Summer School @ U. Helsinki, 2016

The myriad virtues of DAWGs

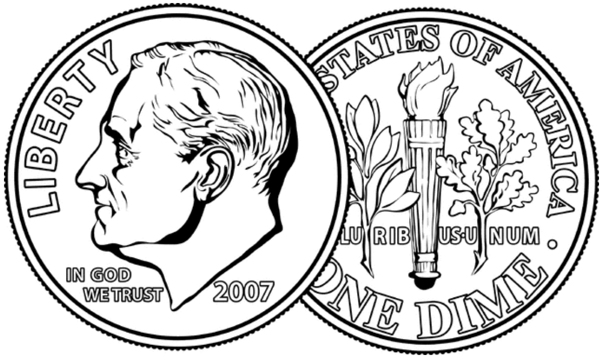
Shunsuke Inenaga
Kyushu University, Japan

Overview of This Course

- Suffix tree [Weiner, 1973] is a fundamental data structure for string processing.
- "The myriad virtues of subword trees" [Apostolico, 1985]
- Directed acyclic word graph (DAWG) [Blumer et al., 1985] is a "dual" data structure for suffix tree.
- In this course, we study some nice properties and usefulness of DAWGs.

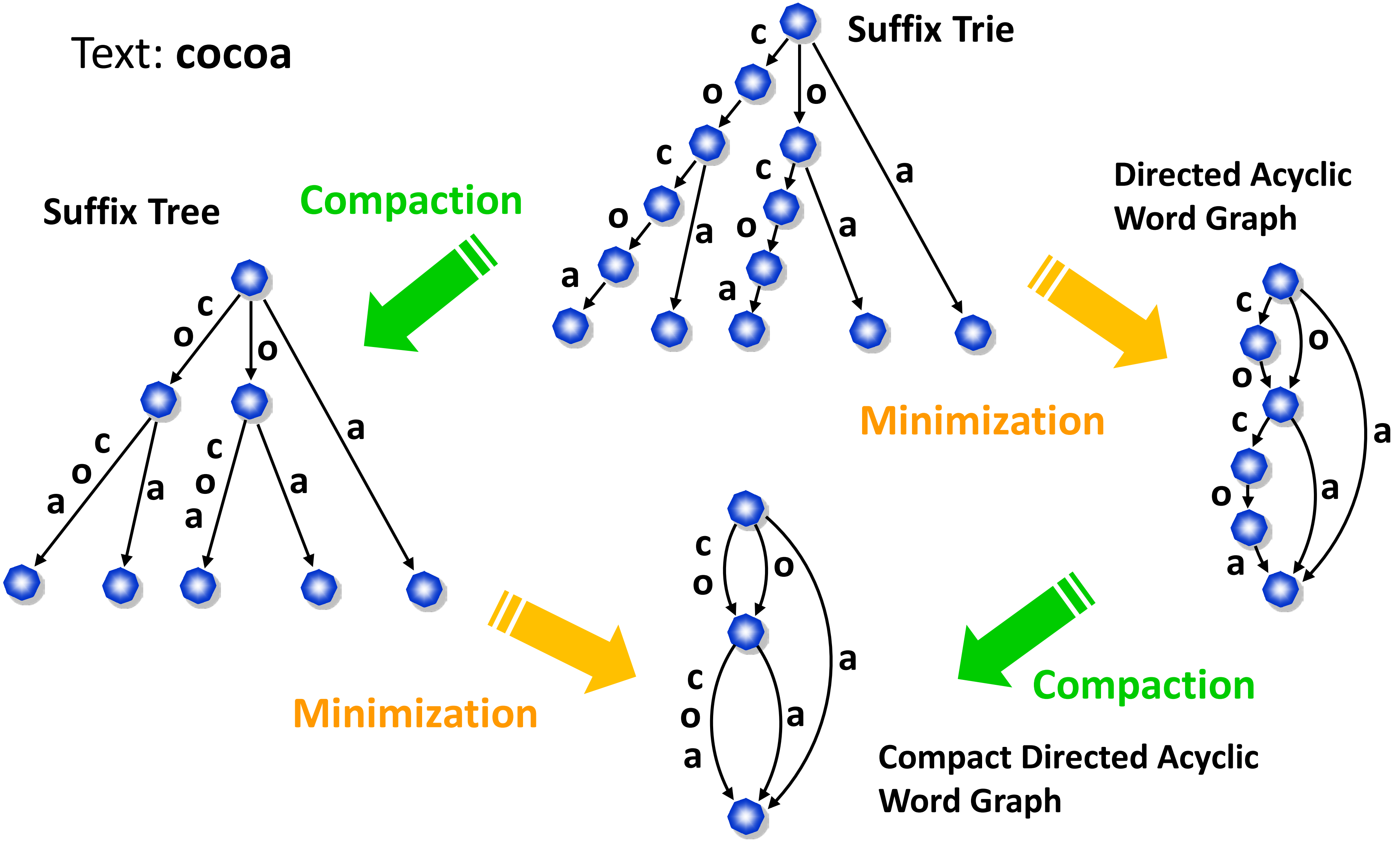
Relation to Previous Course

- This course will share some consequences with the previous course by Djamal & Fabio.
- This happens due to the “duality” between DAWGs and suffix trees, i.e., we will see “the same coin from both sides”.



