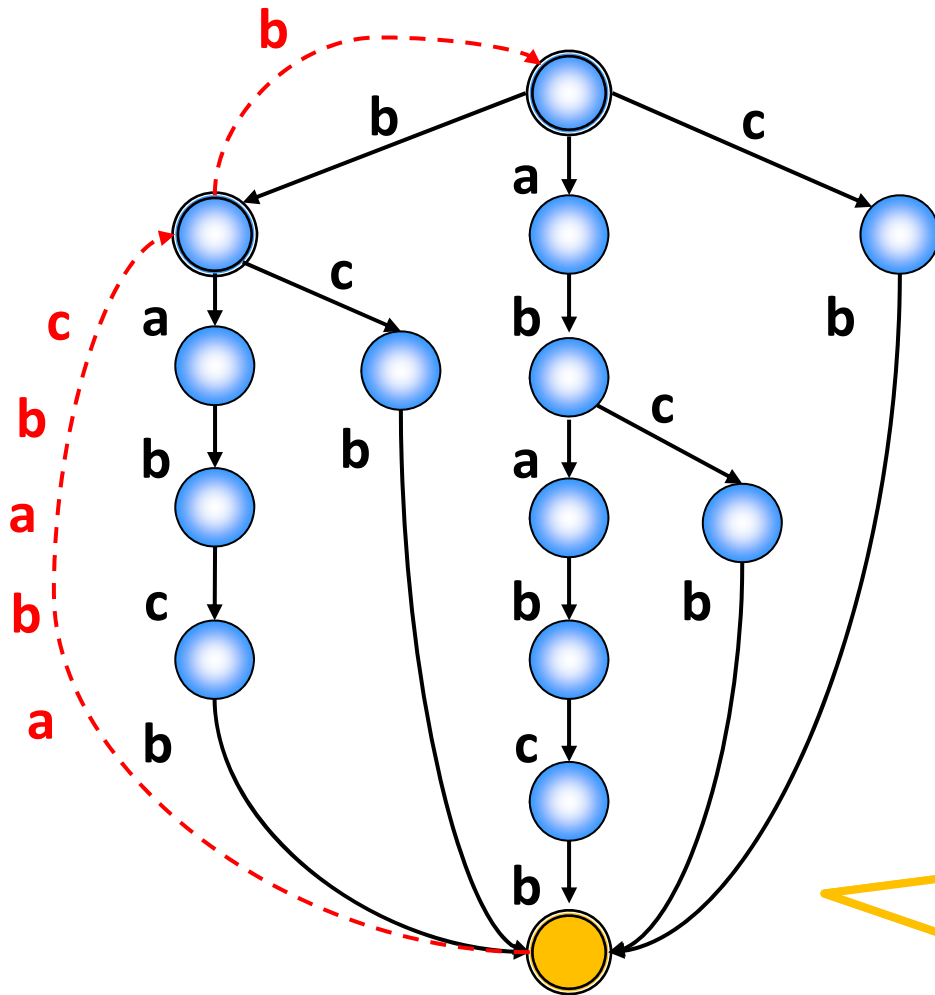
ノードをマージするとき、  
suffix link を同時に  
パス縮約する

# ノードのマージと Suffix Link の関係



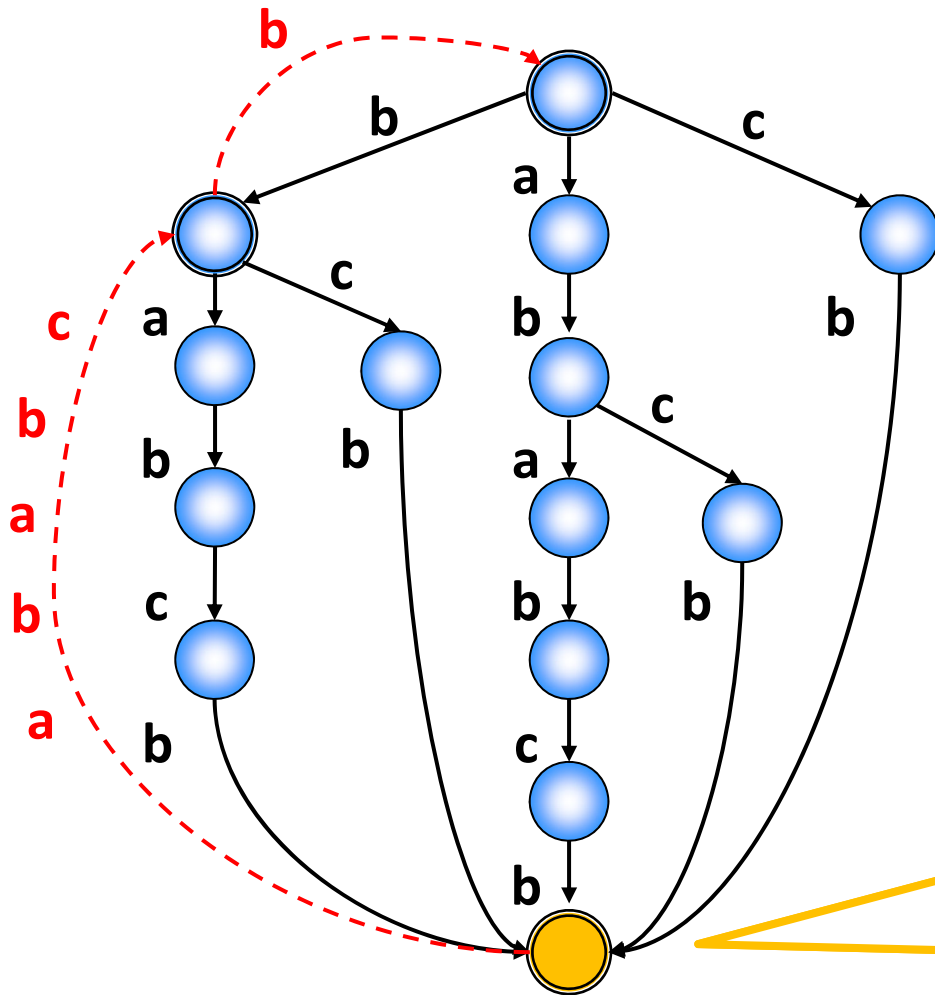
ノードをマージするとき、  
suffix link を同時に  
パス縮約する

# ノードのマージと Suffix Link の関係



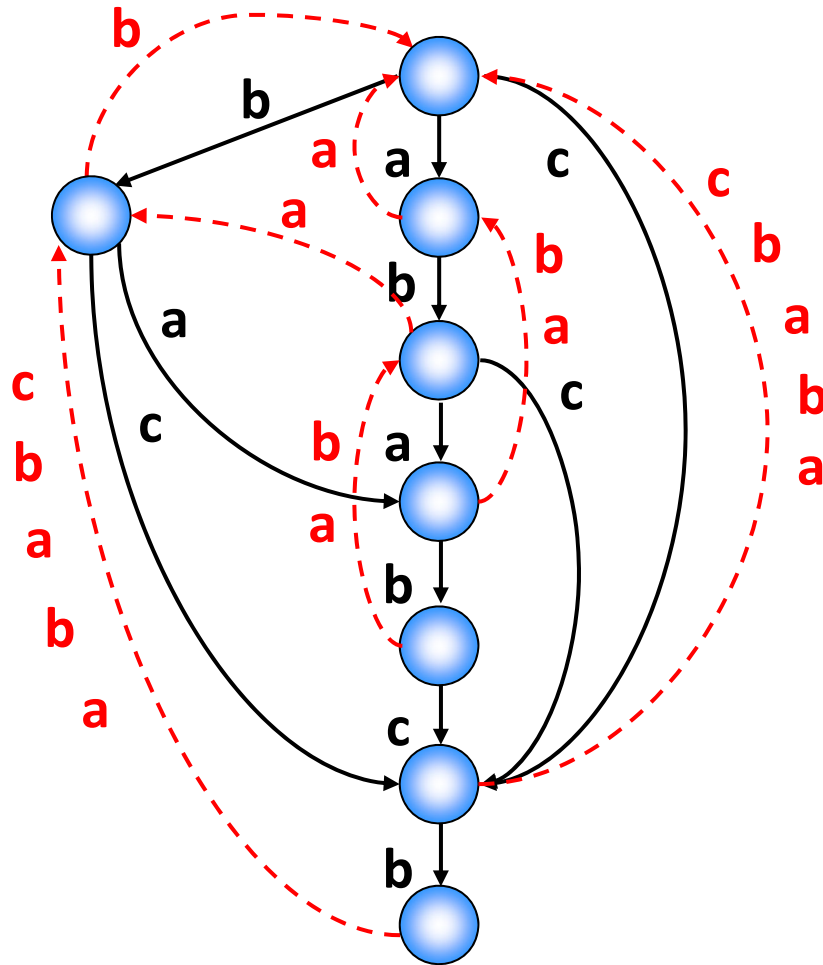
ノードをマージするとき、  
suffix link を同時に  
パス縮約する

# ノードのマージと Suffix Link の関係

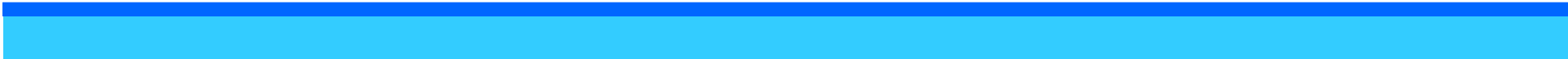


この縮約したパスが  
DAWG のこの頂点の  
suffix link である。

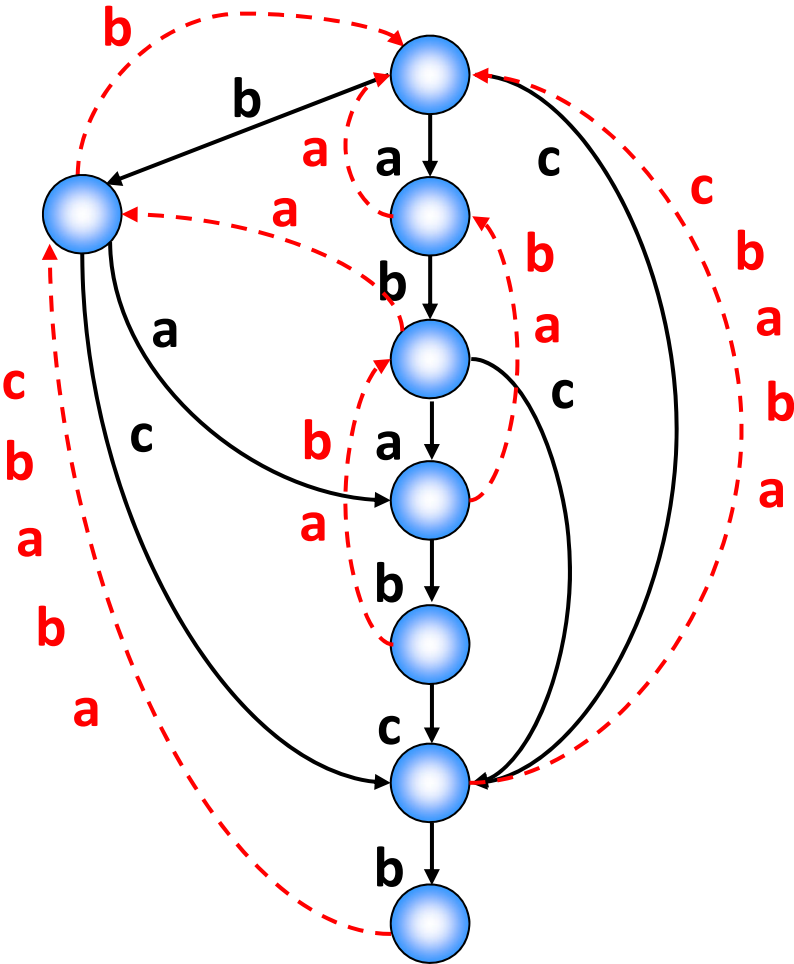
# Suffix Links of DAWG



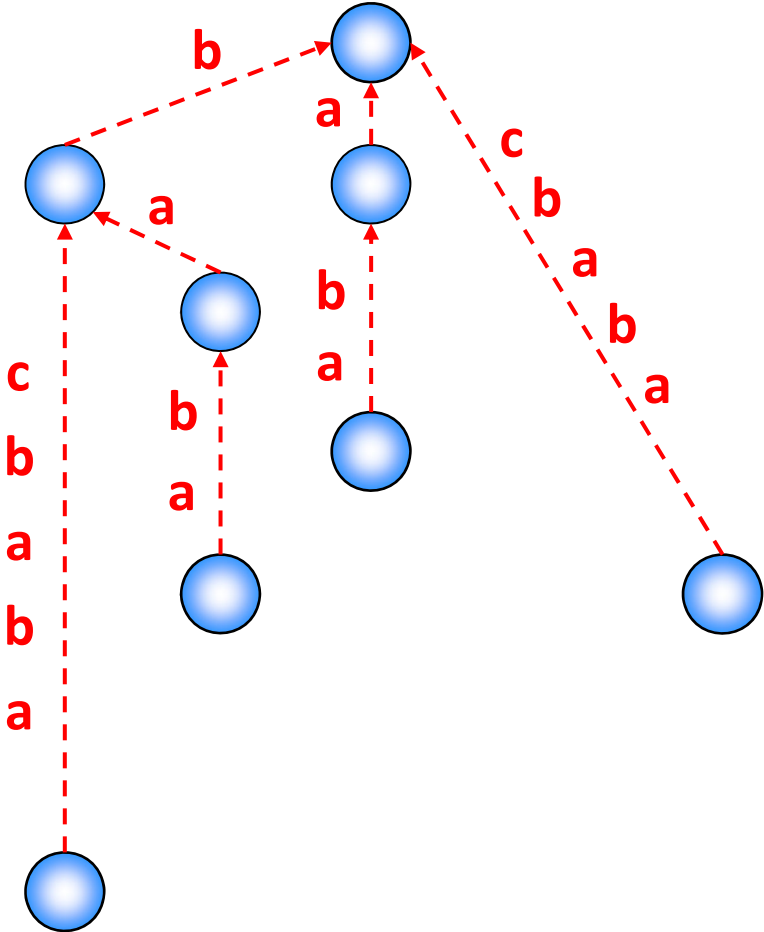
DAWG の suffix link も  
また木をなす



# Suffix Links of DAWG

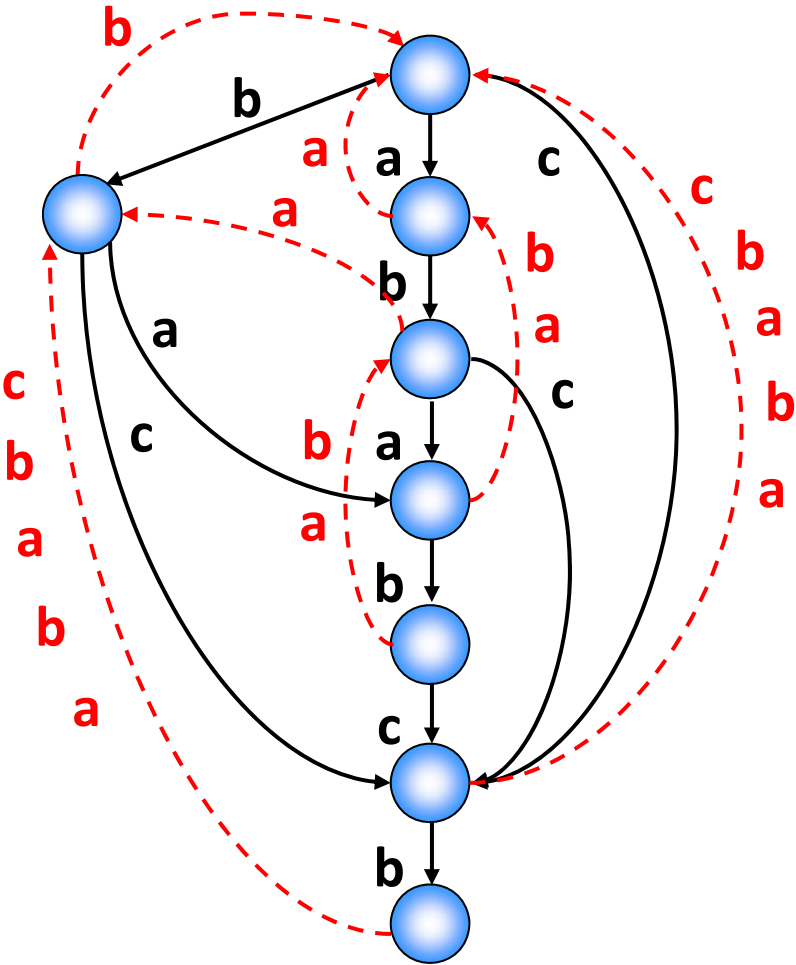


DAWG of **ababcb**

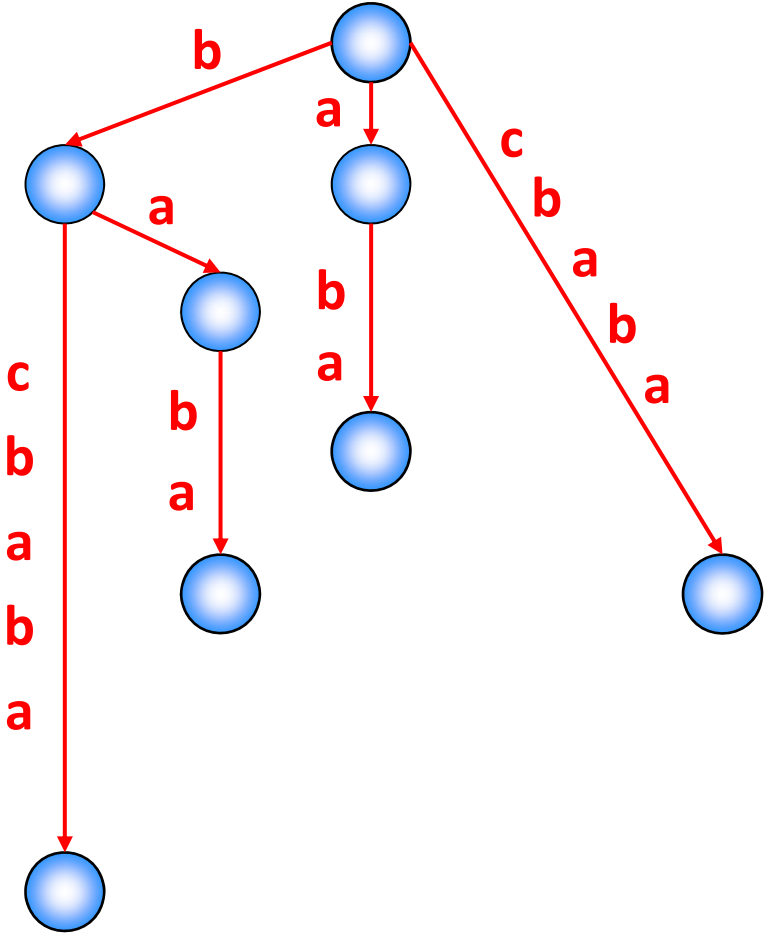


Contracted SLT of **ababcb**

# SLT of DAWG = 反転文字列の Suffix Tree



DAWG of ababcb



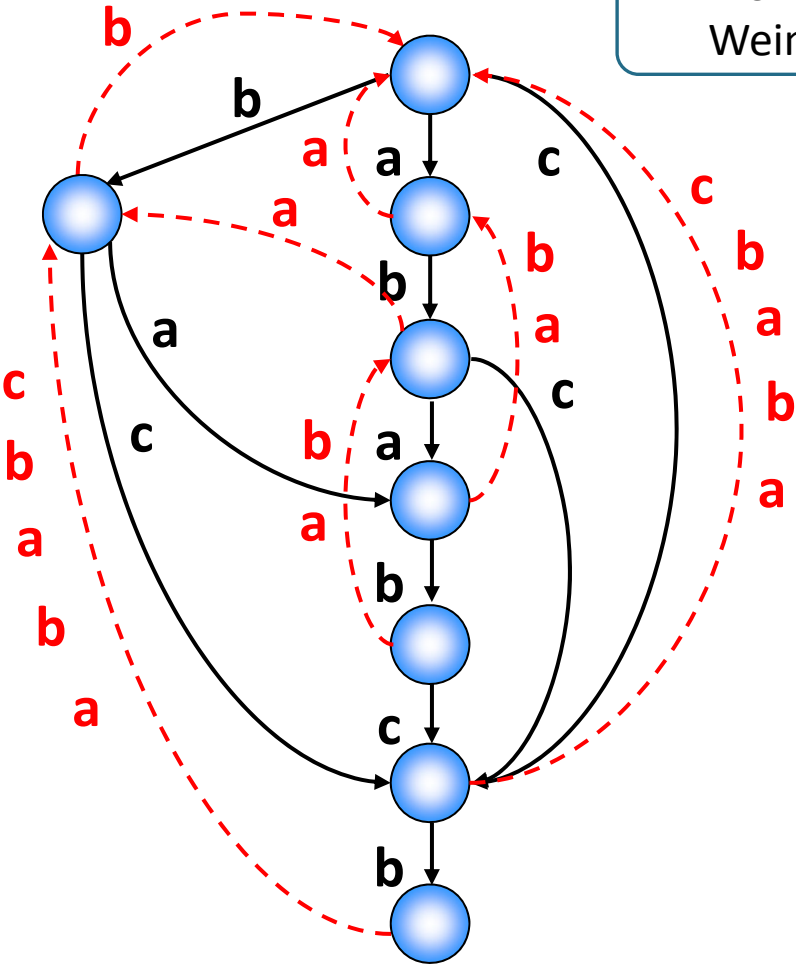
Suffix Tree of bcbaba



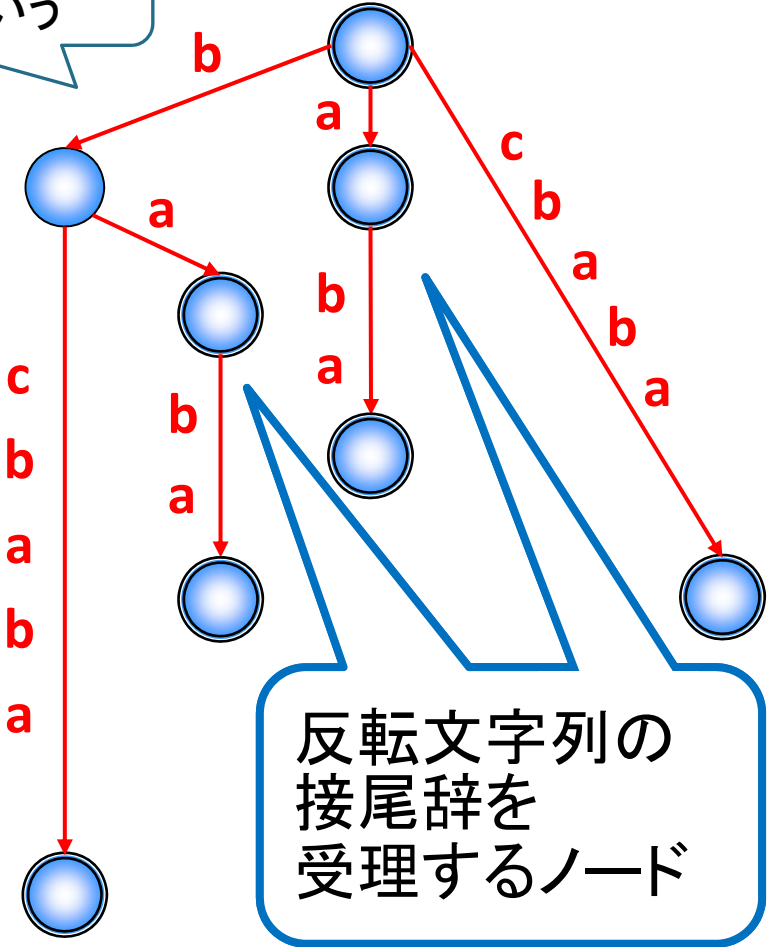


# SLT of DAWG = 反転文字列の Weiner Tree

このような suffix tree を Weiner Tree という



DAWG of ababcb



Suffix Tree of bcbaba



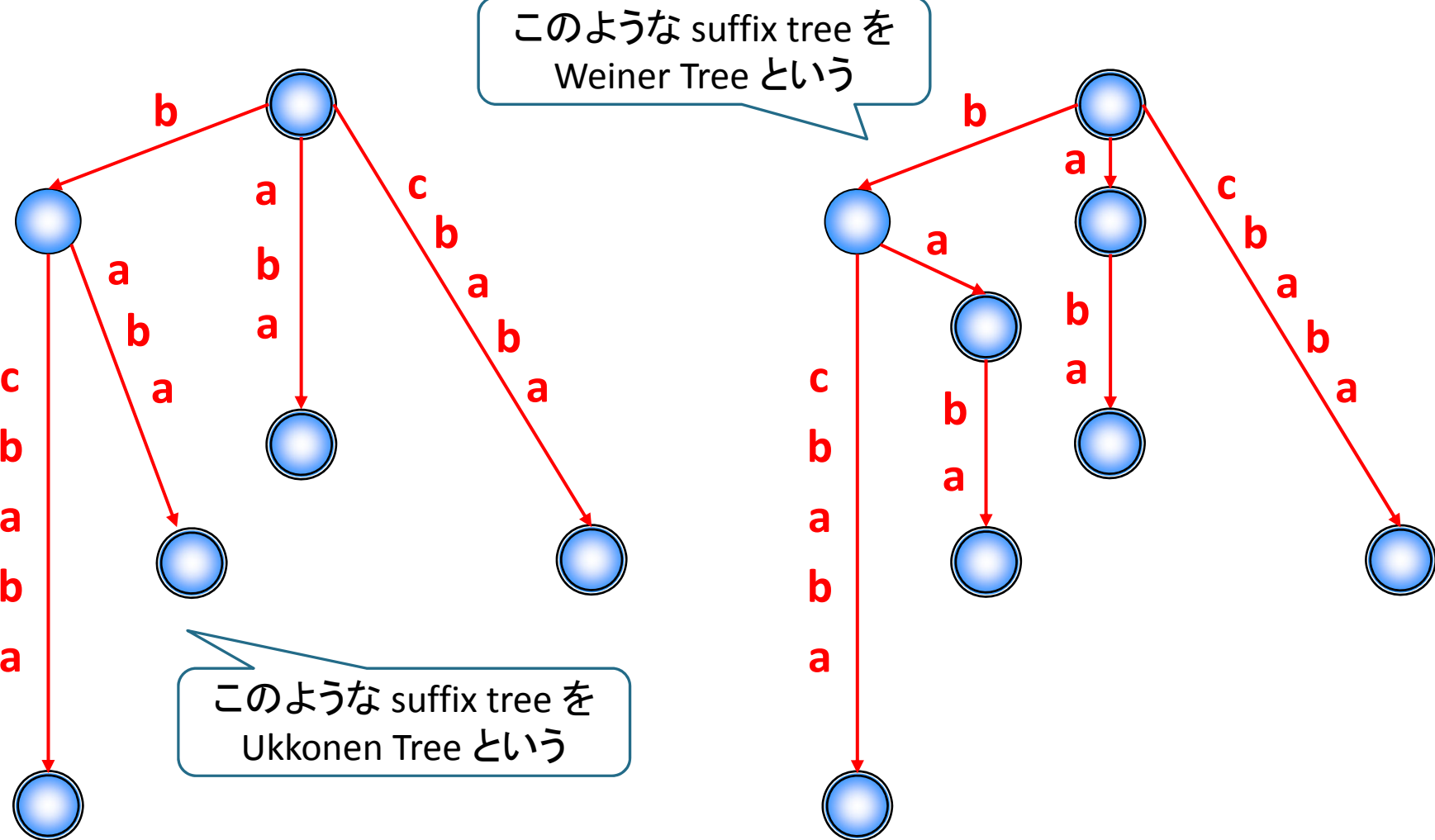
# SLT of DAWG = 反転文字列の Weiner Tree

## 常識3 [A. Blumer et al. 1985]

文字列  $w$  の DAWG の suffix link は  
反転文字列  $w^R$  の Weiner tree に等しい.

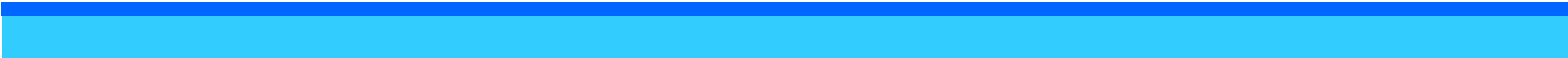
- $w$  の suffix trie のノードをマージしながら suffix link をパス縮約することと,  
 $w^R$  の suffix trie の辺をパス圧縮するのは  
同じことだから.

# 寄り道 : Ukkonen Tree と Weiner Tree



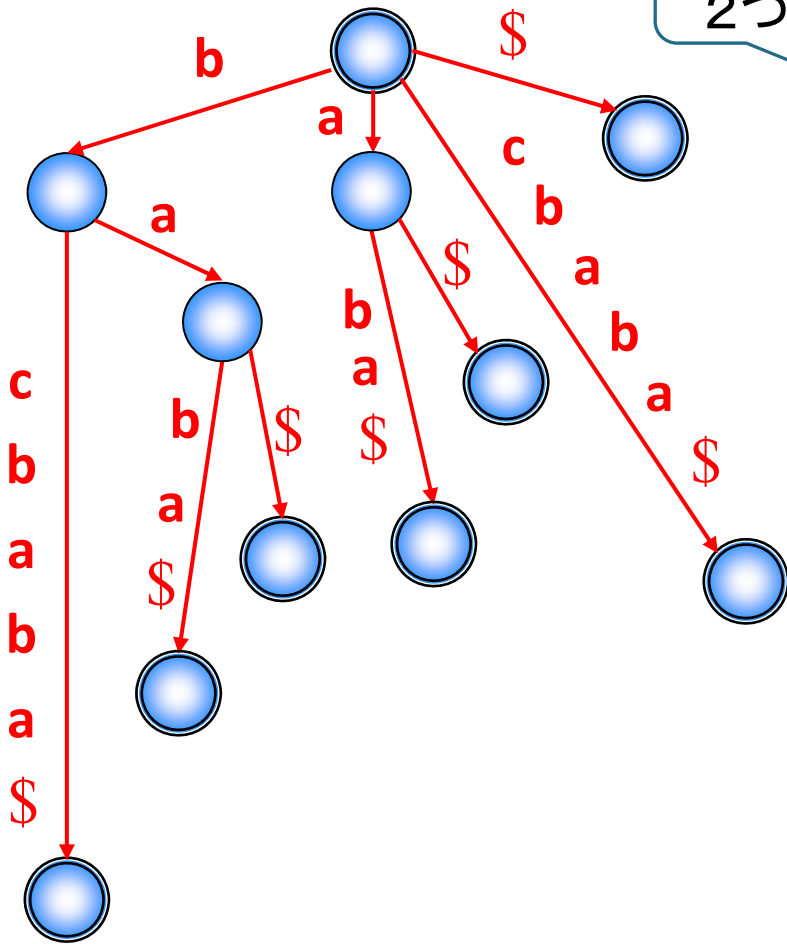
Ukkonen Tree of **bcbaba**

Weiner Tree of **bcbaba**

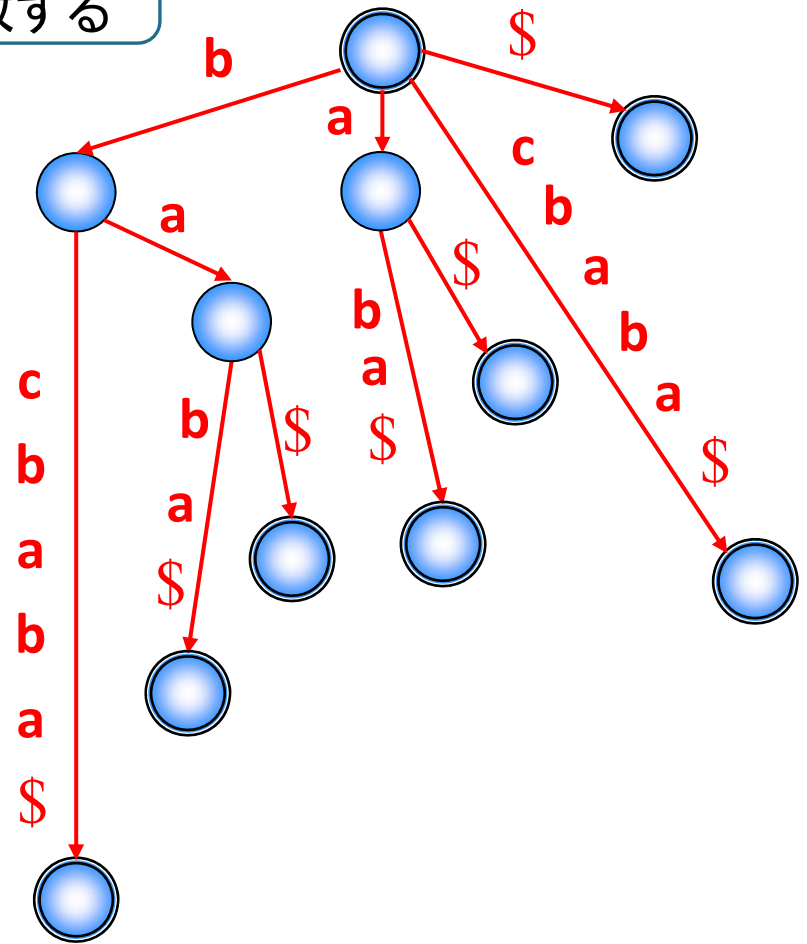


# 寄り道 : Ukkonen Tree と Weiner Tree

末尾に \$ を付けると  
2つの木は一致する



Ukkonen Tree of **bcbaba**\$



Weiner Tree of **bcbaba**\$



# DAWG の頂点数

常識4 [A. Blumer et al. 1985]

DAWG の頂点数は高々  $2n-1$  である。  
 $n$  は文字列の長さ。

- Weiner tree (Suffix tree) の頂点数は高々  $2n-1$  だから、簡単ですね！

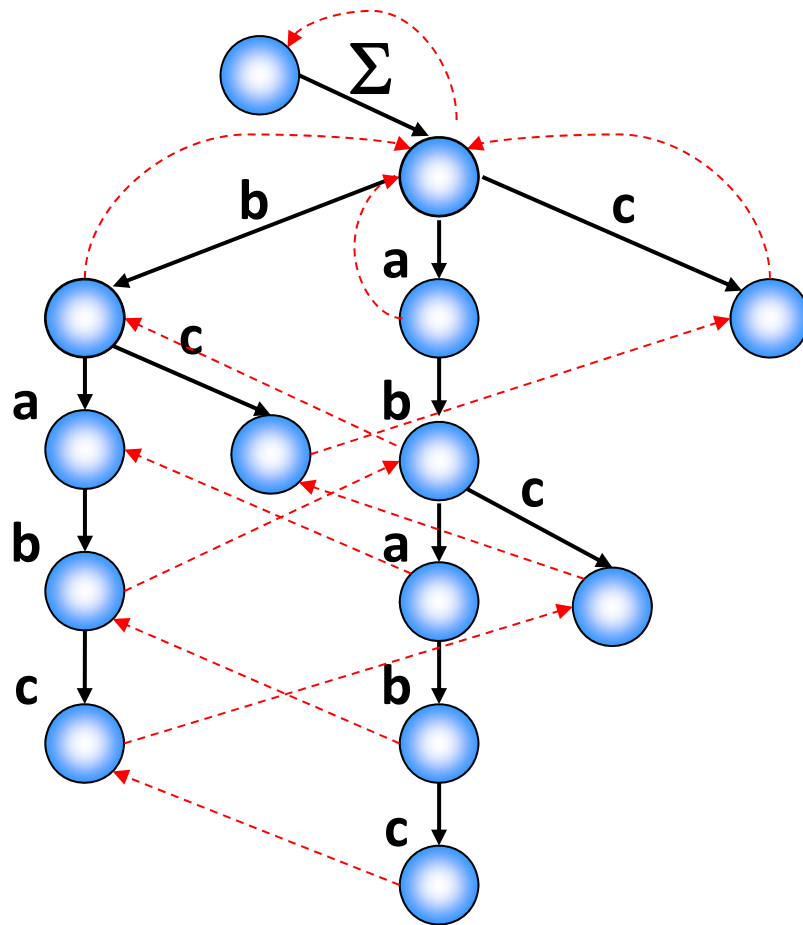
# DAWG の辺数

常識5 [A. Blumer et al. 1985]

DAWG の辺数は高々  $3n-2$  である。  
 $n$  は文字列の長さ。

- 非自明ですが、勘のいい人は10分くらい考えれば分かるでしょう。
- 次回 StringBeginners までの宿題！

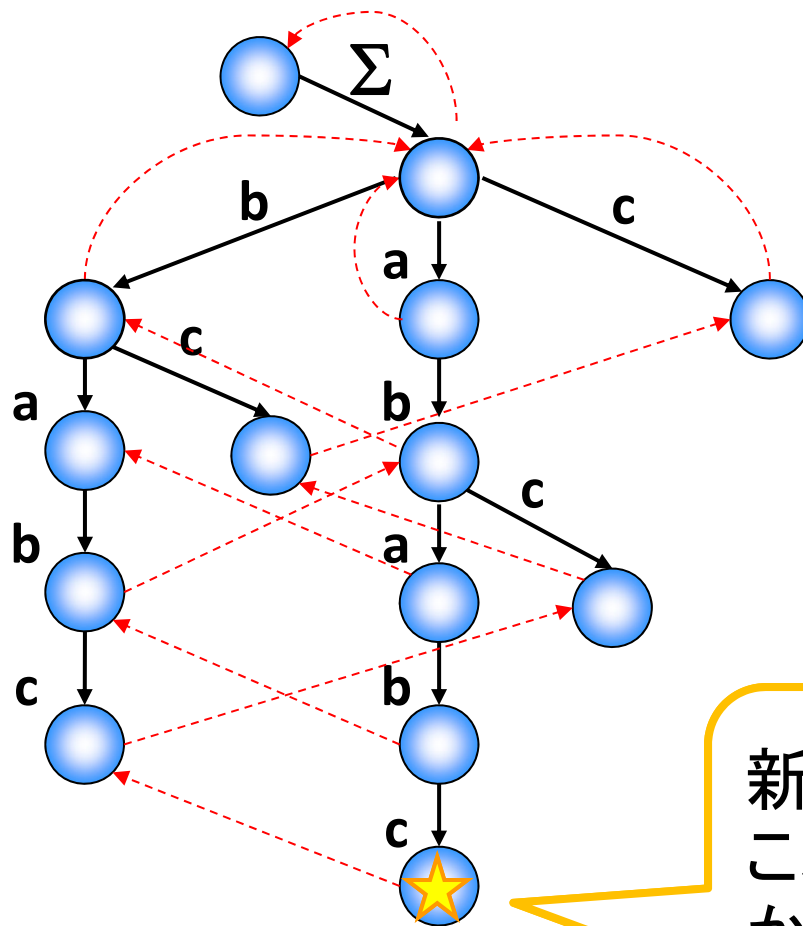
# Suffix Trie の左→右オンライン構築



ababcb

まずは、すべての  
基礎である  
Suffix Trie から！

# Suffix Trie の左→右オンライン構築

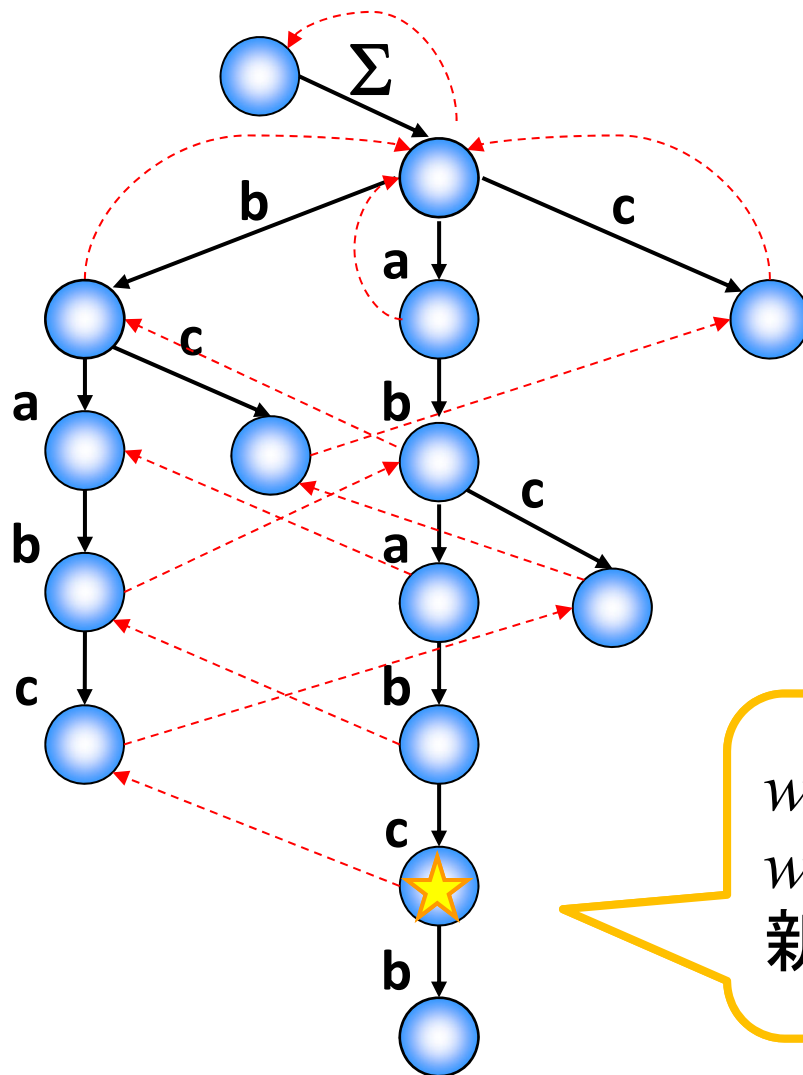


ababcb

新しい文字  $w[i]$  を読み込んだら、  
これまでの最長の葉  $w[1..i-1]$   
から構築を開始する。



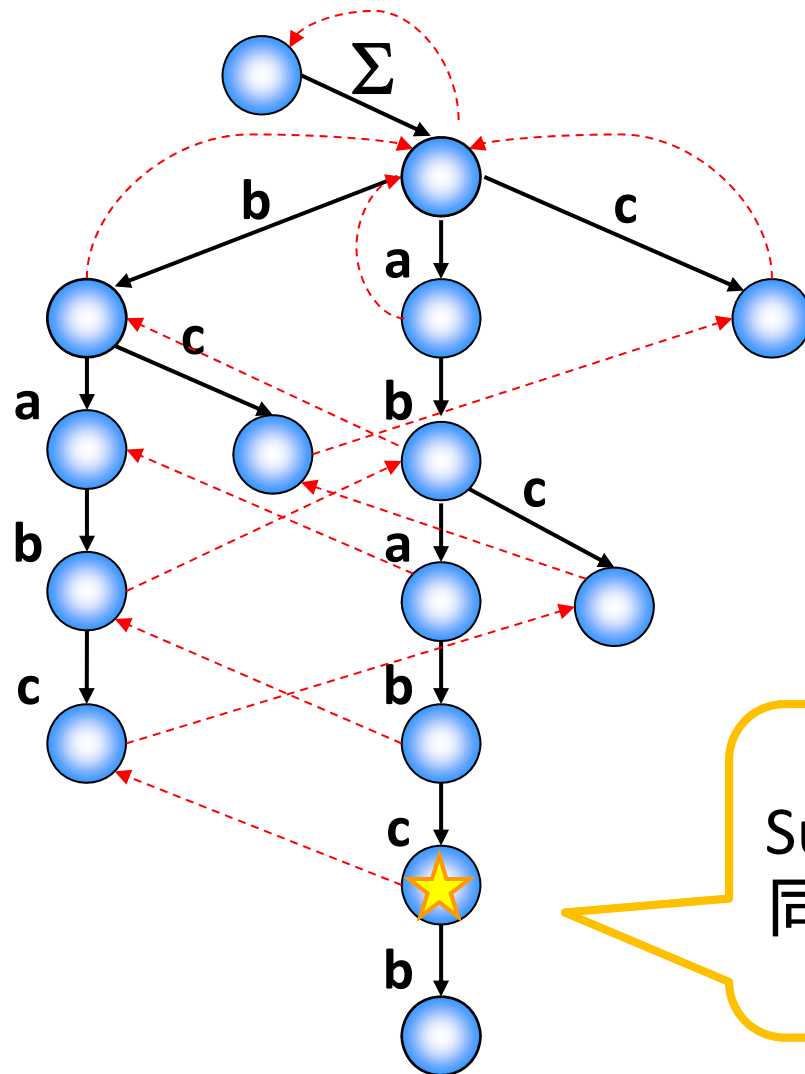
# Suffix Trie の左→右オンライン構築



ababcb

$w[i]$  で辿れない場合は,  
 $w[i]$  でラベル付けされた  
新しい辺と葉を作る.