Text Indexing Structures

Text: **cocoa**



Suffix Trie

Suffix Tree

Directed Acyclic Word Graph

Compact Directed Acyclic Word Graph

Compaction

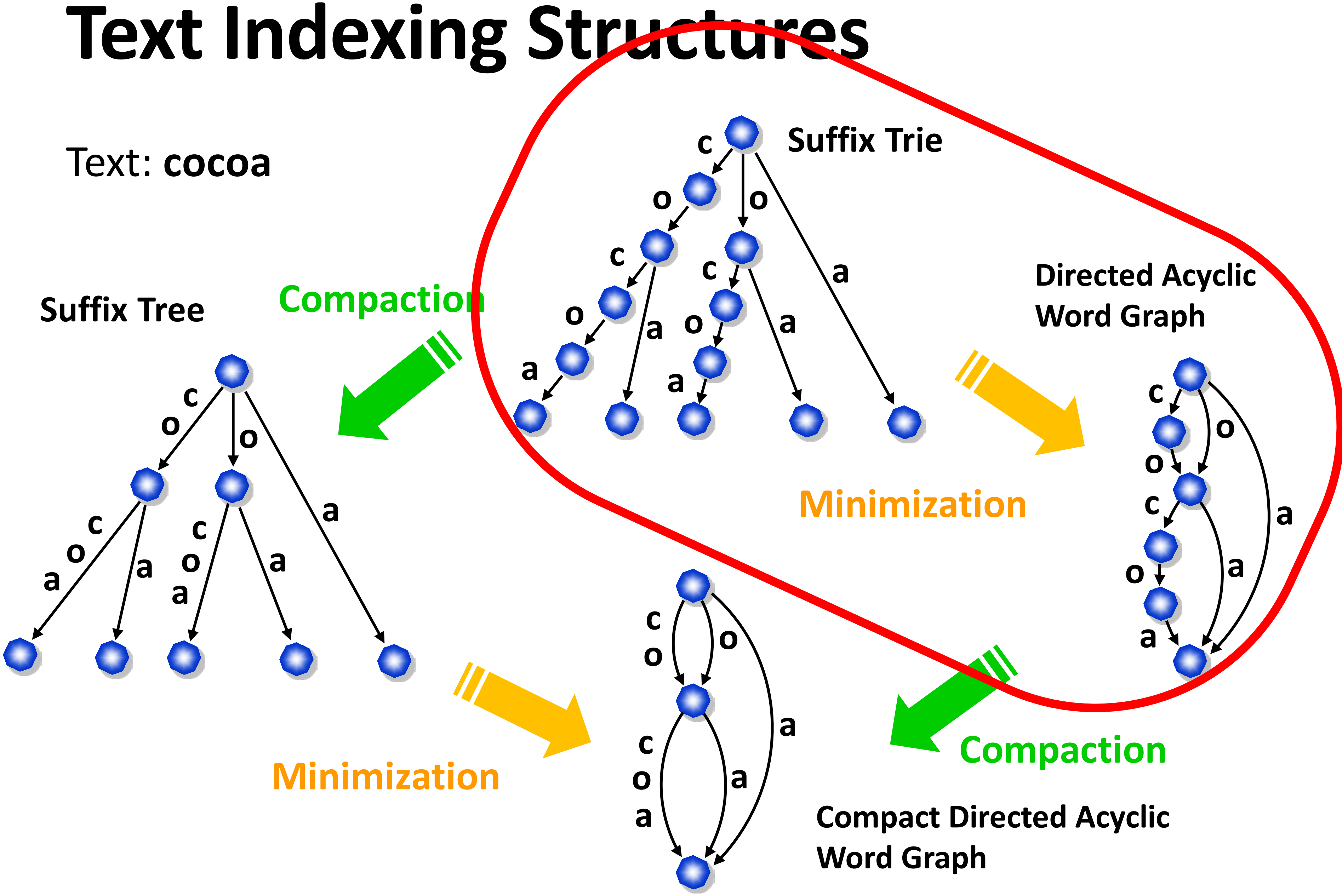
Minimization

Minimization

Compaction

Text Indexing Structures

Text: **cocoa**



Suffix Trie

Suffix Tree

Directed Acyclic Word Graph

Compact Directed Acyclic Word Graph

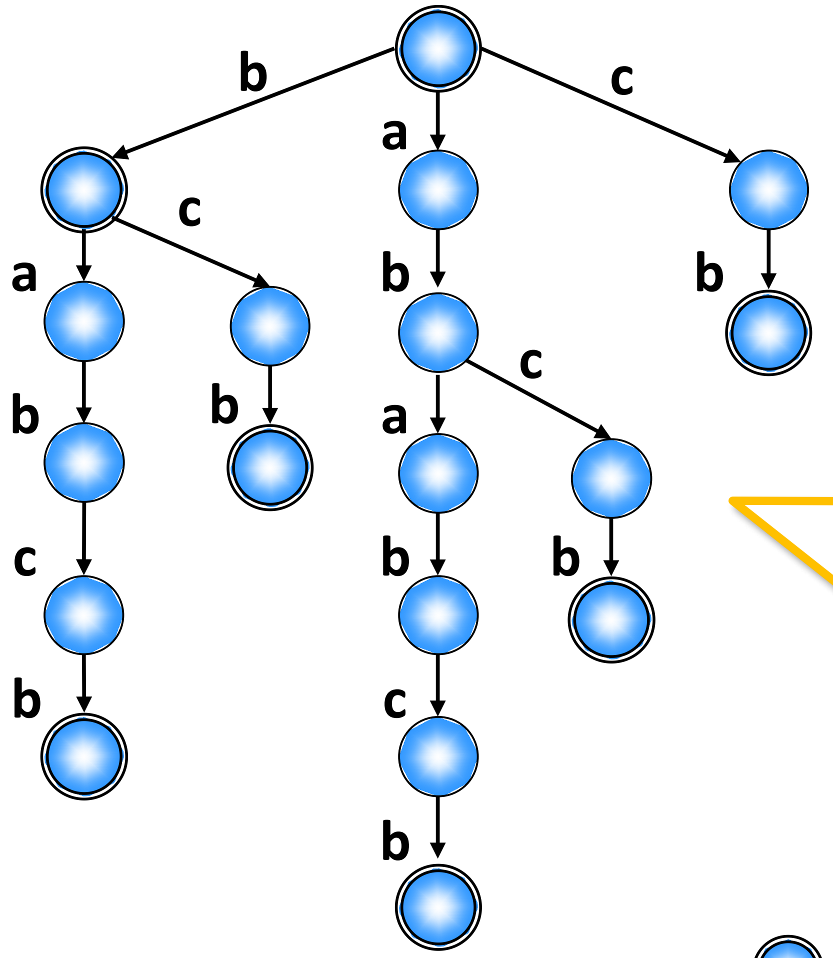
Compaction

Minimization

Minimization

Compaction

From Suffix Trie to DAWG

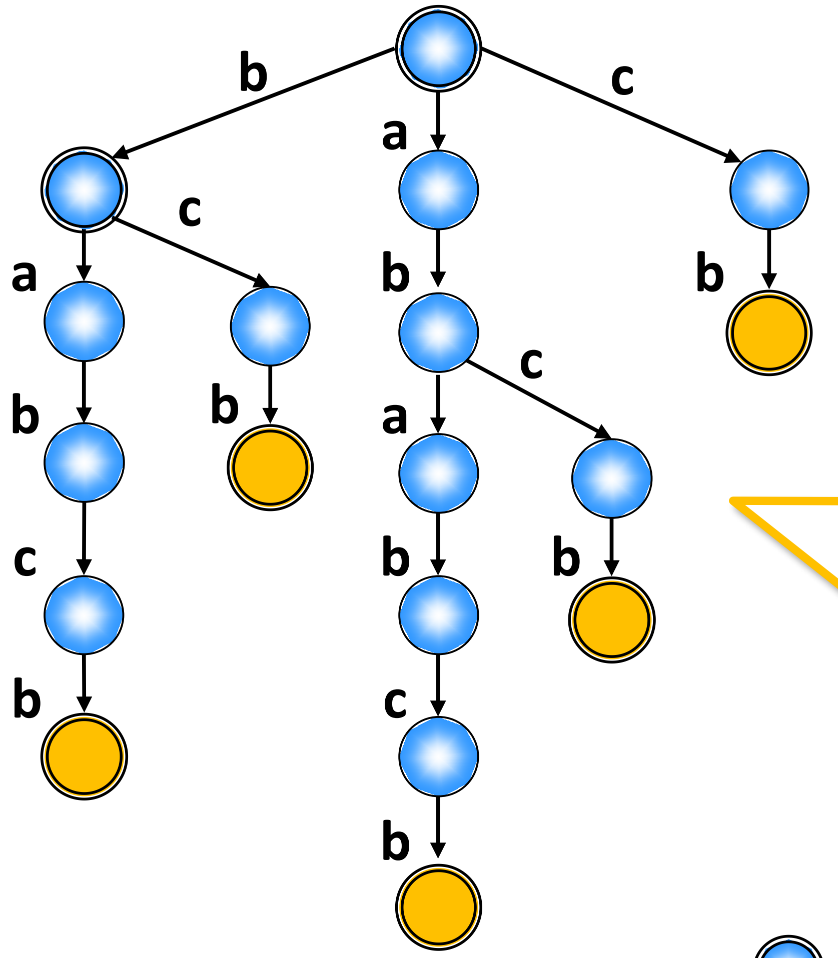


ababcb

Merge isomorphic subtrees (subgraphs) bottom up.

● accepting nodes for suffixes

From Suffix Trie to DAWG

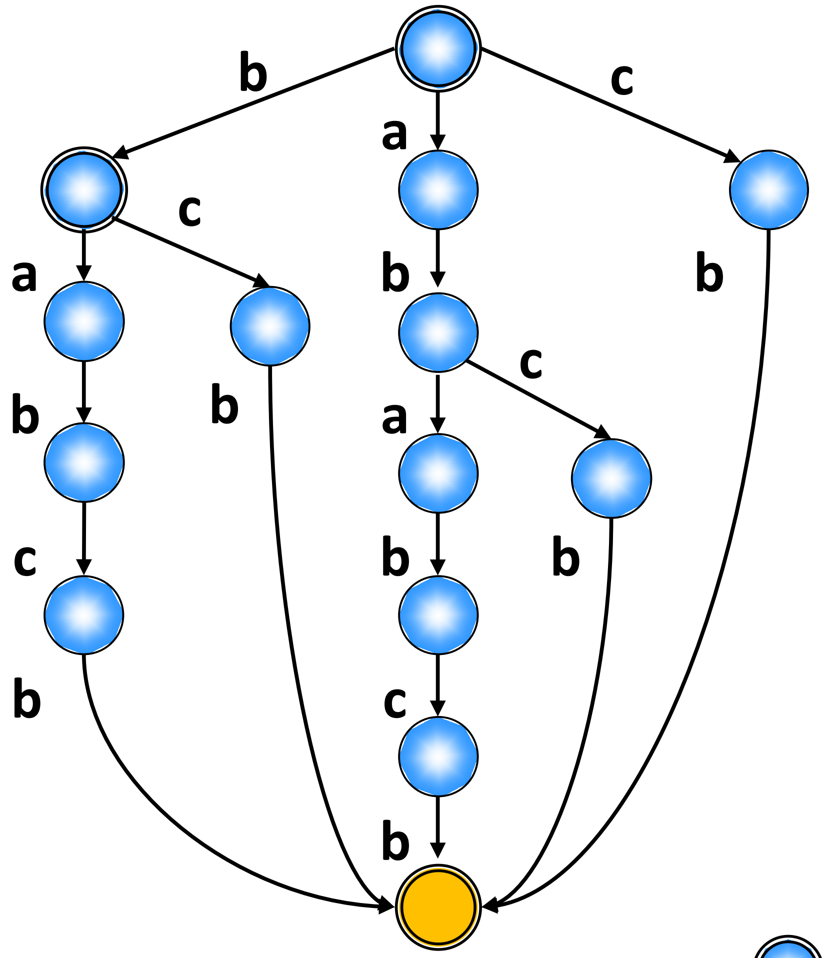


ababcb

Merge isomorphic subtrees (subgraphs) bottom up.

◎ accepting nodes for suffixes

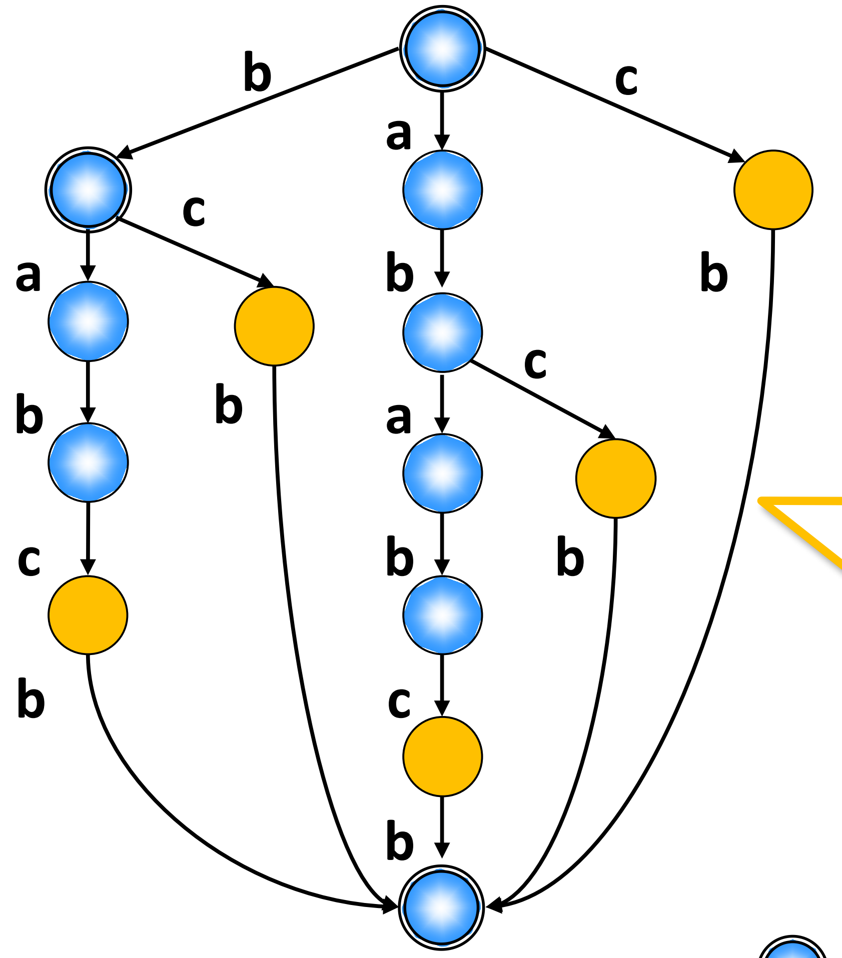
From Suffix Trie to DAWG



ababcb

◎ accepting nodes for suffixes

From Suffix Trie to DAWG

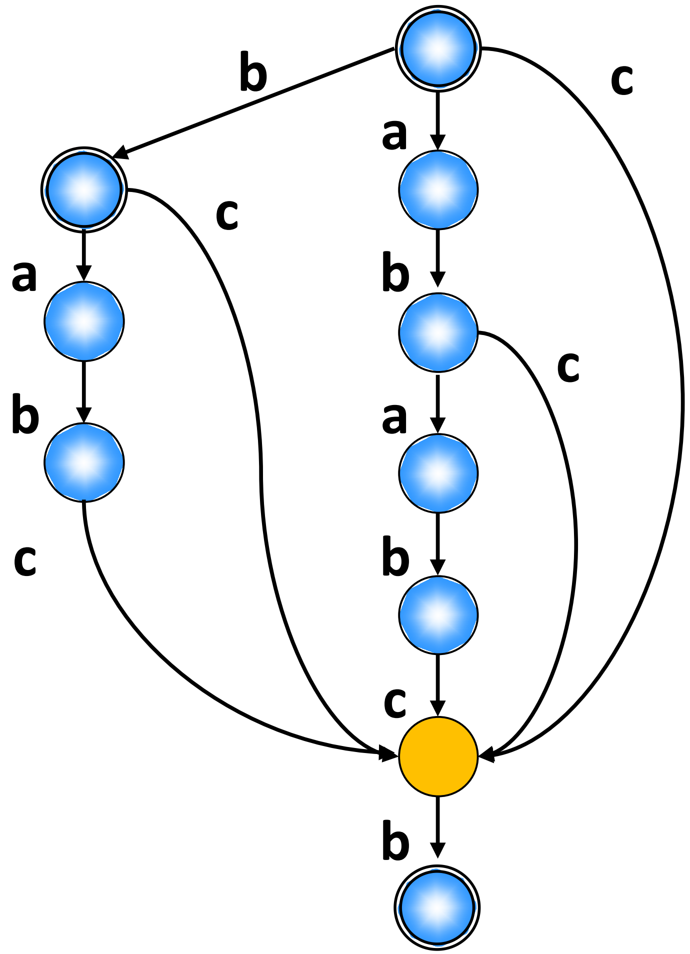


ababcb

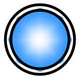
Merge isomorphic subtrees (subgraphs) bottom up.

● accepting nodes for suffixes

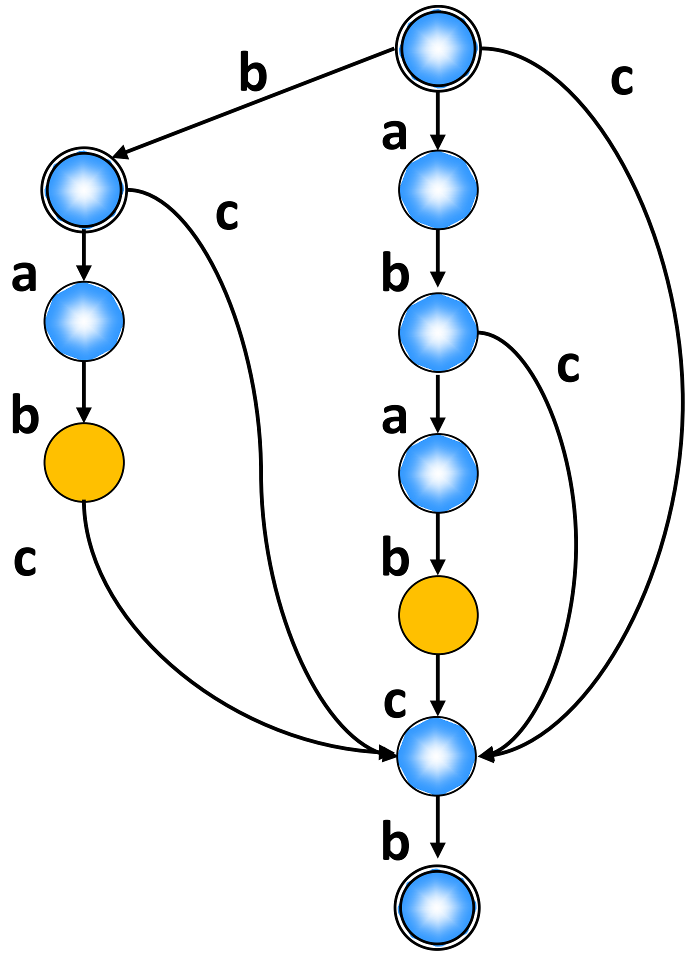
From Suffix Trie to DAWG



ababcb



From Suffix Trie to DAWG

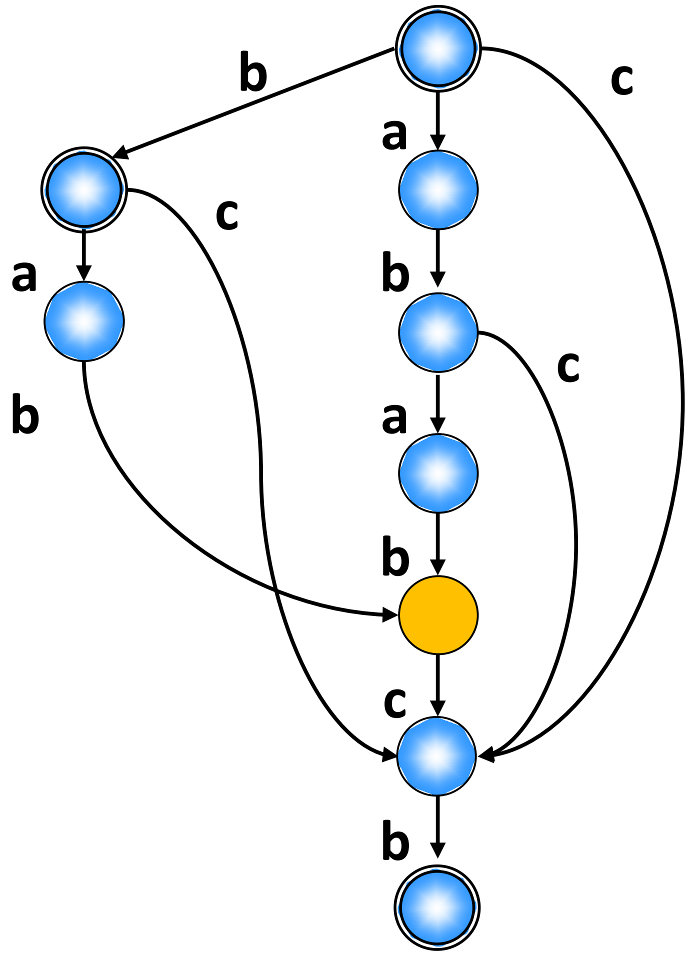


ababcb

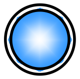
Merge isomorphic subtrees (subgraphs) bottom up.

◎ accepting nodes for suffixes

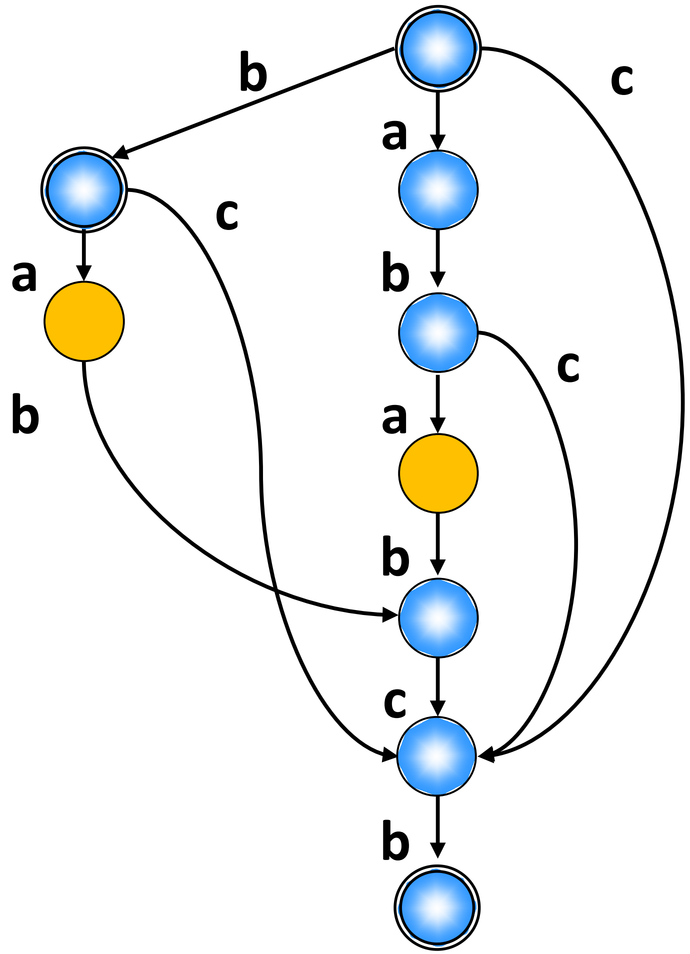
From Suffix Trie to DAWG



ababcb

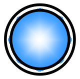


From Suffix Trie to DAWG

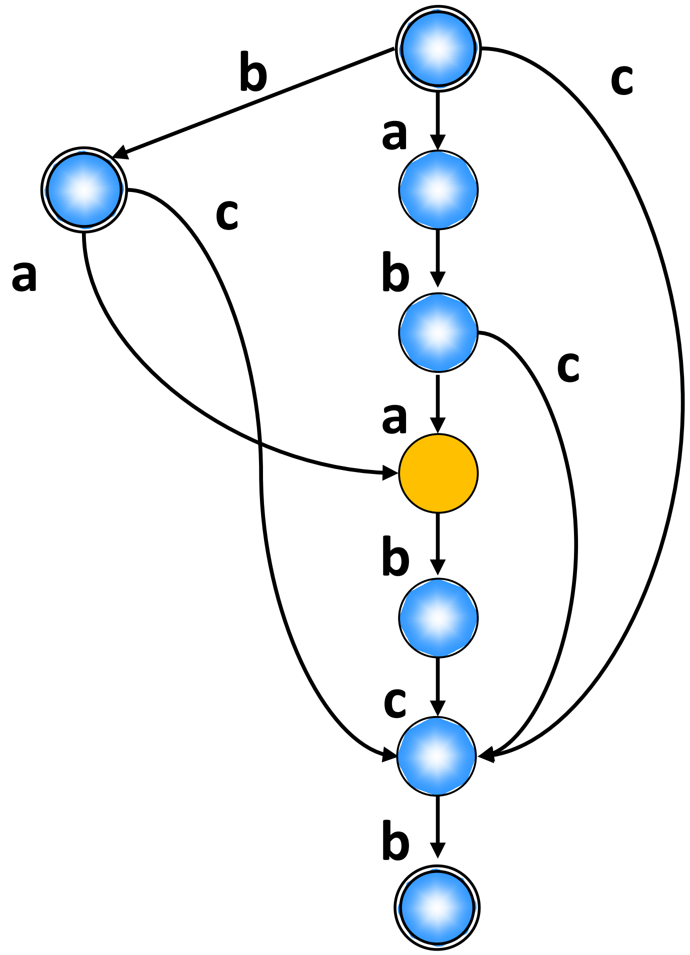


ababcb

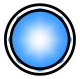
Merge isomorphic subtrees (subgraphs) bottom up.