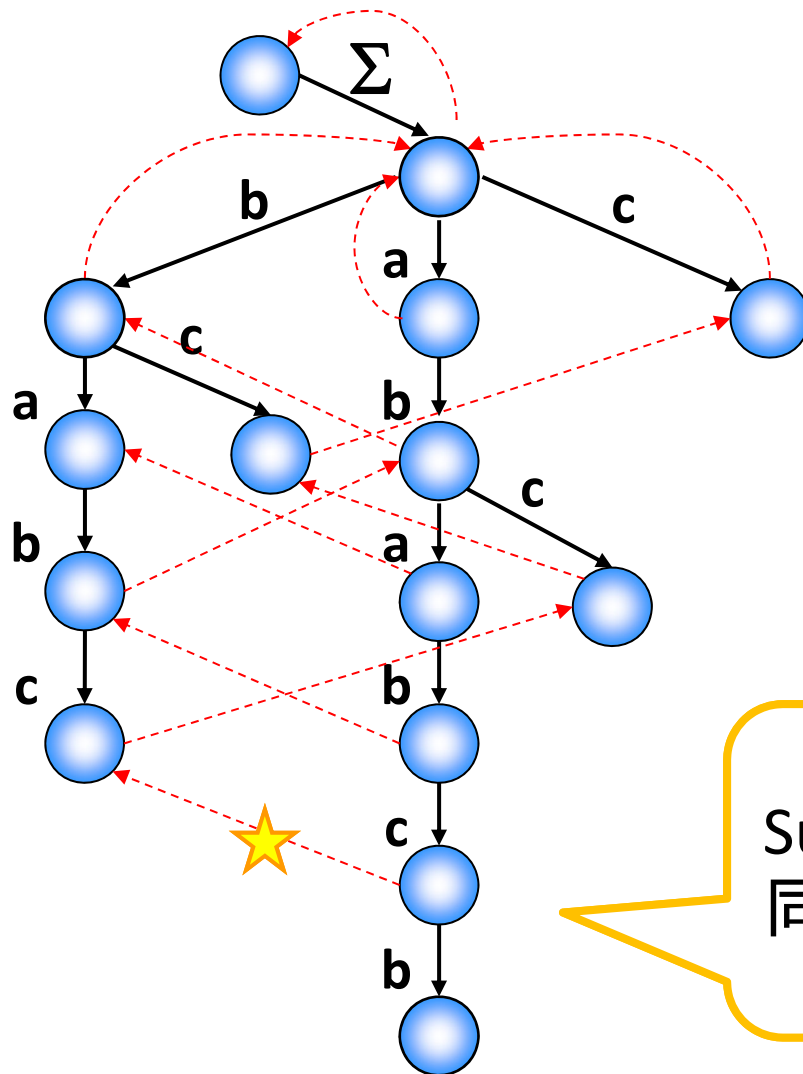
# Suffix Trie の左→右オンライン構築



ababcb

Suffix link を辿った先で,  
同じことを繰り返す.

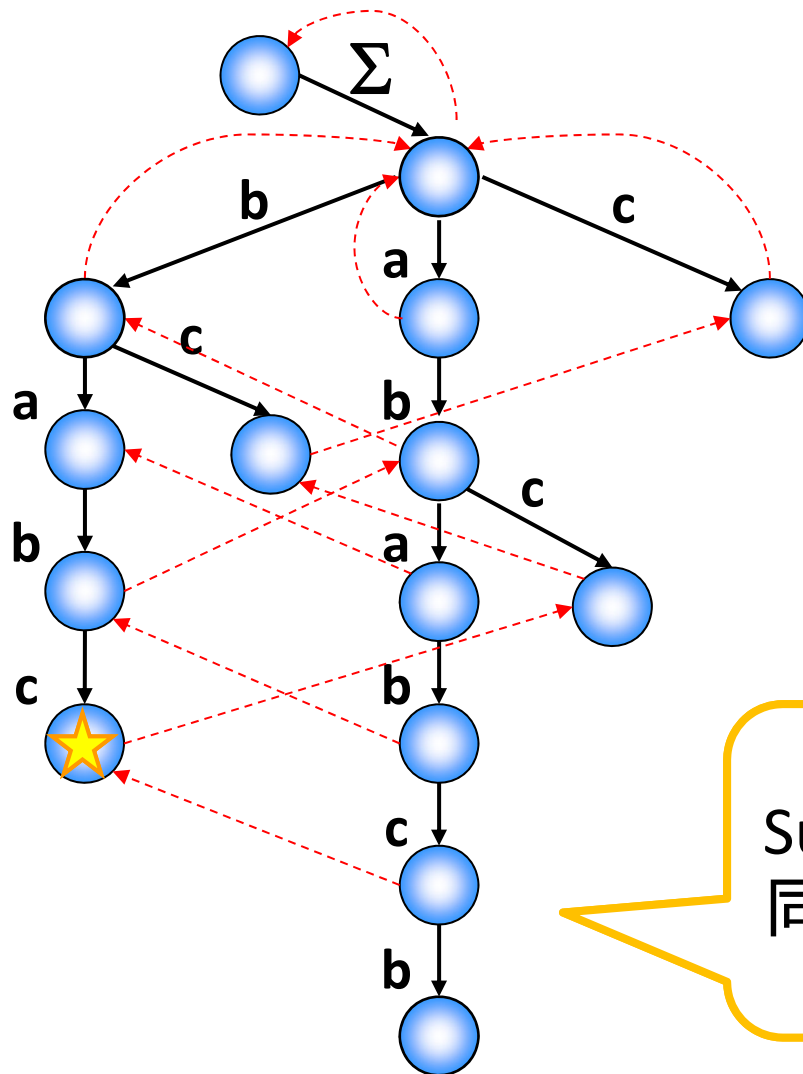
# Suffix Trie の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

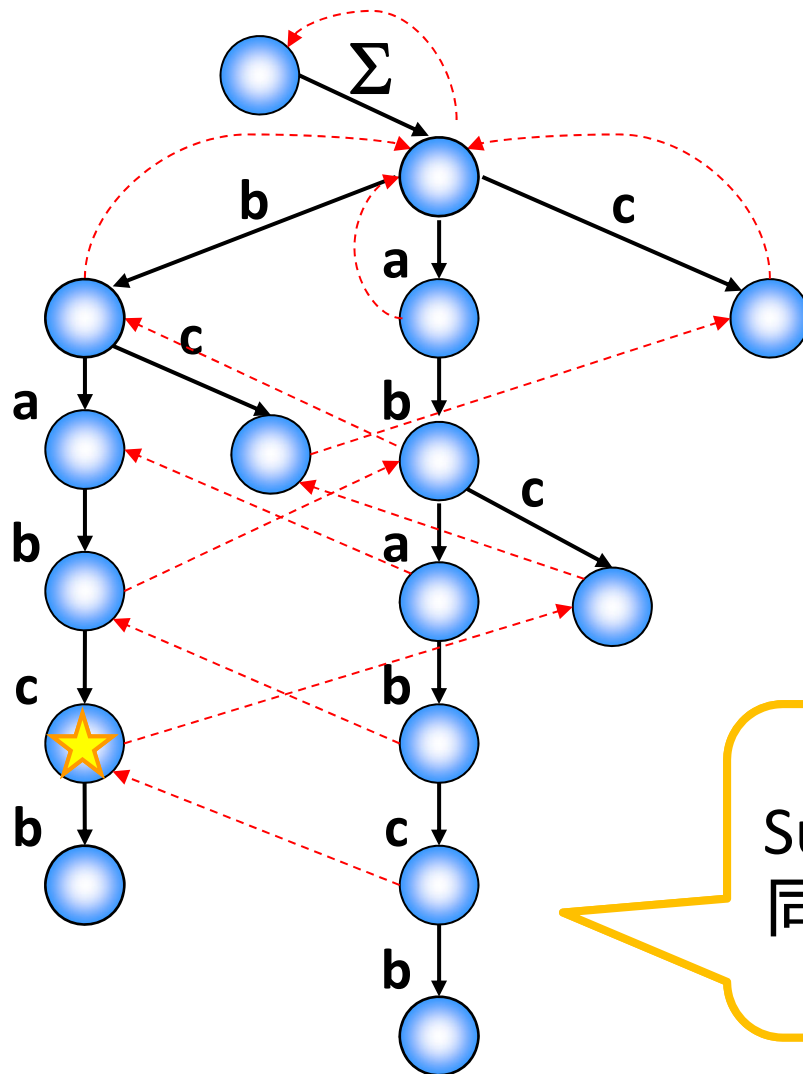
# Suffix Trie の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

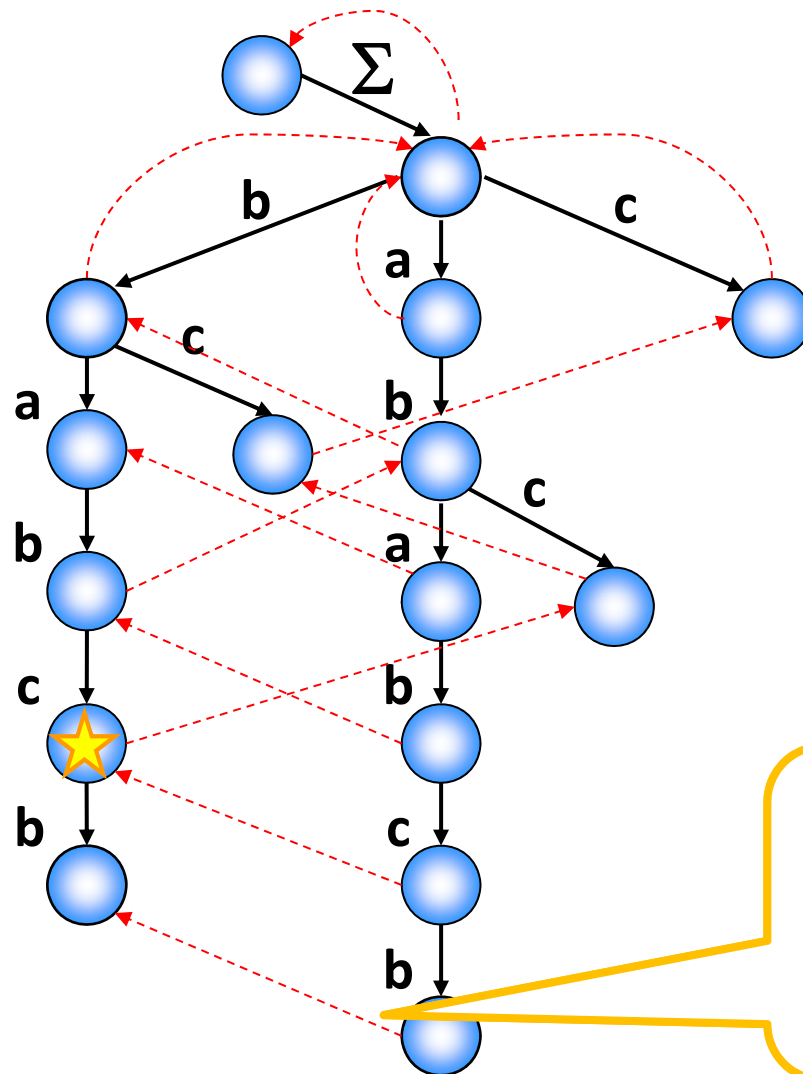
# Suffix Trie の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

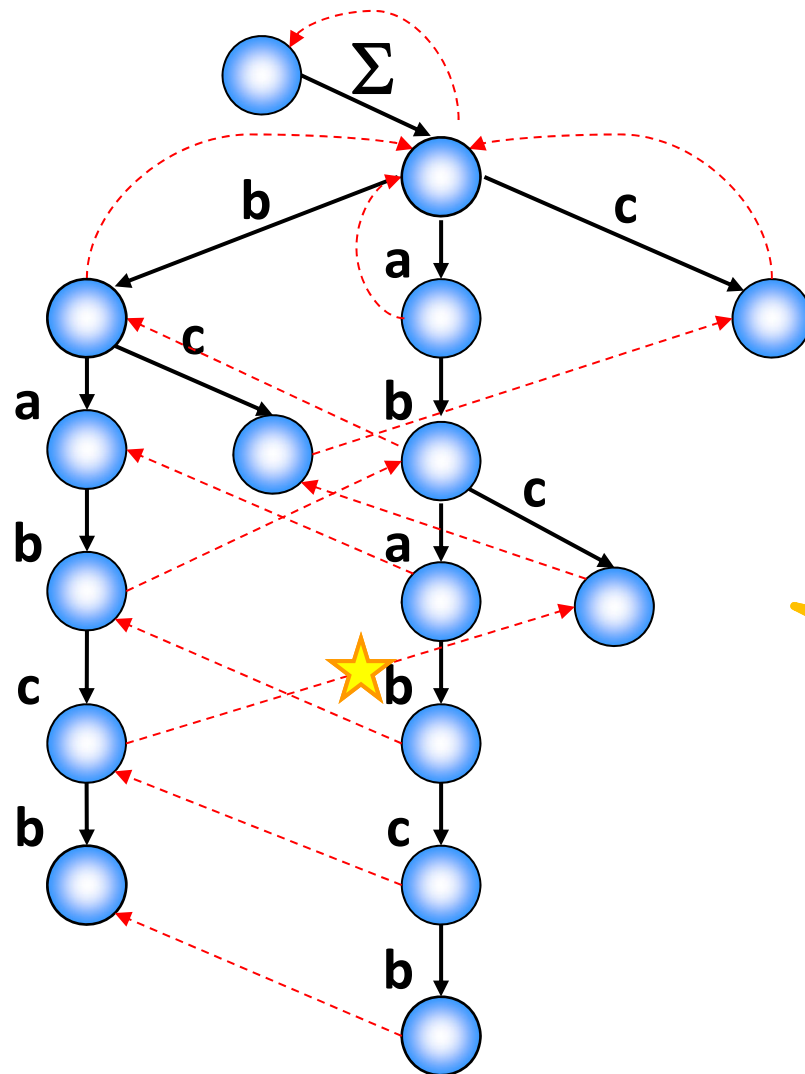
# Suffix Trie の左→右オンライン構築



ababcb

葉と葉を繋ぐ suffix link も  
このときに追加する.

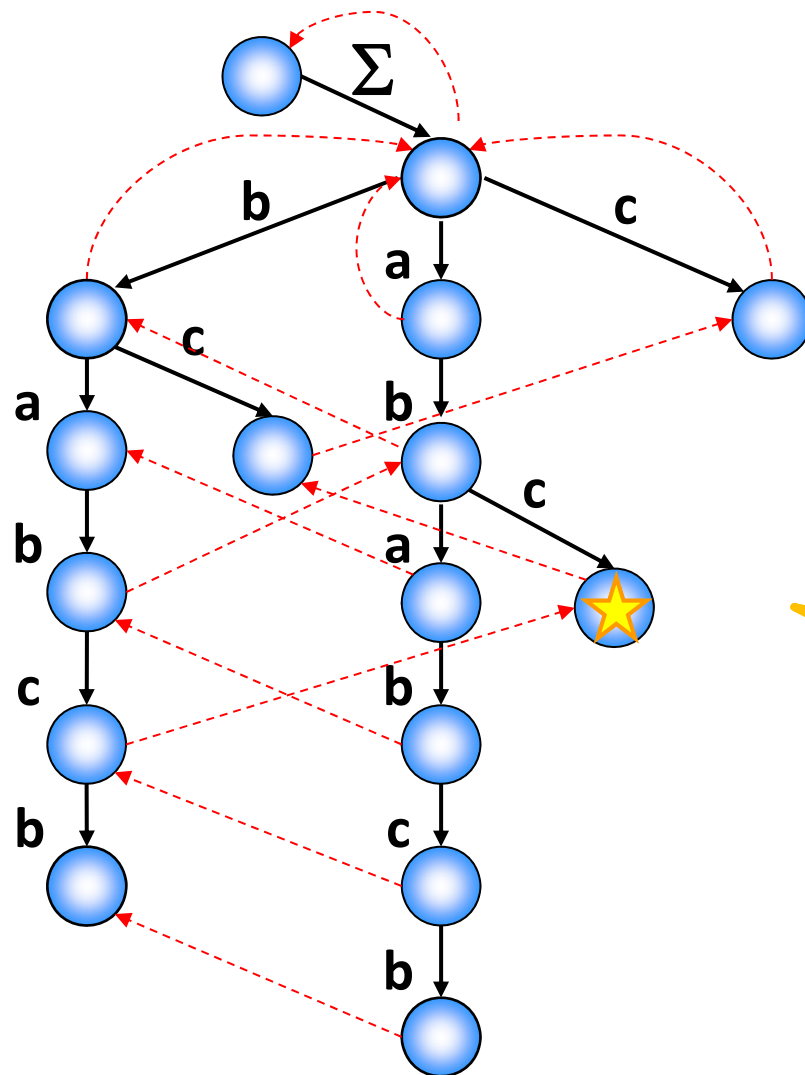
# Suffix Trie の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

# Suffix Trie の左→右オンライン構築

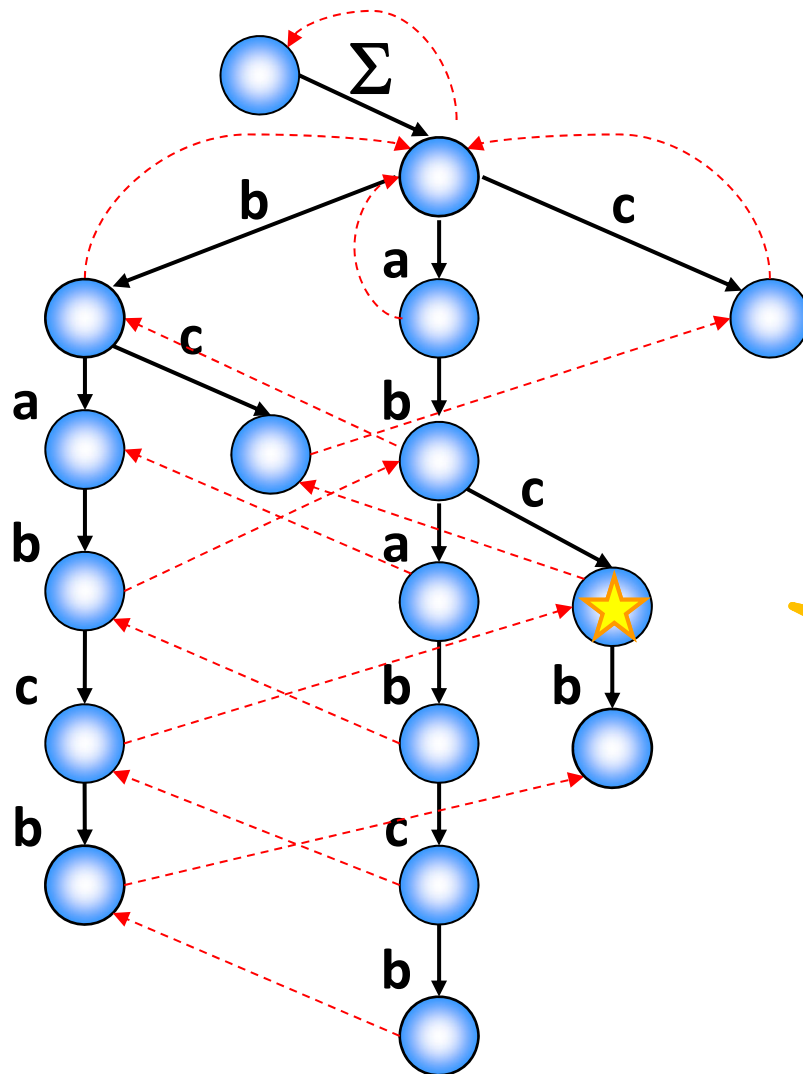


ababcb

Suffix link を辿った先で、  
同じことを繰り返す。



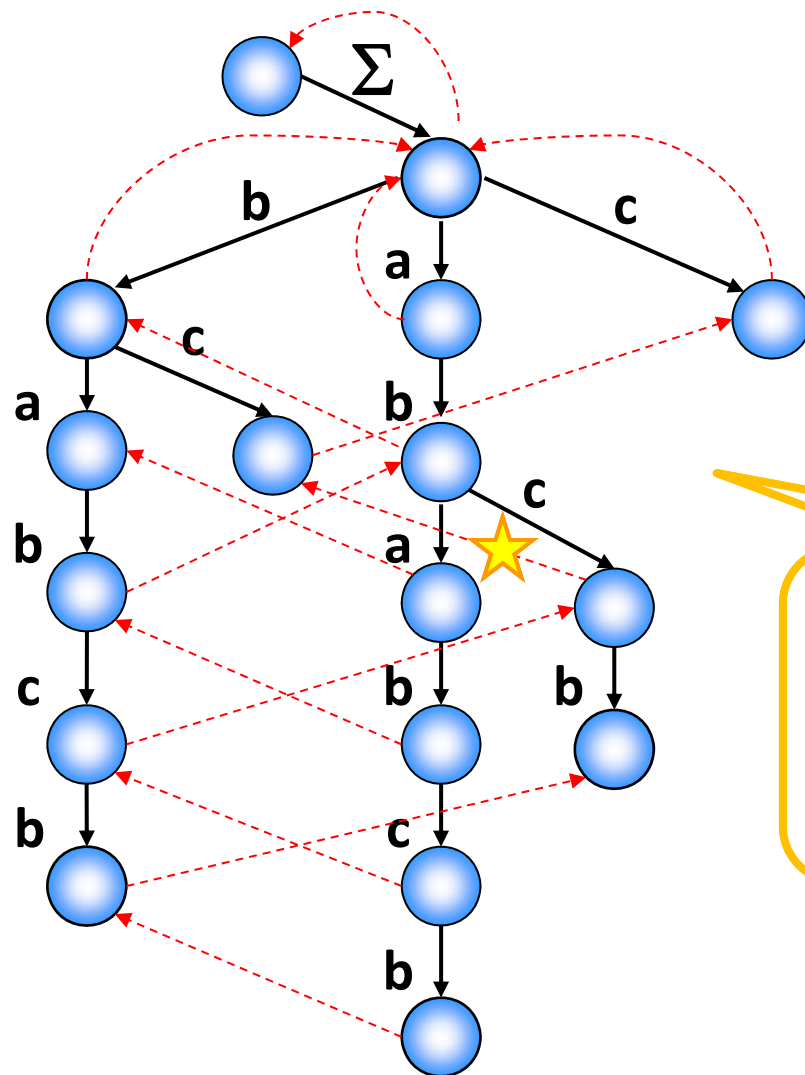
# Suffix Trie の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

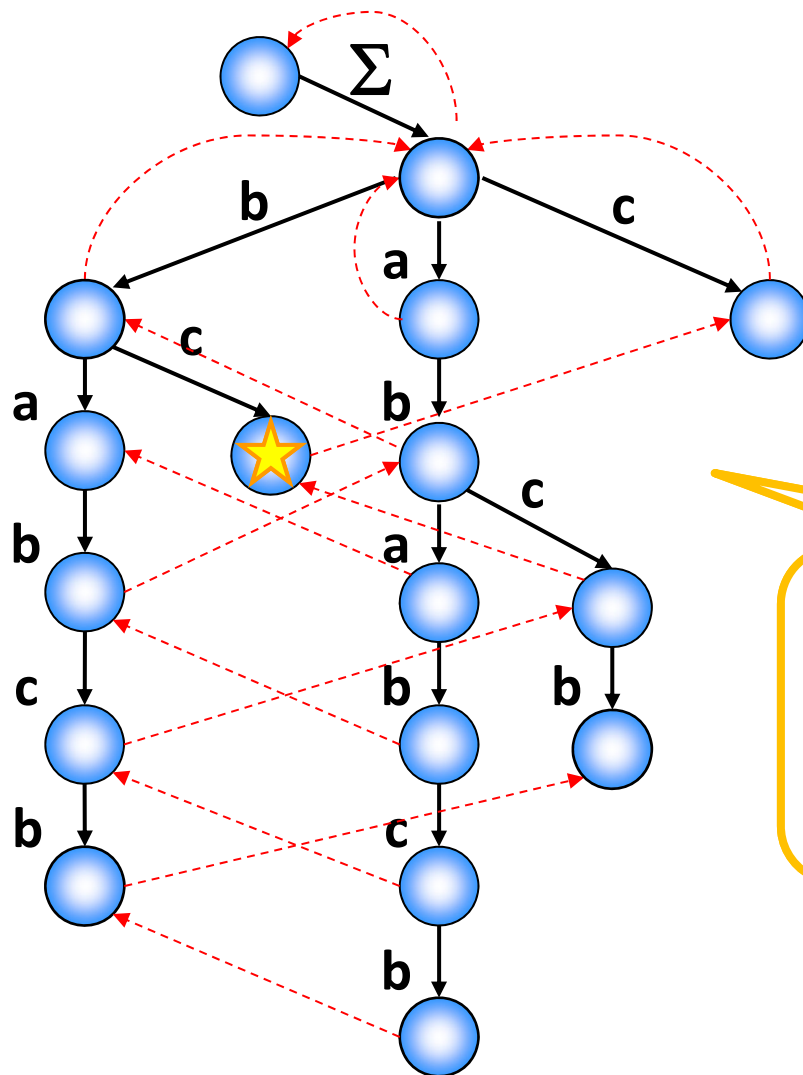
# Suffix Trie の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

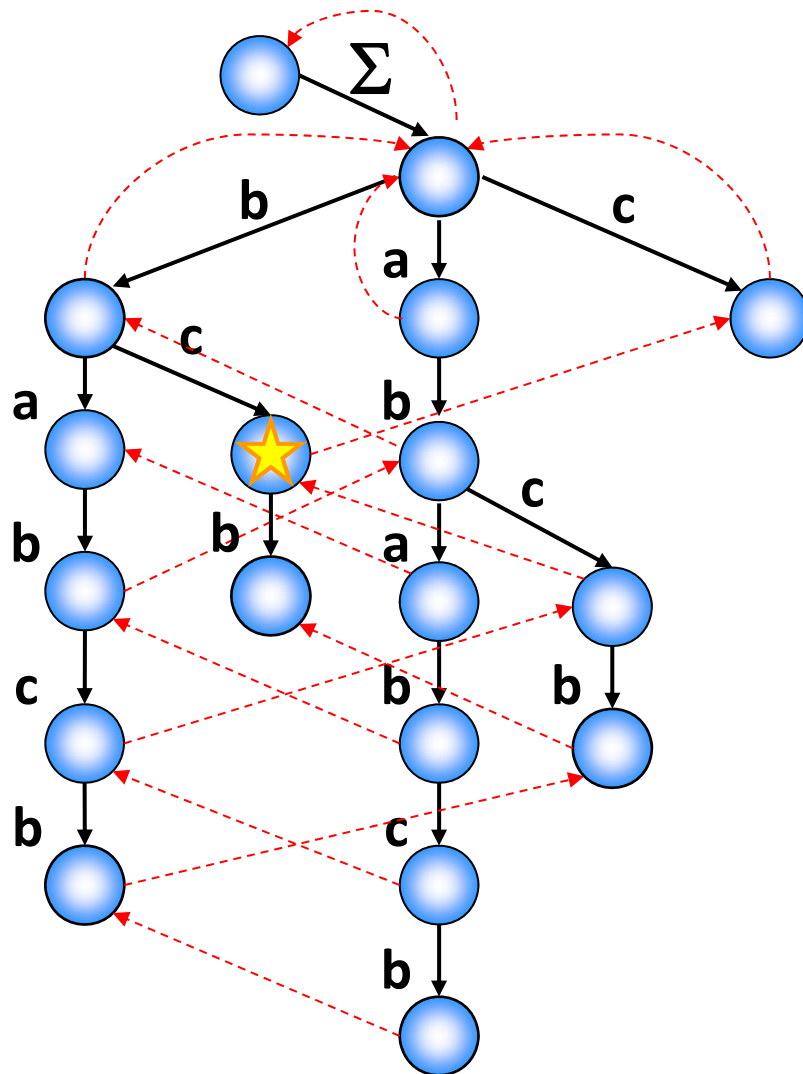
# Suffix Trie の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

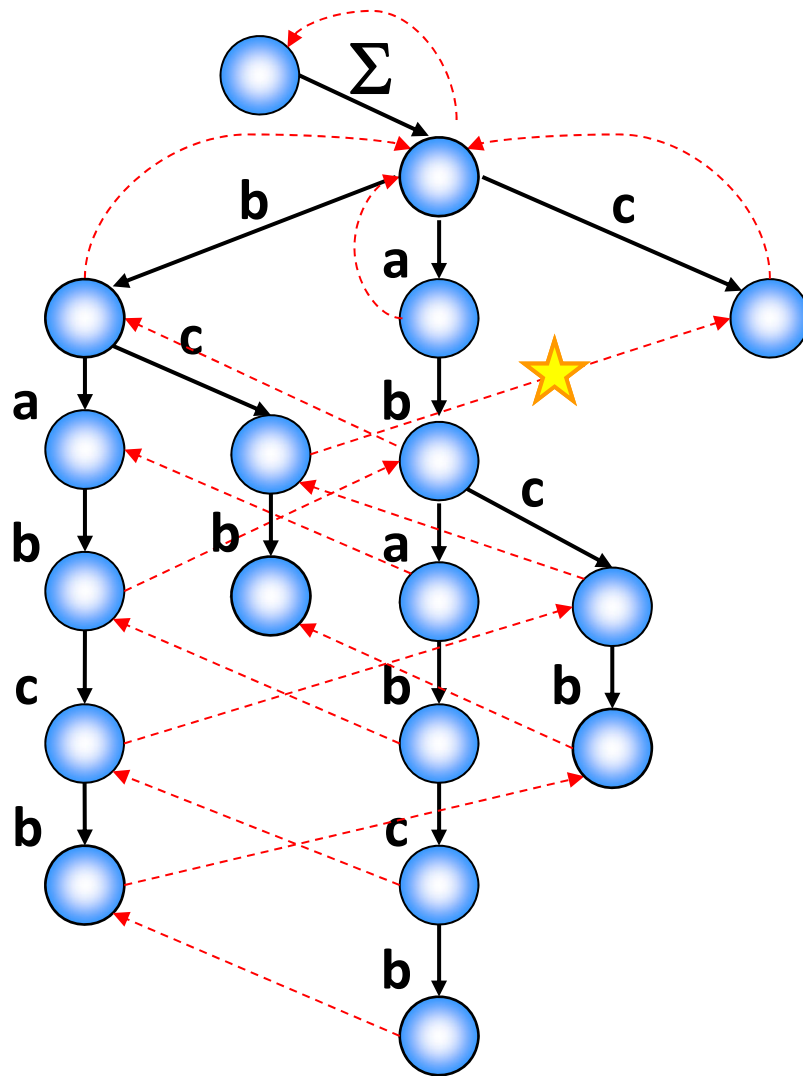
# Suffix Trie の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

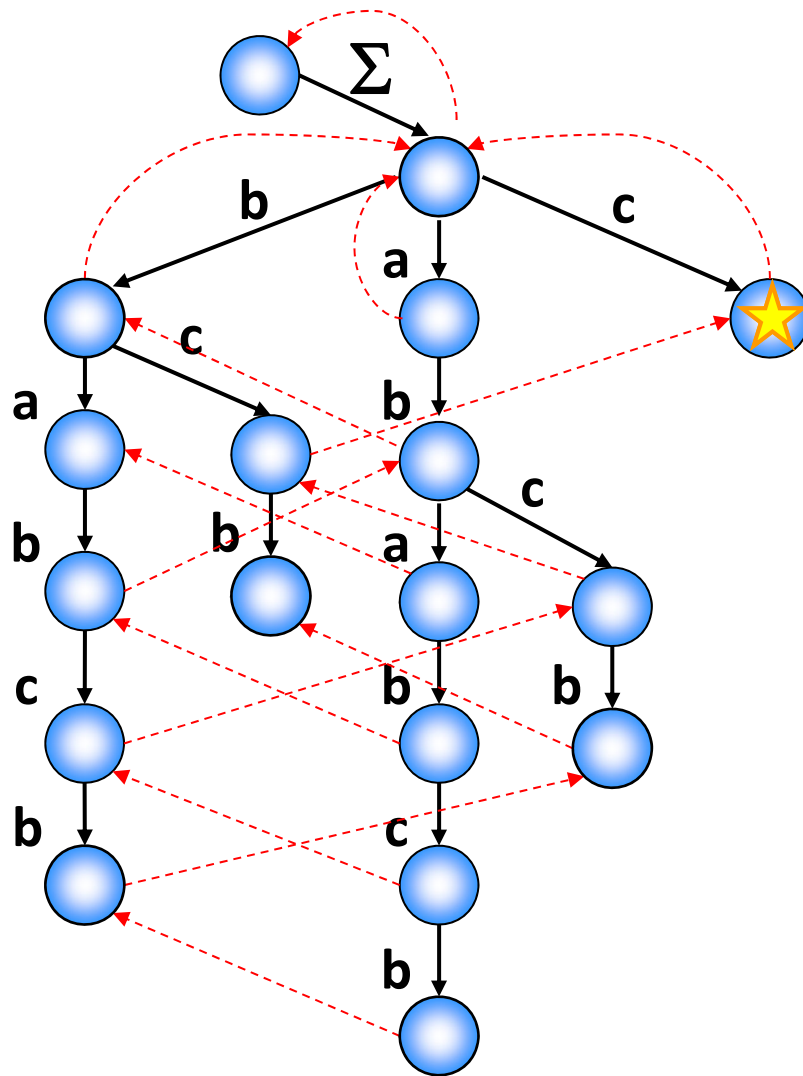
# Suffix Trie の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

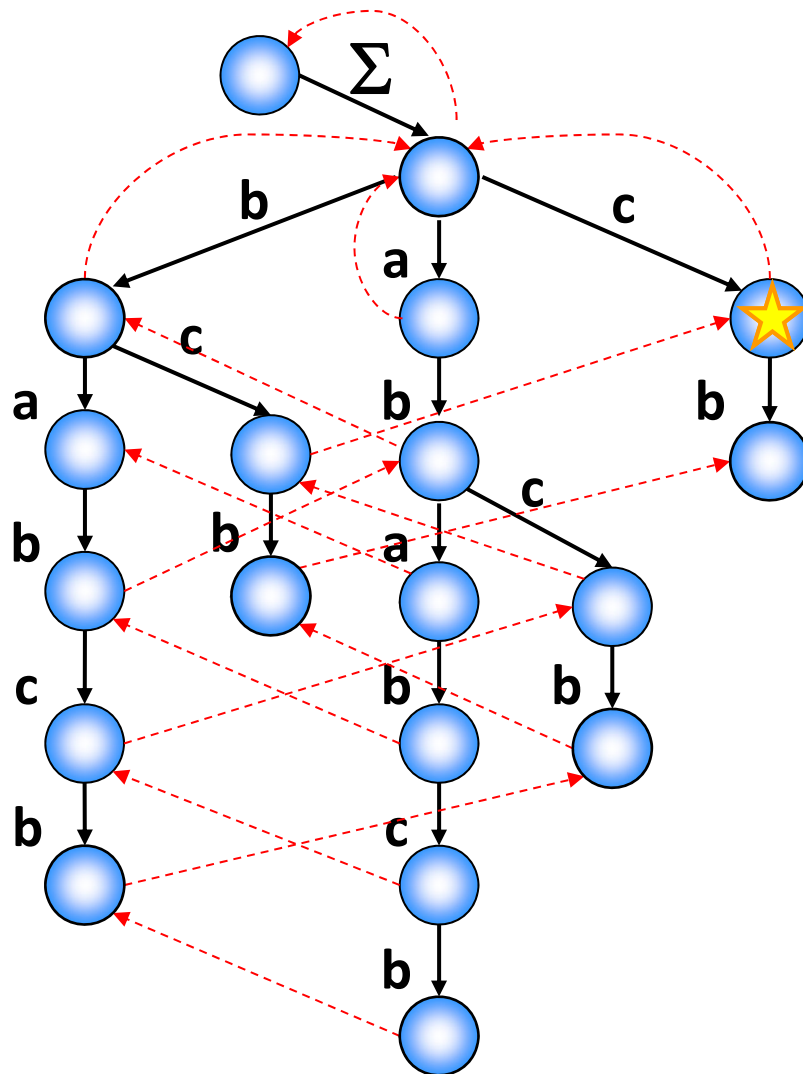
# Suffix Trie の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