accepting nodes for suffixes

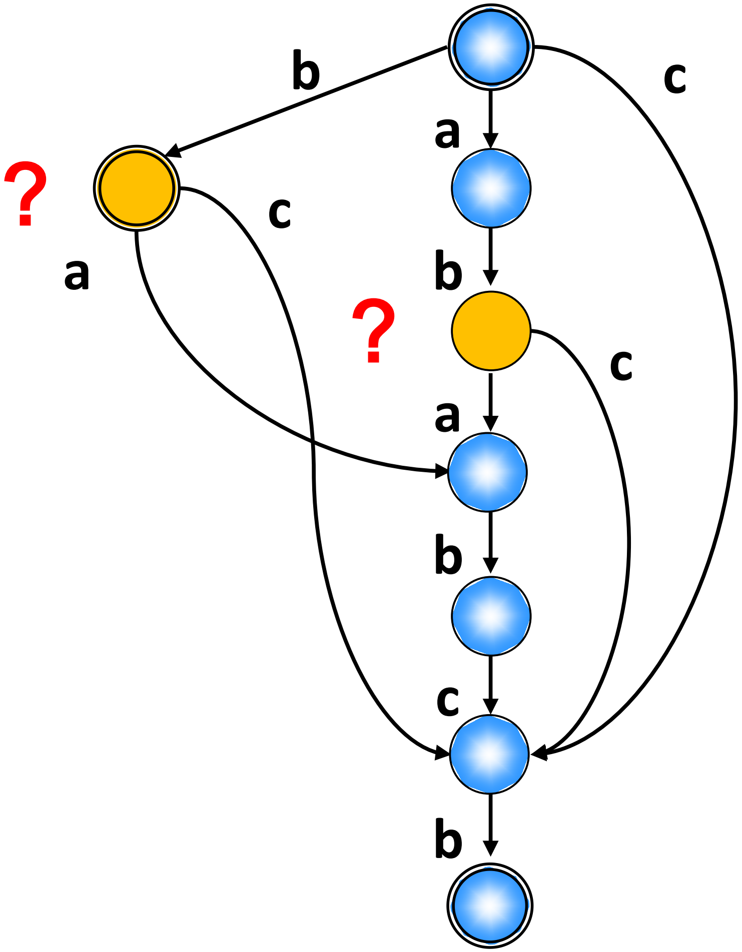
From Suffix Trie to DAWG



ababcb

 accepting nodes for suffixes

From Suffix Trie to DAWG

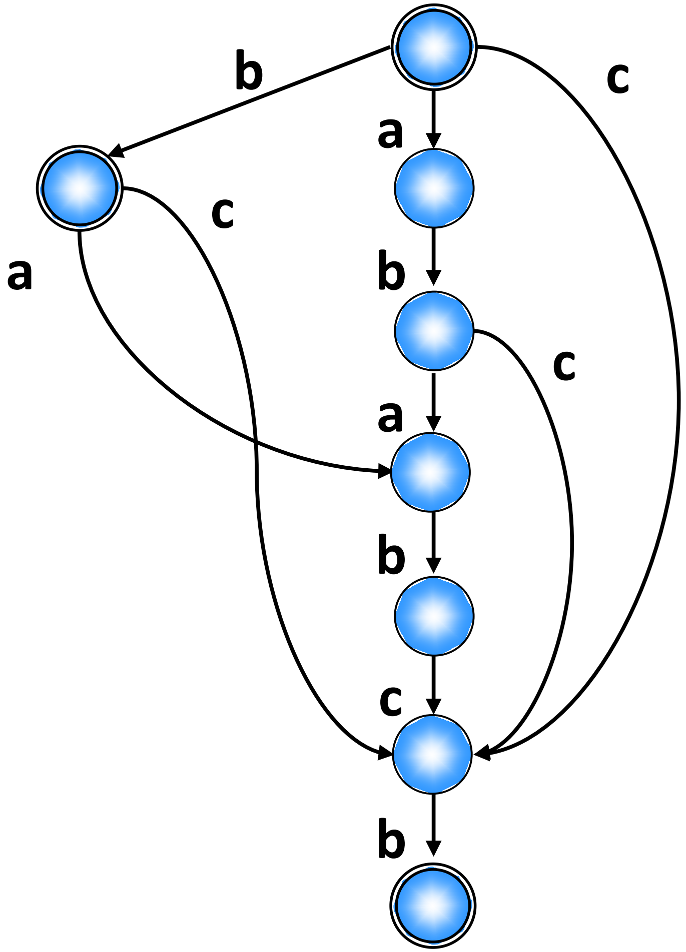


ababcb

One is accepting state, while the other isn't. Don't merge them.

● accepting nodes for suffixes

From Suffix Trie to DAWG



ababcb

DAWG of string
ababcb

◎ accepting nodes for suffixes

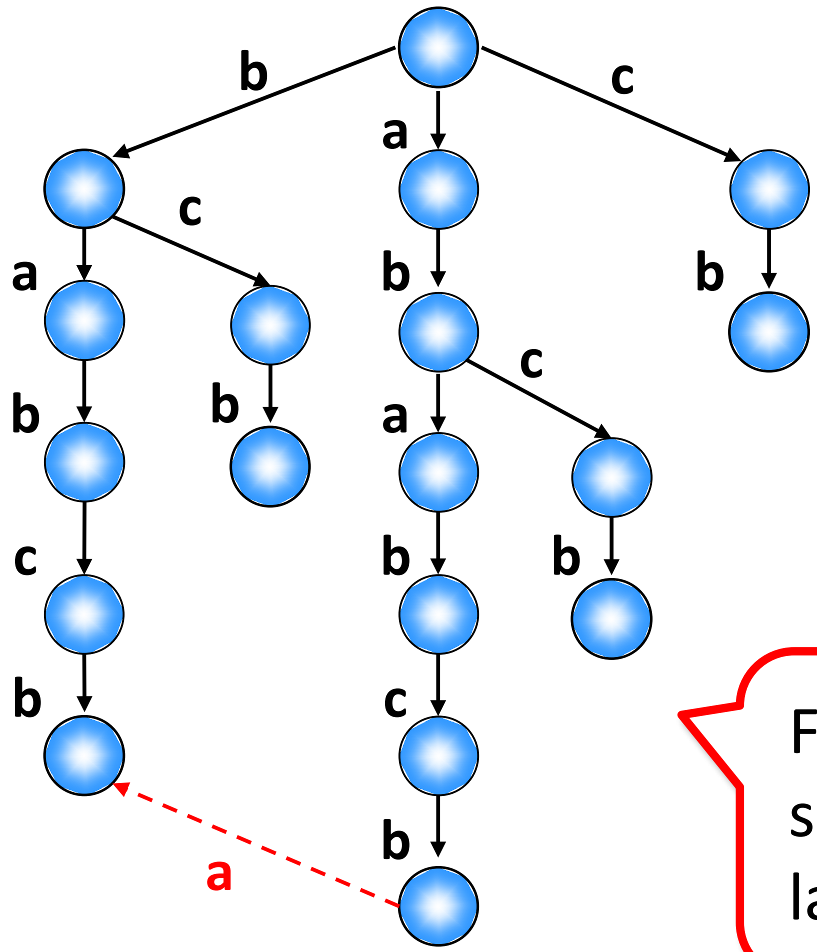
Minimality of DAWG

Theorem 1

The DAWG of string w is the **smallest automaton** that recognizes all suffixes of w.

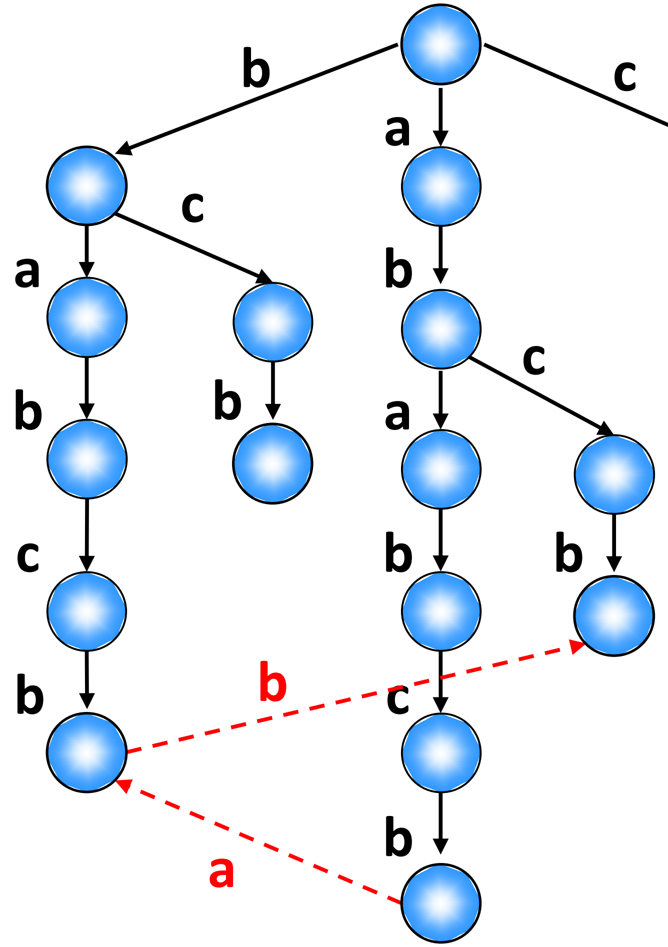
- Clearly, the suffix trie of w recognizes all suffixes of w, so does the DAWG of w.
- By construction, the DAWG is the smallest such automaton.

Suffix Links of Suffix Trie



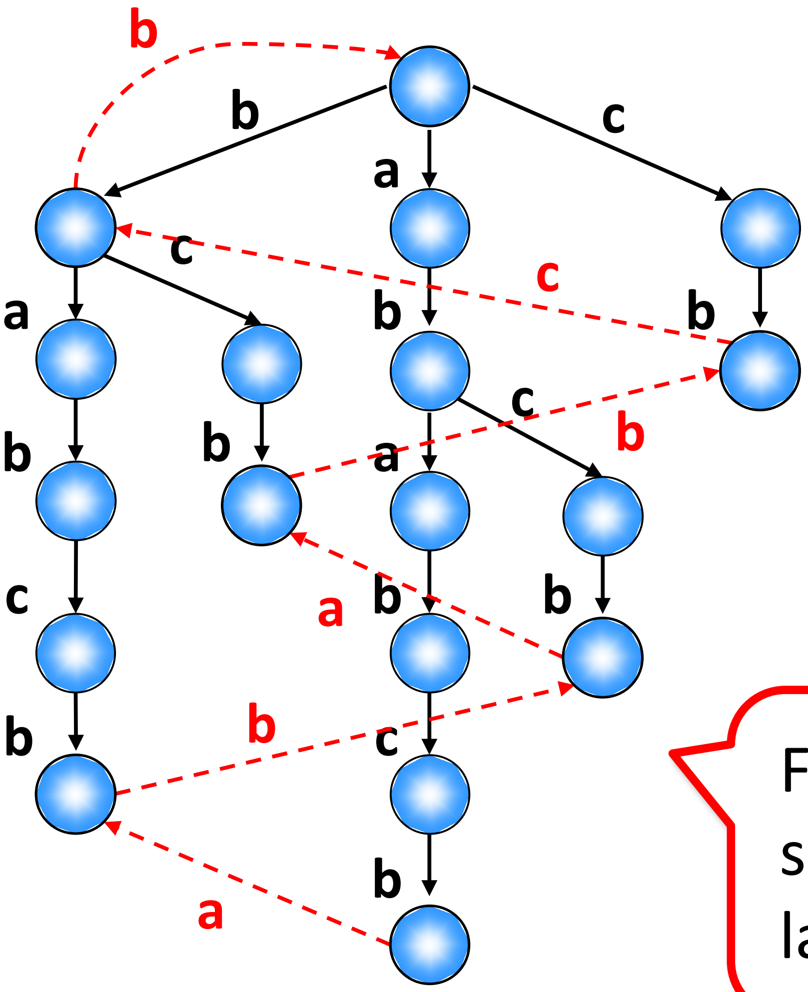
For letter ***a*** and string ***x***, suffix link from node ***ax*** is labeled ***a*** and goes to node ***x***.

Suffix Links of Suffix Trie



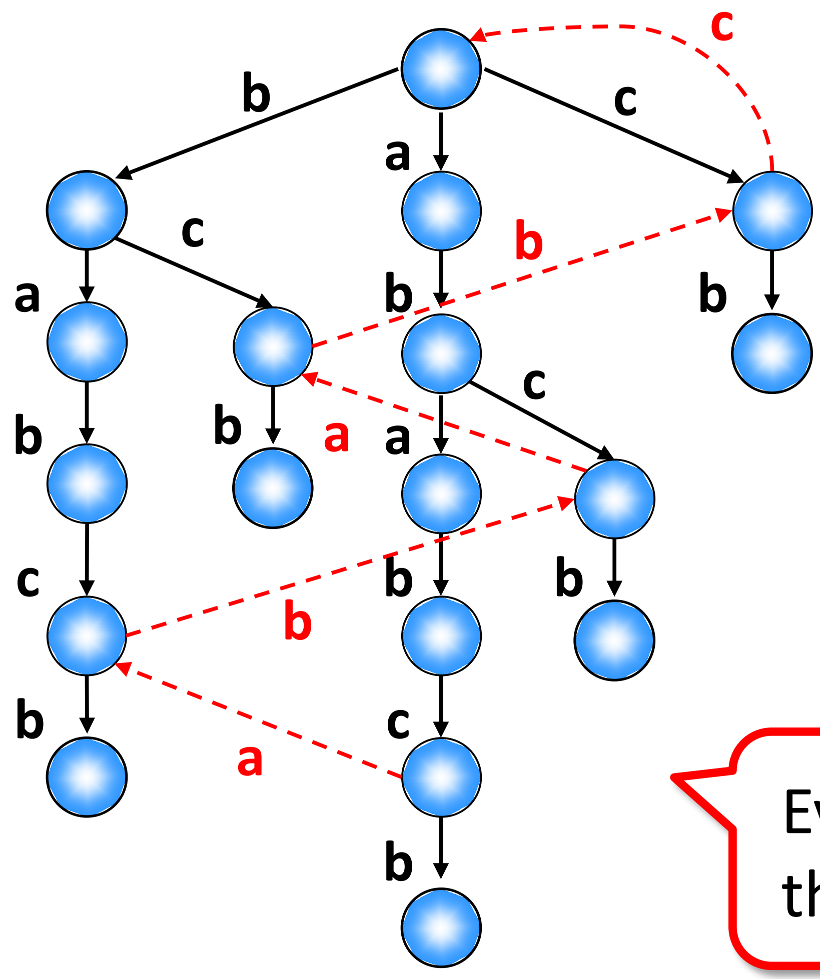
For letter a and string x, suffix link from node ax is labeled a and goes to node x.

Suffix Links of Suffix Trie



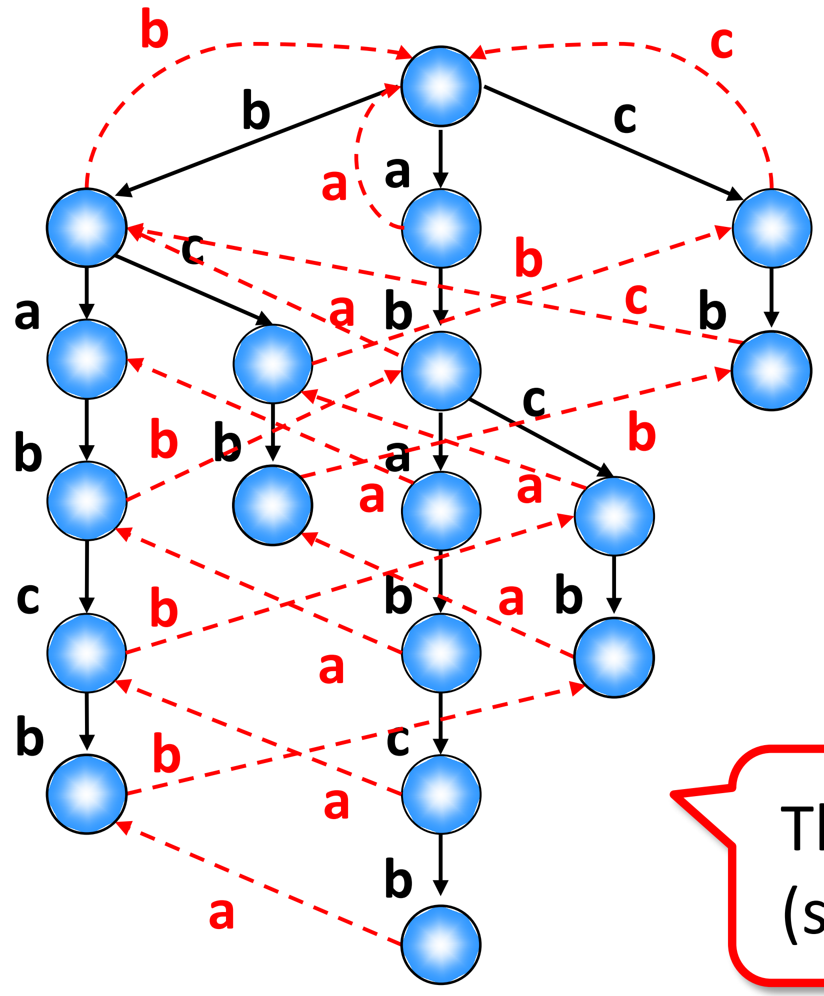
For letter ***a*** and string ***x***, suffix link from node ***ax*** is labeled ***a*** and goes to node ***x***.

Suffix Links of Suffix Trie



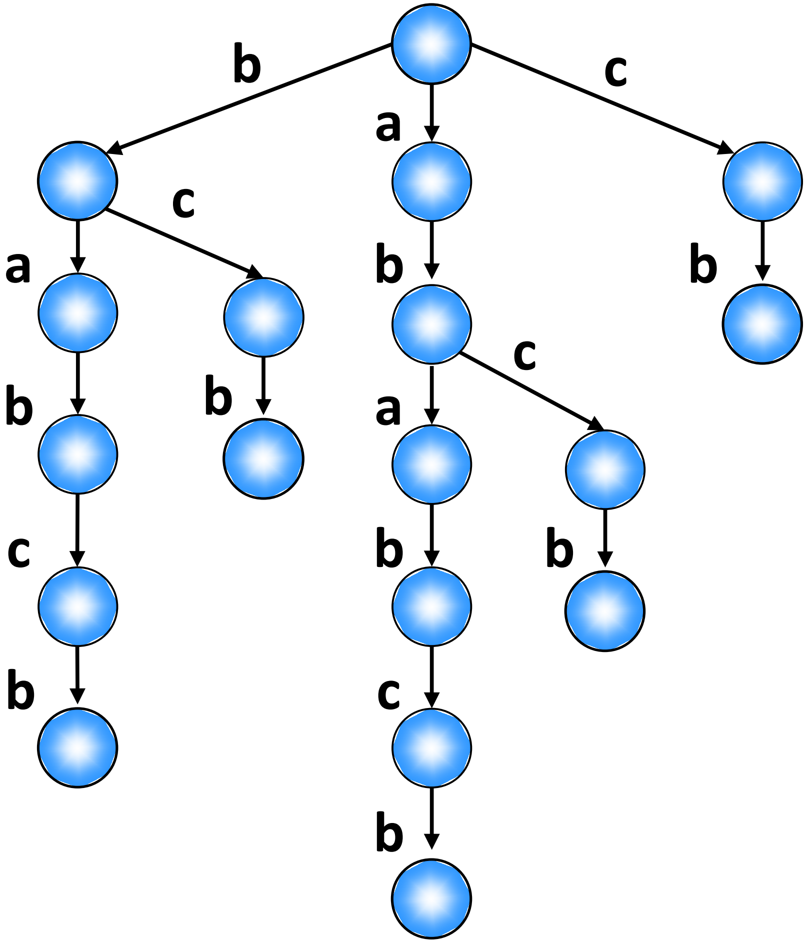
Every suffix link path reaches the root.

Suffix Link Tree of Suffix Trie

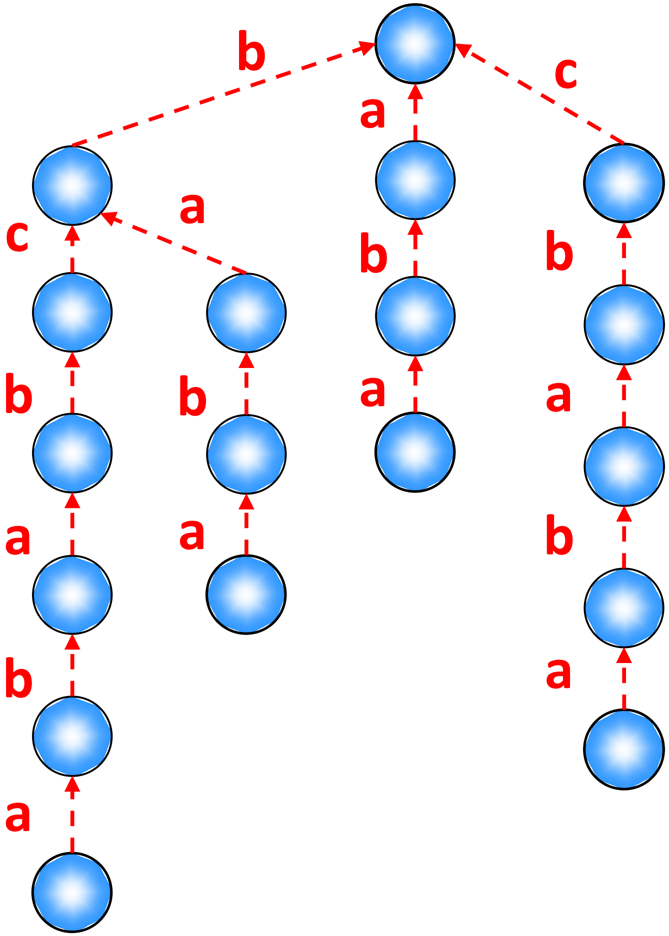


The suffix links form a tree (suffix link tree).