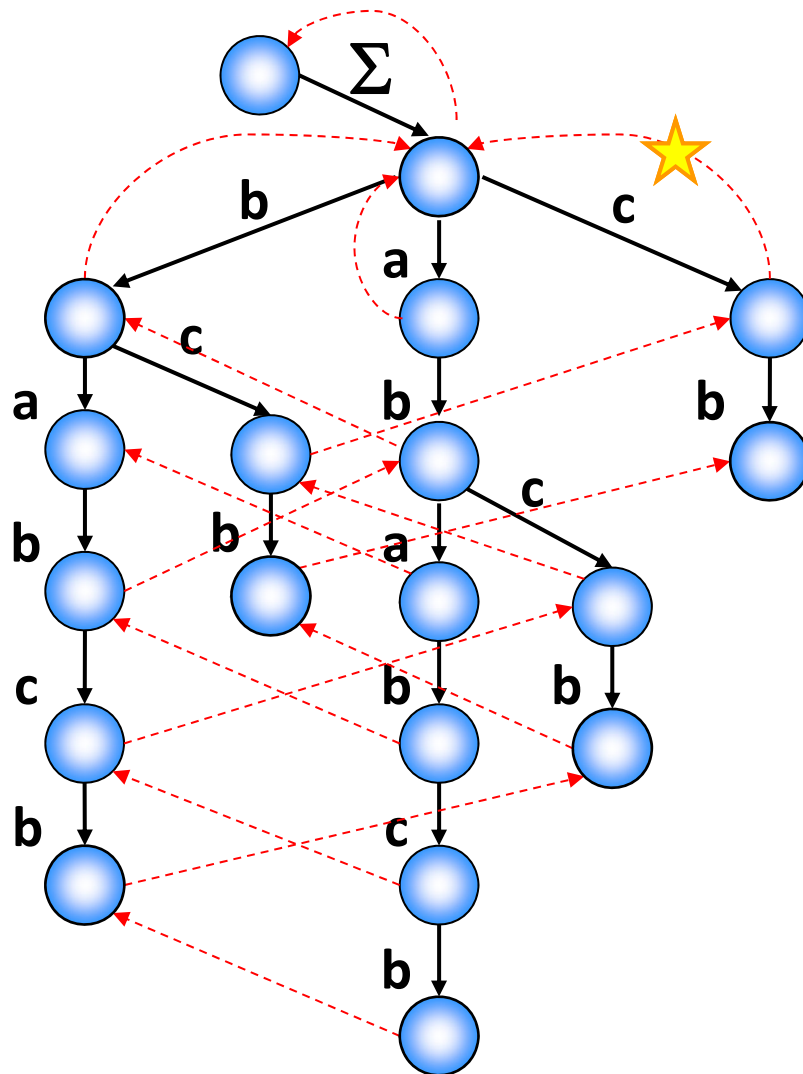
# Suffix Trie の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

# Suffix Trie の左→右オンライン構築

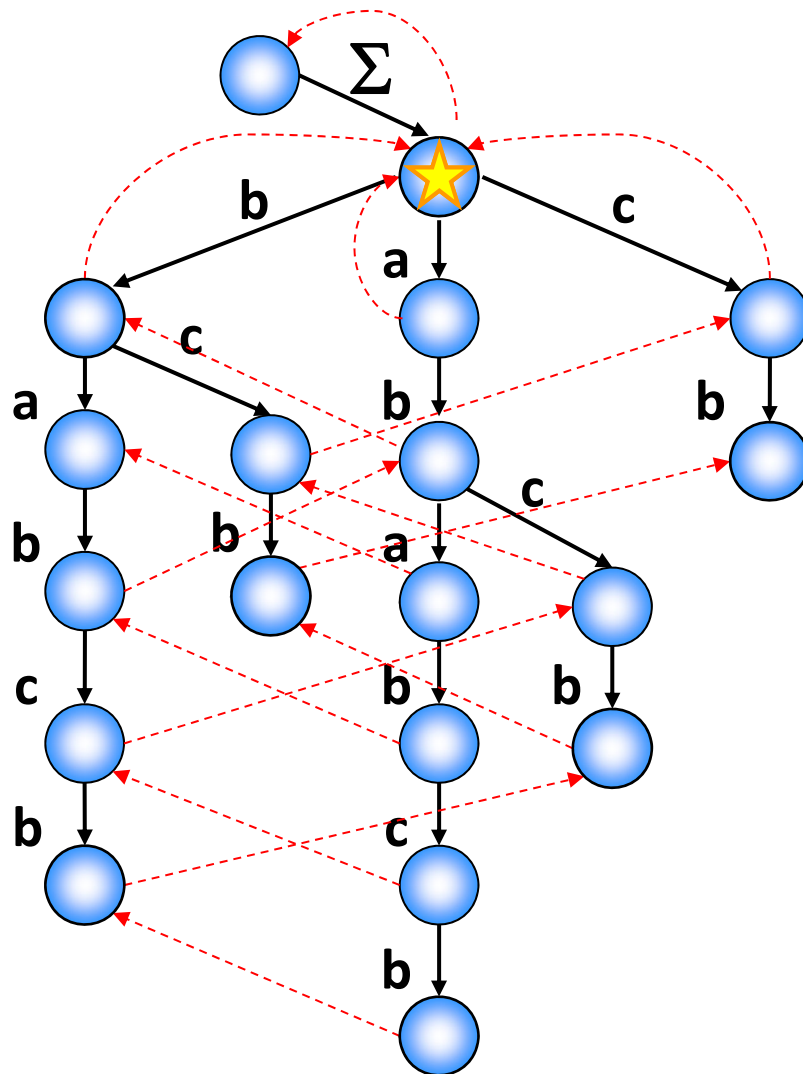


ababcb

Suffix link を辿った先で、  
同じことを繰り返す。



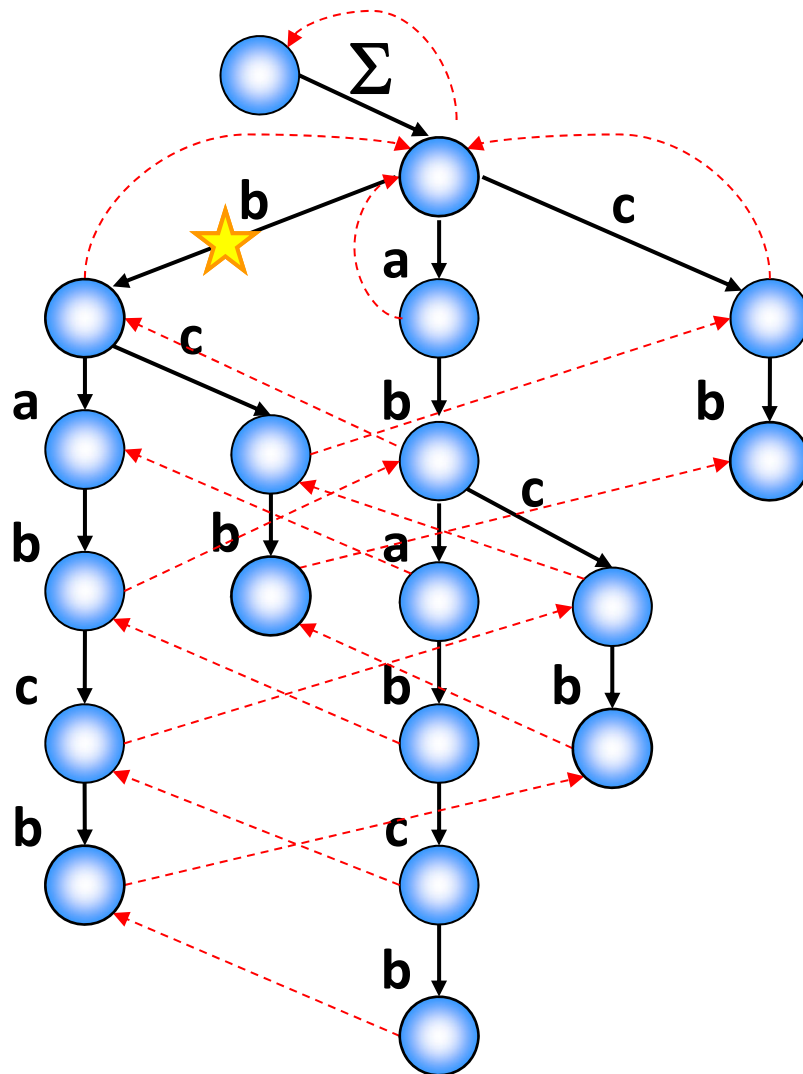
# Suffix Trie の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

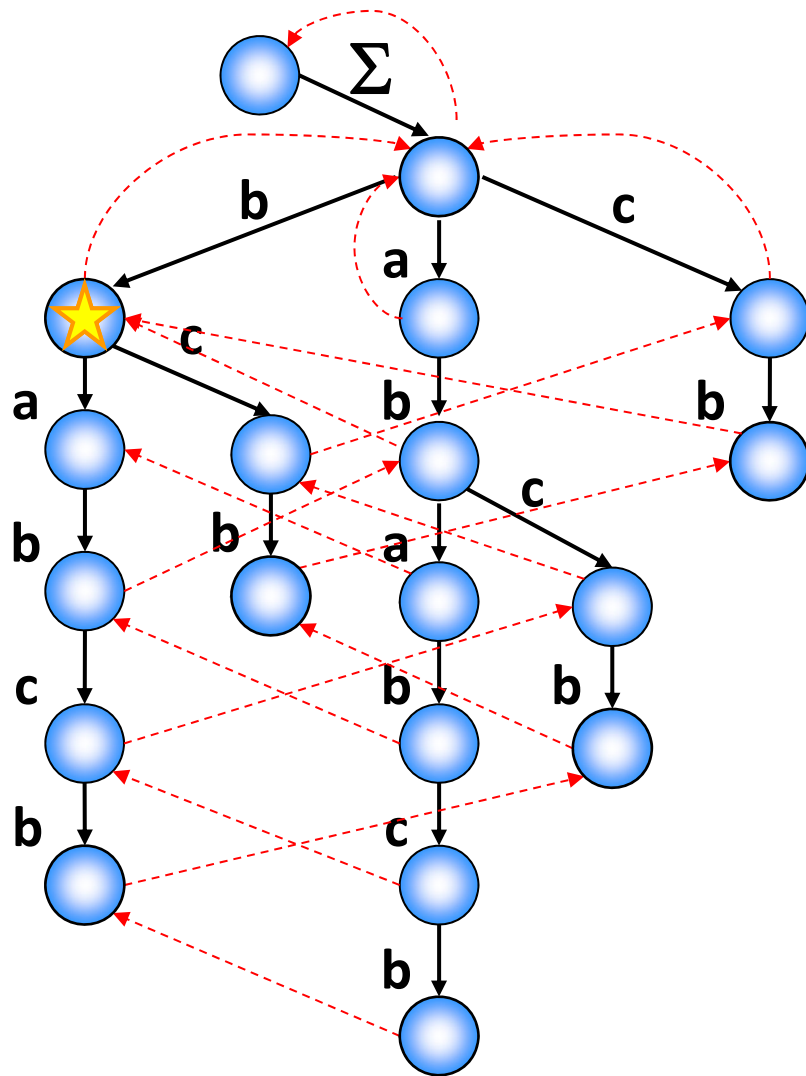
# Suffix Trie の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

# Suffix Trie の左→右オンライン構築



ababcb

$w[i]$  で迎れたら、  
最後に suffix link を追加  
してこのステップは終了。

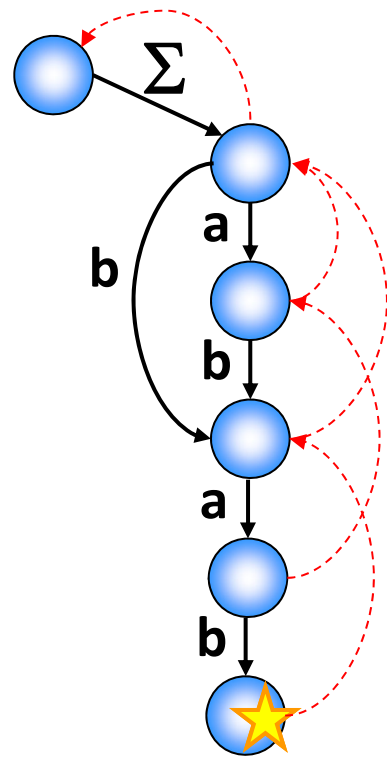
# Suffix Trie の左→右オンライン構築

## 常識7 [Ukkonen 1995]

Suffix trie を  $O(n^2 \log \sigma)$  時間・  
 $O(n^2)$  領域で左→右にオンライン構築できる。

- $n$  は文字列長,  $\sigma$  はアルファベットサイズ.
- $w[i]$  で辿れるかの判定を,  
2分探索木を使って毎回  $O(\log \sigma)$  時間で行う.

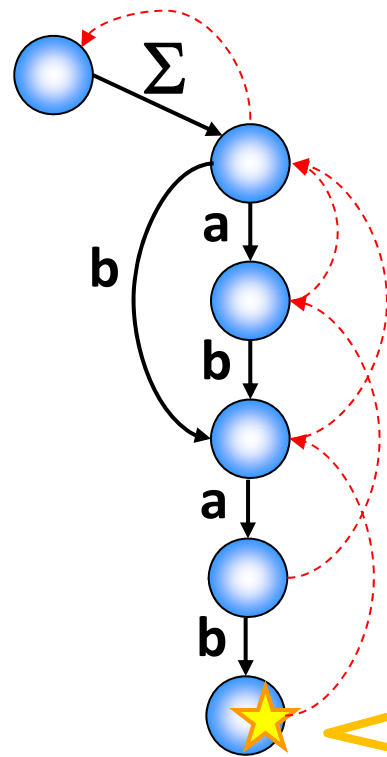
# DAWG の左→右オンライン構築



ababcb

やっと本題の1つへ！

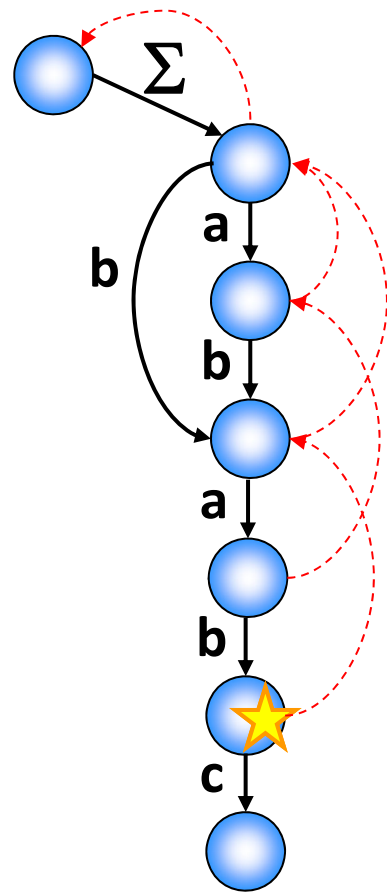
# DAWG の左→右オンライン構築



ababcb

新しい文字  $w[i]$  を読み込んだら、  
現在の sink ノードから構築を  
開始する。

# DAWG の左→右オンライン構築

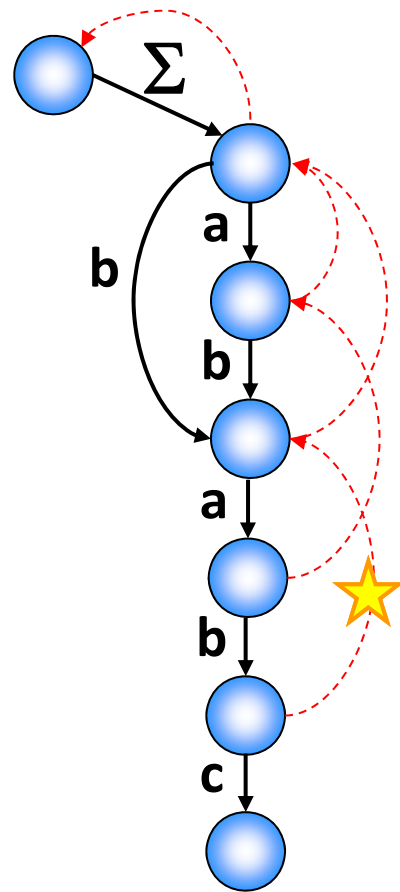


ababcb

新しい葉を複数作る代わりに、  
新しい(1つの) sink ノードを作る。

これがノードのマージに対応！

# DAWG の左→右オンライン構築

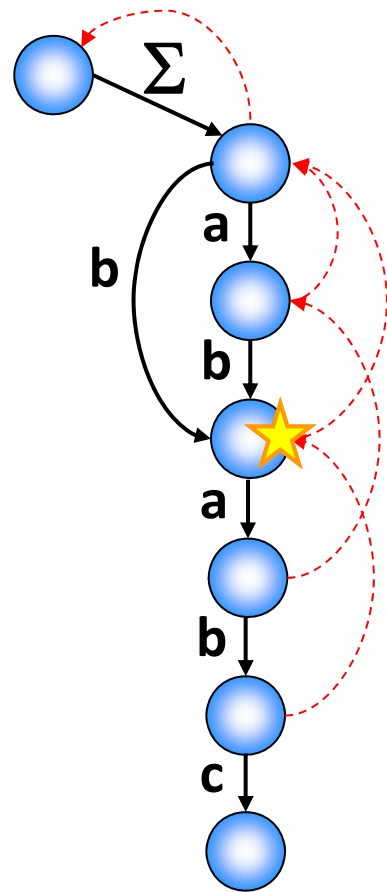


ababcb





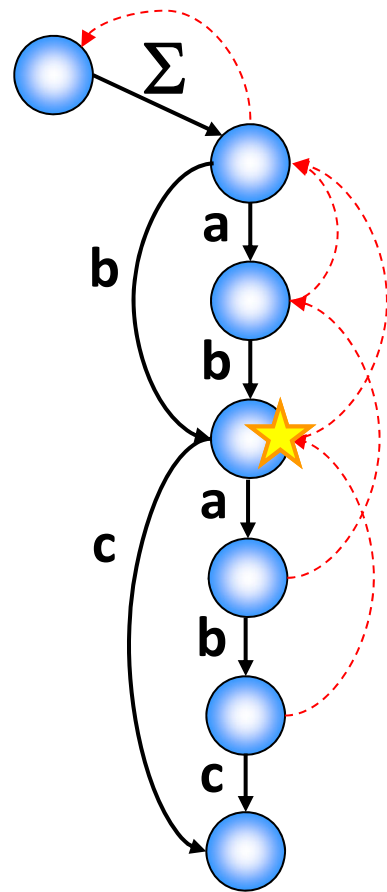
# DAWG の左→右オンライン構築



ababcb

$w[i]$  で辿れない場合は,  
 $w[i]$  でラベル付けされた  
新しい辺を sink に向かって  
作る.

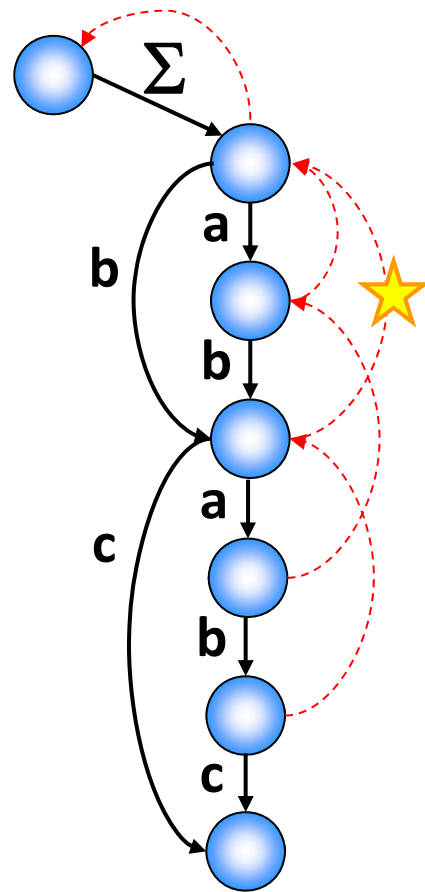
# DAWG の左→右オンライン構築



ababcb

$w[i]$  で辿れない場合は,  
 $w[i]$  でラベル付けされた  
新しい辺を sink に向かって  
作る.

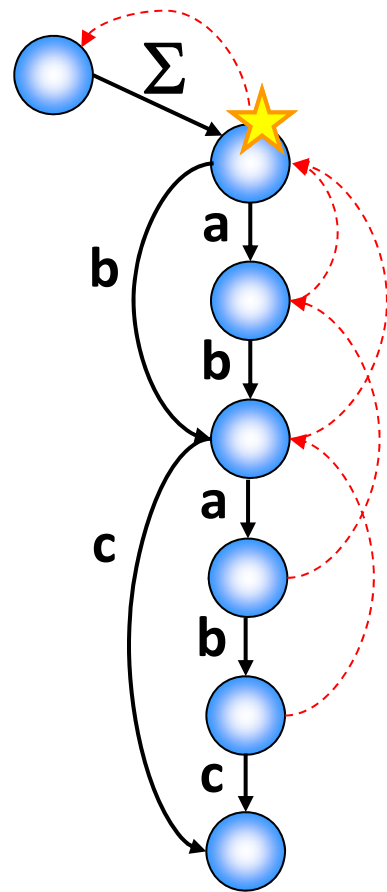
# DAWG の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

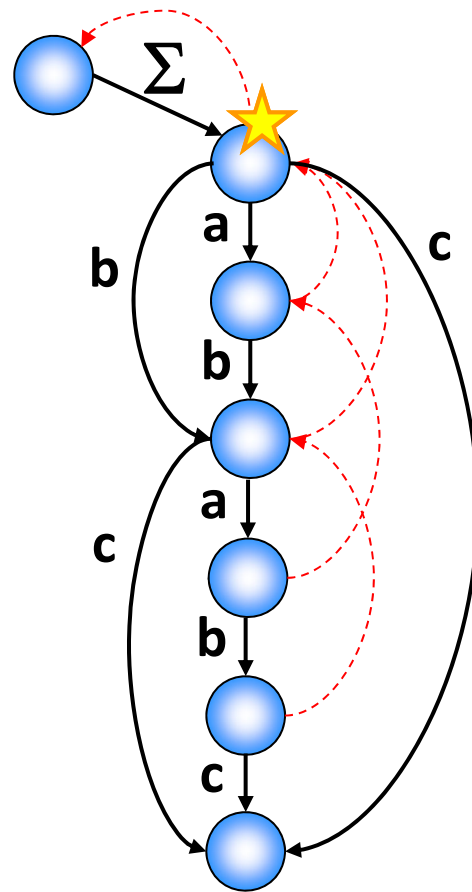
# DAWG の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

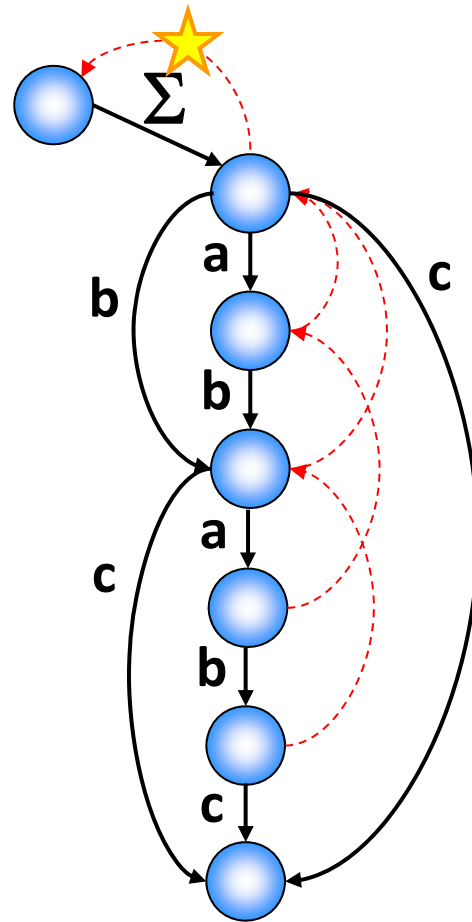
# DAWG の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

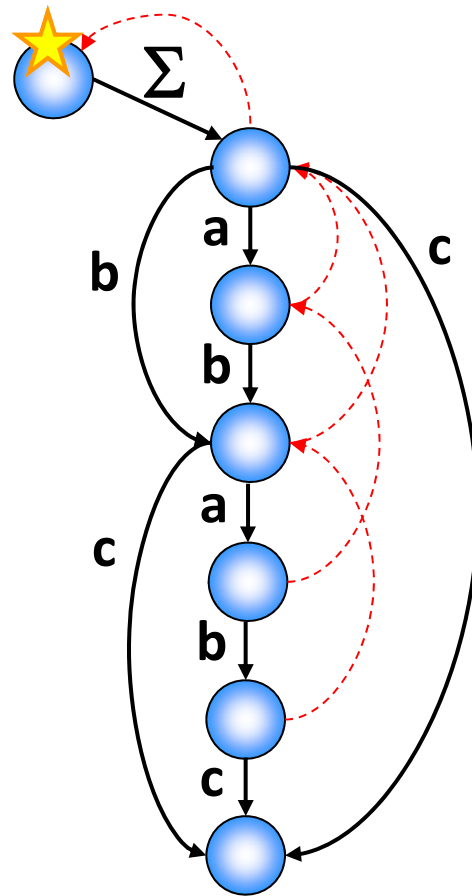
# DAWG の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

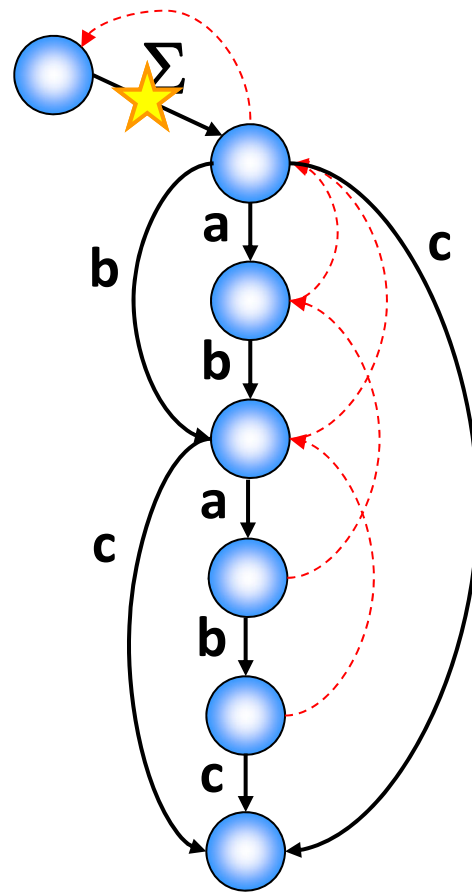
# DAWG の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

# DAWG の左→右オンライン構築

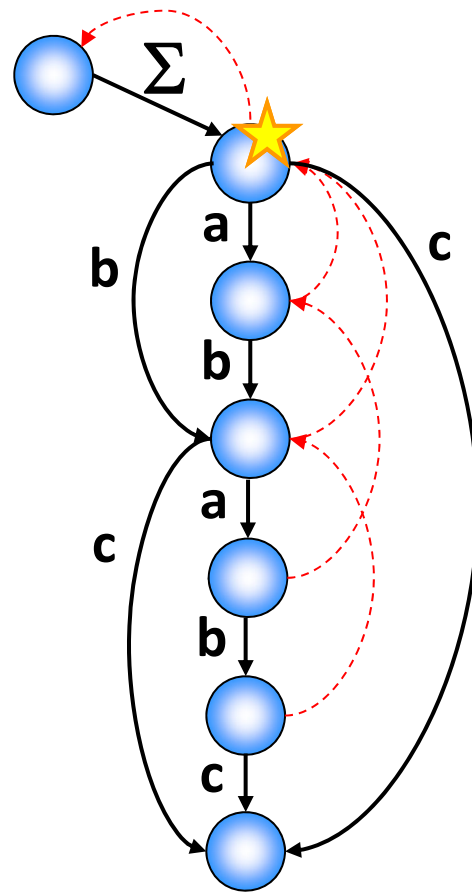


ababcb

Suffix link を辿った先で、  
同じことを繰り返す。



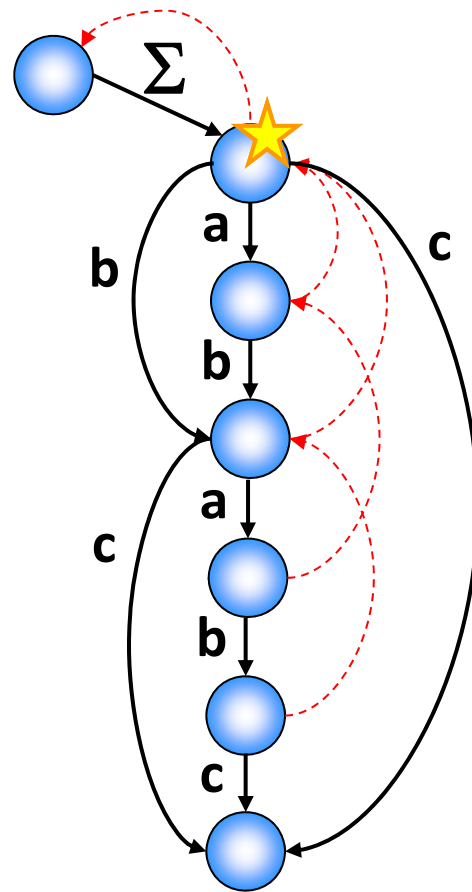
# DAWG の左→右オンライン構築



ababcb

Suffix link を辿った先で、  
同じことを繰り返す。

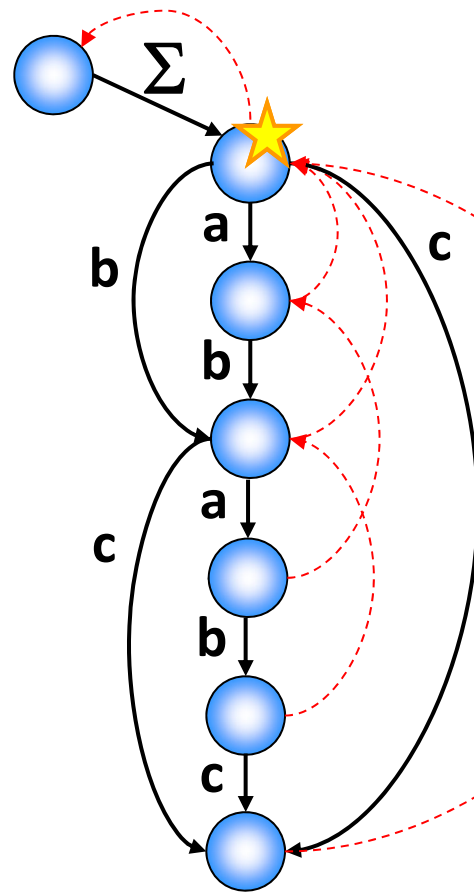
# DAWG の左→右オンライン構築



ababcb

$w[i]$  で迎れたら,  
最後に suffix link を追加  
してこのステップは終了.

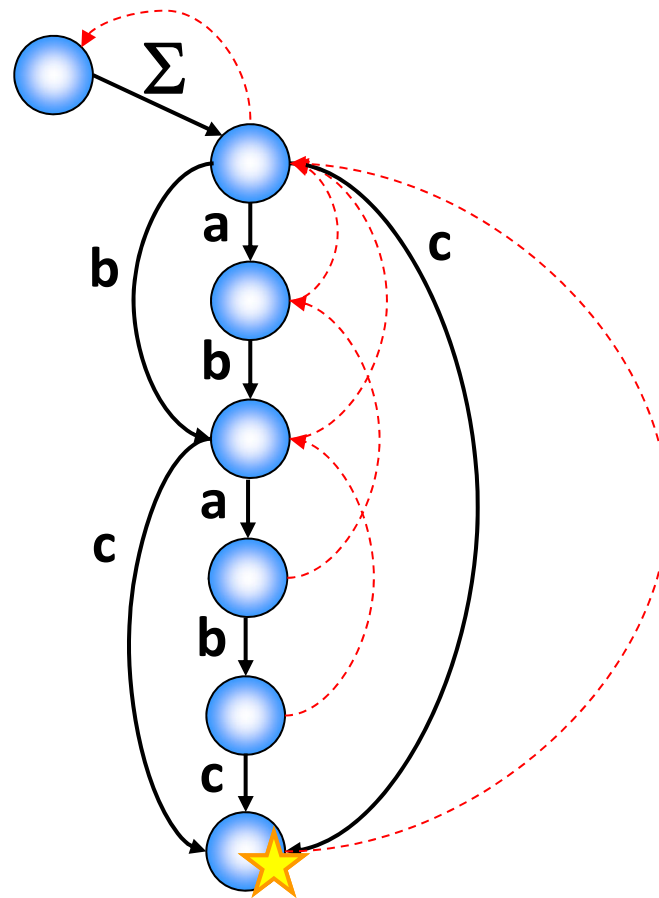
# DAWG の左→右オンライン構築



ababcb

$w[i]$  で迎れたら,  
最後に suffix link を追加  
してこのステップは終了.