Suffix Link Tree of Suffix Trie

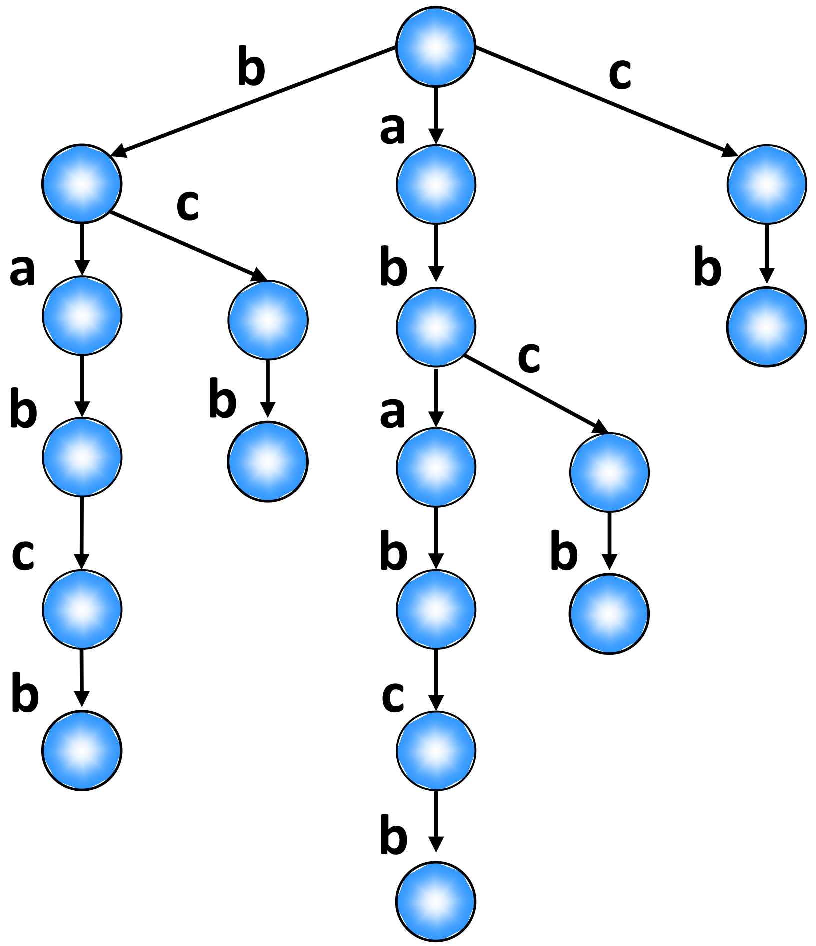


Suffix Trie of **ababcb**

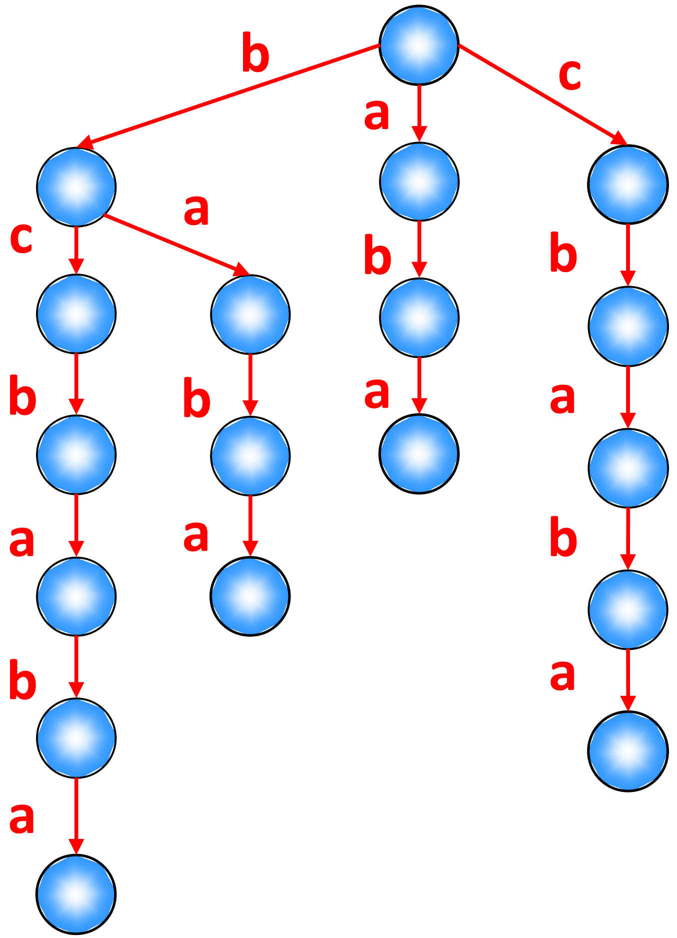


Suffix Link Tree of **ababcb**

Suffix Link Tree = Suffix Trie of reverse



Suffix Trie of **ababcb**



Suffix Trie of **bcbaba**

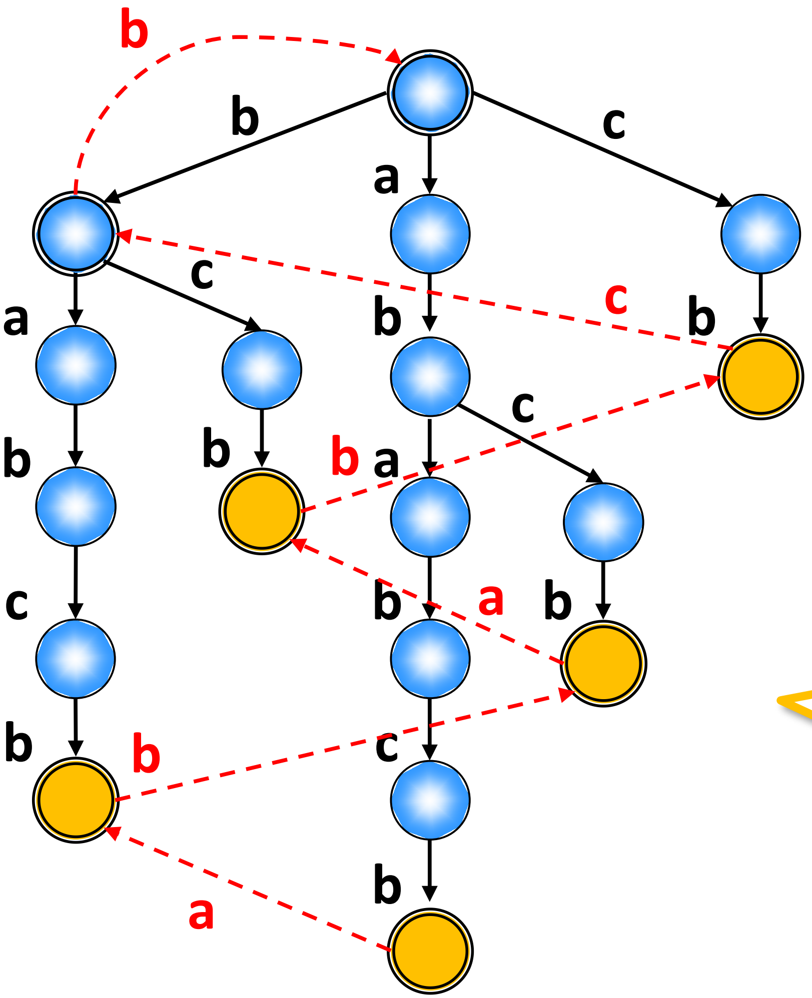
Suffix Link Tree = Suffix Trie of reverse

Lemma 1

The suffix link tree of the suffix trie of string w forms the suffix trie of reversed string w^R.

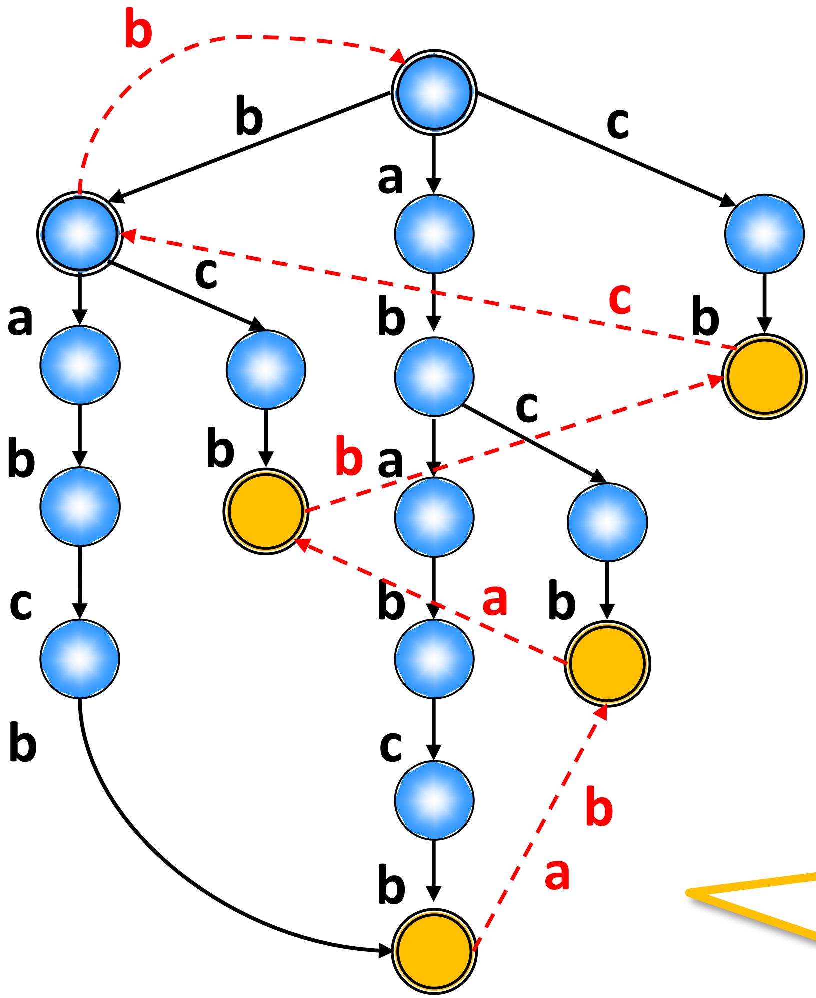
- For any node of the suffix trie of w which represents substring x of w, the node represents subtring x^R of w^R in the suffix link tree.

Node Merges and Suffix Links



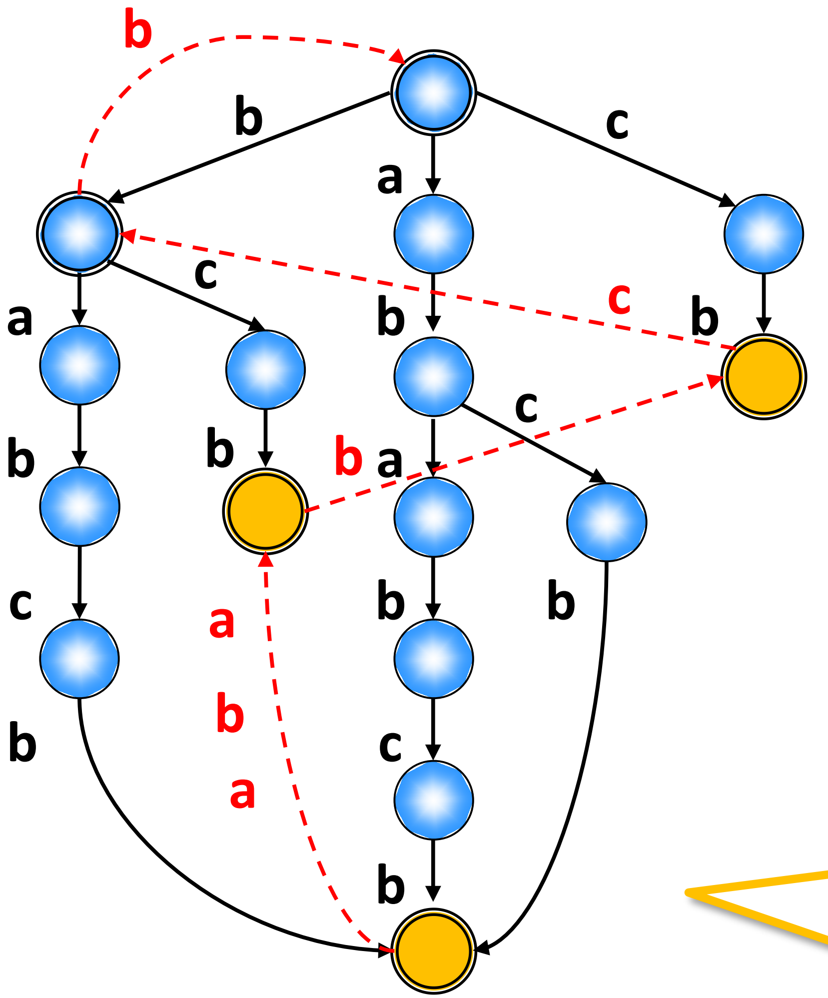
Merged nodes are connected by a chain of suffix links.