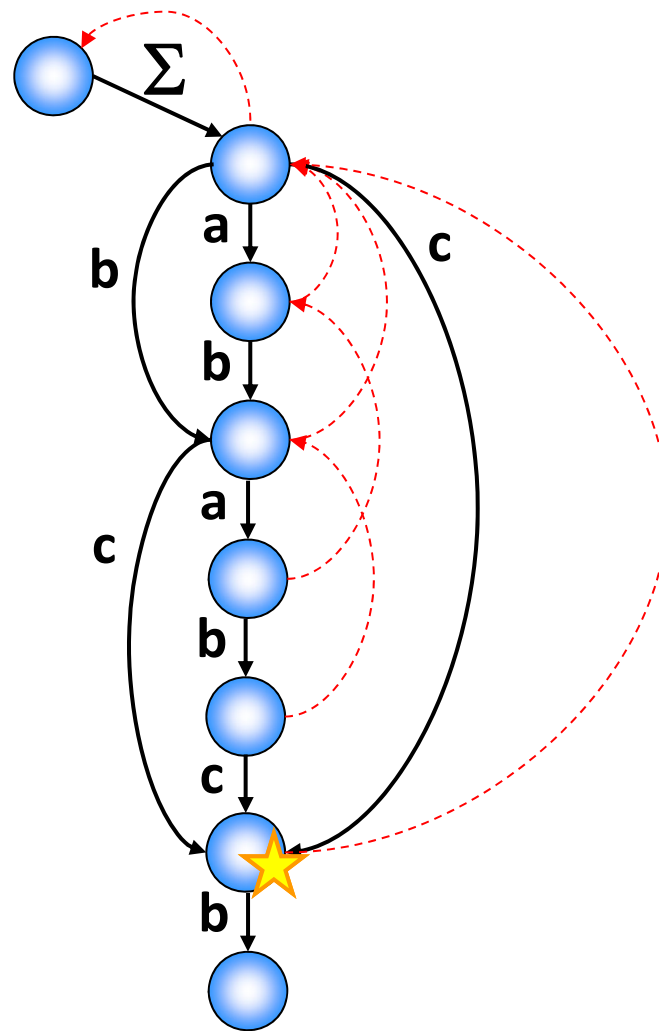
# DAWG の左→右オンライン構築



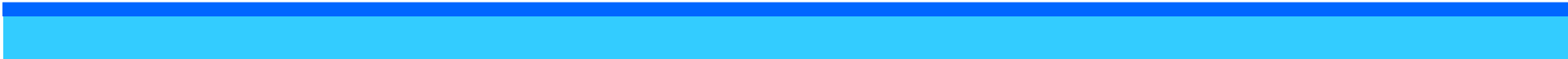
ababcb



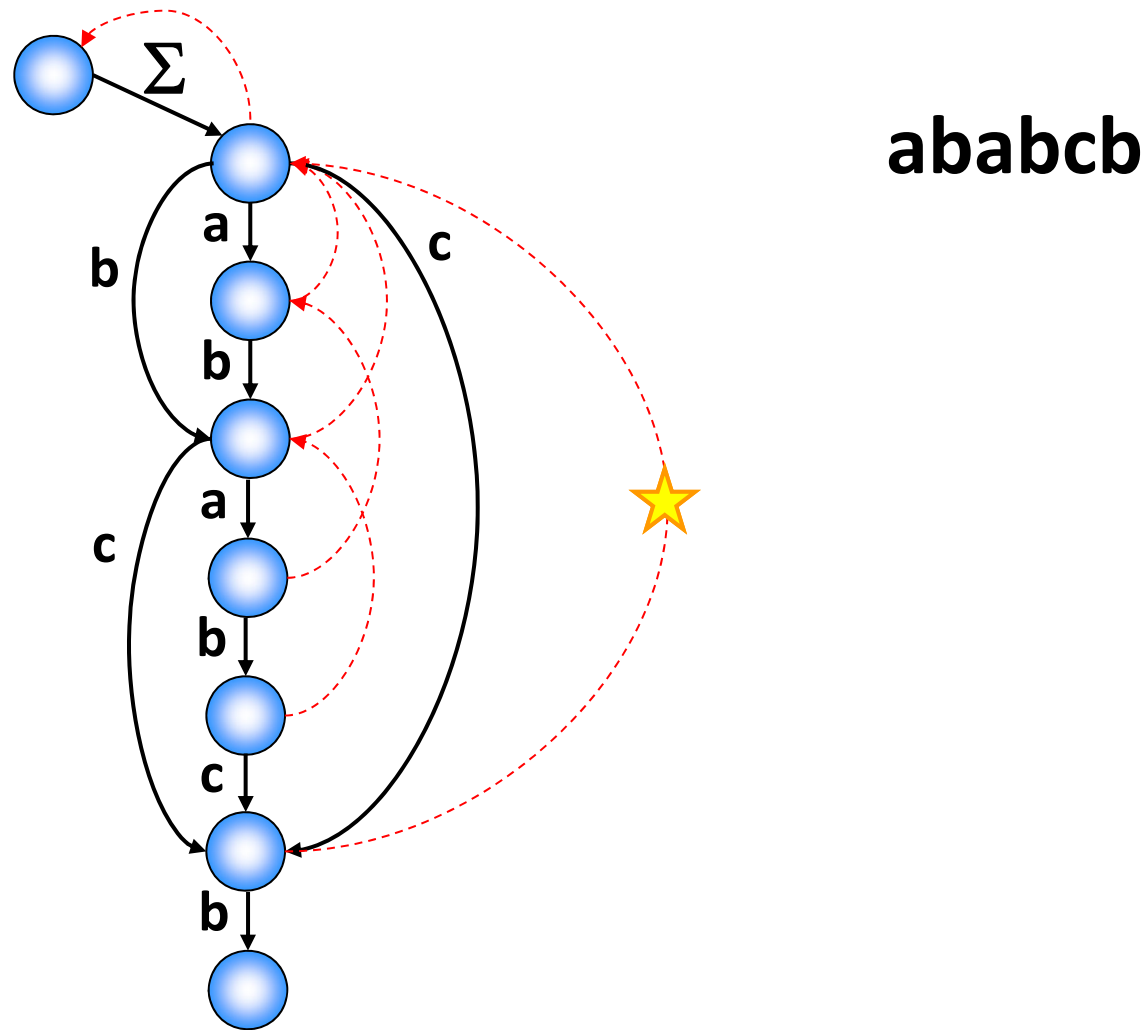
# DAWG の左→右オンライン構築



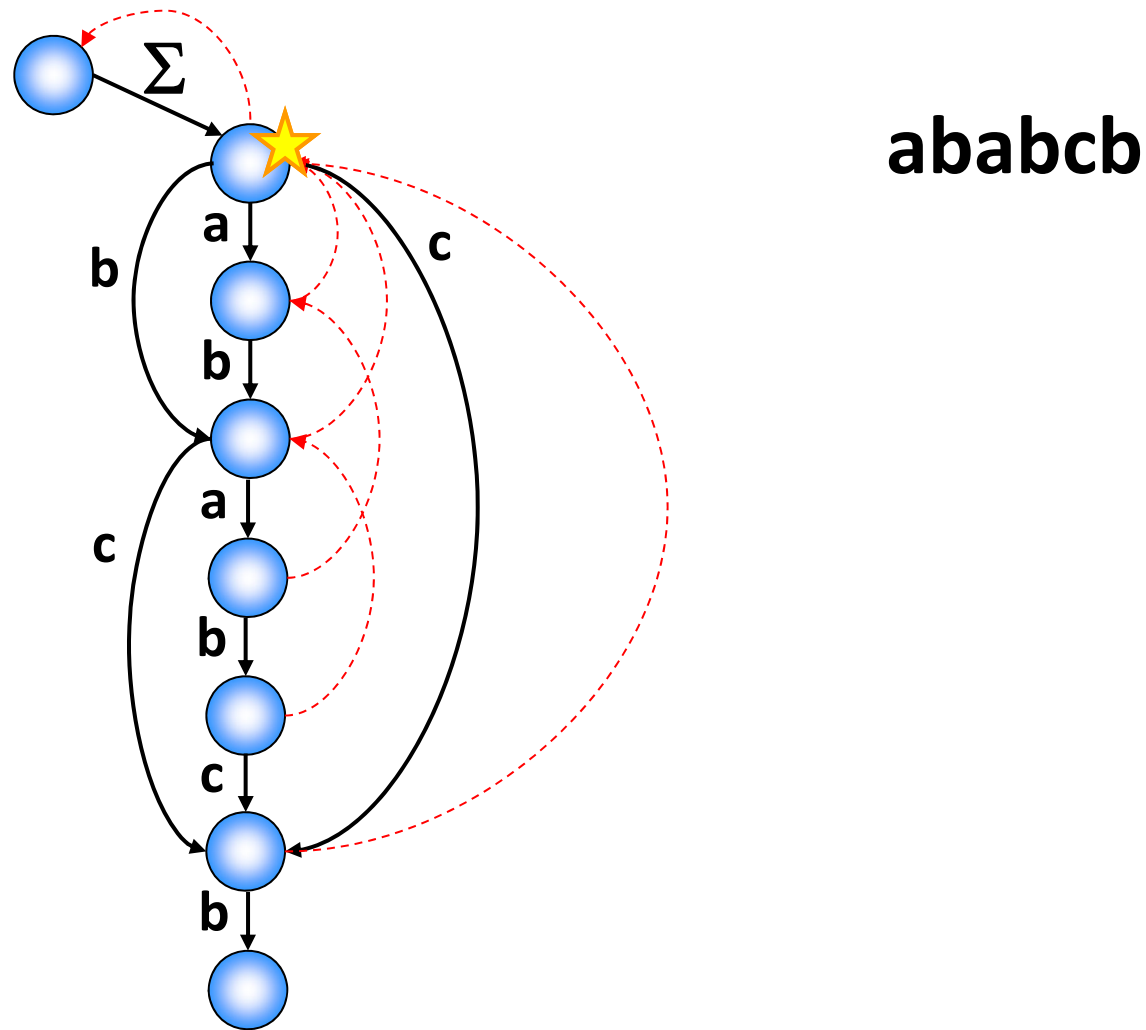
ababcb



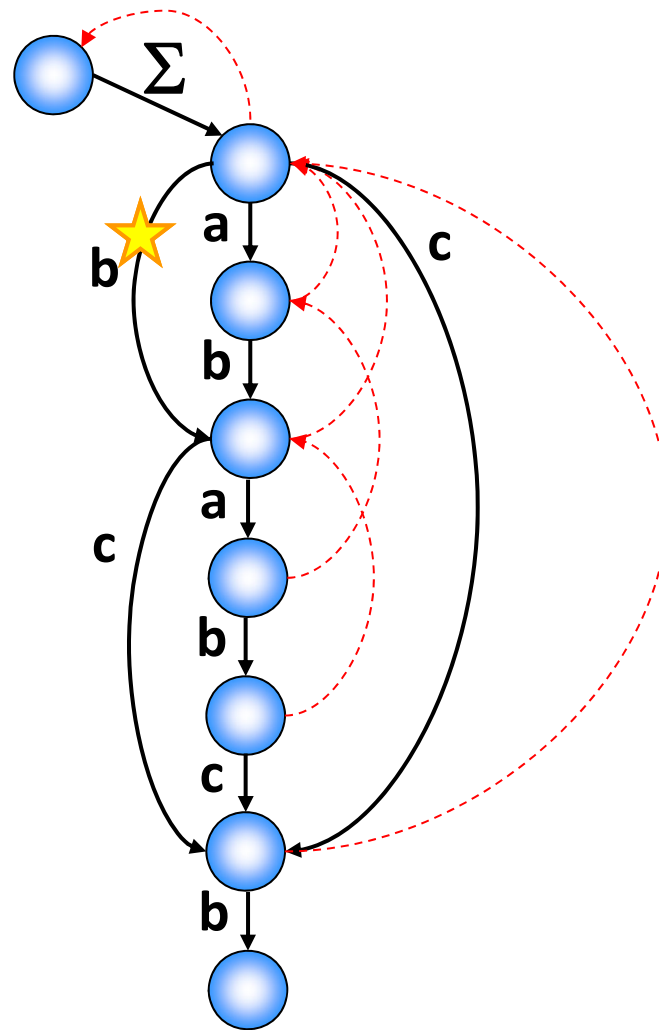
# DAWG の左→右オンライン構築



# DAWG の左→右オンライン構築



# DAWG の左→右オンライン構築

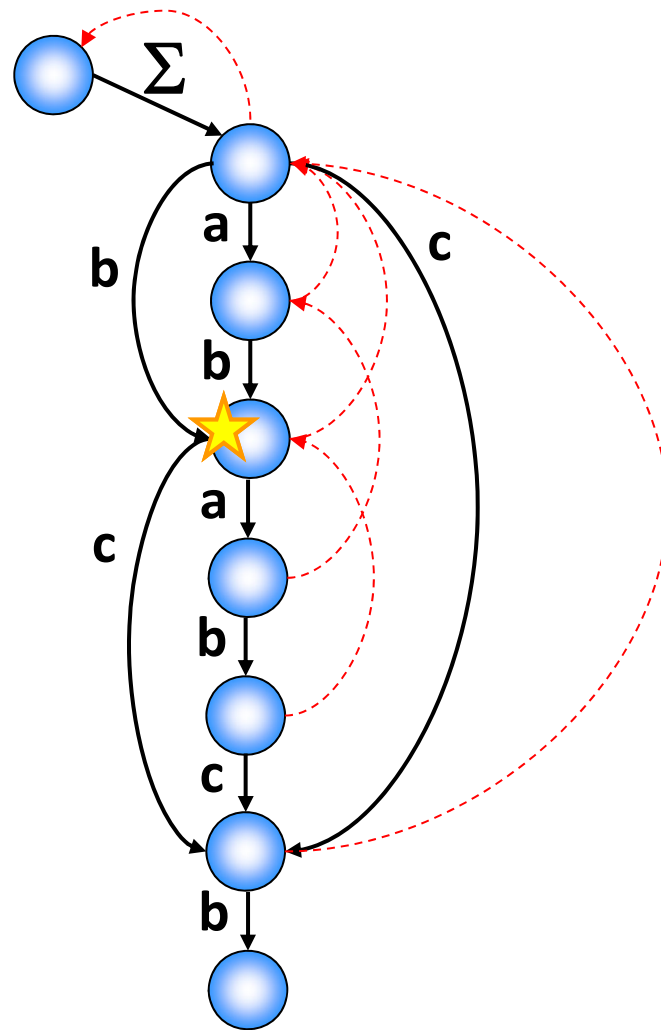


ababcb

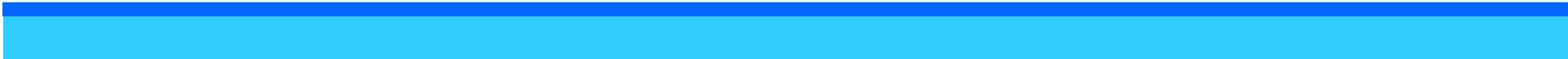




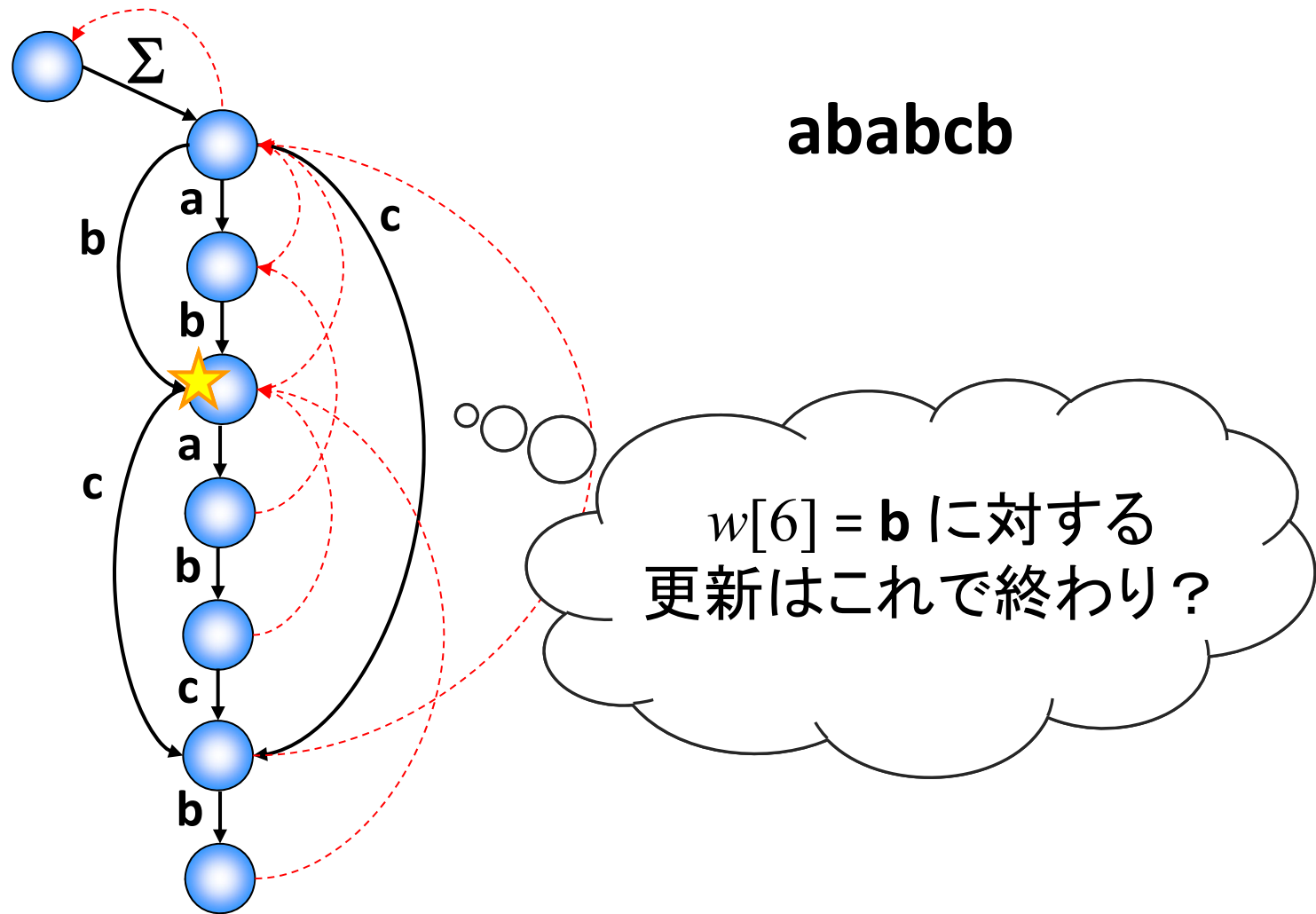
# DAWG の左→右オンライン構築



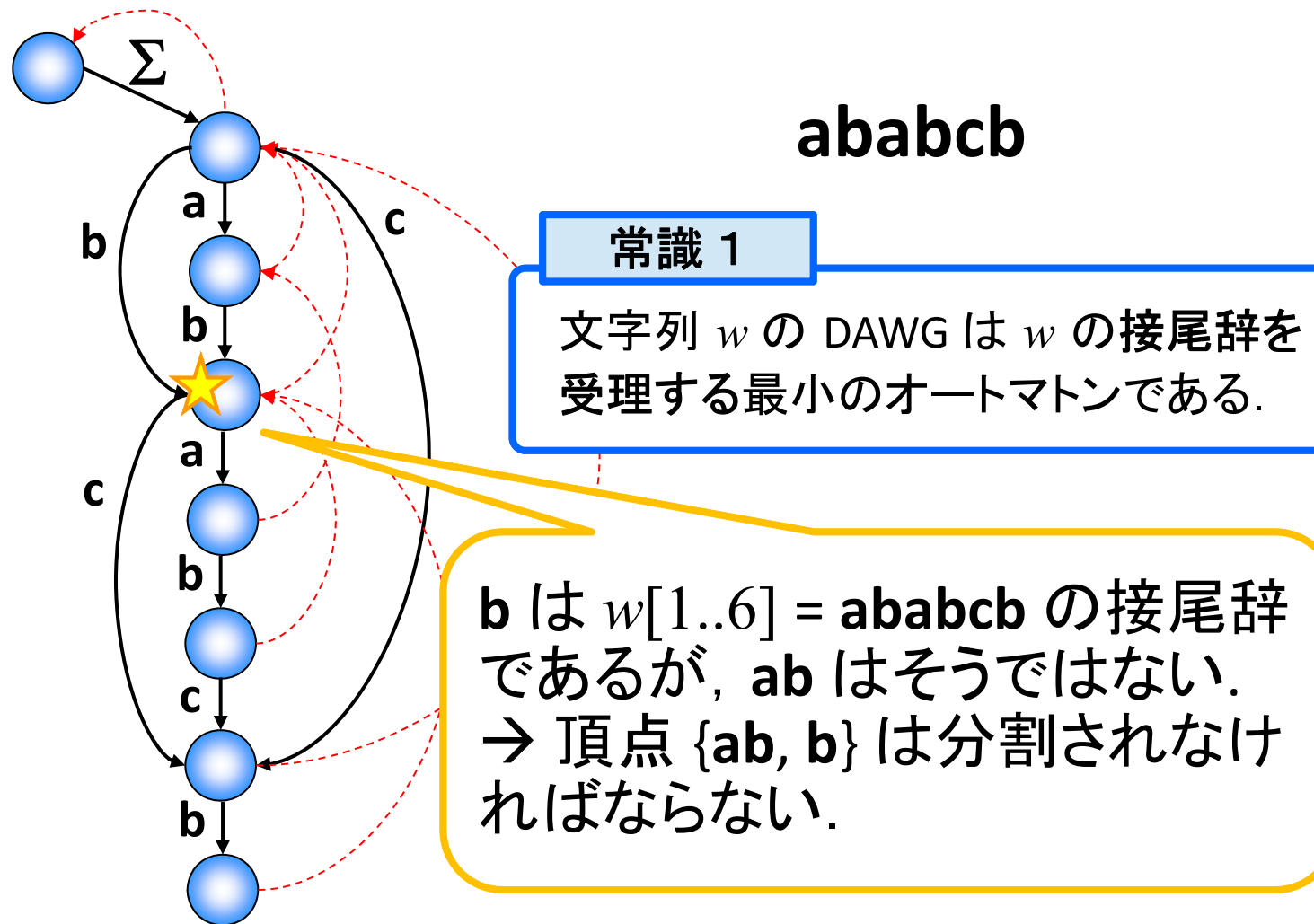
ababcb



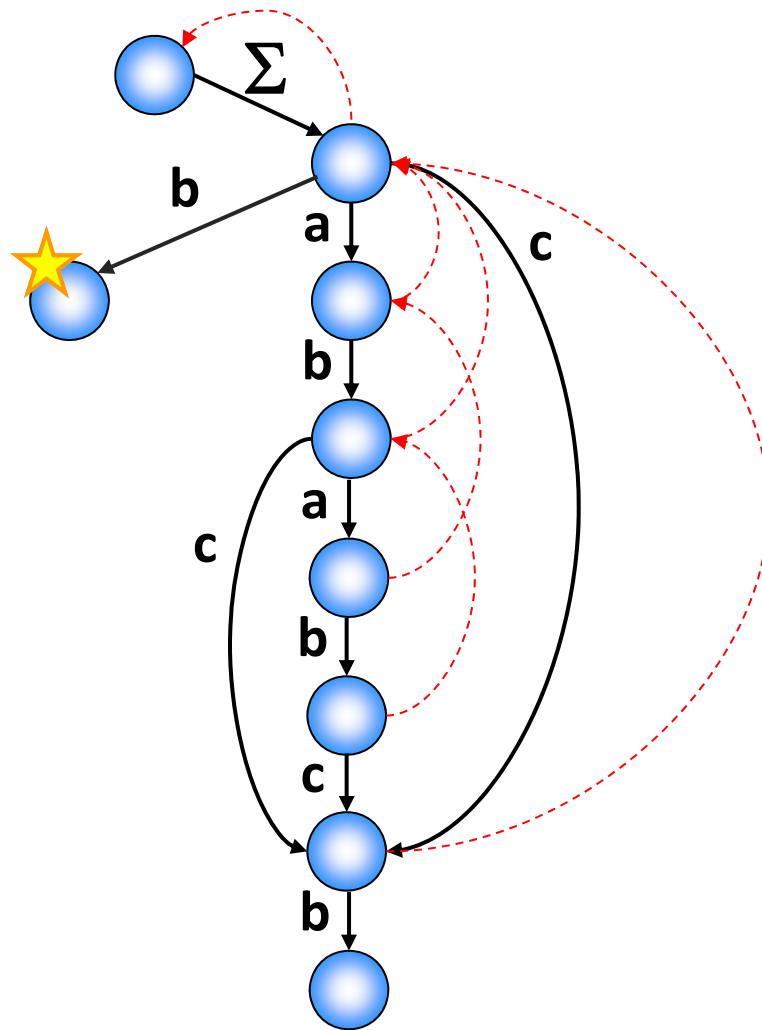
# DAWG の左→右オンライン構築



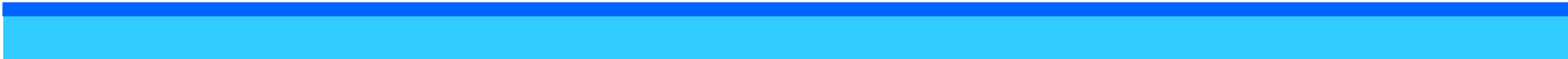
# DAWG の左→右オンライン構築



# DAWG の左→右オンライン構築

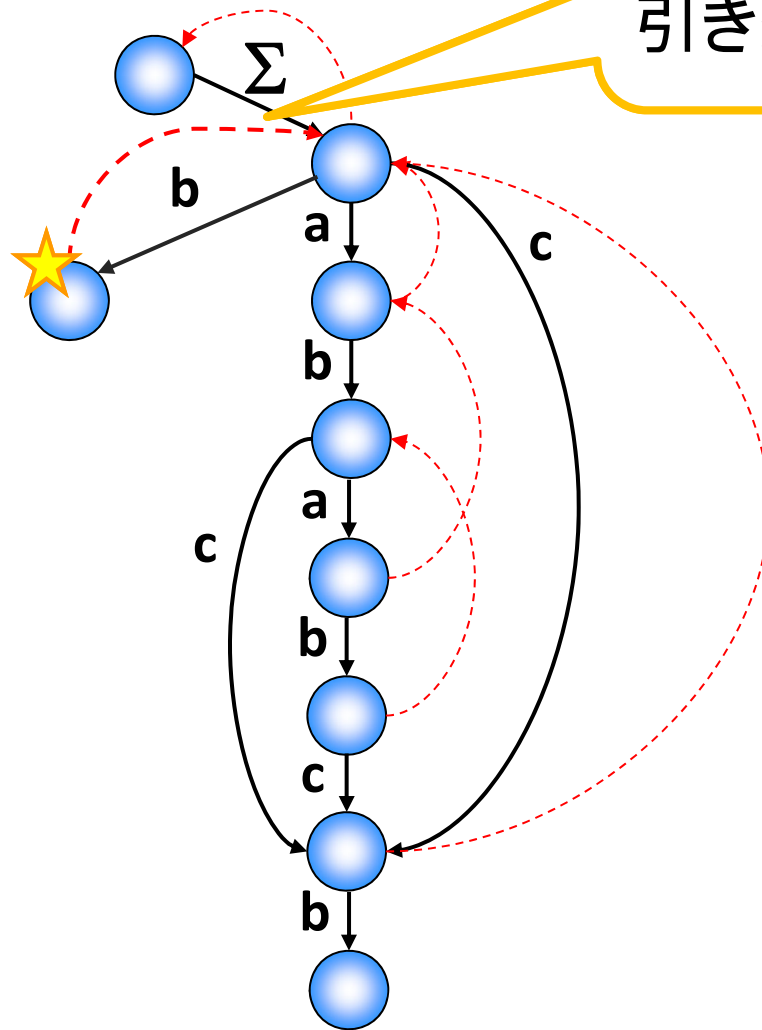


**ababcb**

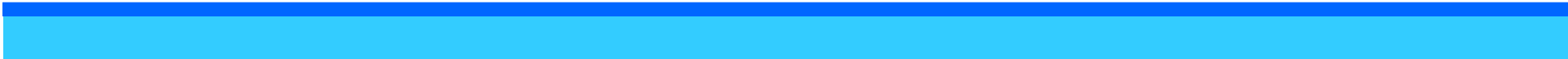


# DAWG の左→右への構築

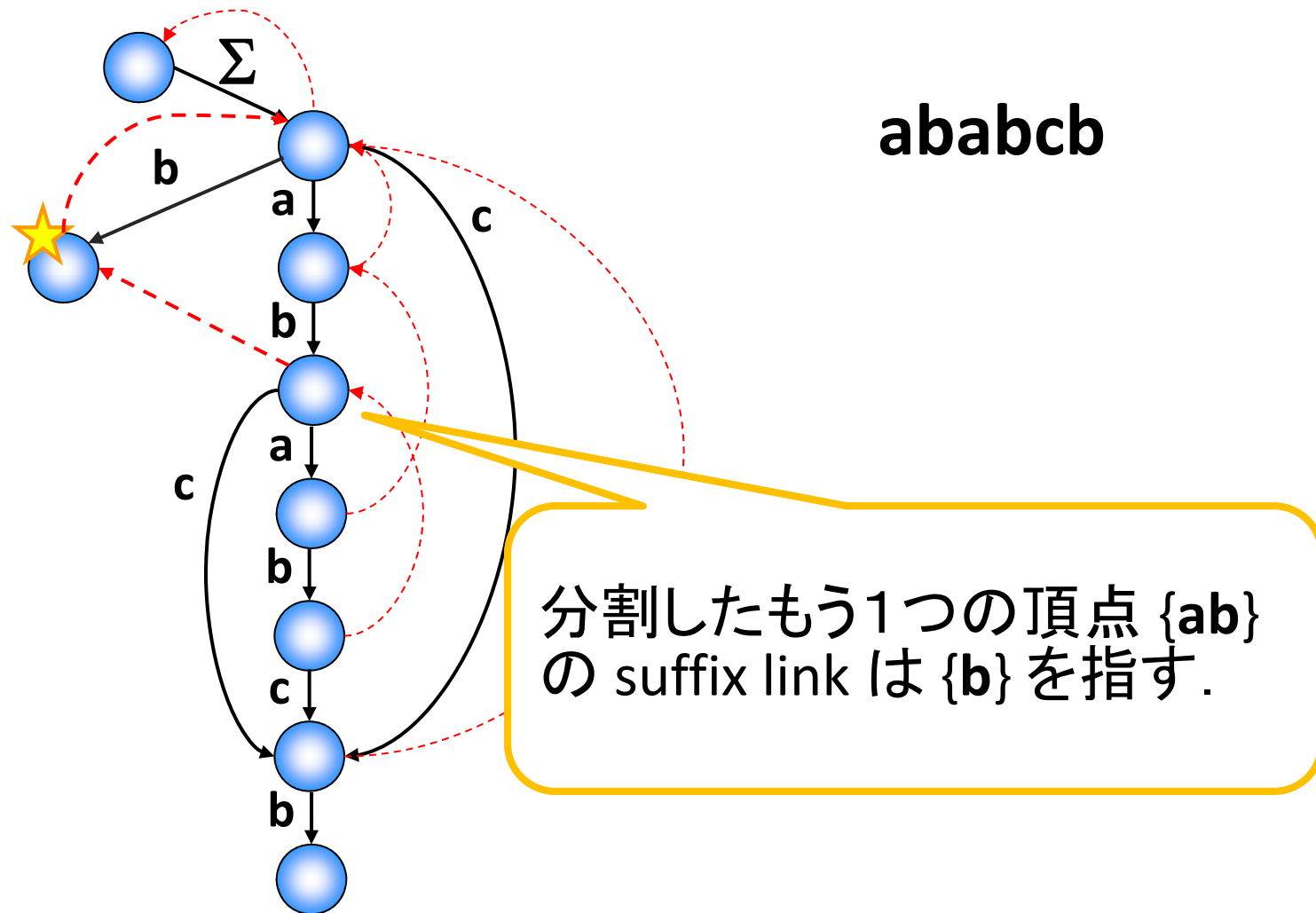
古い頂点 {ab, b} から出ていた suffix link は新しい頂点 {b} に引き継がれる.



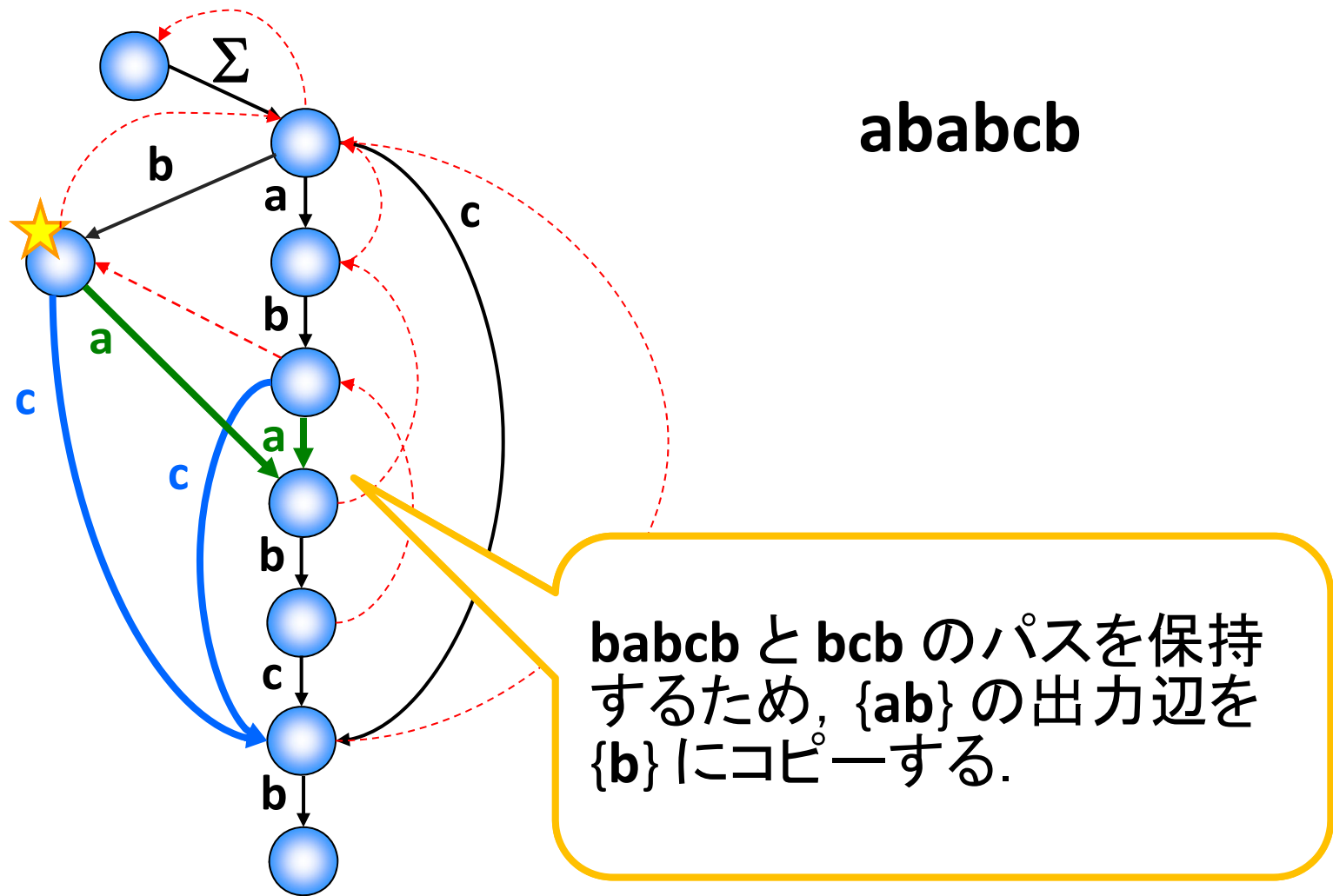
ababcb



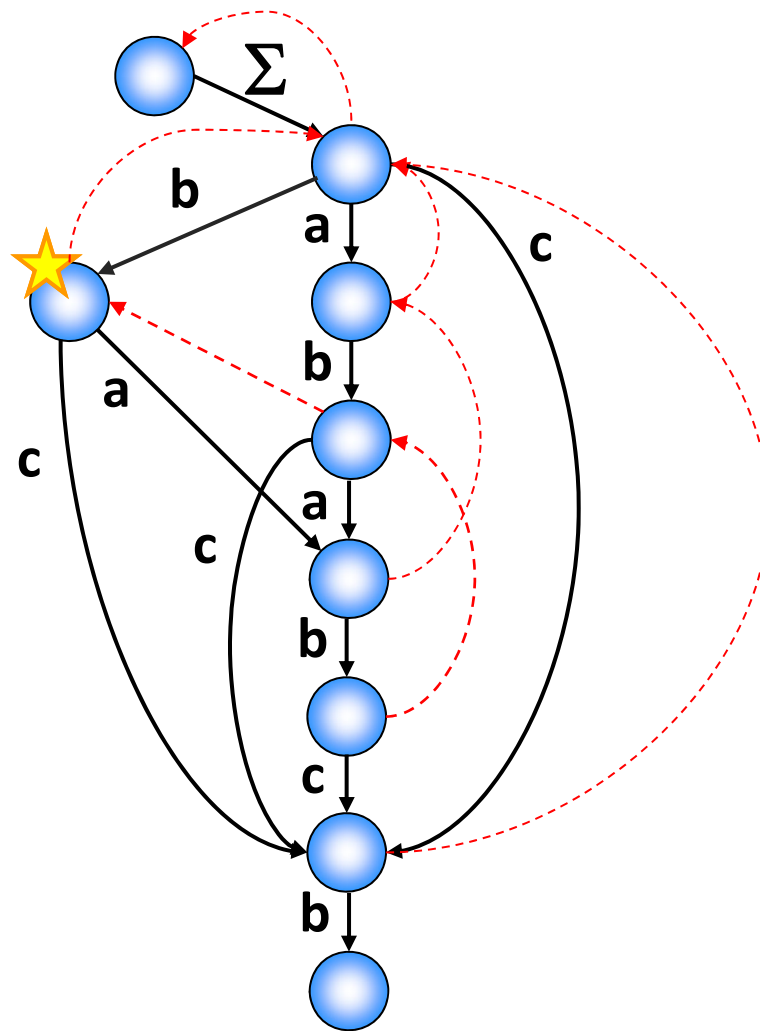
# DAWG の左→右オンライン構築



# DAWG の左→右オンライン構築



# DAWG の左→右オンライン構築

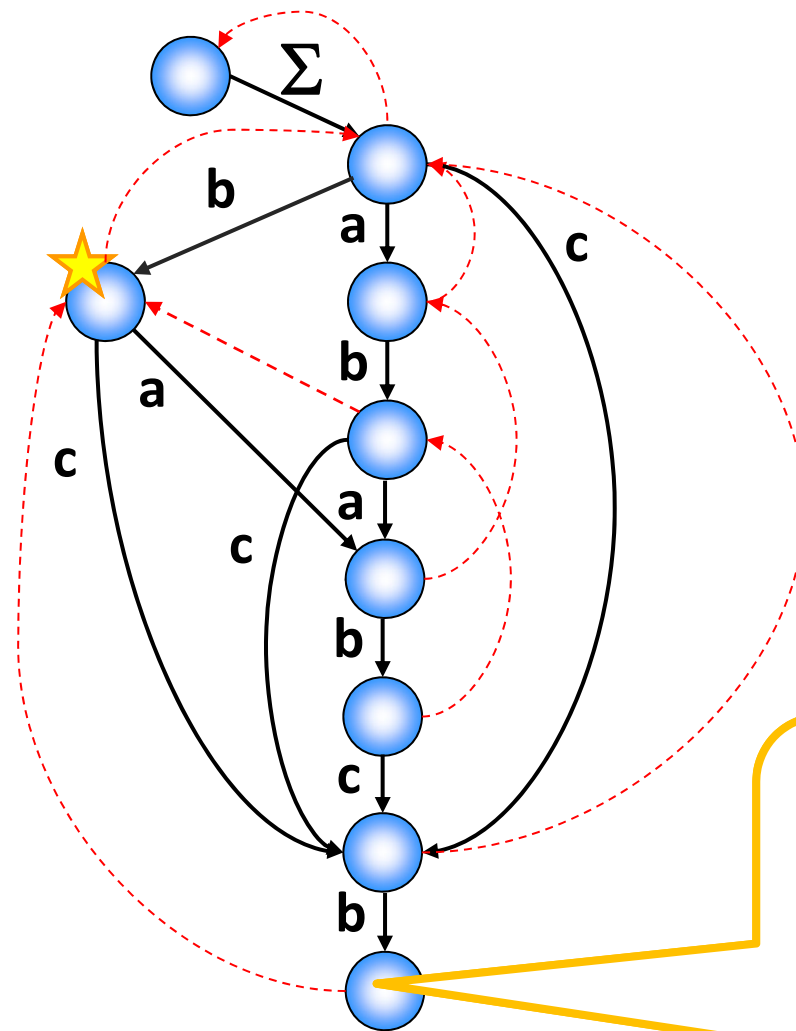


ababcb





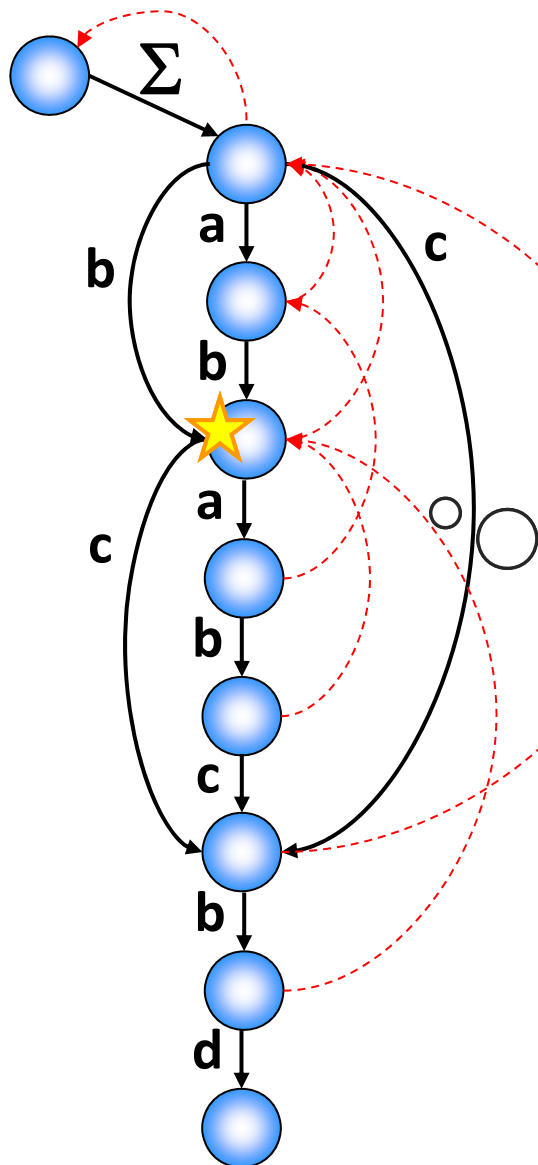
# DAWG の左→右オンライン構築



ababcb

最後に sink の suffix link  
を追加して, ababcb の  
DAWG が完成!

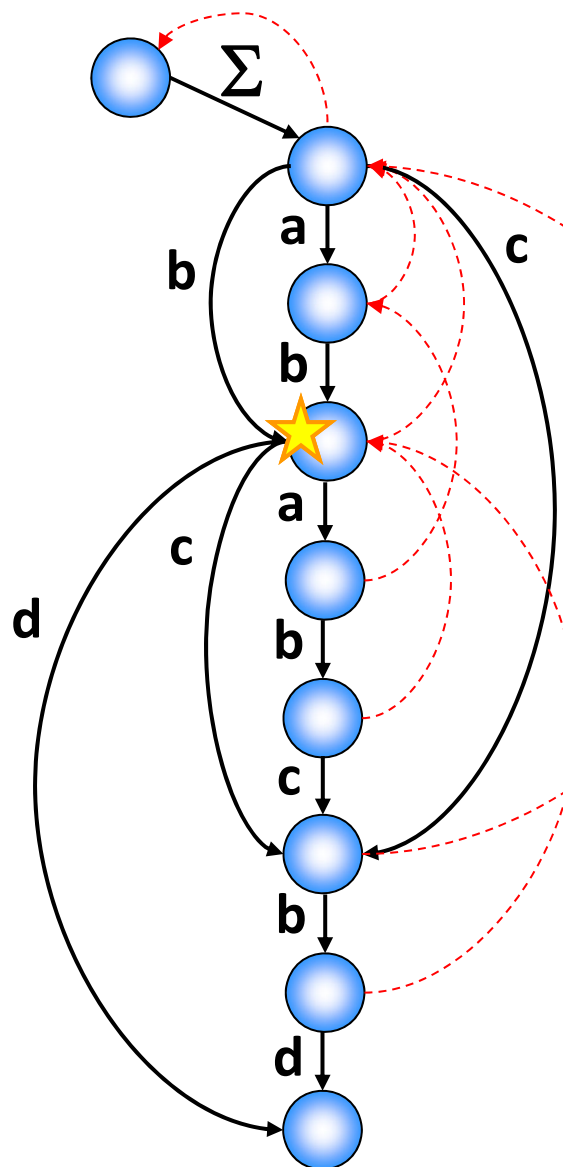
# なぜノード分割が必要か？



ababc**d**

もし {ab, b} を分割してい  
なかったら, 次の文字が  
来たときに困る.

# なぜノード分割が必要か？



ababc**d**

このグラフには abd のパスがあるが、abd は ababc**d** の部分文字列ではない！

※ ちなみに、このようにノードを分割せずにいい加減に作ったグラフを Suffix Oracle という。  
→ false positive があるがフィルタリングには使える。

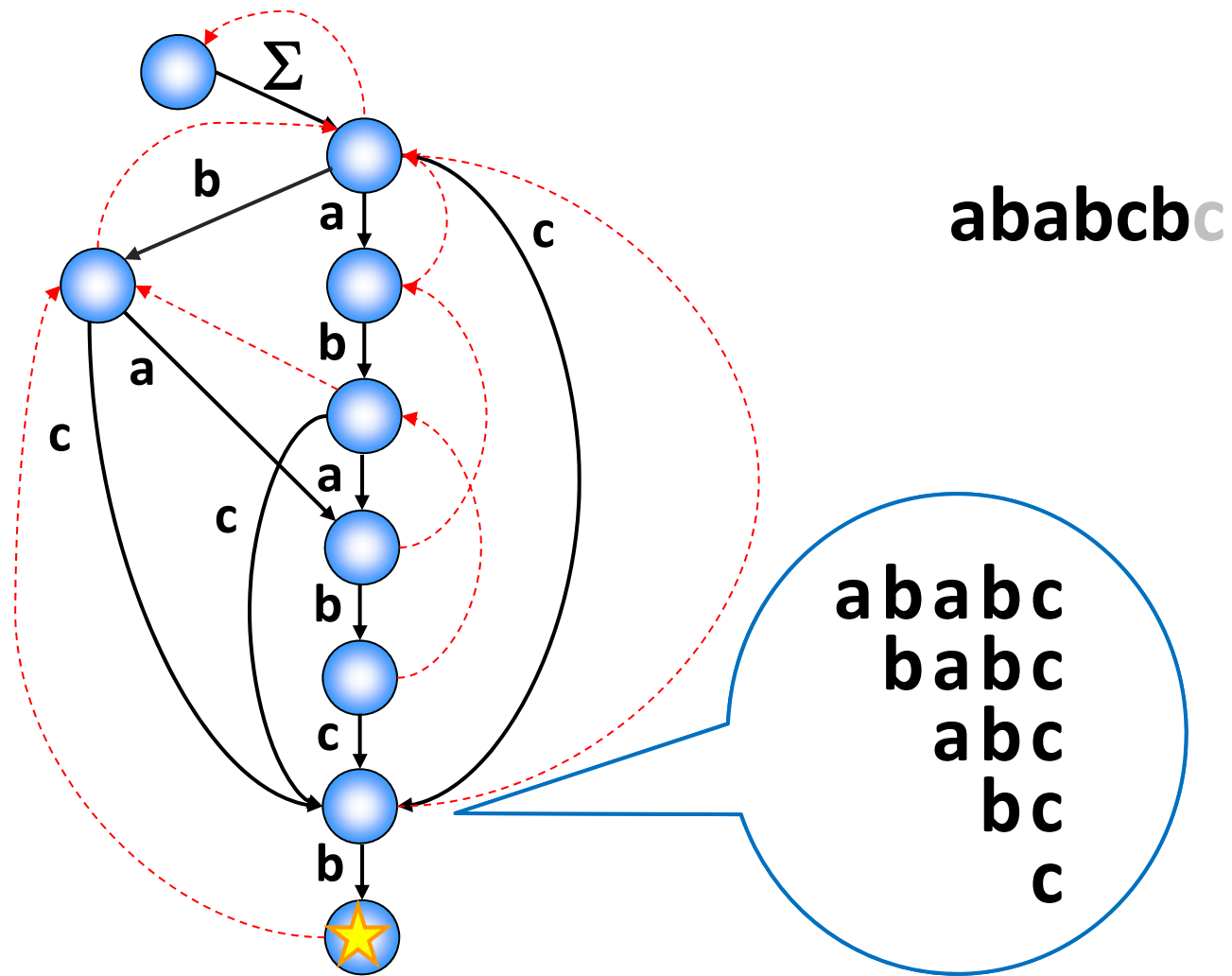
# DAWG の左→右オンライン構築

ちょっと非常識 1 [Blumer et al. 1985]

DAWG を  $O(n \log \sigma)$  時間・ $O(n)$  領域で  
左→右にオンライン構築できる.