Node Merges and Suffix Links



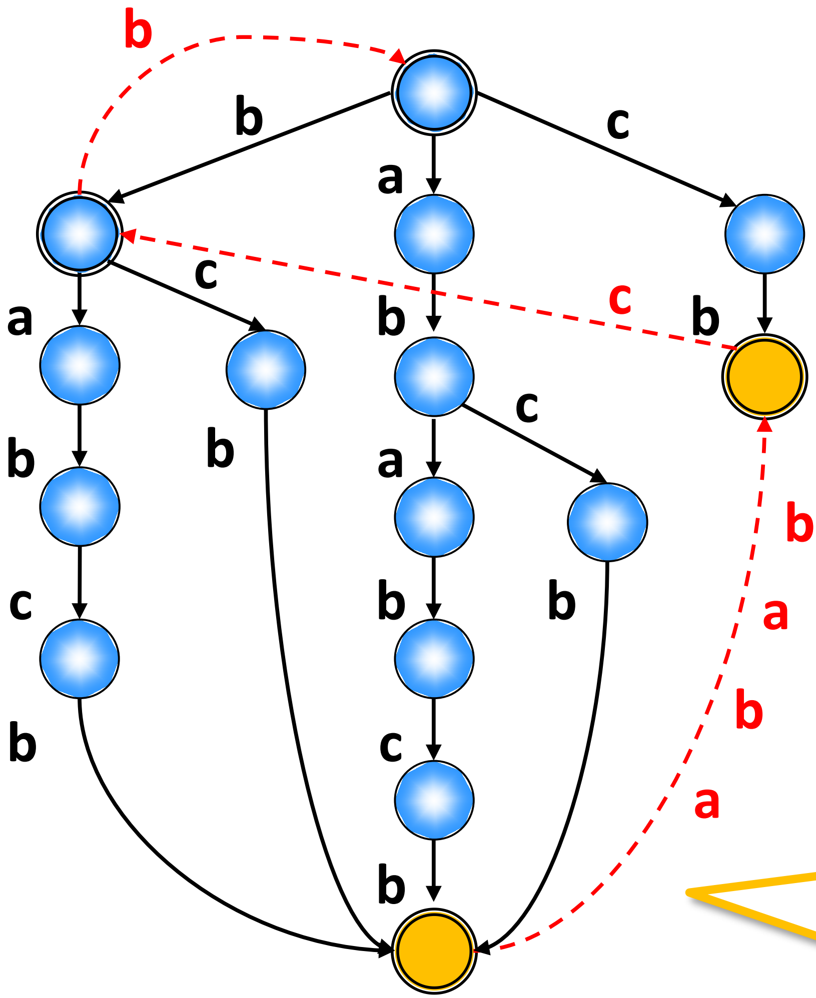
When merging nodes, we also contract the suffix links.

Node Merges and Suffix Links



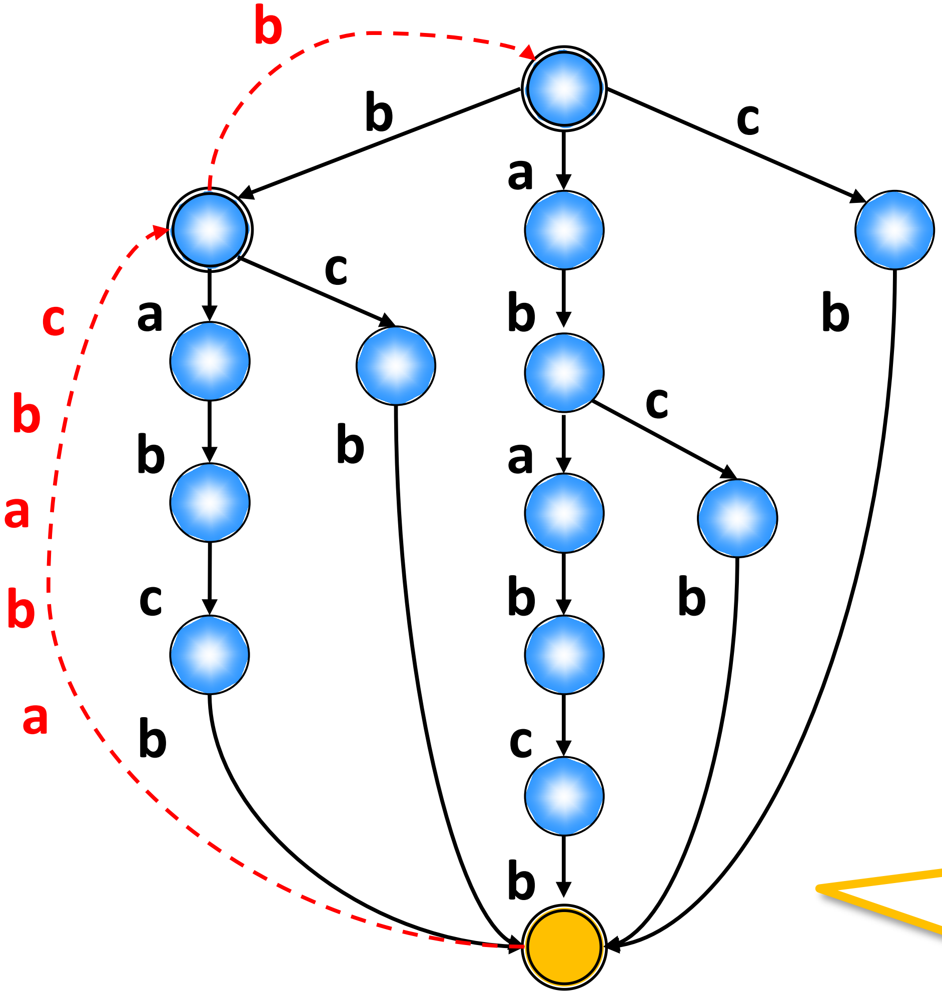
When merging nodes, we also contract the suffix links.

Node Merges and Suffix Links



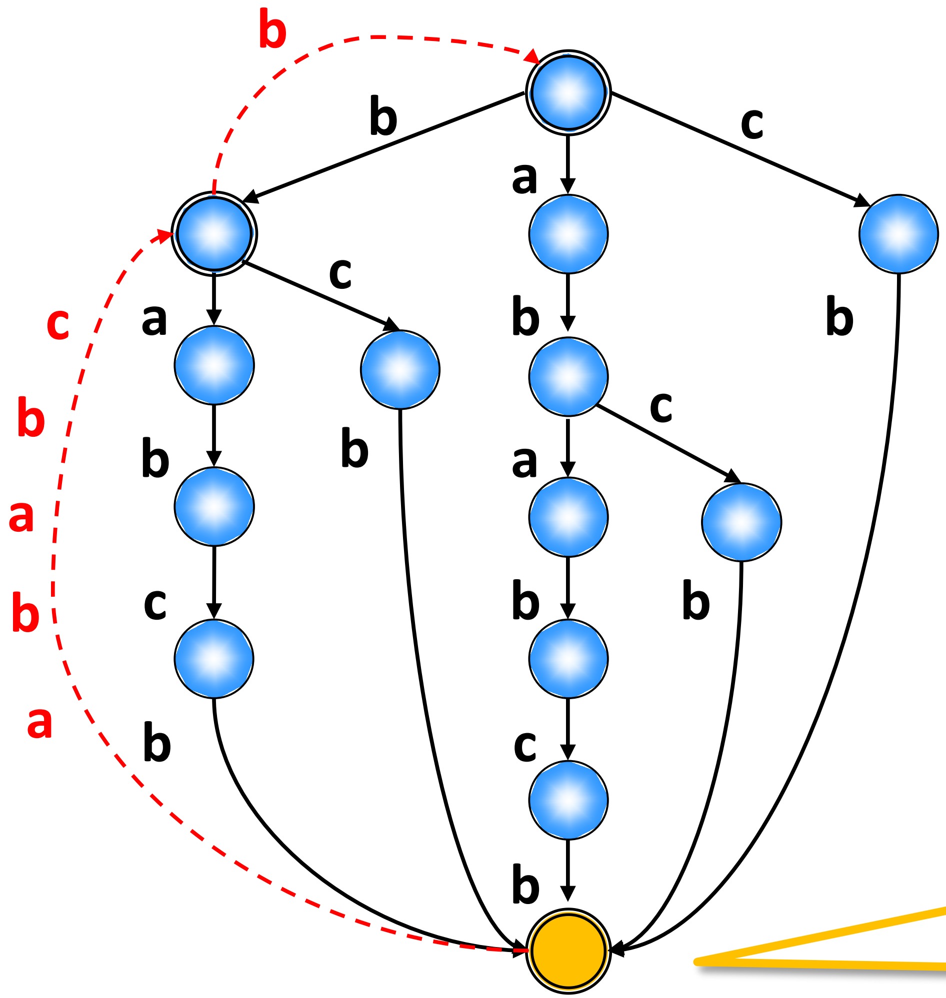
When merging nodes, we also contract the suffix links.

Node Merges and Suffix Links



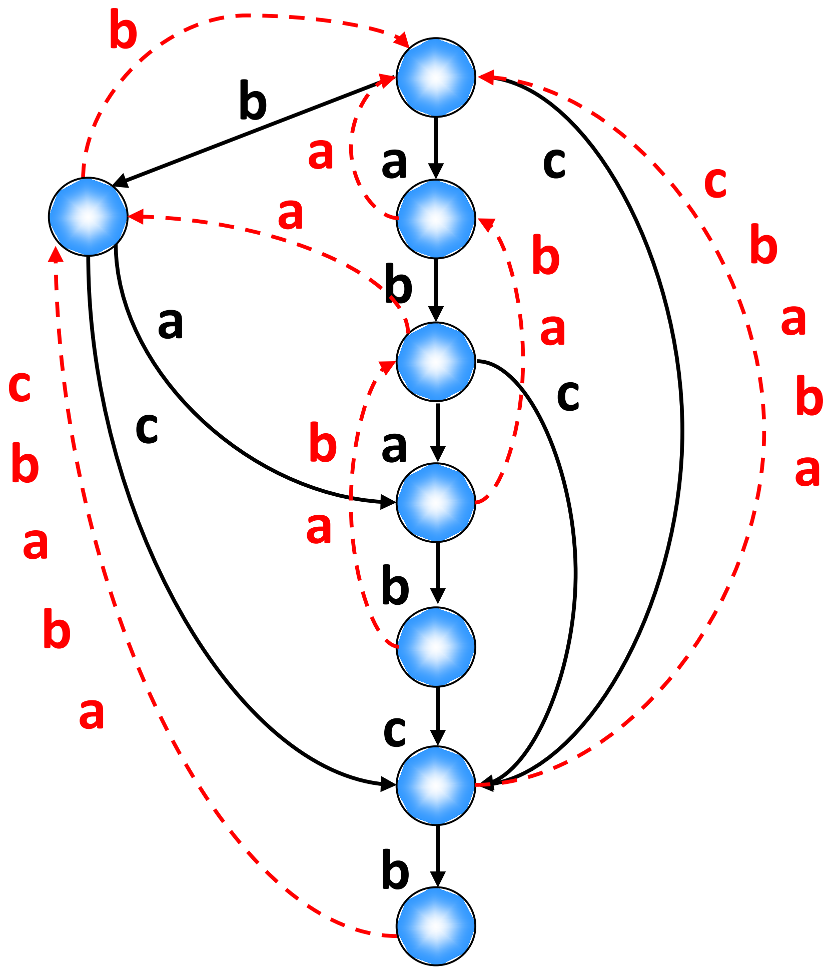
When merging nodes, we also contract the suffix links.