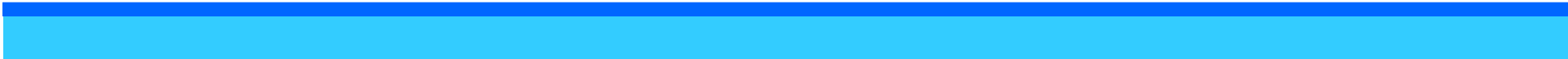
- これまで説明した内容に限れば, 必要な作業はすべて頂点と辺の数に比例している.
- 実際には, ノードの分割のときに, 「辺の差し替え」作業が複数回起きる. しかし, この回数は文字列長  $n$  で均せる.

# 辺の差し替え

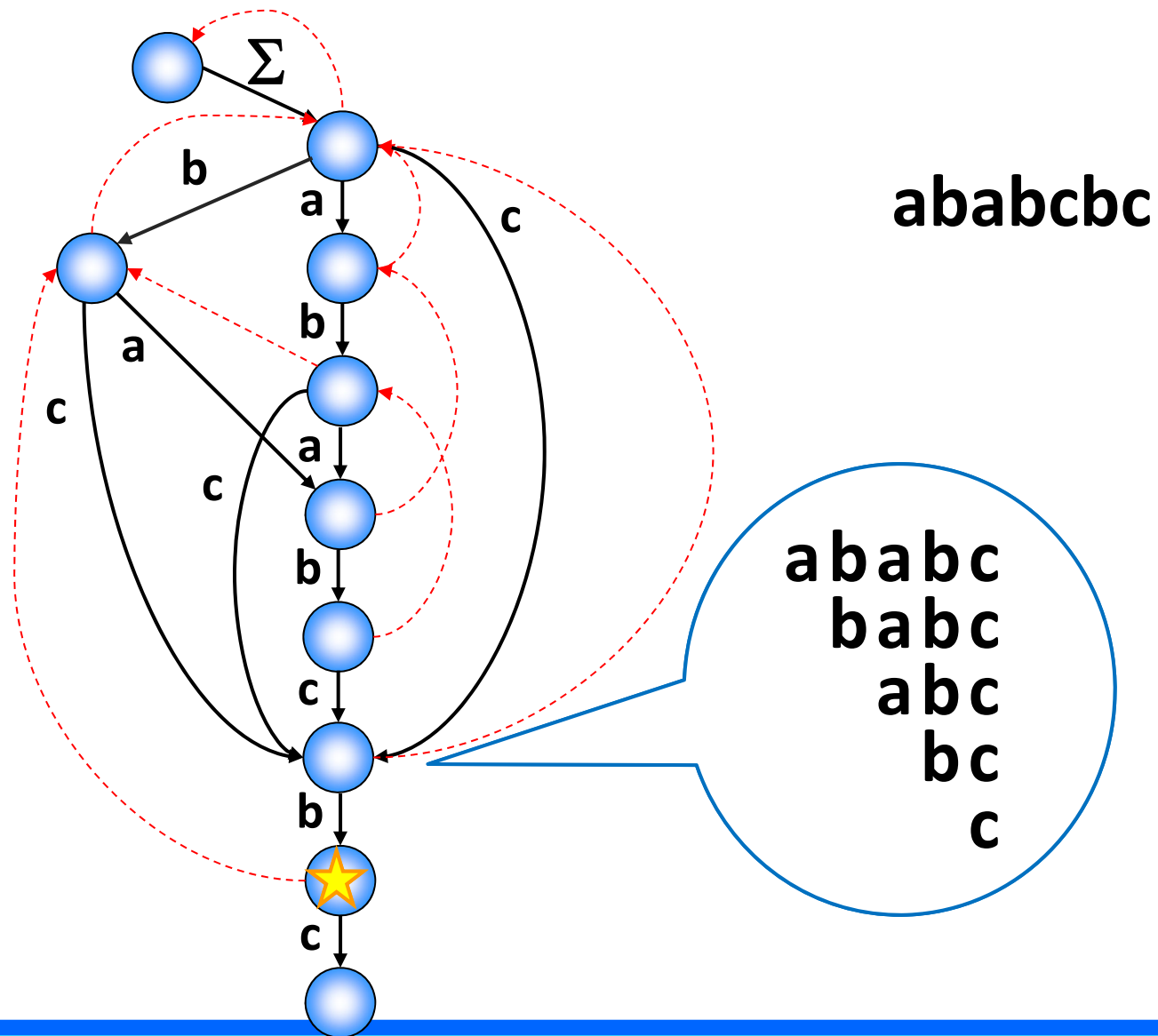


ababcabc

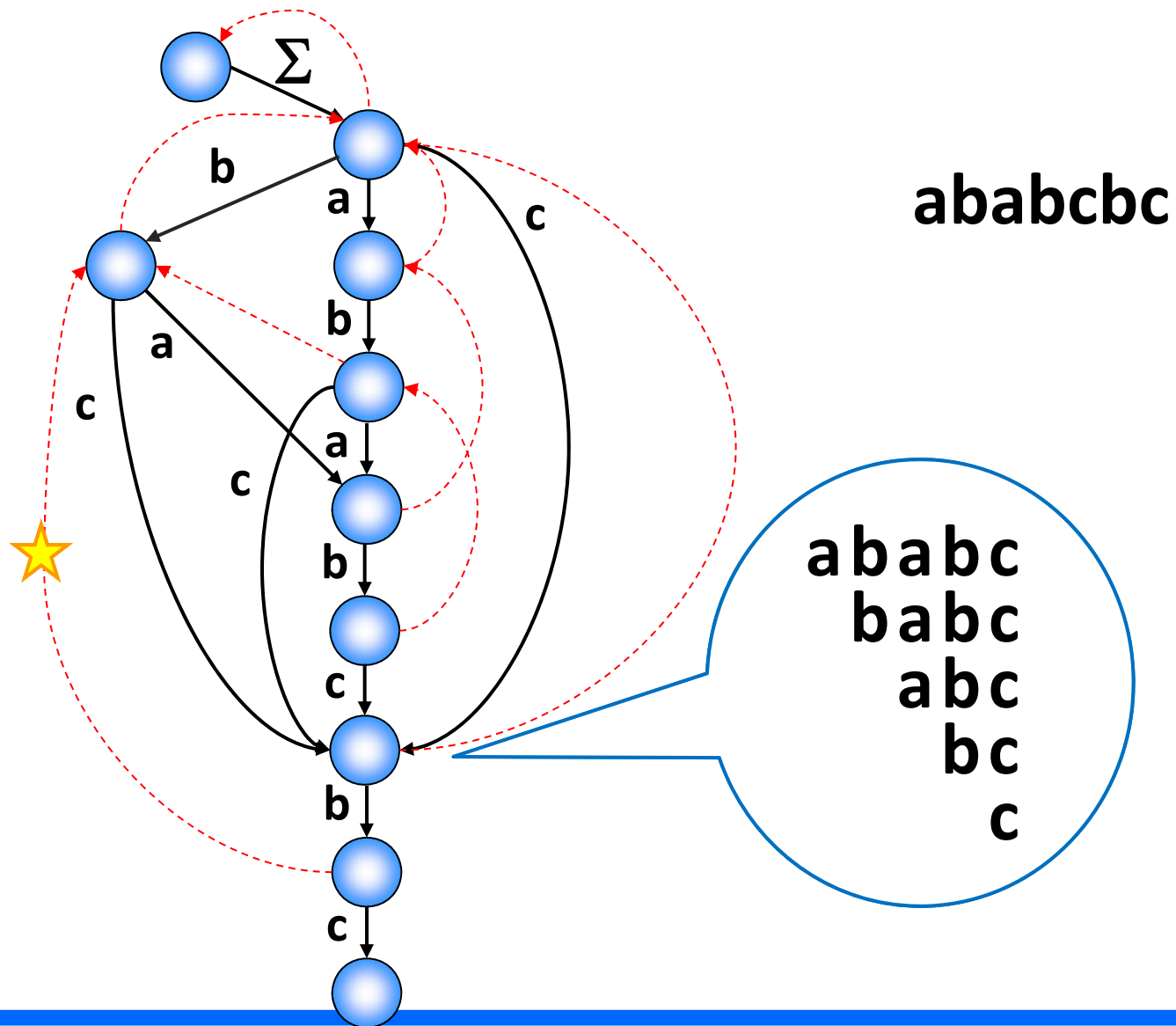
ababc  
babc  
abc  
bc  
c



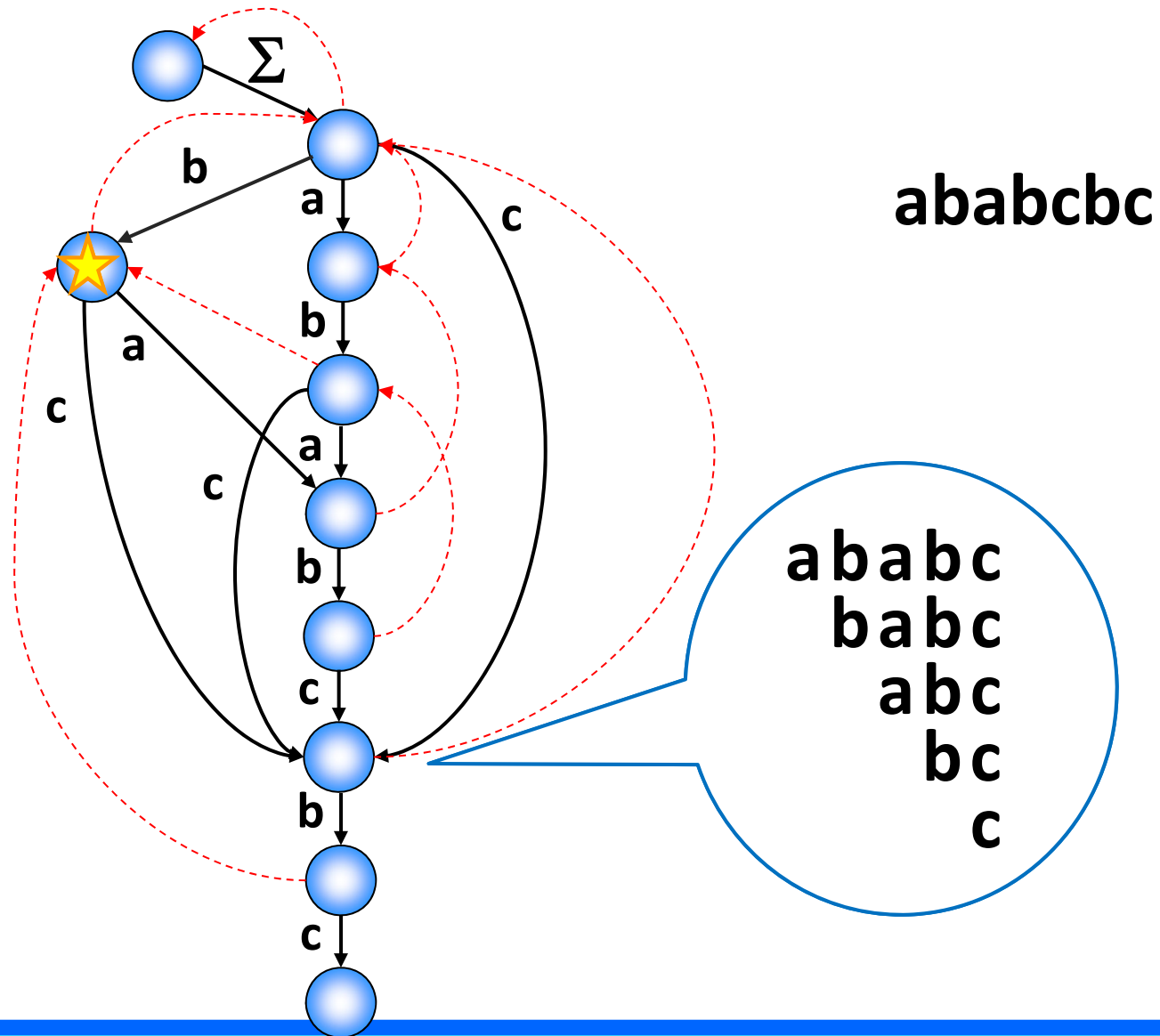
# 辺の差し替え



# 辺の差し替え



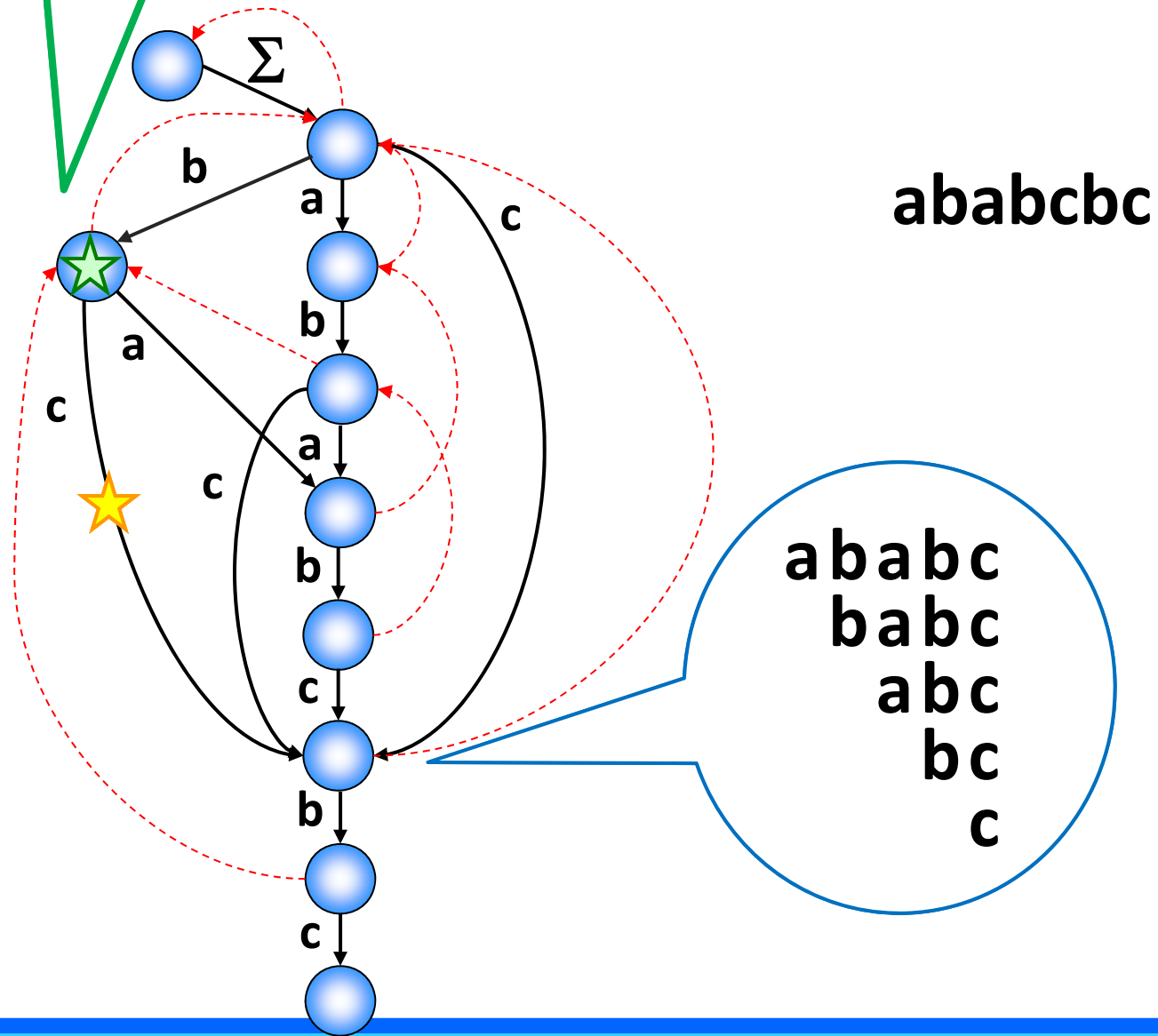
# 辺の差し替え



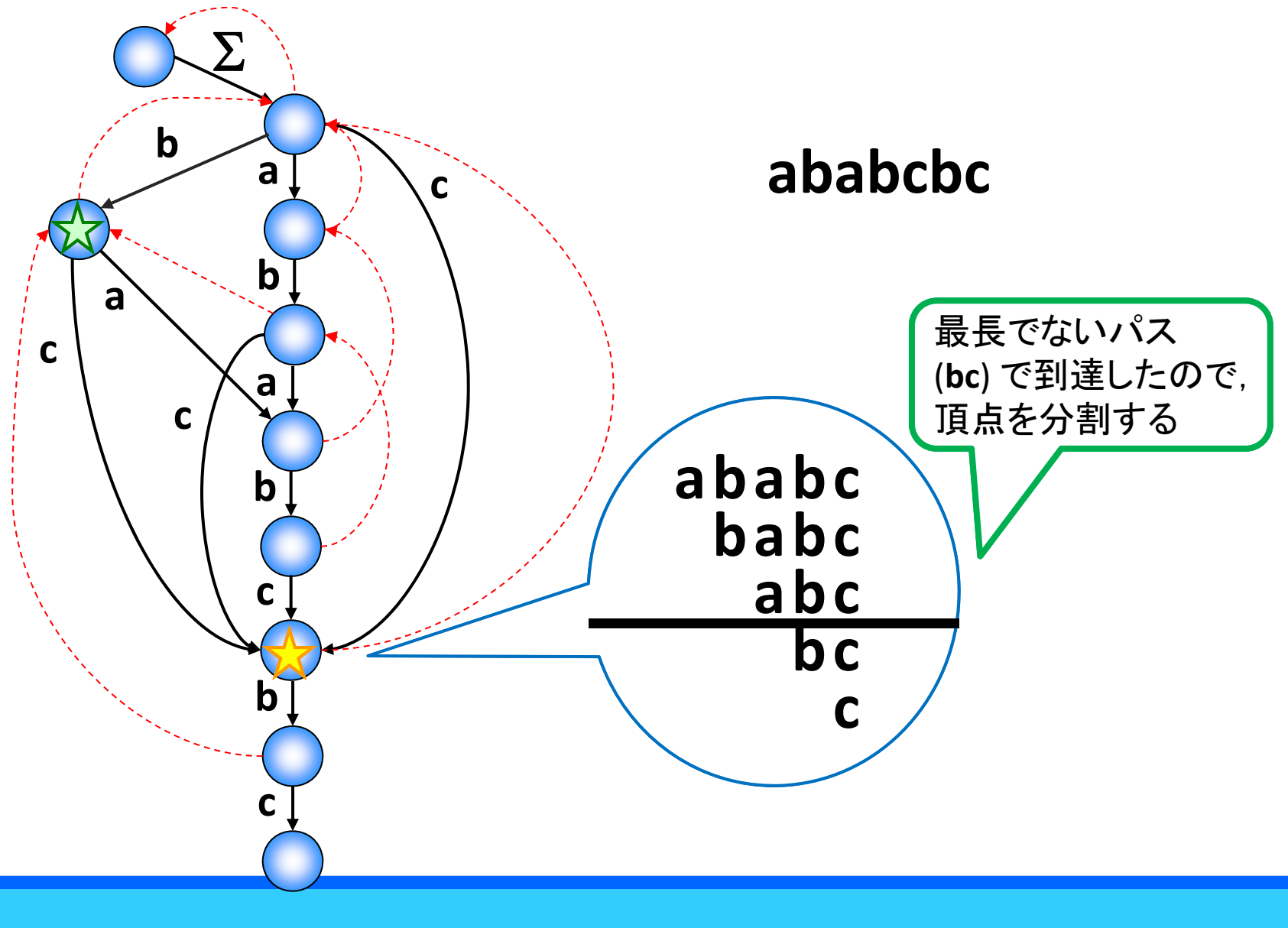


便利のために  
この場所を覚えておく

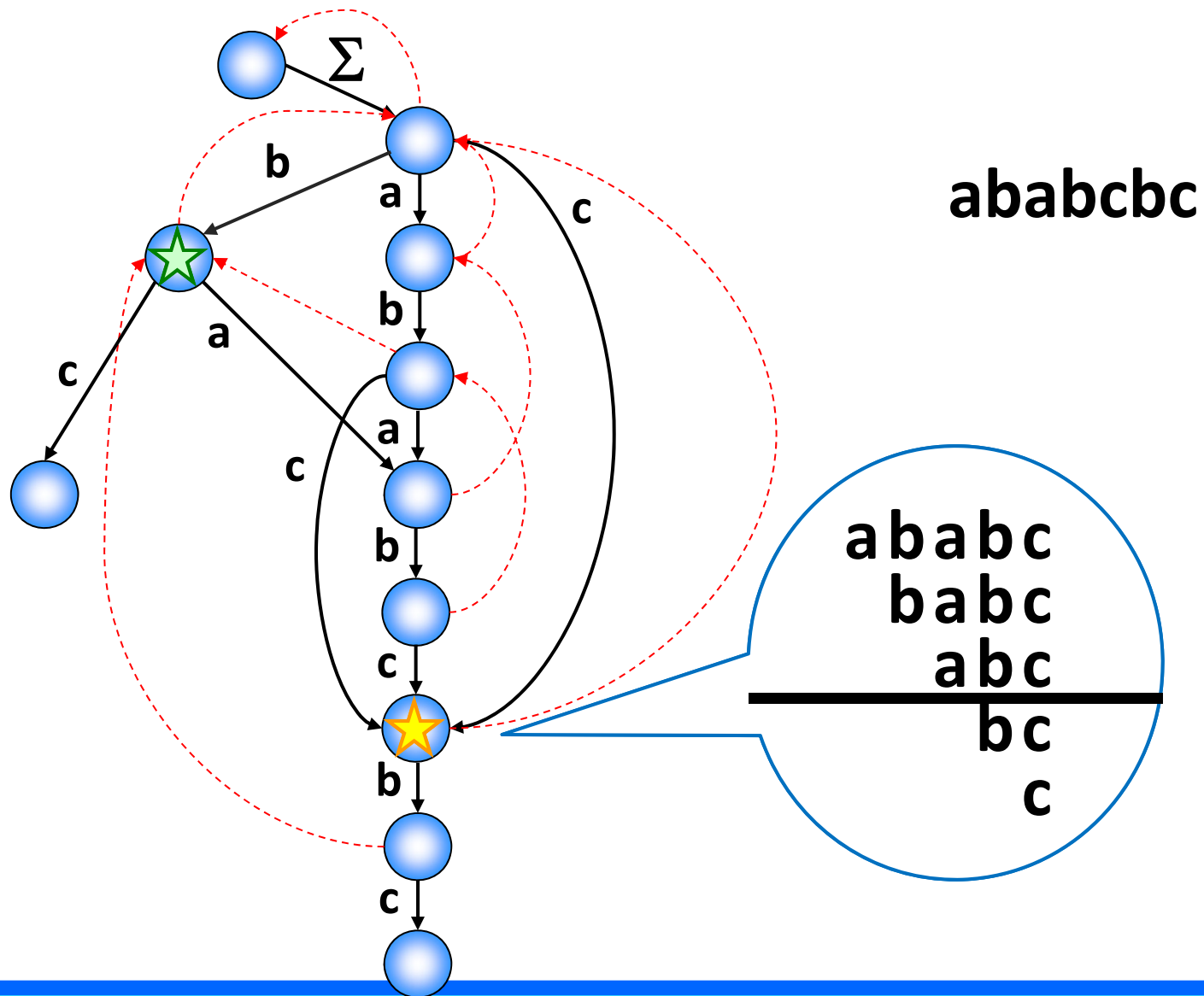
# 差し替え



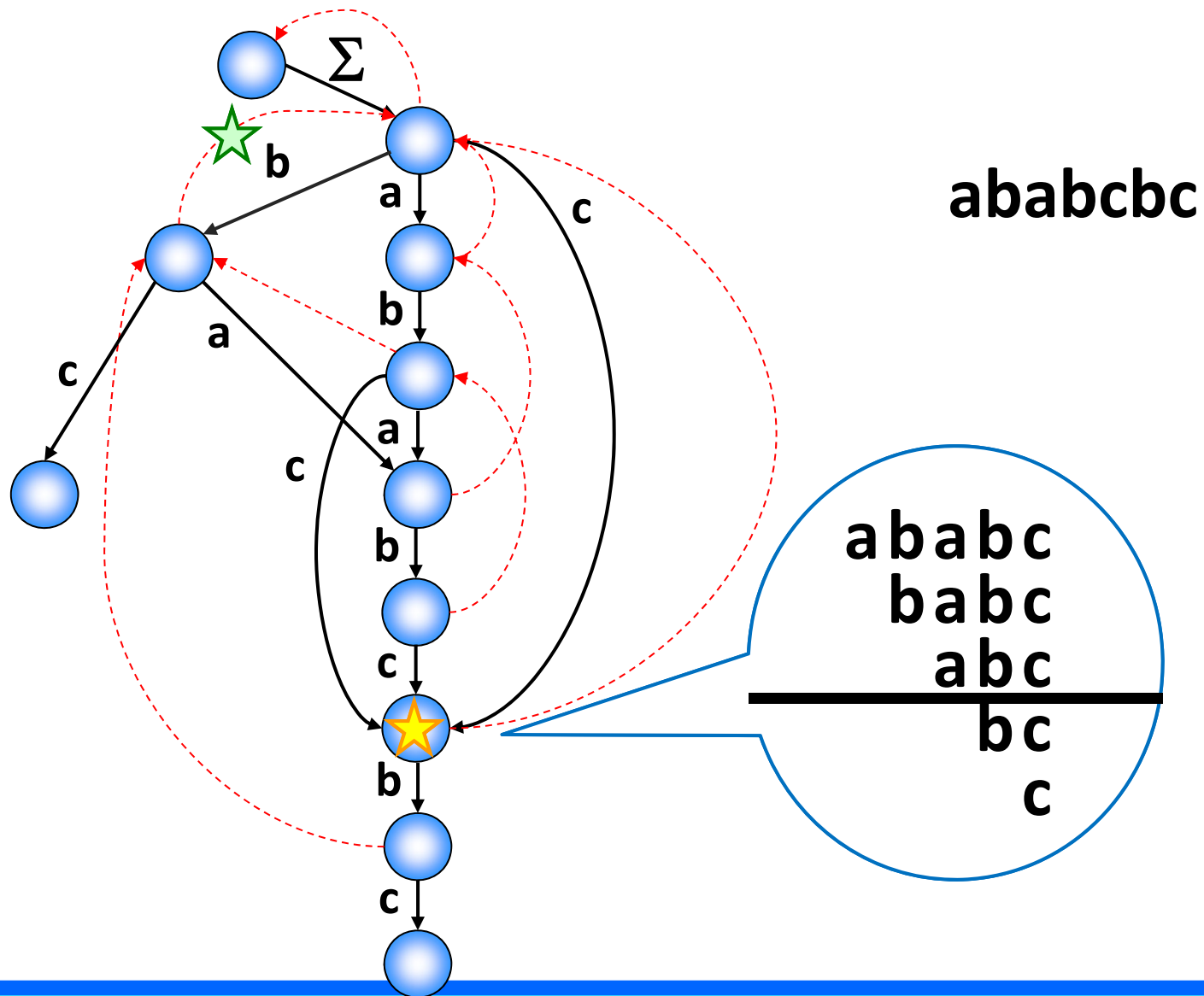
# 辺の差し替え



# 辺の差し替え



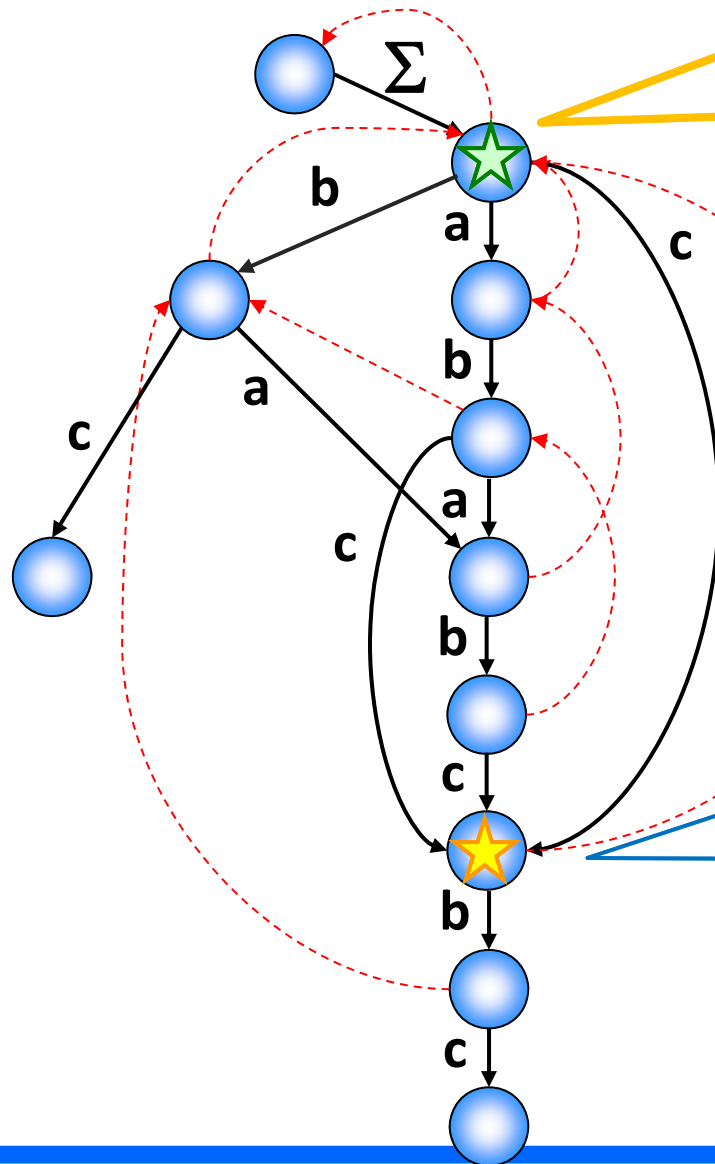
# 辺の差し替え



# 辺の差し替え

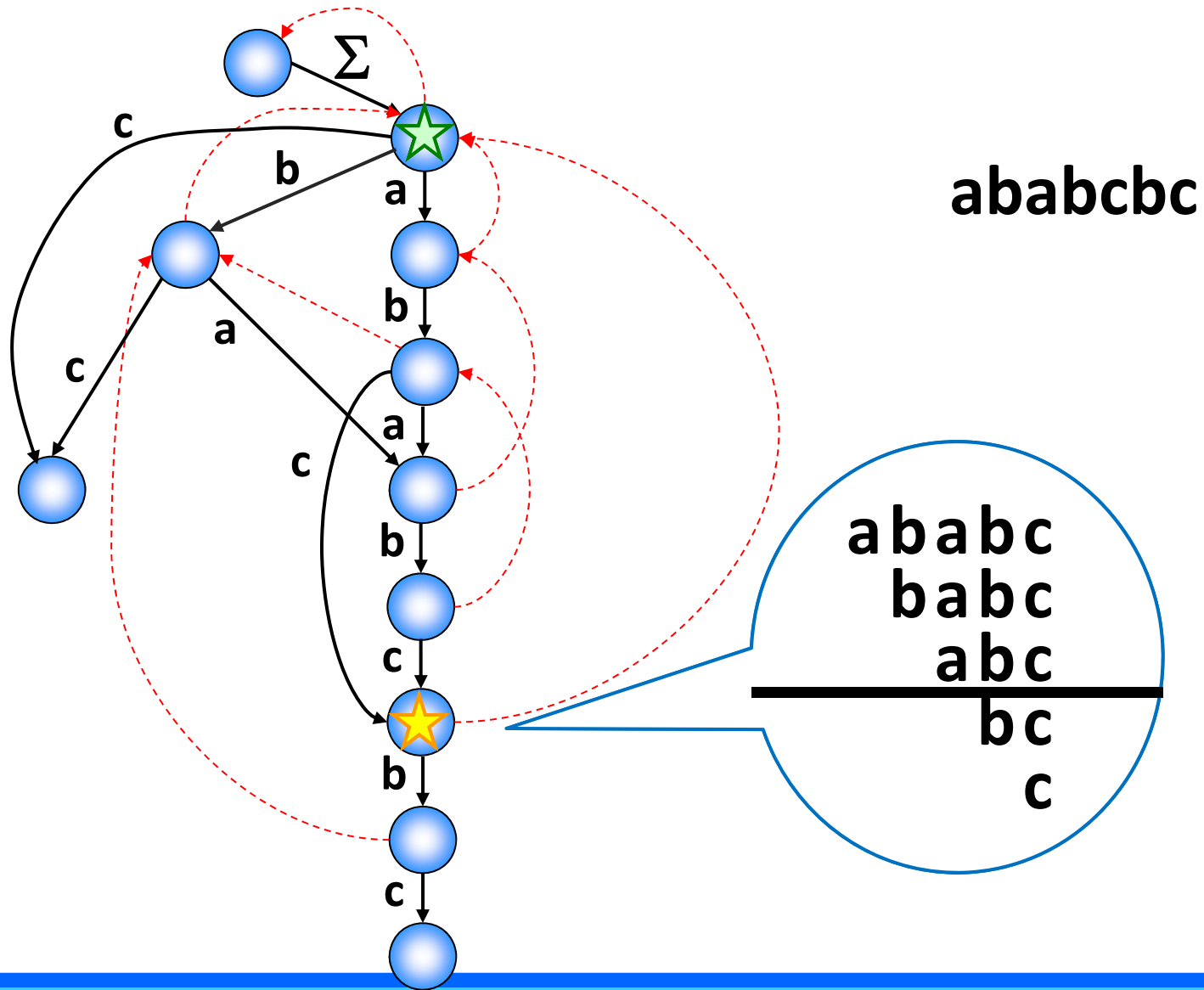
☆ から c で辿った先に  
★ があるので, c の辺を  
差し替える

ababcbc

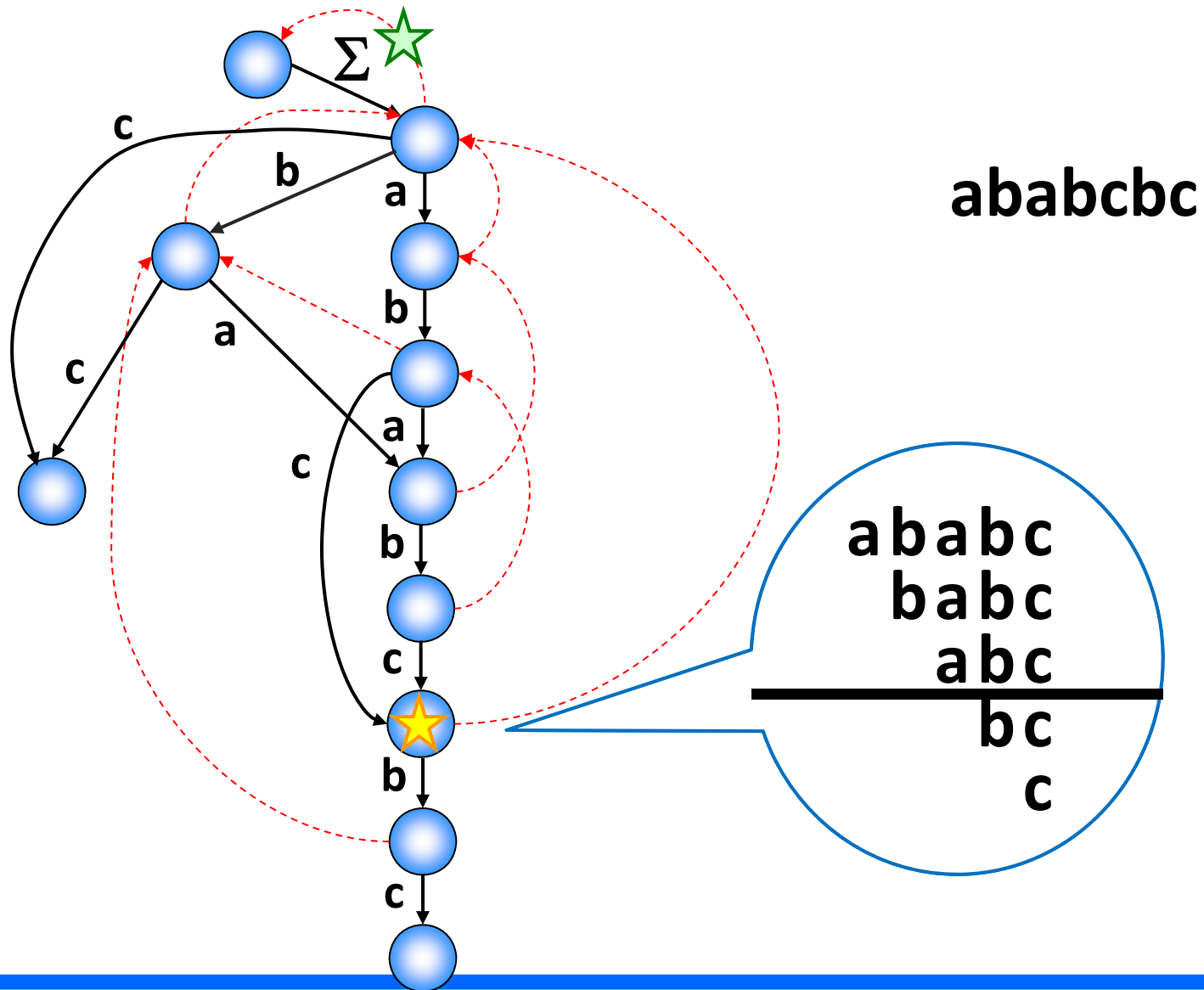


ababc  
babc  
abc  
-----  
bc  
c

# 辺の差し替え

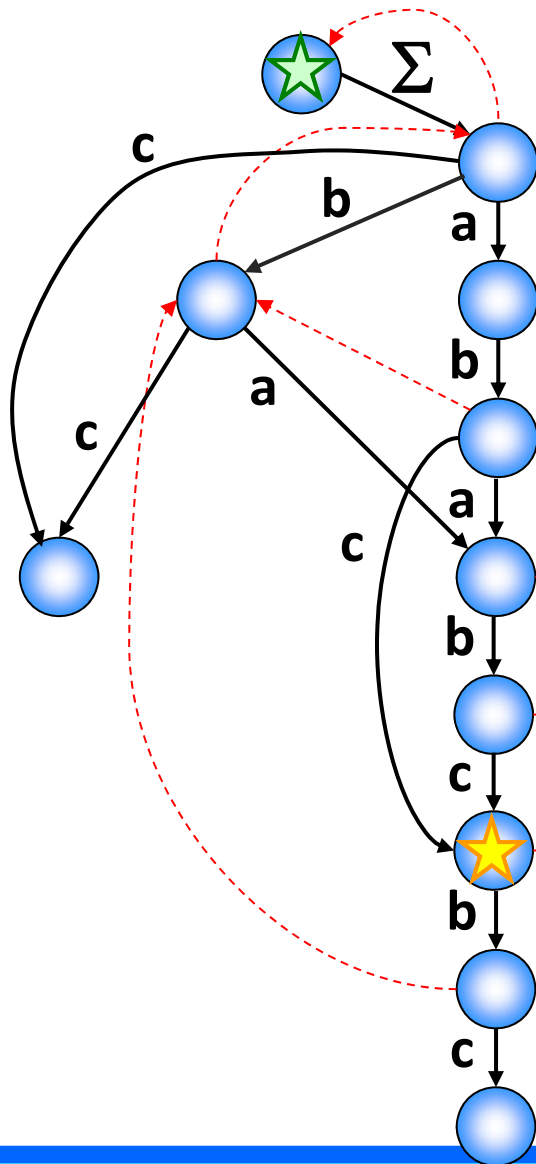


# 辺の差し替え



# 辺の差し替え

☆ から c で辿った先に  
★ がないので、  
辺の差し替えは終わり



ababcbc

ababc  
babc  
abc  
-----  
bc  
c



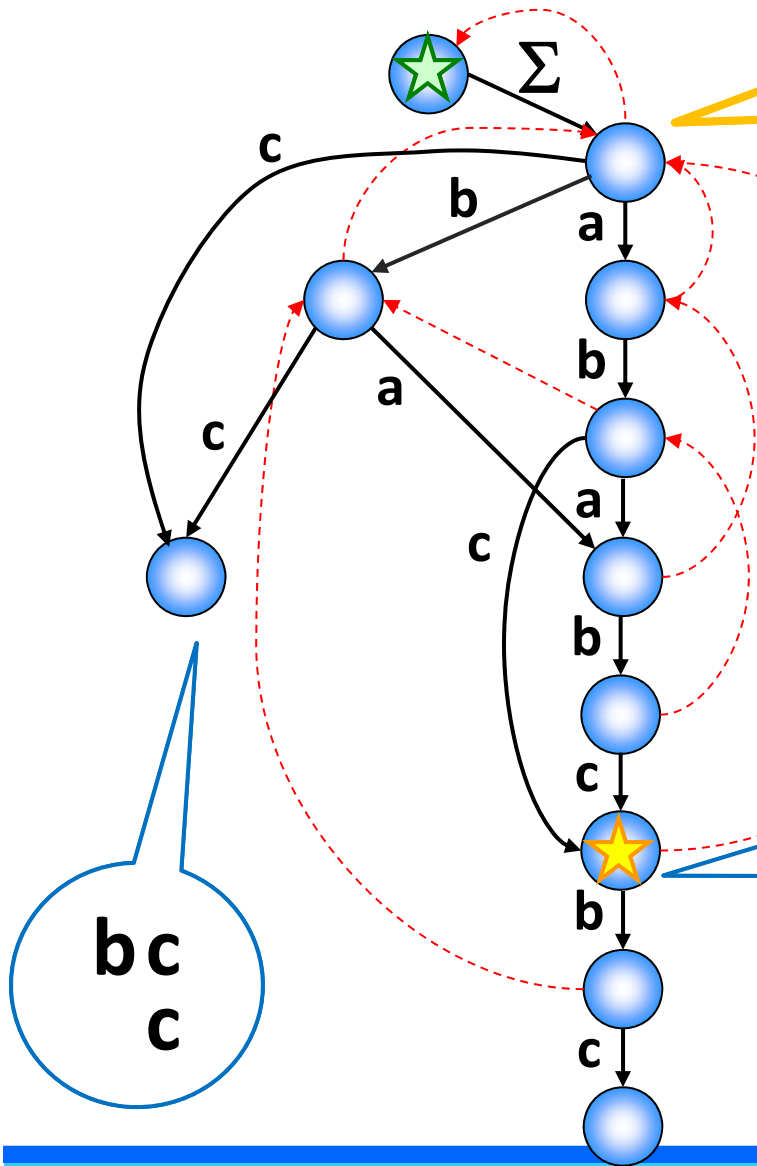
# 辺の差し替え

☆ から c で辿った先に  
★ がないので、  
辺の差し替えは終わり

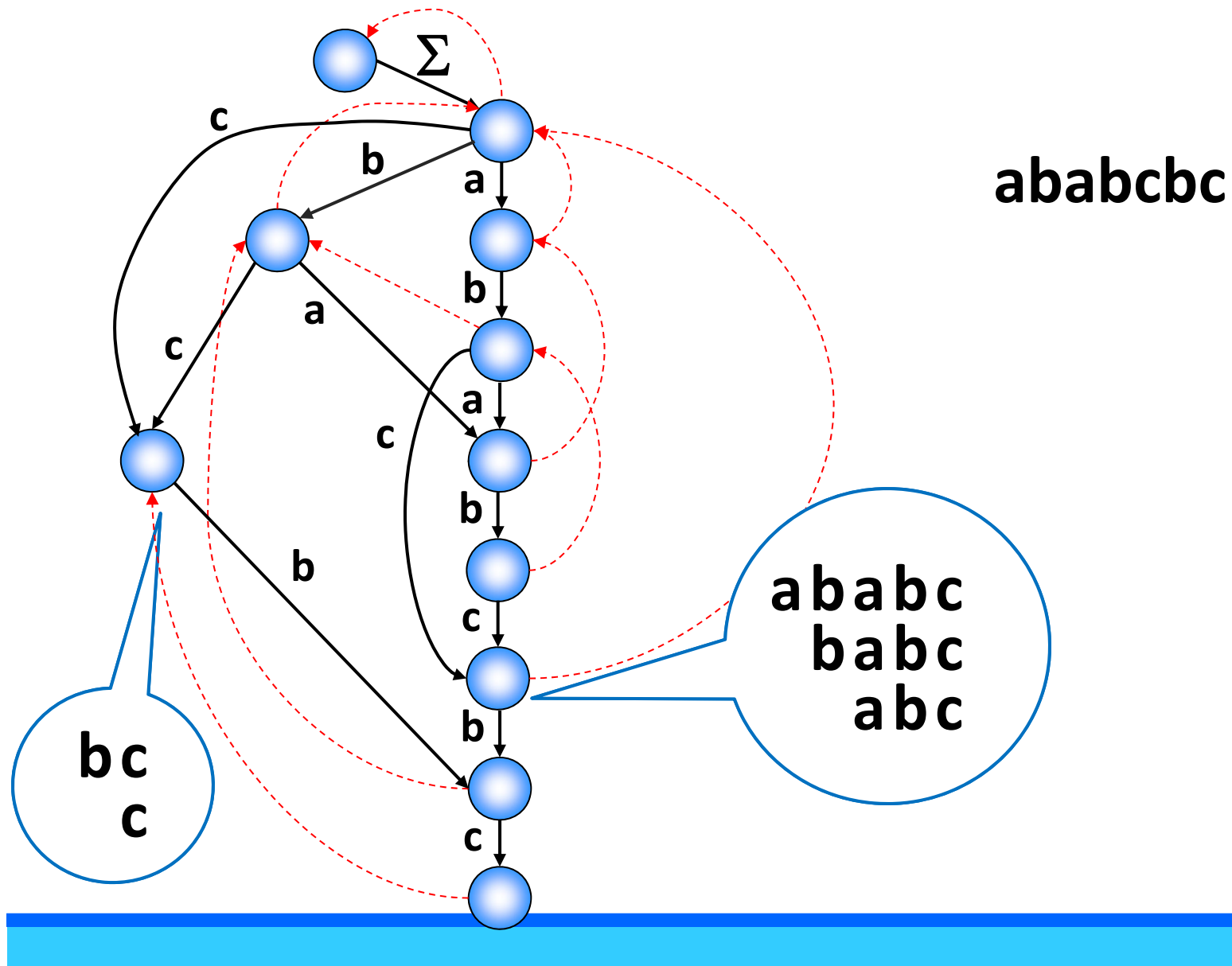
ababcbc

ababc  
babcb  
abc

bc  
c



# 辺の差し替え



# DAWG の左→右オンライン構築 ⇒ Weiner Tree の右→左オンライン構築

常識8 [Weiner 1973, A. Blumer et al. 1985]

Weiner tree を  $O(n \log \sigma)$  時間・ $O(n)$  領域  
で右→左にオンライン構築できる.

- 常識3 (DAWG の suffix link は逆文字列の Weiner tree) と, ちょっと非常識 1 より自明.
- 実はこの逆も成り立つ!  
→ Weiner のアルゴリズムの拡張  
(今日は割愛かな).

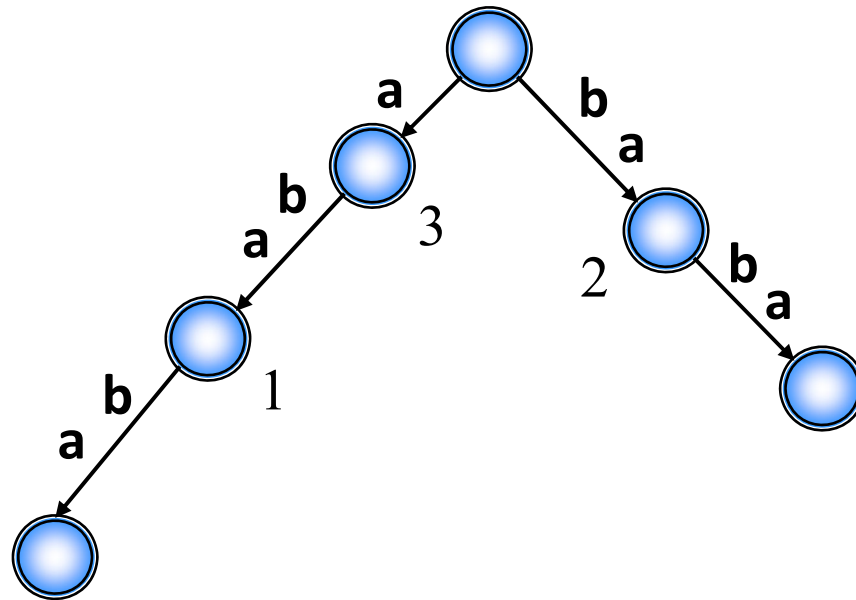
# Weiner Tree の左→右オンライン構築

## 常識 9

Weiner tree の左→右オンライン構築は  $\Omega(n^2)$  時間かかる.

- 文字列  $(ab)^{n/2}$  を考える.

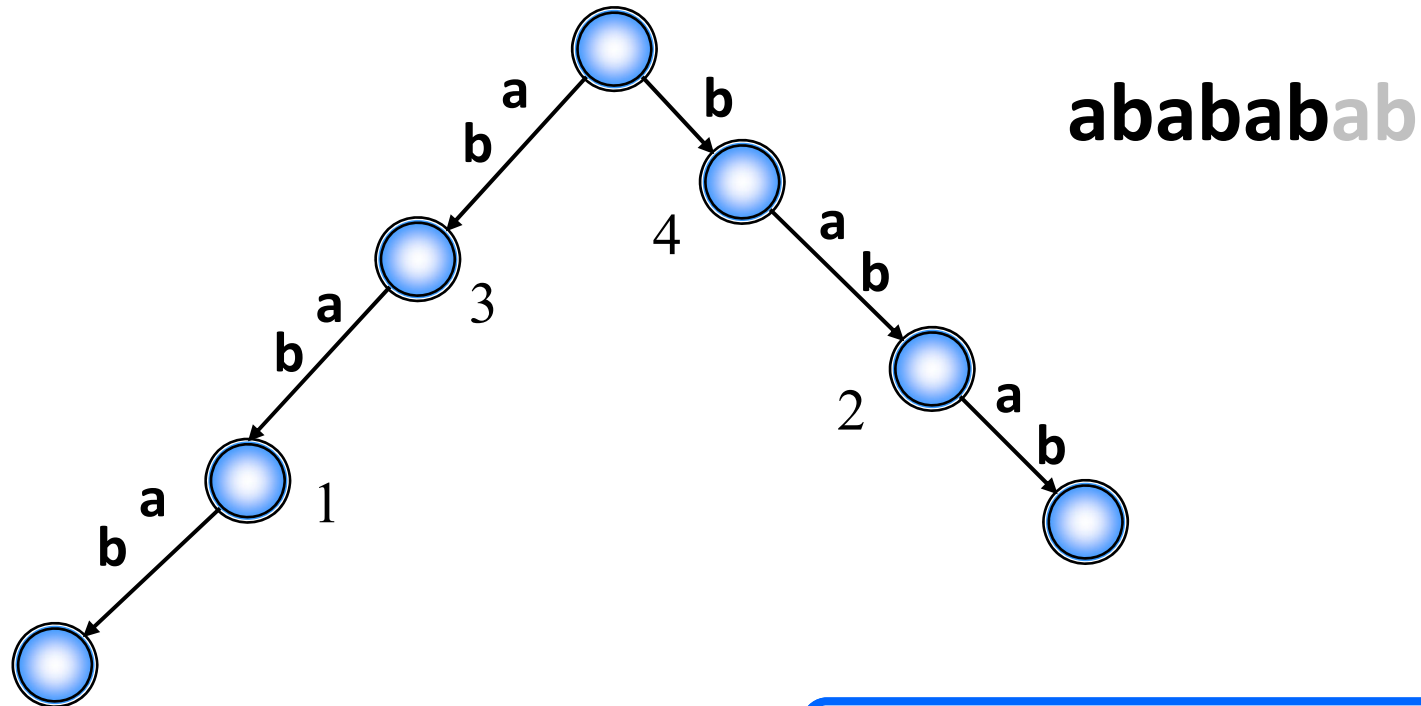
# Weiner Tree の左→右オンライン構築



abababab

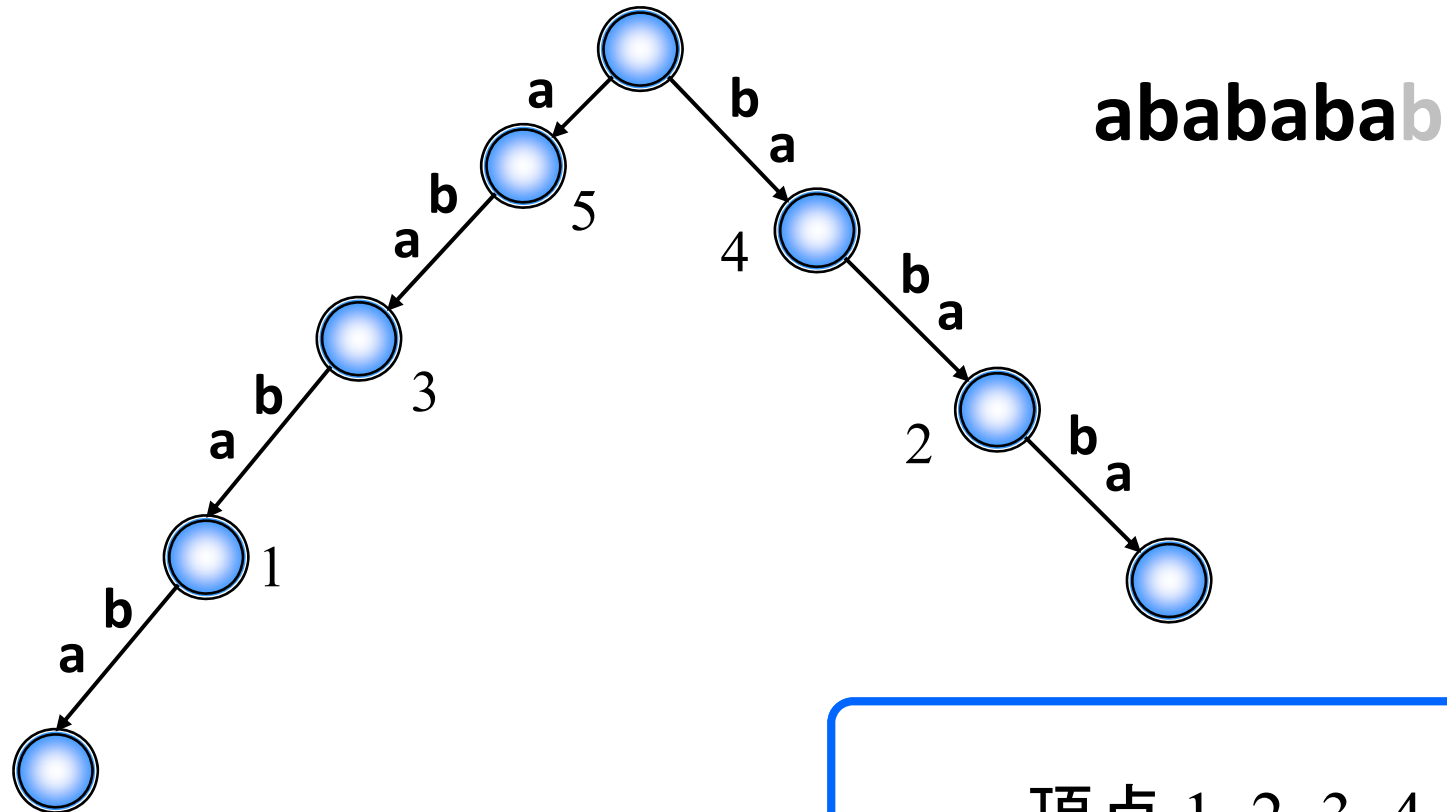


# Weiner Tree の左→右オンライン構築



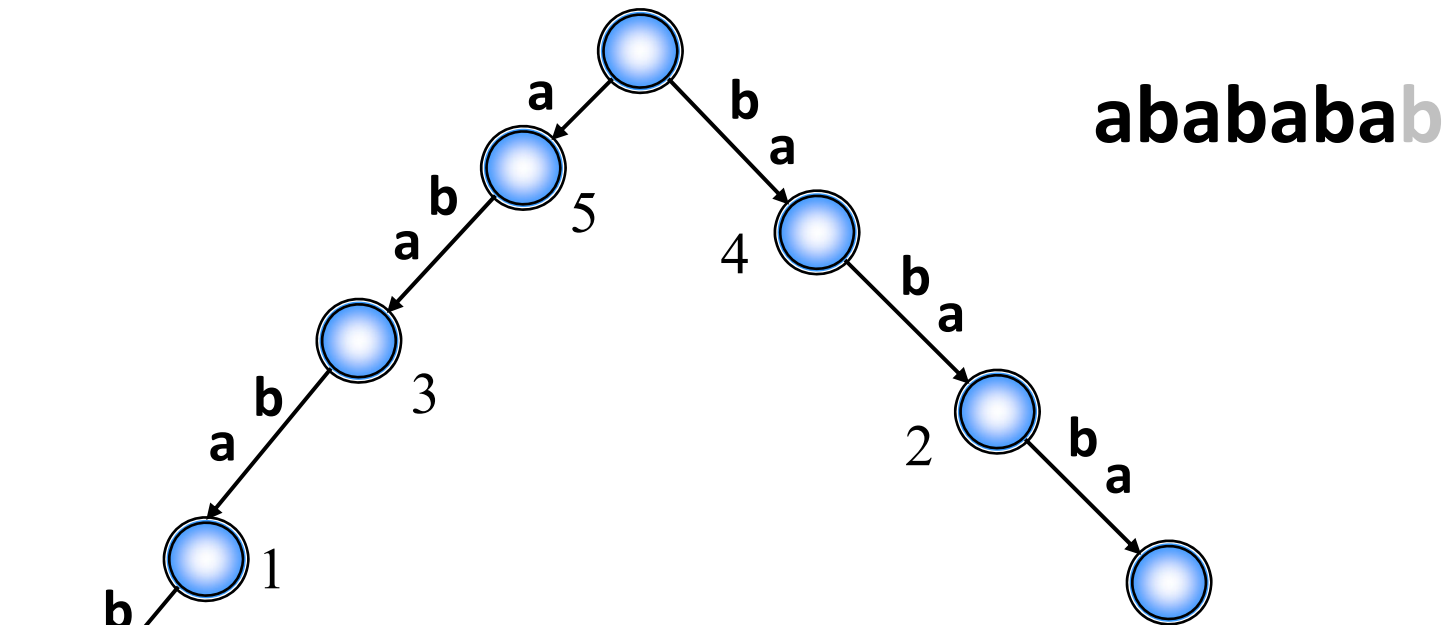
頂点 1, 2, 3 が  
1文字分下にズれる.

# Weiner Tree の左→右オンライン構築



頂点 1, 2, 3, 4 が  
1文字分下にズれる。

# Weiner Tree の左→右オンライン構築



一般に、1文字追加ごとに  
 $\Omega(n)$  個の内部頂点を  
下にズラす必要がある。

※ この例は周期的だが、  
周期的でない嫌な例も存在する。  
abcxabcyabczabcwabcv... など。



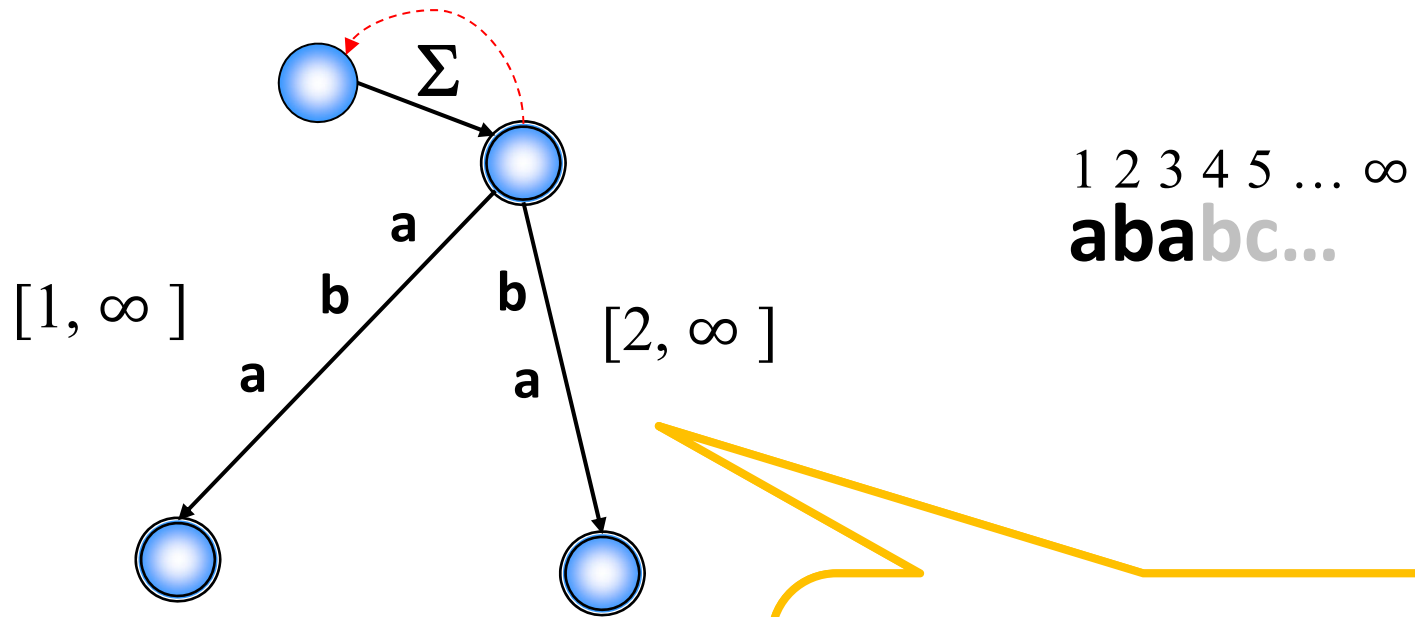
# ちょっと寄り道： DAWG の右→左オンライン構築

## 常識 10

DAWG の右→左オンライン構築は  
 $\Omega(n^2)$  時間かかる。

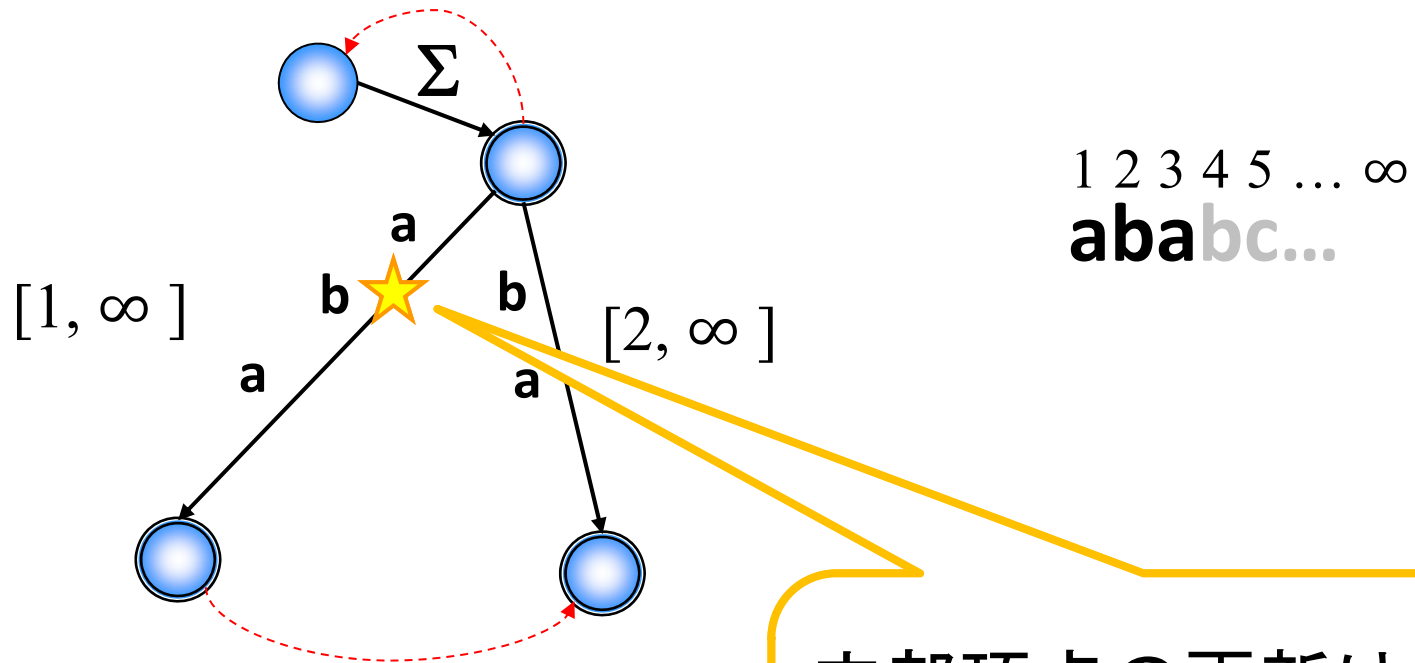
- 常識 3より,  $w^R$  の Weiner tree と  $w$  の DAWG は頂点を共有する。
- よって, 常識 9より  $\Omega(n^2)$  時間かかる。

# Ukkonen Tree の左→右オンライン構築



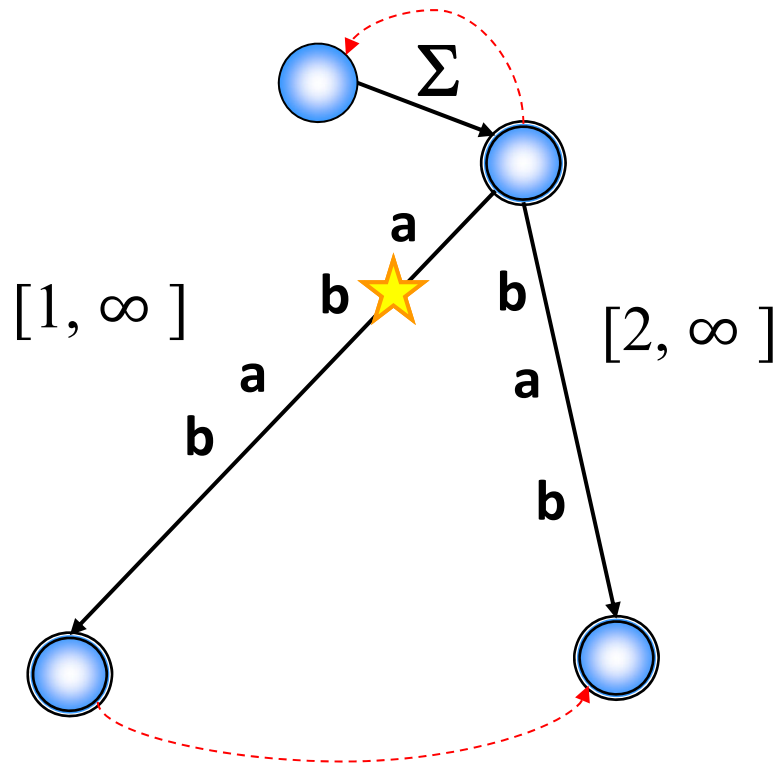
葉の入力辺は  
 $[i, \infty]$  でエンコードする  
→ 葉を”自動的に”更新

# Ukkonen Tree の左→右オンライン構築

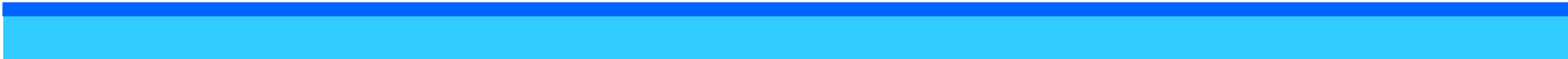


内部頂点の更新は、  
葉ではない最長の suffix  
から始める → active point

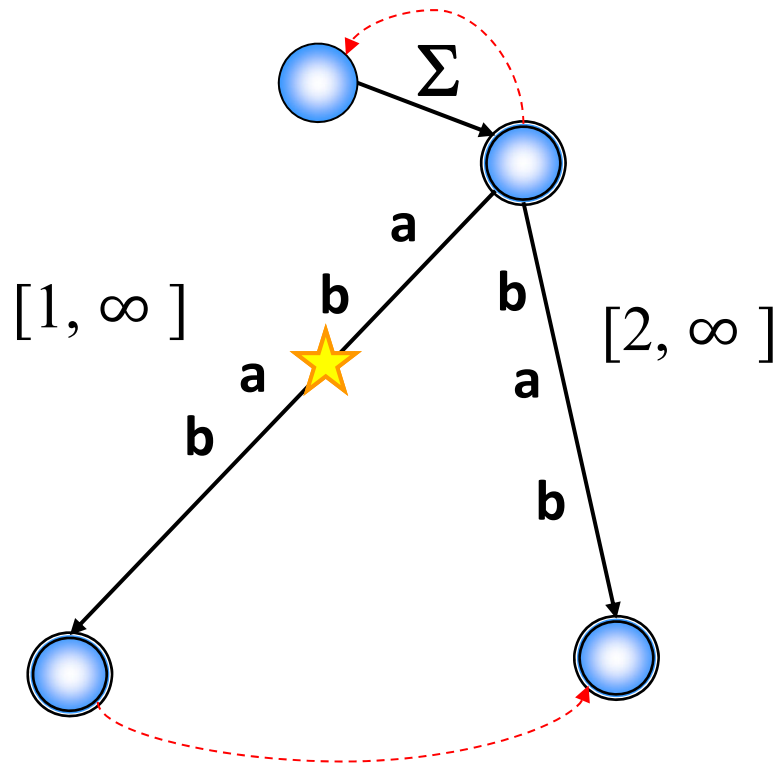
# Ukkonen Tree の左→右オンライン構築



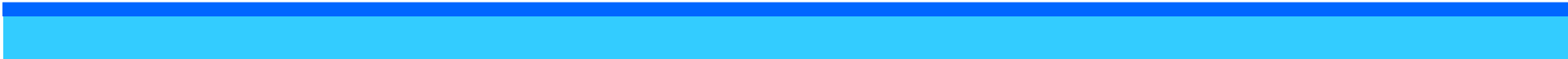
1 2 3 4 5 ...  $\infty$   
**ababc...**



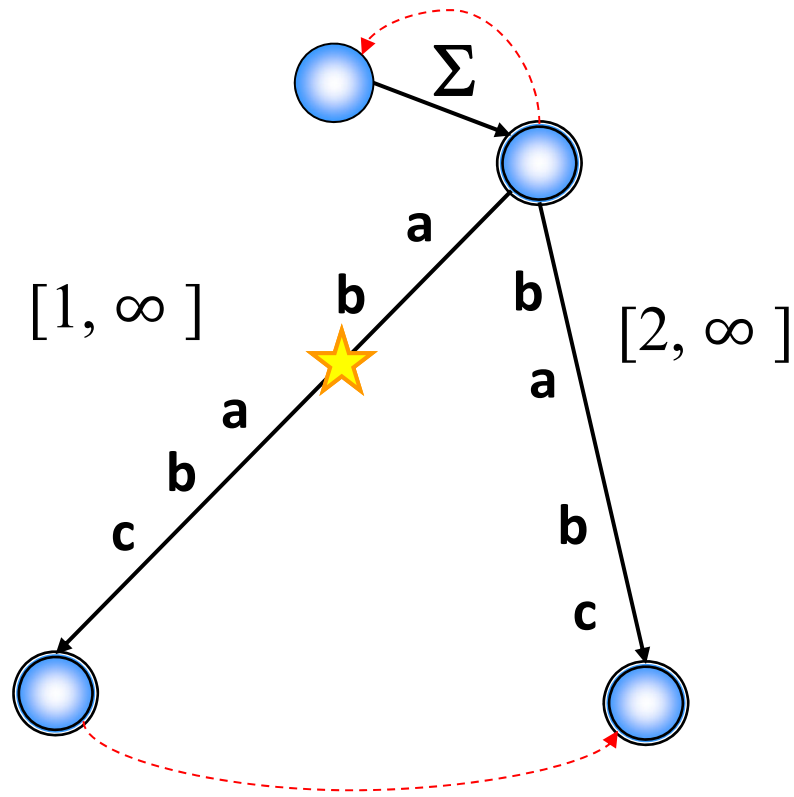
# Ukkonen Tree の左→右オンライン構築



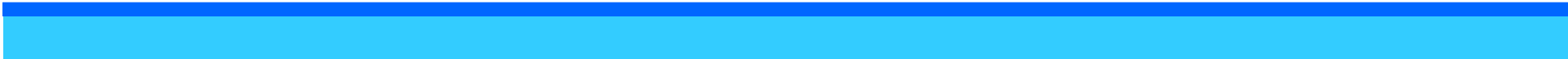
1 2 3 4 5 ...  $\infty$   
**ababc...**



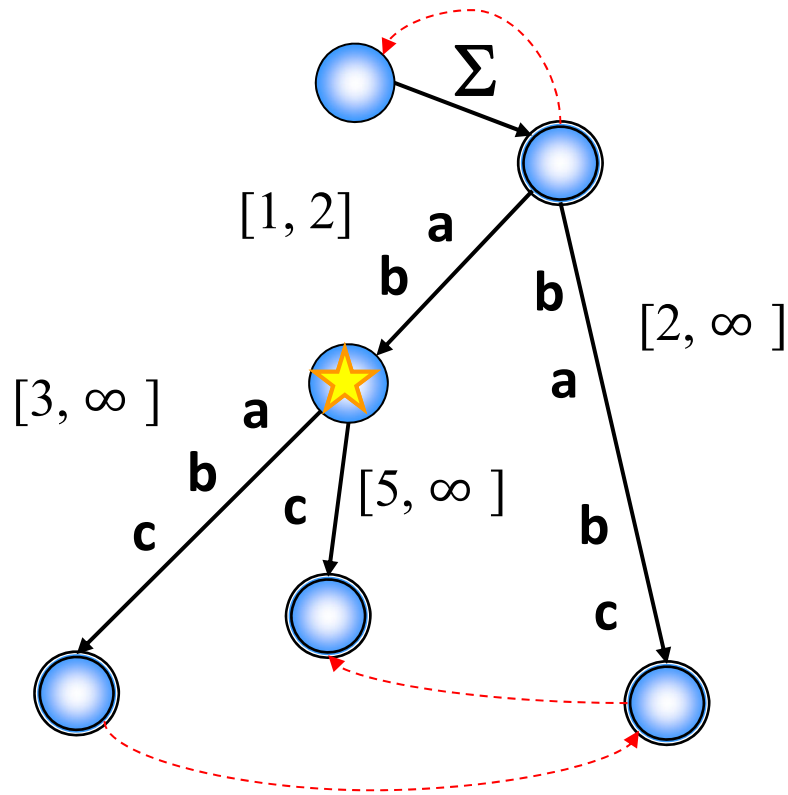
# Ukkonen Tree の左→右オンライン構築



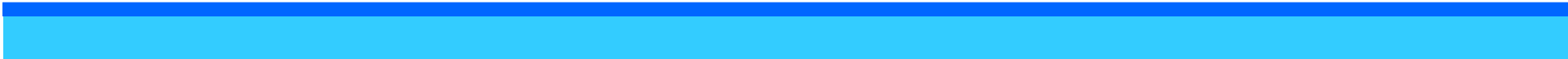
1 2 3 4 5 ...  $\infty$   
**ababc...**



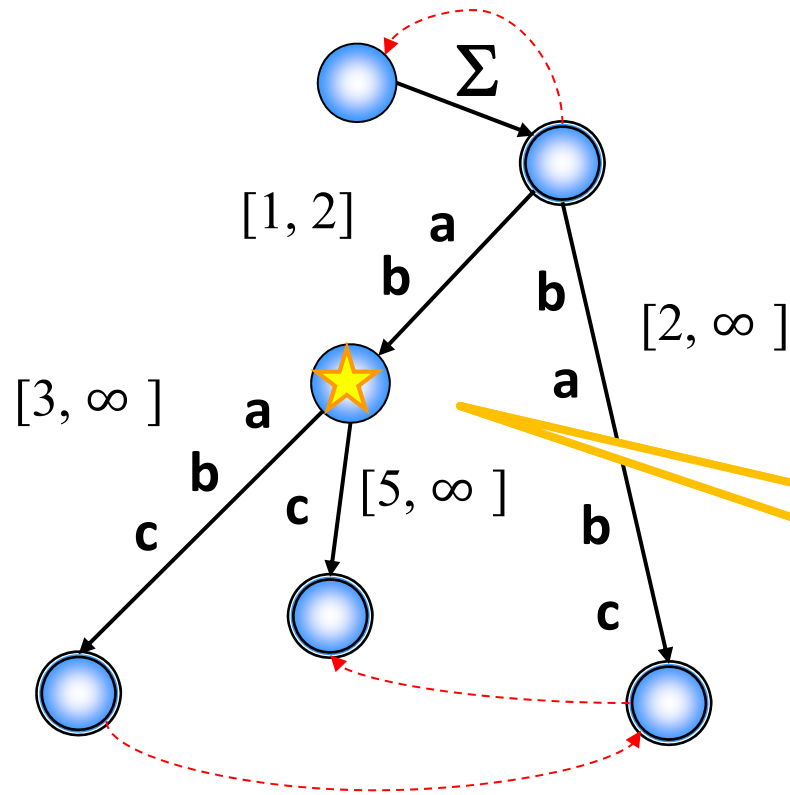
# Ukkonen Tree の左→右オンライン構築



1 2 3 4 5 ...  $\infty$   
**ababc...**



# Ukkonen Tree の左→右オンライン構築

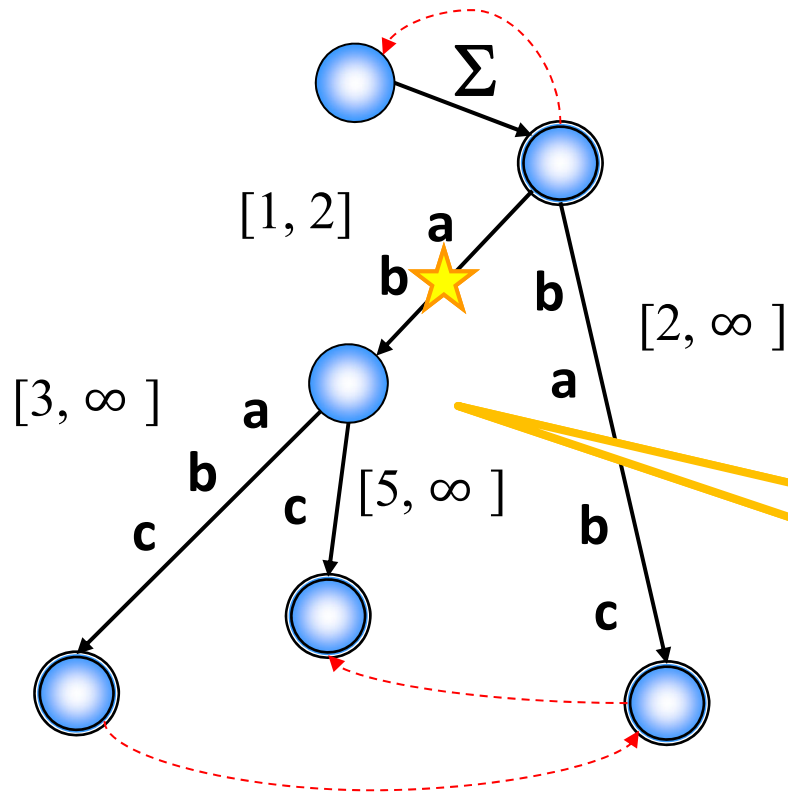


1 2 3 4 5 ... ∞  
**ababc...**

新しい頂点 (ab) の  
suffix link がまだないので、  
親の suffix link で代用.

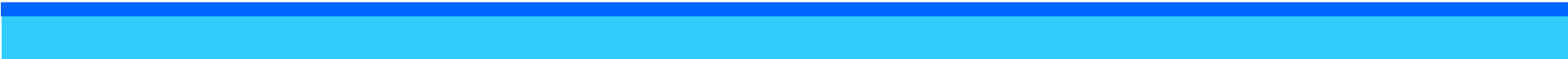


# Ukkonen Tree の左→右オンライン構築

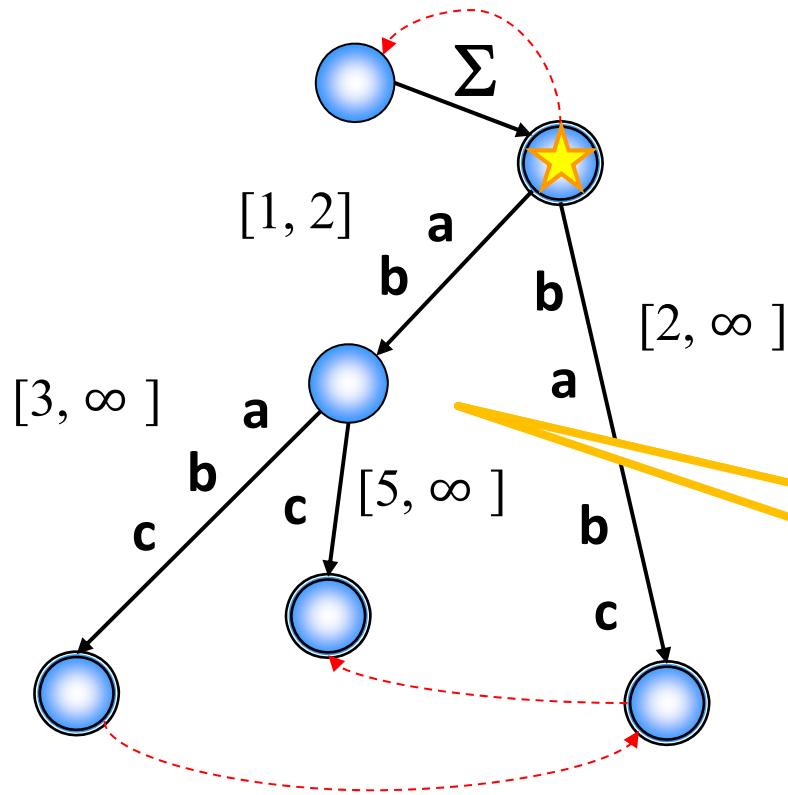


1 2 3 4 5 ...  $\infty$   
ababc...

新しい頂点 (ab) の suffix link がまだないので、親の suffix link で代用.



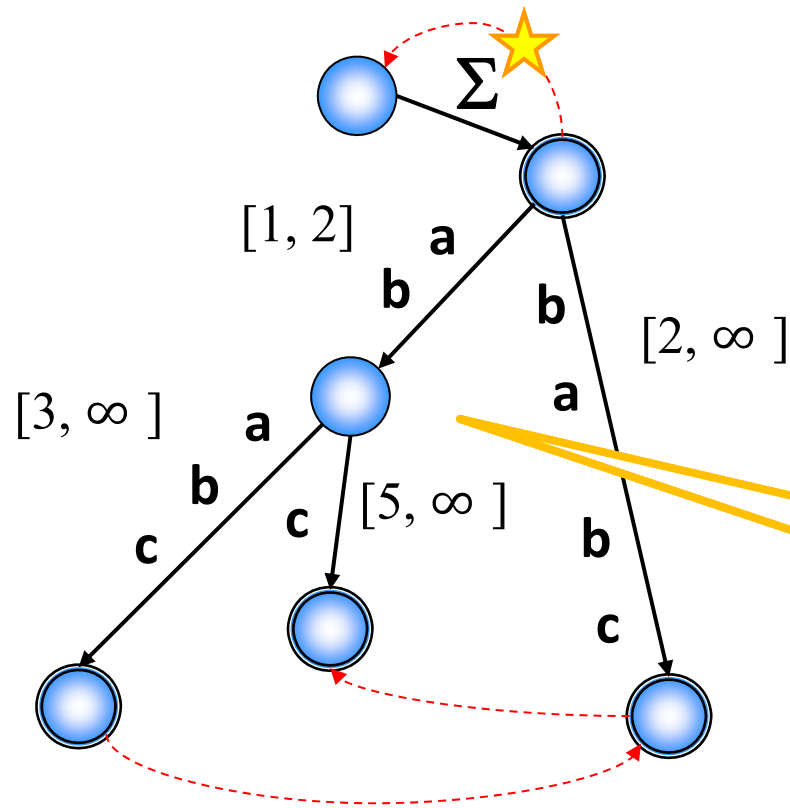
# Ukkonen Tree の左→右オンライン構築



1 2 3 4 5 ...  $\infty$   
**ababc...**

新しい頂点 (ab) の  
suffix link がまだないので、  
親の suffix link で代用.

# Ukkonen Tree の左→右オンライン構築

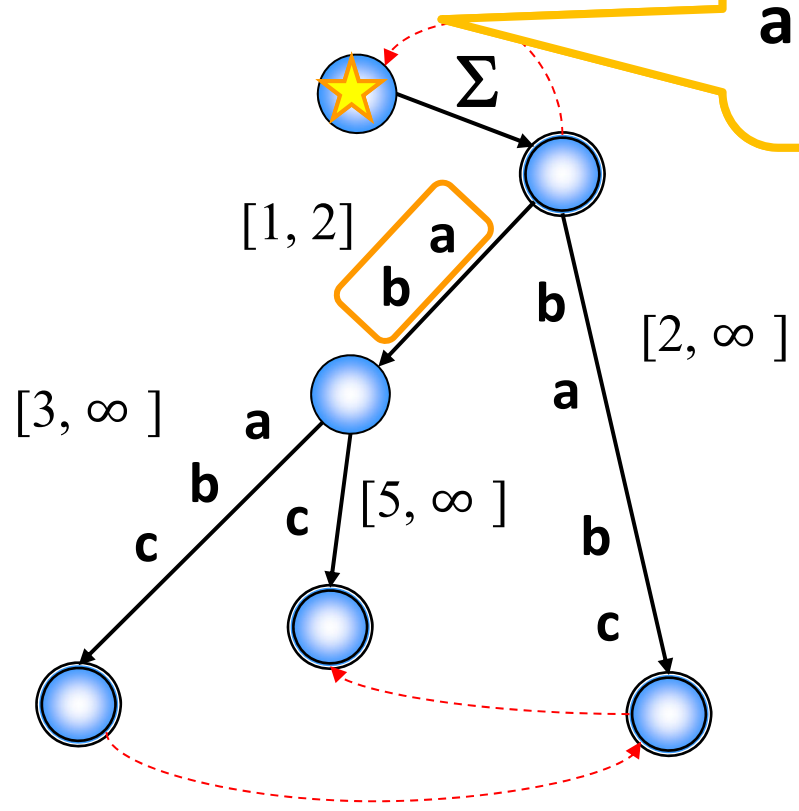


1 2 3 4 5 ...  $\infty$   
**ababc...**

新しい頂点 (ab) の  
suffix link がまだないので、  
親の suffix link で代用.

# Ukkonen Tree の左 →

ここから親へのラベル  
ab で降れる.

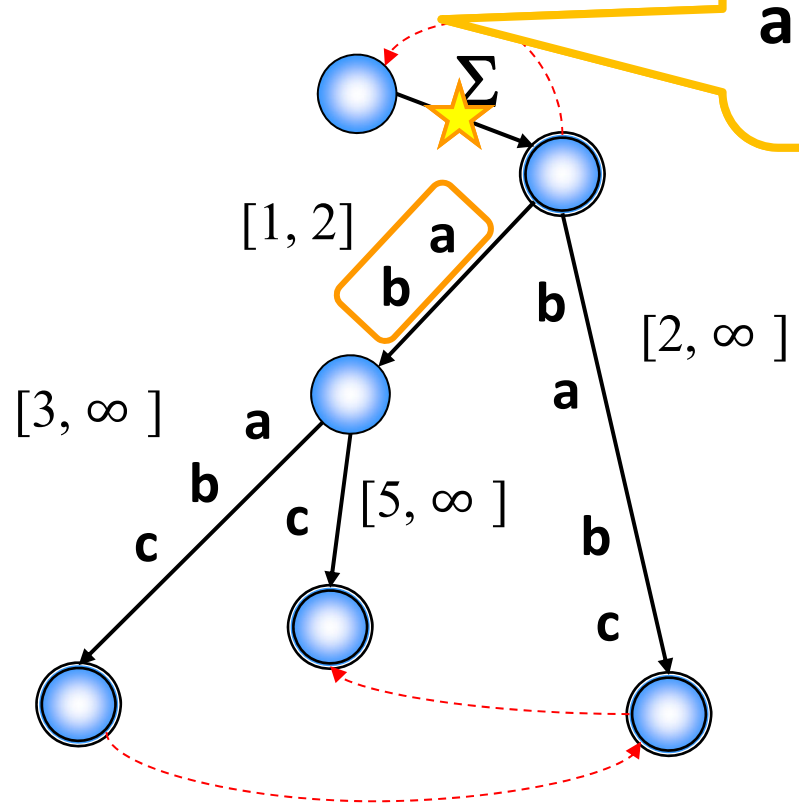


1 2 3 4 5 ...  $\infty$   
**ababc...**



# Ukkonen Tree の左 →

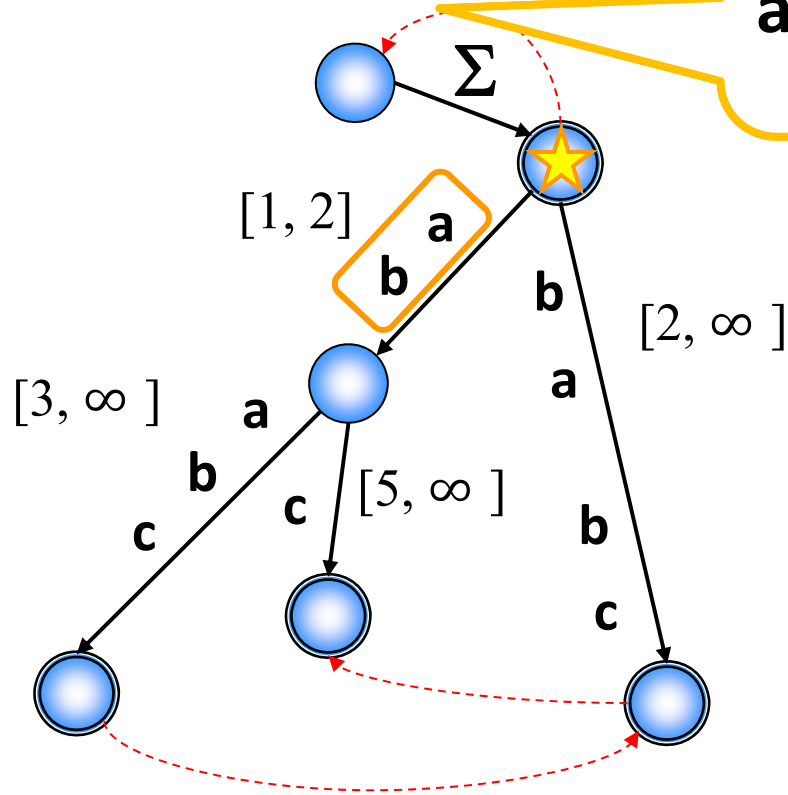
ここから親へのラベル  
ab で降れる.



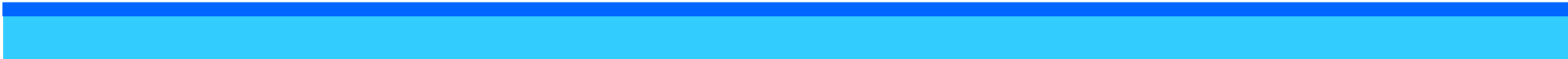
1 2 3 4 5 ...  $\infty$   
**ababc...**



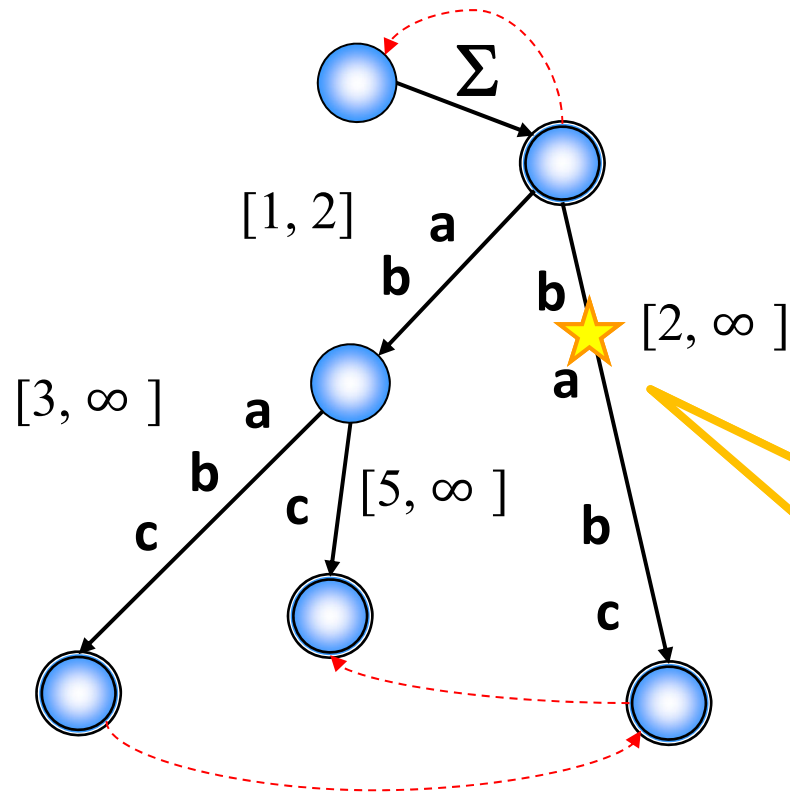
Ukkonen Tree の左 → ここから親へのラベル ab で降れる.



1 2 3 4 5 ...  $\infty$   
**ababc...**



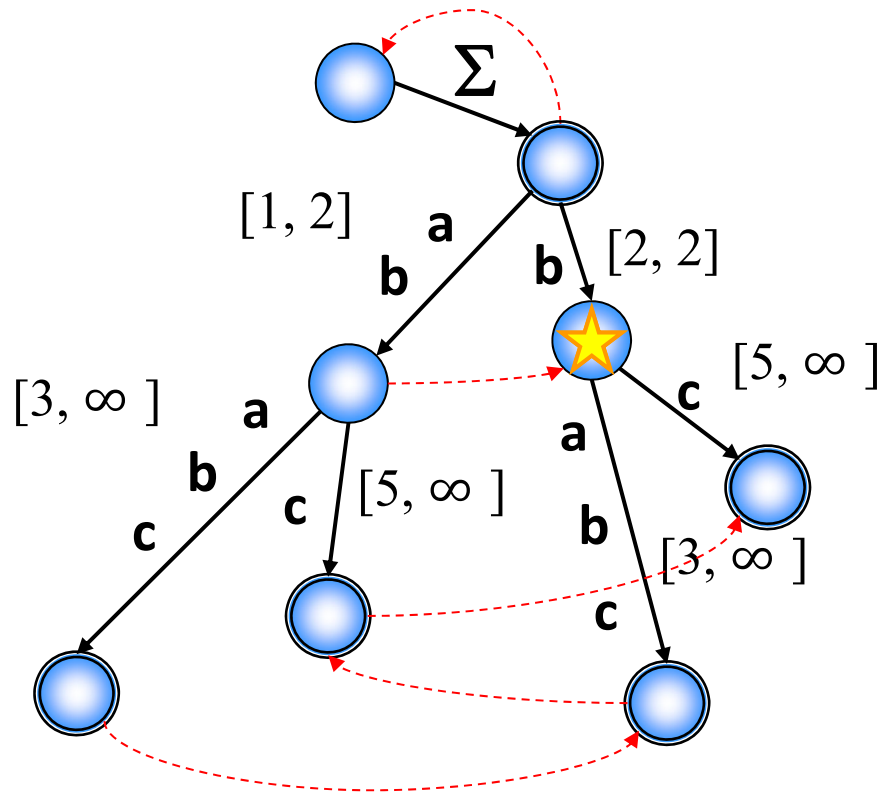
# Ukkonen Tree の左→右オンライン構築



1 2 3 4 5 ...  $\infty$   
**ababc...**

ab の次の suffix b に到着

# Ukkonen Tree の左→右オンライン構築

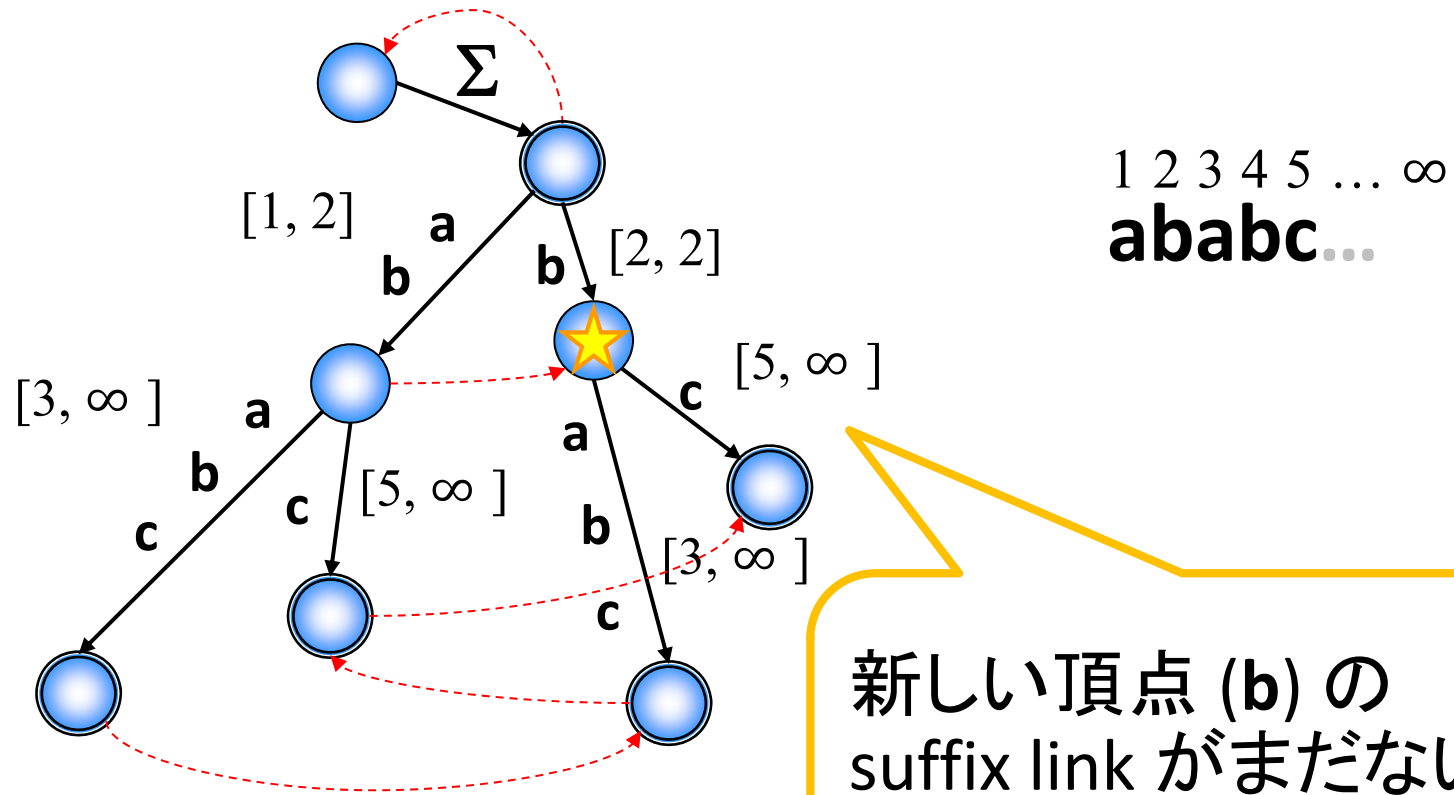


1 2 3 4 5 ...  $\infty$   
**ababc...**

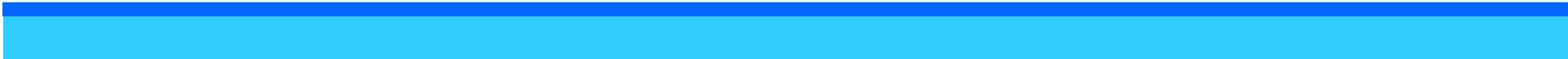




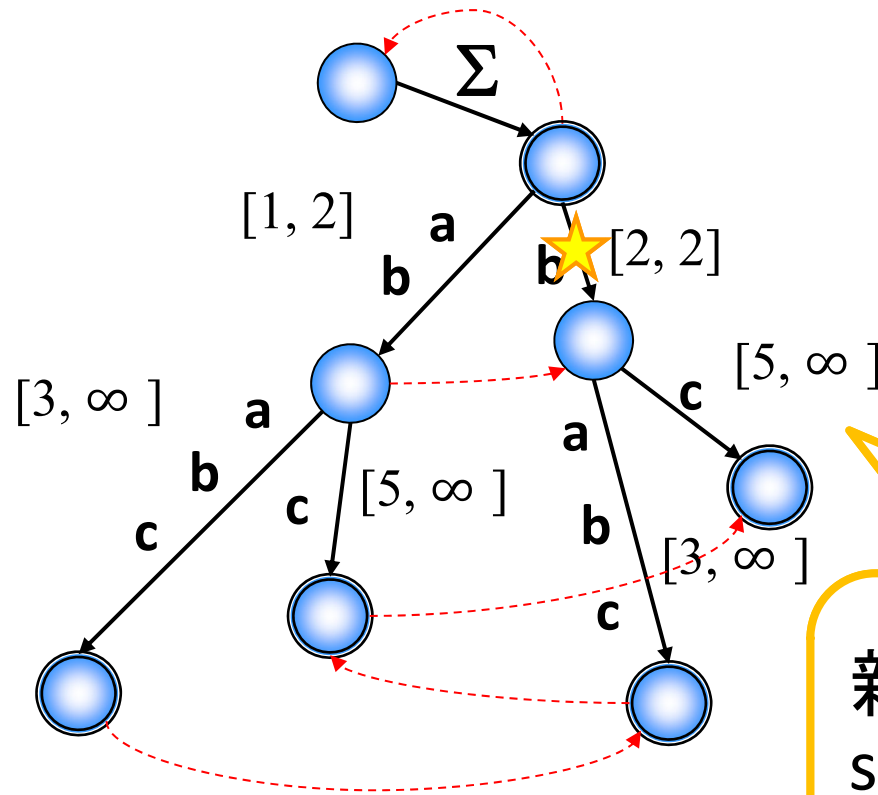
# Ukkonen Tree の左→右オンライン構築



新しい頂点 (b) の suffix link がまだないので、親の suffix link で代用.

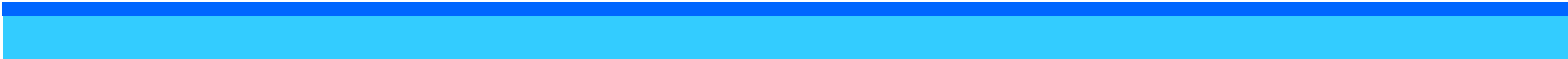


# Ukkonen Tree の左→右オンライン構築

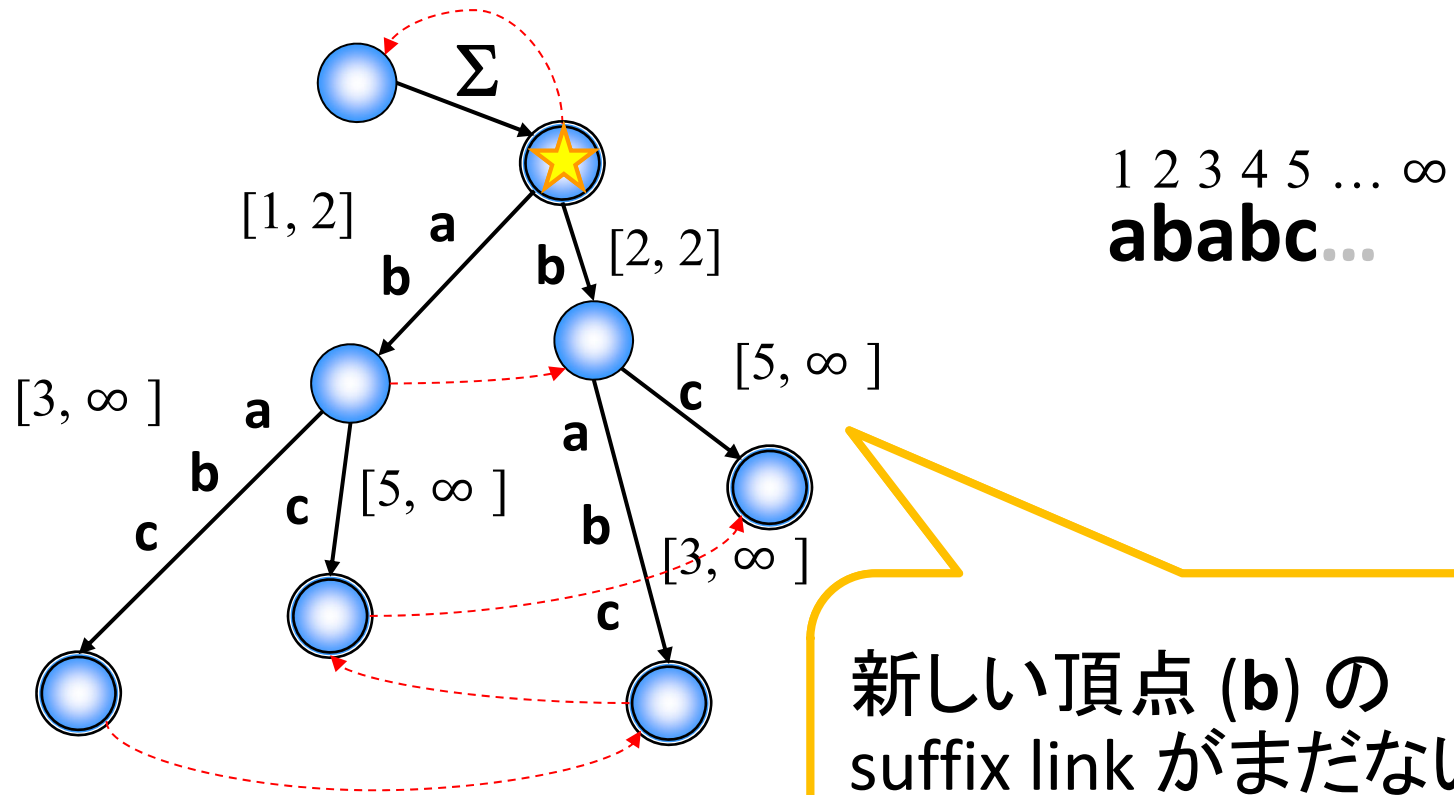


1 2 3 4 5 ... ∞  
**ababc...**

新しい頂点 (b) の suffix link がまだないので、親の suffix link で代用.



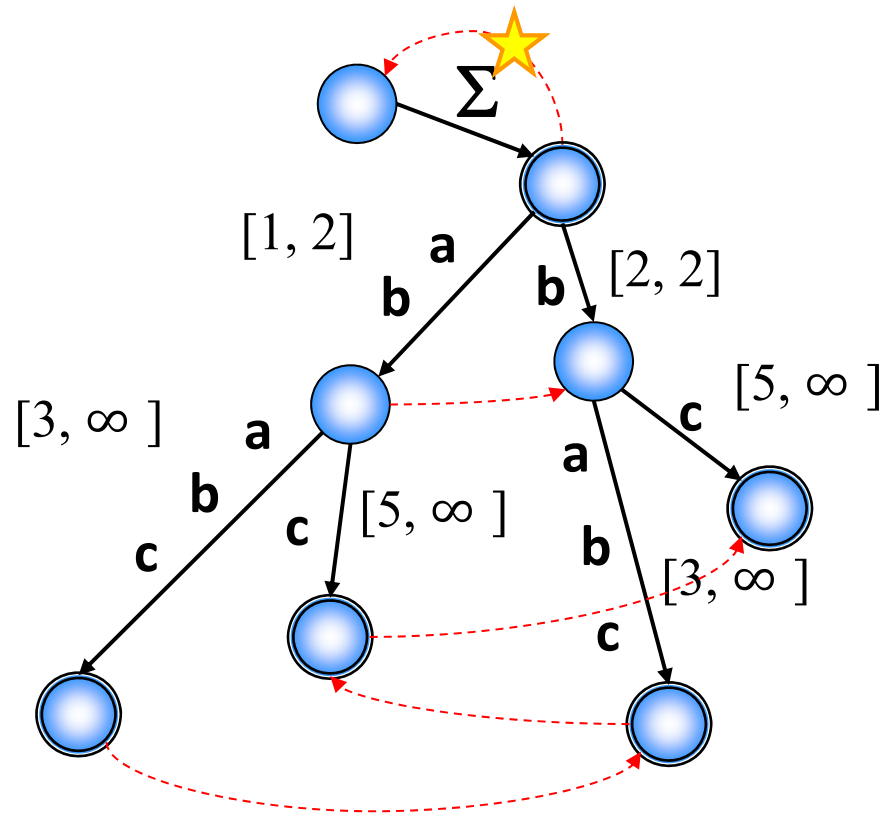
# Ukkonen Tree の左→右オンライン構築



新しい頂点 (b) の suffix link がまだないので、親の suffix link で代用.



# Ukkonen Tree の左→右オンライン構築

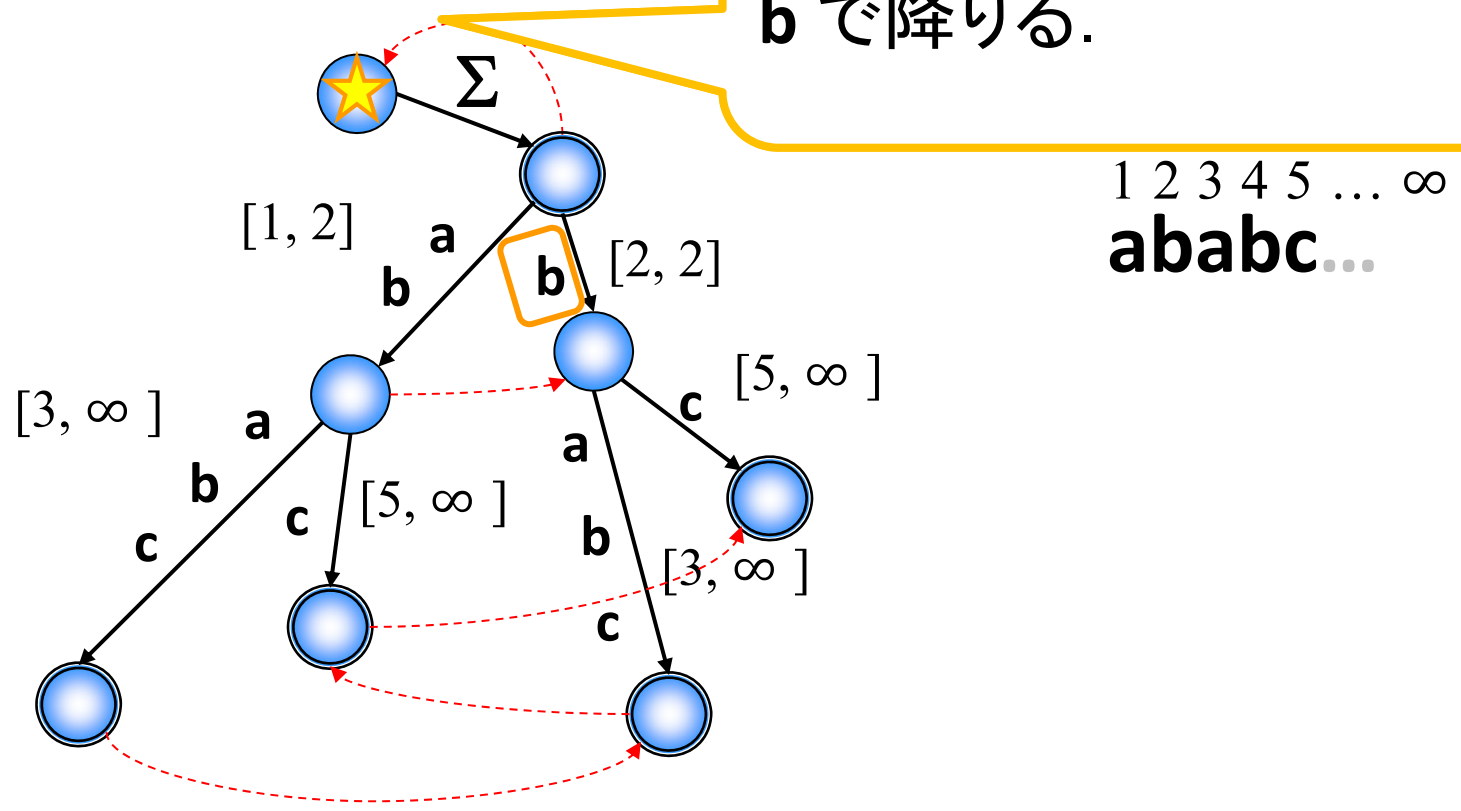


1 2 3 4 5 ...  $\infty$   
**ababc...**

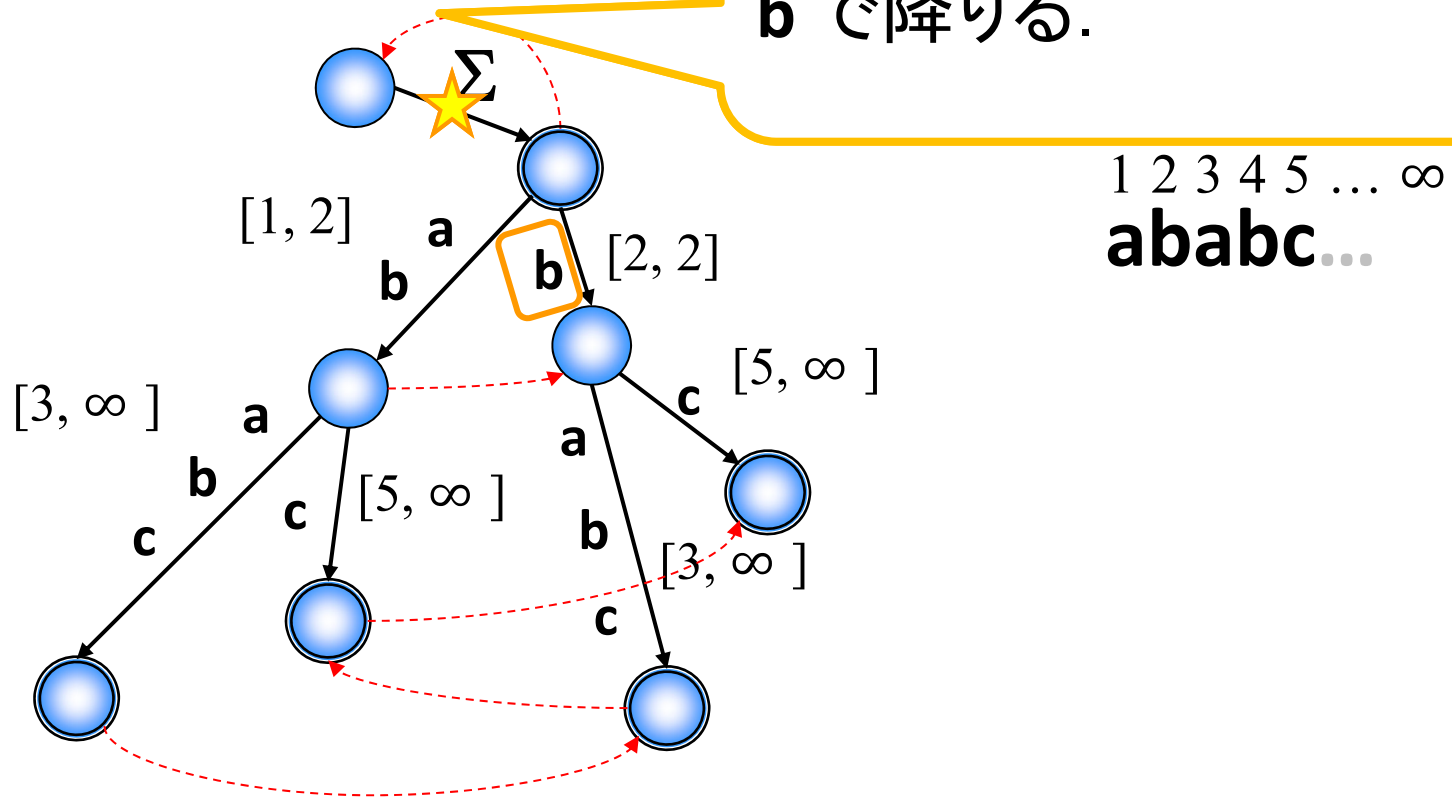


# Ukkonen Tree の左→

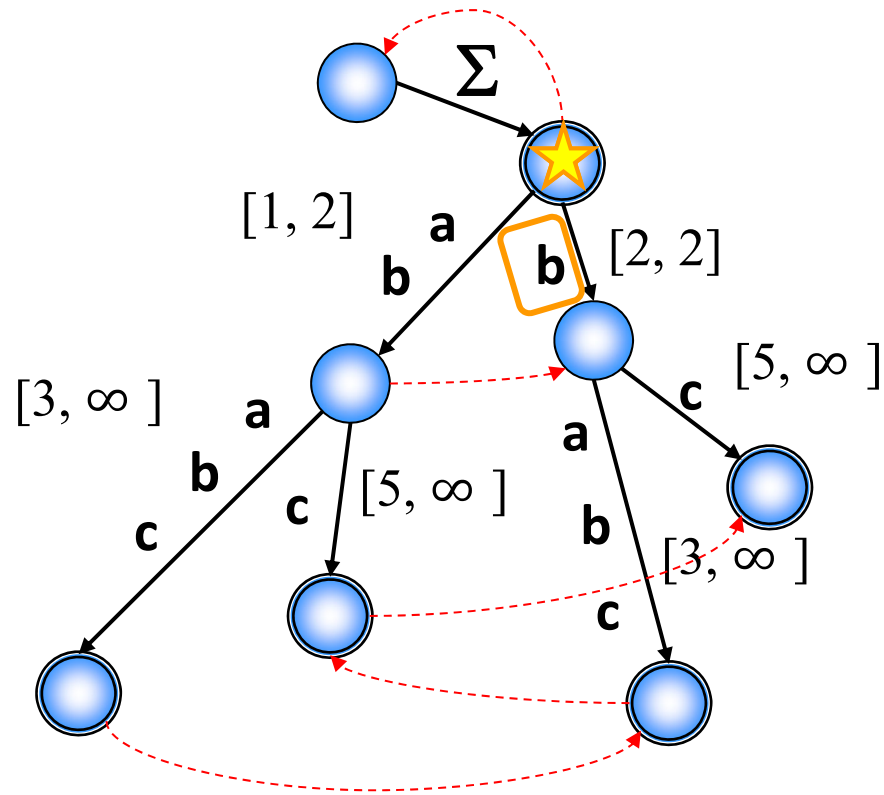
ここから親へのラベル  
bで降れる。



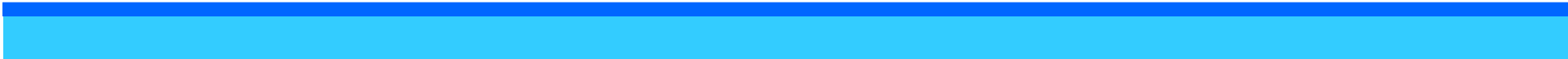
# Ukkonen Tree の左 → ここから親へのラベル **b** で降る.



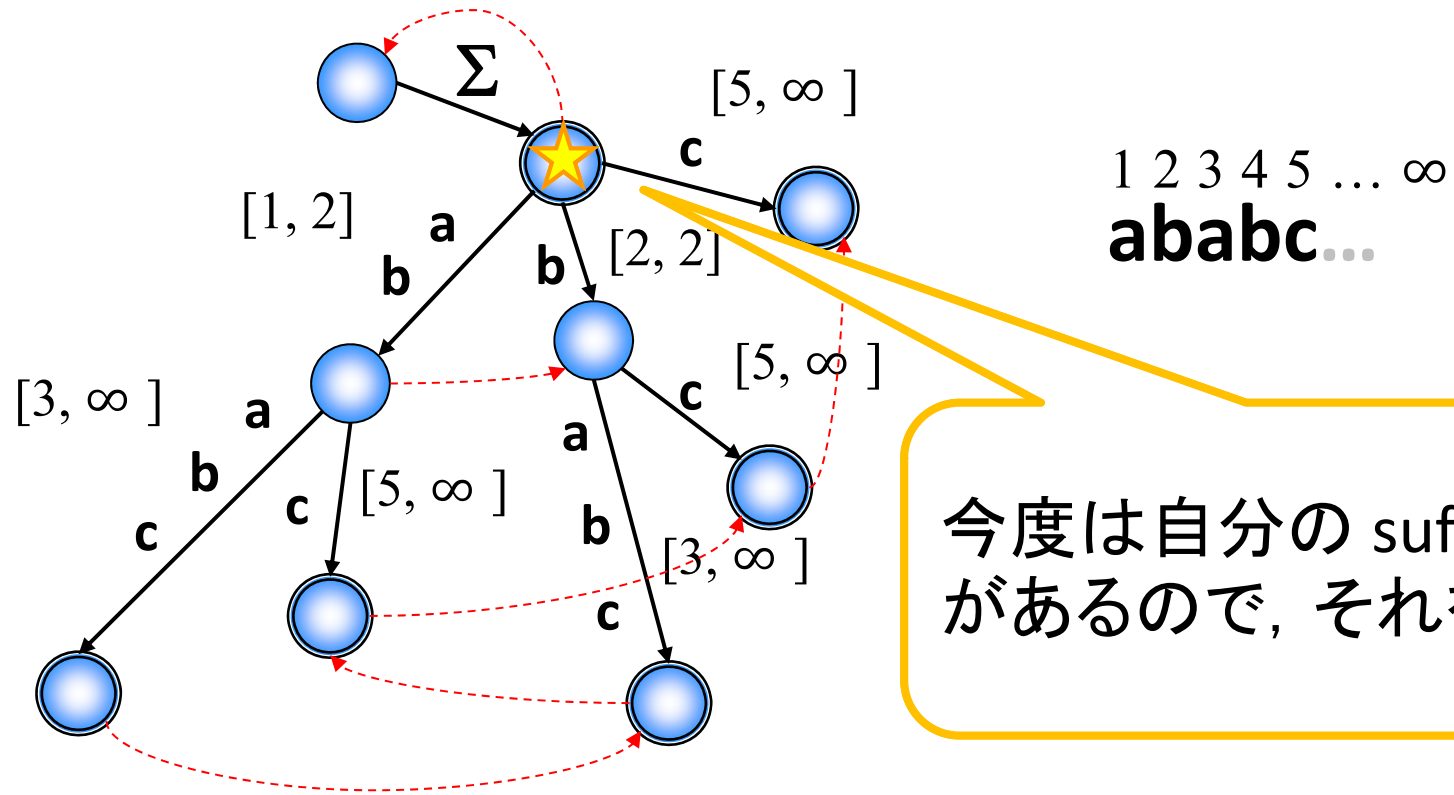
# Ukkonen Tree の左→右オンライン構築



1 2 3 4 5 ...  $\infty$   
**ababc...**



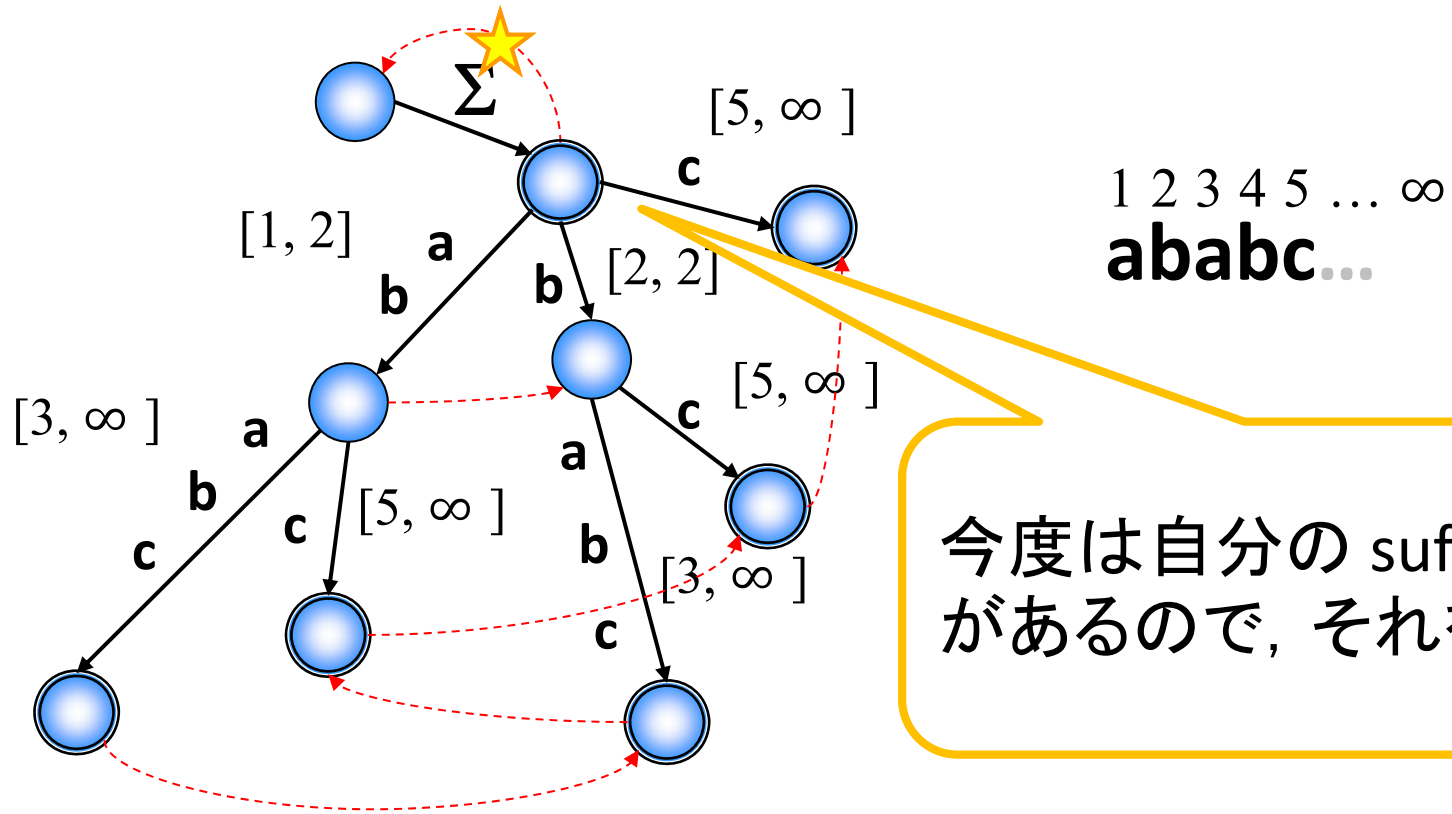
# Ukkonen Tree の左→右オンライン構築



今度は自分の suffix link  
があるので, それを使う.

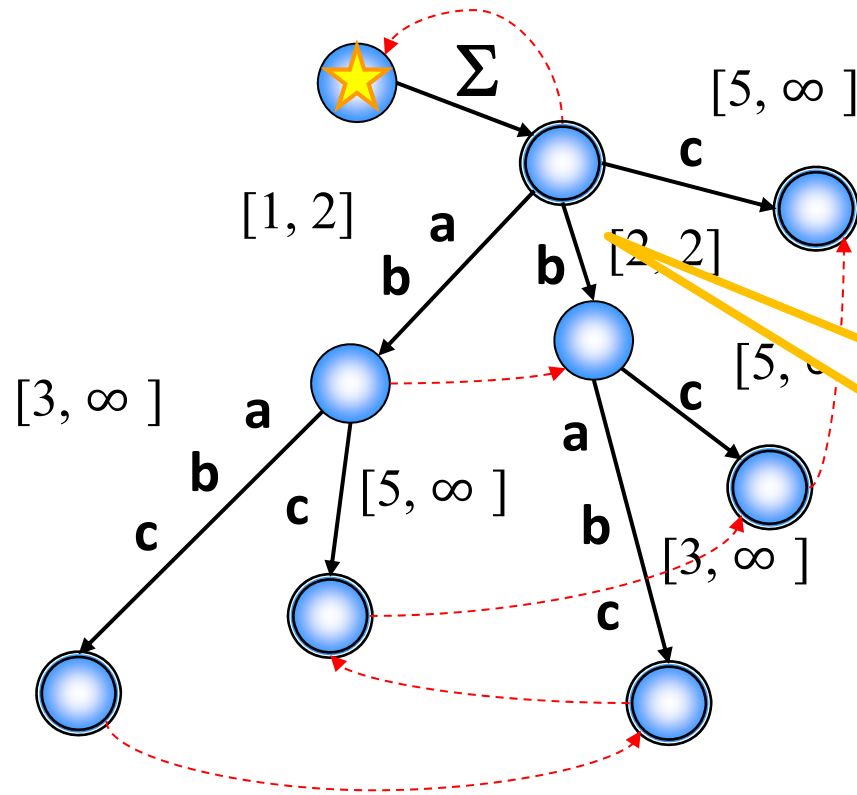


# Ukkonen Tree の左→右オンライン構築



今度は自分の suffix link  
があるので, それを使う.

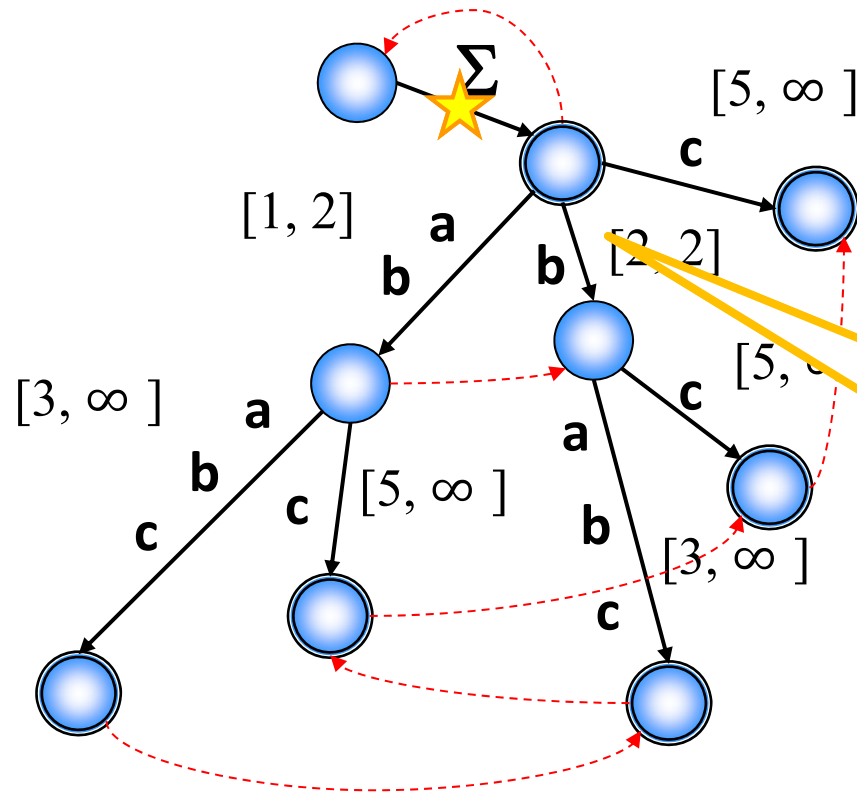
# Ukkonen Tree の左→右オンライン構築



1 2 3 4 5 ...  $\infty$   
**ababc...**

ここから  $w[5] = c$  で  
辿れるので、  
このステップは終わり.

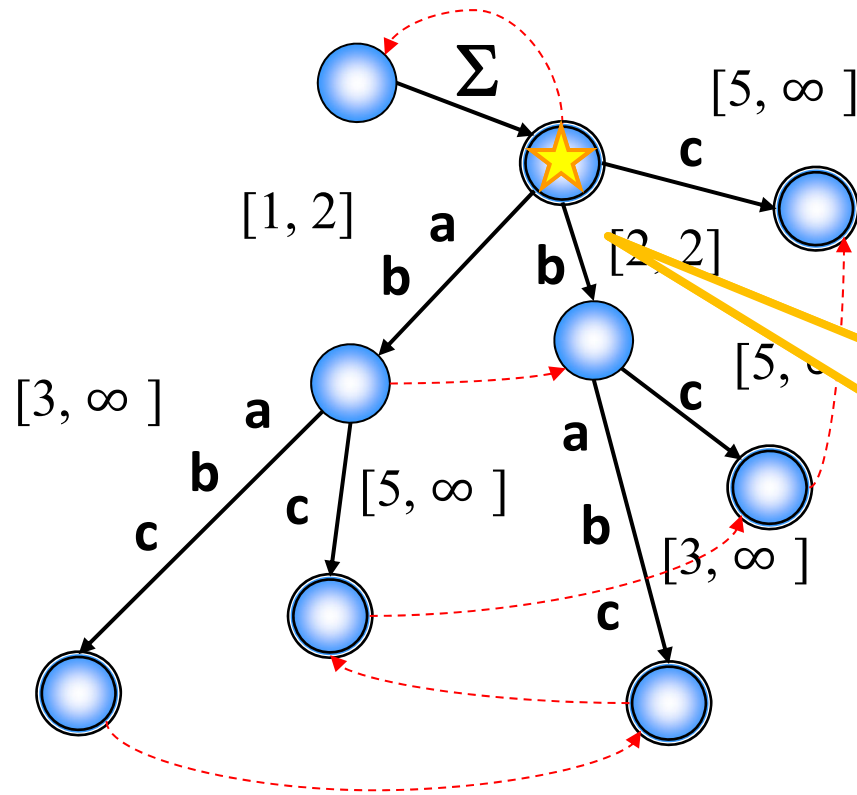
# Ukkonen Tree の左→右オンライン構築



1 2 3 4 5 ... ∞  
**ababc...**

ここから  $w[5] = c$  で  
辿れるので、  
このステップは終わり.

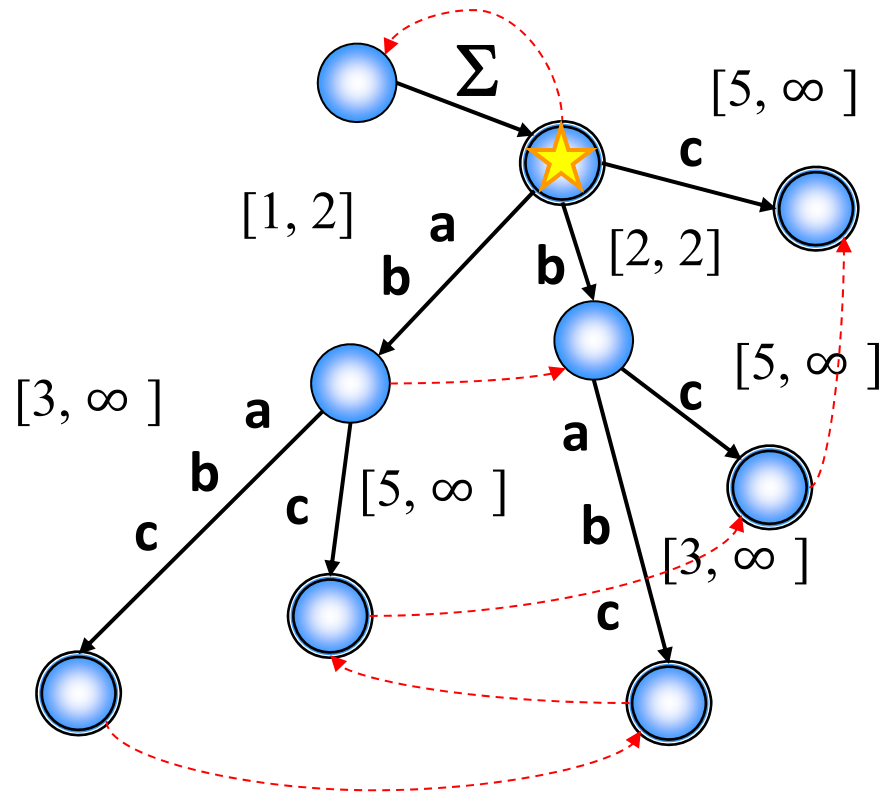
# Ukkonen Tree の左→右オンライン構築



1 2 3 4 5 ...  $\infty$   
**ababc...**

ここから  $w[5] = c$  で  
辿れるので、  
このステップは終わり.

# Ukkonen Tree の左→右オンライン構築



1 2 3 4 5 ...  $\infty$   
**ababc...**



# Ukkonen Tree の左→右オンライン構築

## ちょっと非常識2 [Ukkonen 1995]

Ukkonen tree を  $O(n \log \sigma)$  時間・ $O(n)$  領域で左→右にオンライン構築できる.

- 1ステップで  $O(n)$  個の葉を作る場合があるが、その数は文字列長  $n$  で均せる.
- 親の suffix link を代用するときのコストも  $n$  で均せる.

# Real-time オンライン構築

- これまで紹介したアルゴリズムはすべて、1文字ごとの更新を  $O(\log \sigma)$  ならし時間で行う。  
→ 最悪の場合、1文字あたり  $\Omega(n)$  時間使う。
- ストリーミング処理などの場合には、delay が大きいと嬉しくない。
- そこで、最悪時の更新時間を保証するアルゴリズムがいくつか知られている。  
これを real-time と呼んだりする。

# Weiner tree の Real-time 右→左オンライン構築

## 非常識1 [Breslauer & Italiano, 2013]

定数アルファベットの場合, Weiner tree を  
1文字あたり  $O(\log \log n)$  時間で更新できる.

- Weiner のアルゴリズムを Fringe Nearest Marked Ancestor という特別な NMA に帰着する.
- この FNMA に  $O(\log \log n)$  最悪時間で応答するアルゴリズムを考案.
- $O(\sigma n)$  領域使ってしまう.



# Ukkonen tree の Real-time 左→右オンライン構築

## 非常識2 [Breslauer & Italiano, 2013]

定数アルファベットの場合, Ukkonen tree を  
1文字あたり  $O(\log \log n)$  時間で更新できる.

- 葉を挿入するタスクをスタックに積んでおいて, 新しい文字が来たら定数個(2~3個)の葉を浅いほうから追加する.  
→ active point は木の上から降りてくるので, 浅いほうからメンテしておけばよい.
- 浅いほうから葉を挿入するために, Weiner tree のアルゴリズムをバックグラウンドで走らせる.

# Weiner tree の Real-time 右→左オンライン構築

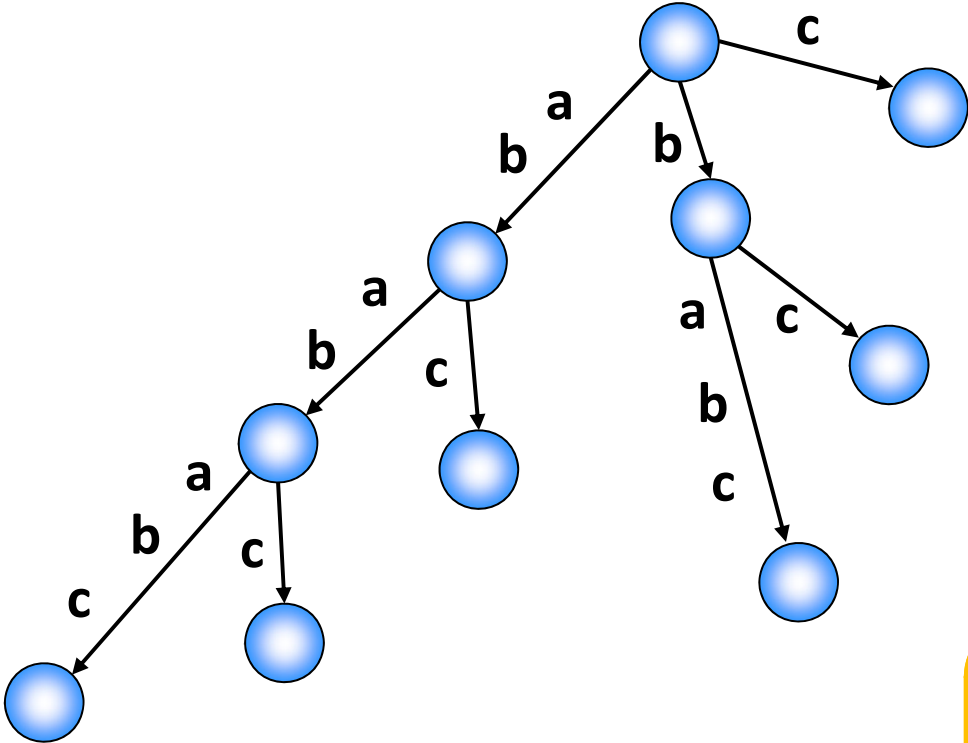
## 黒魔術1 [Fischer & Gawrychowski, 2015]

Weiner tree を1文字あたり

$O(\log\log n + (\log\log \sigma)^2/\log\log\log \sigma)$  時間で更新できる.

- FNMA に  $O(\log\log n)$  最悪時間・ $O(n)$  領域で応答するアルゴリズムを考案.
- 辺の探索に Weighted exponential search tree を使う →  $O((\log\log \sigma)^2/\log\log\log \sigma)$  最悪時間

# スライド窓の Ukkonen tree



1 2 3 4 5 6 7 8 ...  
**abababca...**

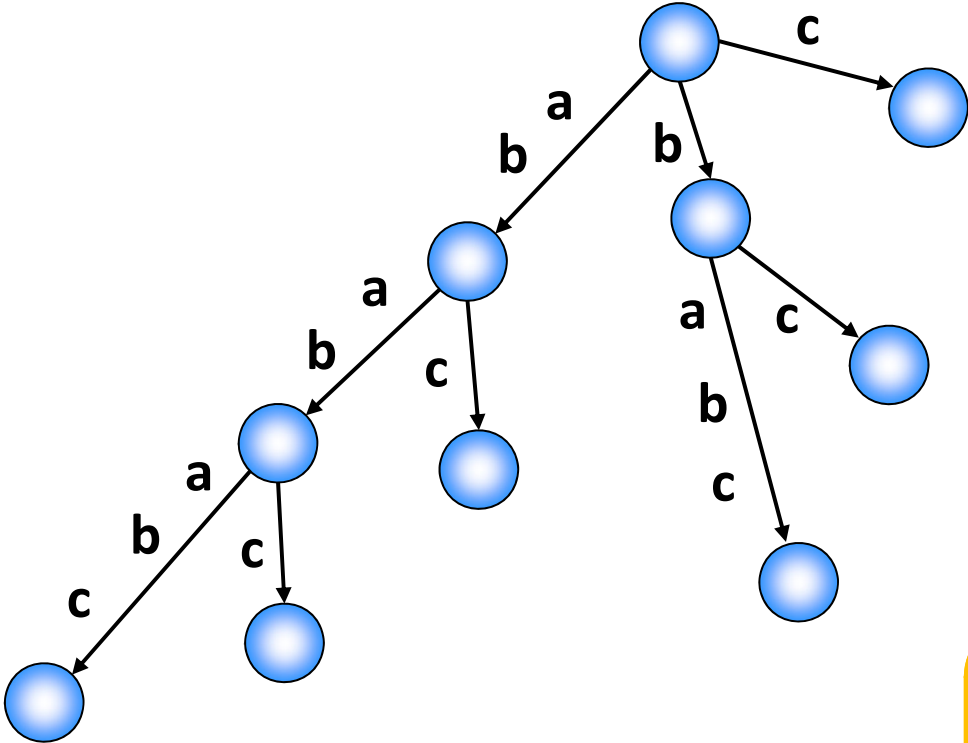


1 2 3 4 5 6 7 8 ...  
**abababca...**

窓の右端をスライドするのは、Ukkonenのアルゴリズムでできる。



# スライド窓の Ukkonen tree

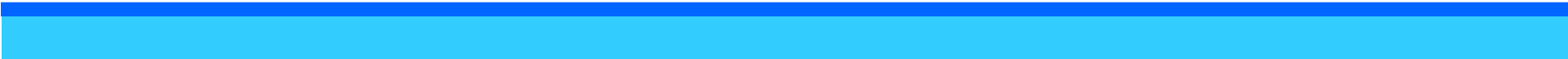


1 2 3 4 5 6 7 8 ...  
**abababca...**

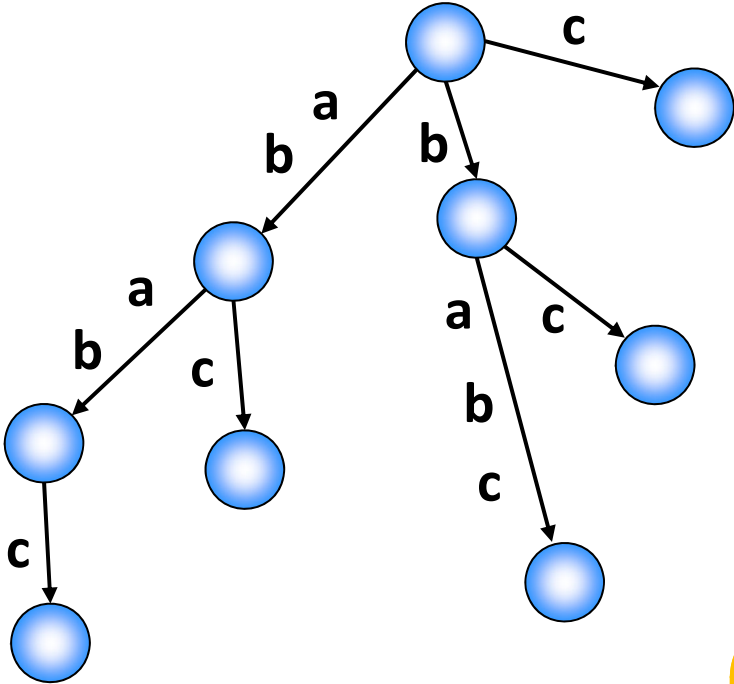


1 2 3 4 5 6 7 8 ...  
**abababca...**

窓の左端をスライドするには、最も長い suffix を表す葉を削除すればよい。



# スライド窓の Ukkonen tree

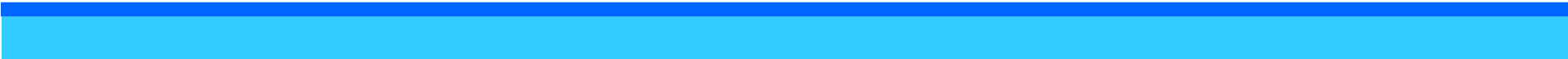


1 2 3 4 5 6 7 8 ...  
**abababca...**

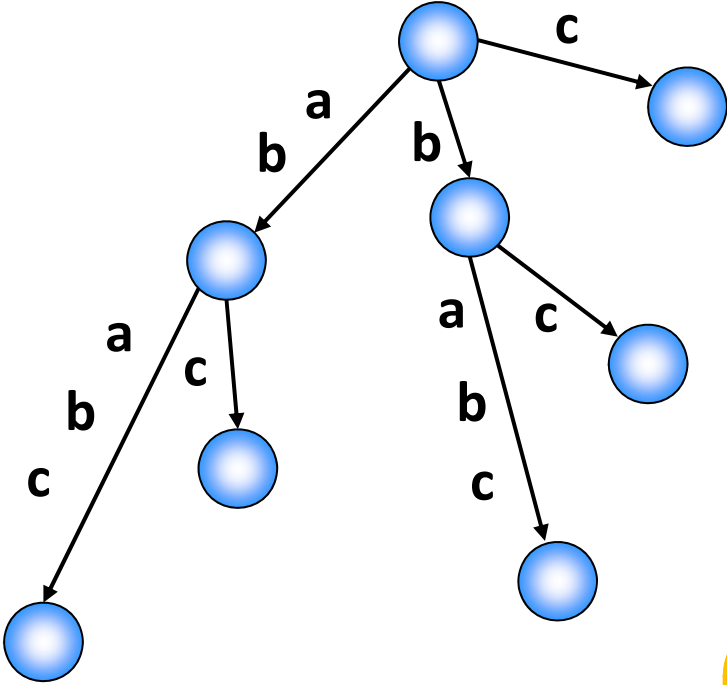


1 2 3 4 5 6 7 8 ...  
**abababca...**

窓の左端をスライドするには、最も長い suffix を表す葉を削除すればよい。



# スライド窓の Ukkonen tree

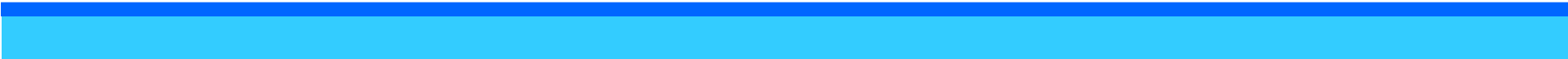


1 2 3 4 5 6 7 8 ...  
**abababca...**



1 2 3 4 5 6 7 8 ...  
**abababca...**

窓の左端をスライドするには、最も長い suffix を表す葉を削除すればよい。



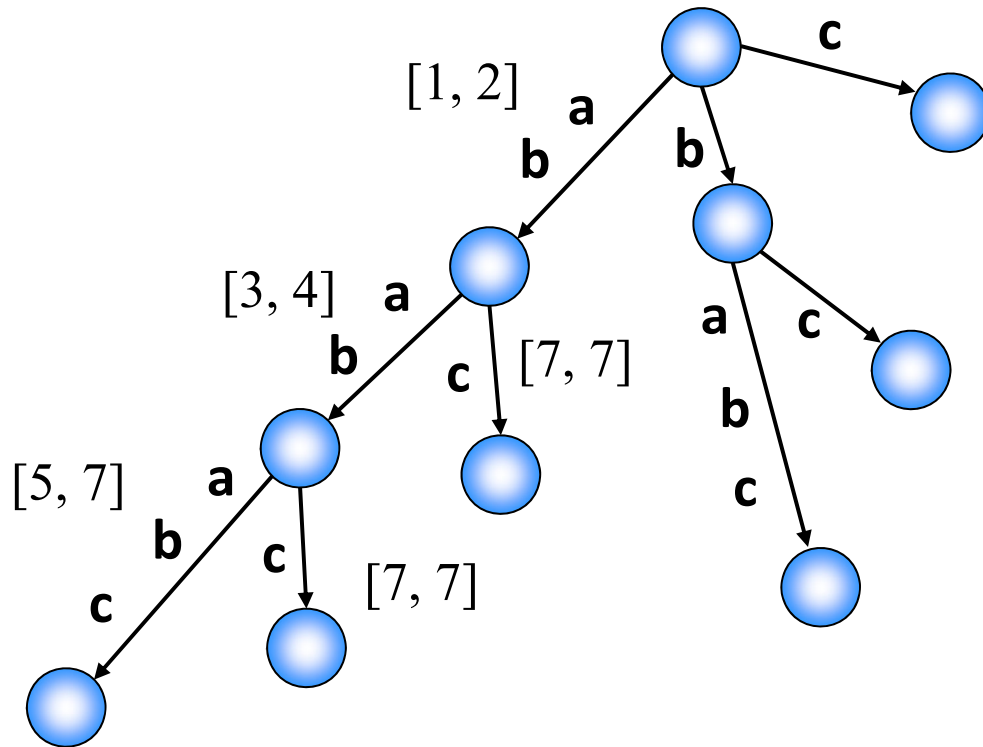
# スライド窓の Ukkonen tree

## ちょっと非常識3 [Larsson 1996]

長さ  $n$  の文字列上の幅  $d$  のスライド窓に対して、 $O(n \log \sigma)$  時間・ $O(d)$  領域で Ukkonen tree を左→右にオンラインに構築できる。

- 基本的に前述の通りだが、  
辺ラベルのエンコードに気をつける必要あり。  
→ ならし  $O(n)$  時間で常に窓内の位置を参照するようにメンテできる [Fiala & Greene 1989].

# スライド窓の Ukkonen tree



1 2 3 4 5 6 7 8 ...  
**abababca...**

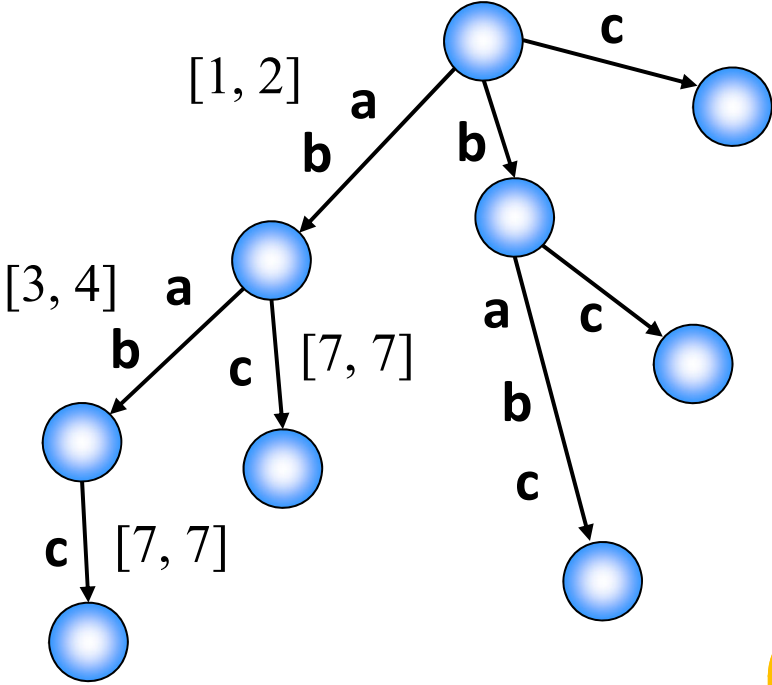


1 2 3 4 5 6 7 8 ...  
**abababca...**

窓の左端をスライドする  
には、最も長い suffix を  
表す葉を削除すればよい。



# スライド窓の Ukkonen tree



1 2 3 4 5 6 7 8 ...  
**abababca...**



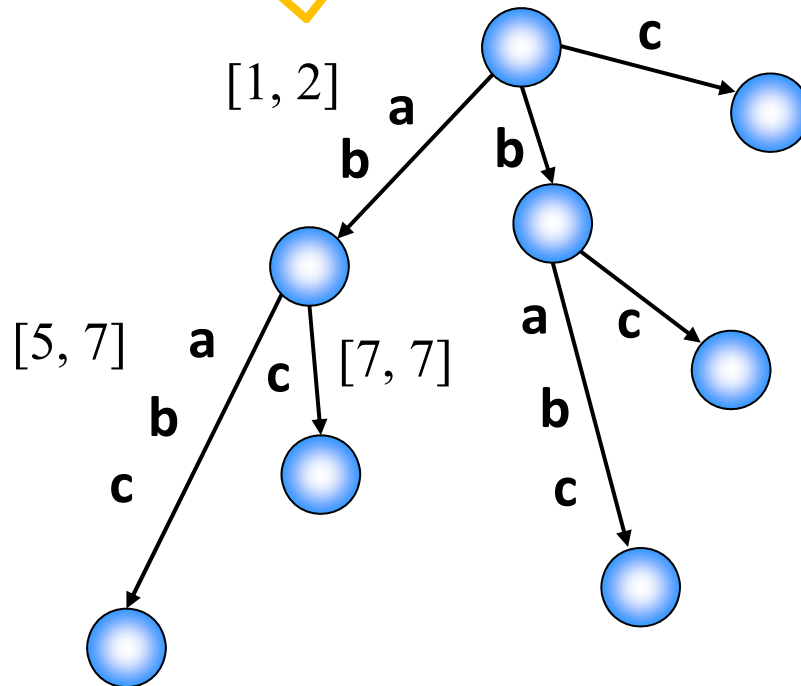
1 2 3 4 5 6 7 8 ...  
**abababca...**

窓の左端をスライドするには、最も長い suffix を表す葉を削除すればよい。



[1, 2] の 1 が窓の外!

# Ukkonen tree



1 2 3 4 5 6 7 8 ...  
**abababca...**

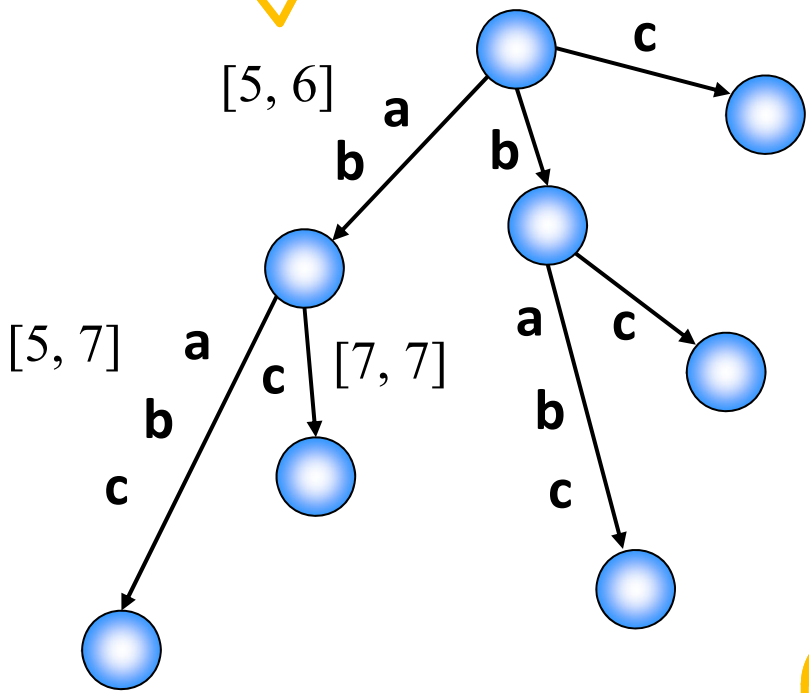


1 2 3 4 5 6 7 8 ...  
**abababca...**

窓の左端をスライドするには、最も長い suffix を表す葉を削除すればよい。

# Ukkonen tree

[5, 6] に更新



1 2 3 4 5 6 7 8 ...  
**abababca...**



1 2 3 4 5 6 7 8 ...  
**abababca...**

窓の左端をスライドするには、最も長い suffix を表す葉を削除すればよい。



# スライド窓の DAWG

ちょっと非常識4 [J. Blumer 1987]

長さ  $n$  の文字列上の幅  $d$  のスライド窓に対する DAWG の左→右オンライン構築には、1文字あたり  $\Omega(d)$  時間かかる。

- 1文字スライドするごとに、 $\Omega(d)$  個の辺・頂点を更新しなければならない嫌な例がある。


# スライド窓の Weiner tree

## ちょっと非常識5


長さ  $n$  の文字列上の幅  $d$  のスライド窓に対する Weiner tree の右→左オンライン構築には、1文字あたり  $\Omega(d)$  時間かかる。

- ちょっと非常識4よりただちに成り立つ。

# その他の関連する話題 (1/3)

- Suffix tree の双方向オンライン構築 [Inenaga 2003]
  - CDAWG の左→右オンライン構築 [Inenaga et al. 2005]
  - CDAWG の左→右スライド窓  
[Inenaga et al. 2004, Senft 2008]
  - 複数文字列に対する Weiner tree の  
右→左オンライン構築 [Takagi et al. 2017]
  - 複数文字列に対する DAWG, Ukkonen tree の  
左→右オンライン構築 [Takagi et al. 2017]
- 

# その他の関連する話題 (2/3)

- Affix tree の双方向オンライン構築 [Maass 2003]
  - Suffix trist の左→右オンライン構築 [Cole et al. 2013]
  - Position heap の右→左オンライン構築 [Ehrenfeucht et al., 2011]
  - Position heap の左→右オンライン構築 [Kucherov 2013]
  - Linear-size suffix trie の左→右および右→左オンライン構築 [Diptarama et al. 2019]
- 

# その他の関連する話題 (3/3)

- Parameterized suffix tree の左→右オンライン構築  
[Shibuya 2004, Lee et al. 2011]
- Parameterized suffix tree の右→左オンライン構築  
[Fujisato et al. 2019 (unpublished)]

忘れてる/知らないのもまだあるかも...?  
サーベイ論文書こうと思います。