Node Merges and Suffix Links



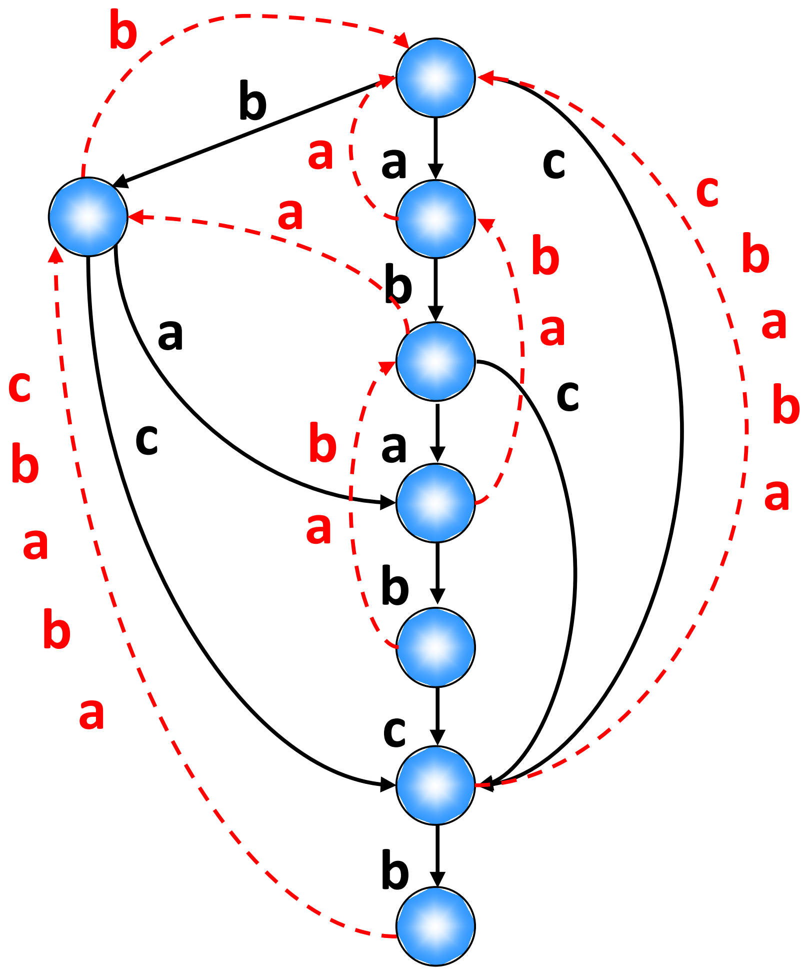
This is the suffix link of this DAWG node.

Suffix Links of DAWG

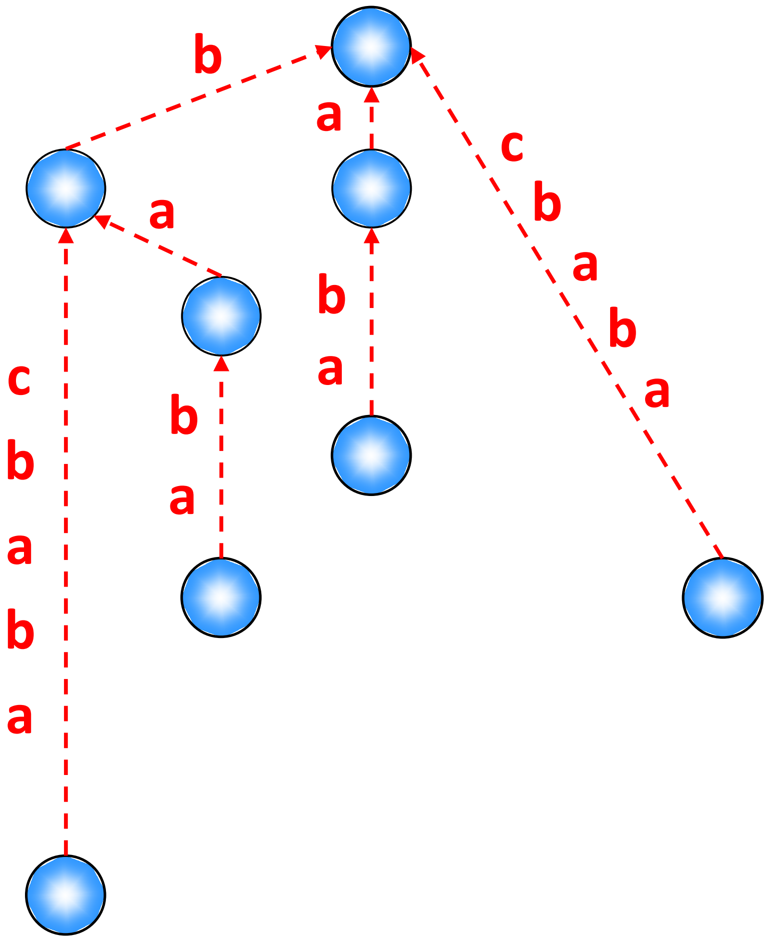


The suffix links of DAWG also forms a tree.

Suffix Links of DAWG

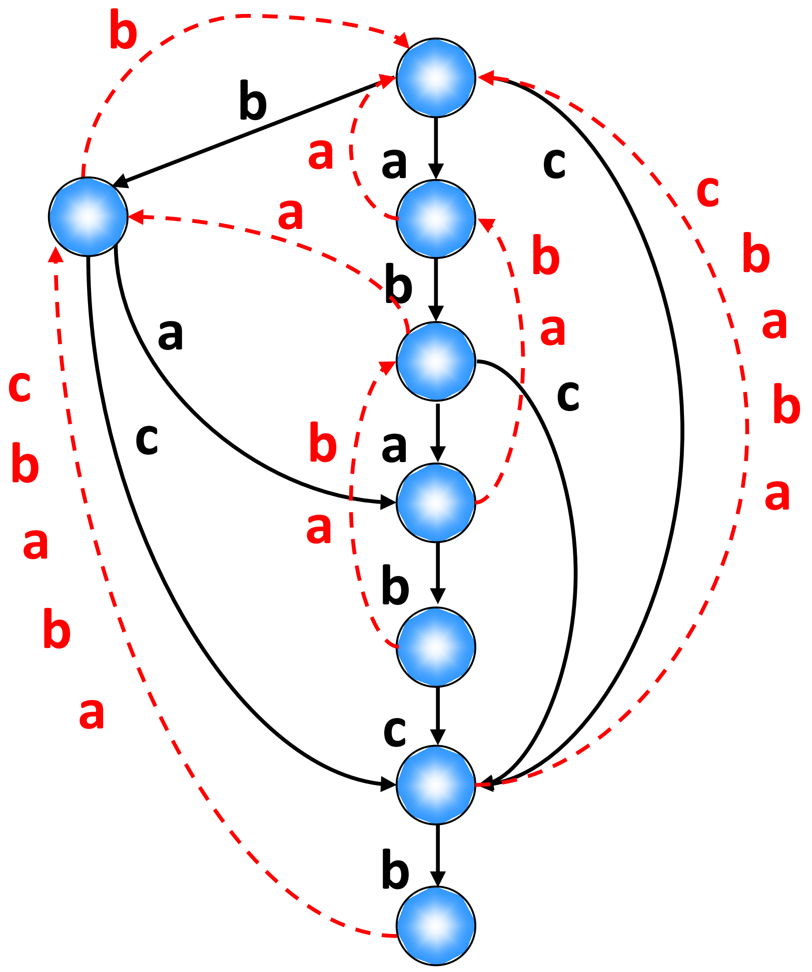


DAWG of **ababcb**

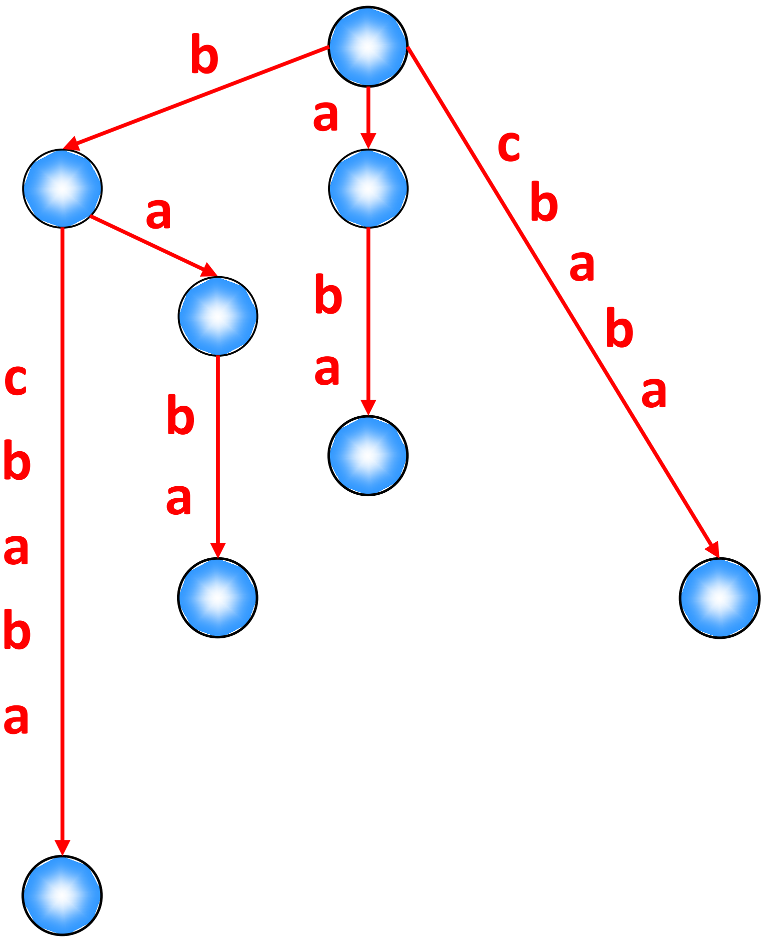


Contracted SLT of **ababcb**

SLT of DAWG = Suffix Tree of reverse

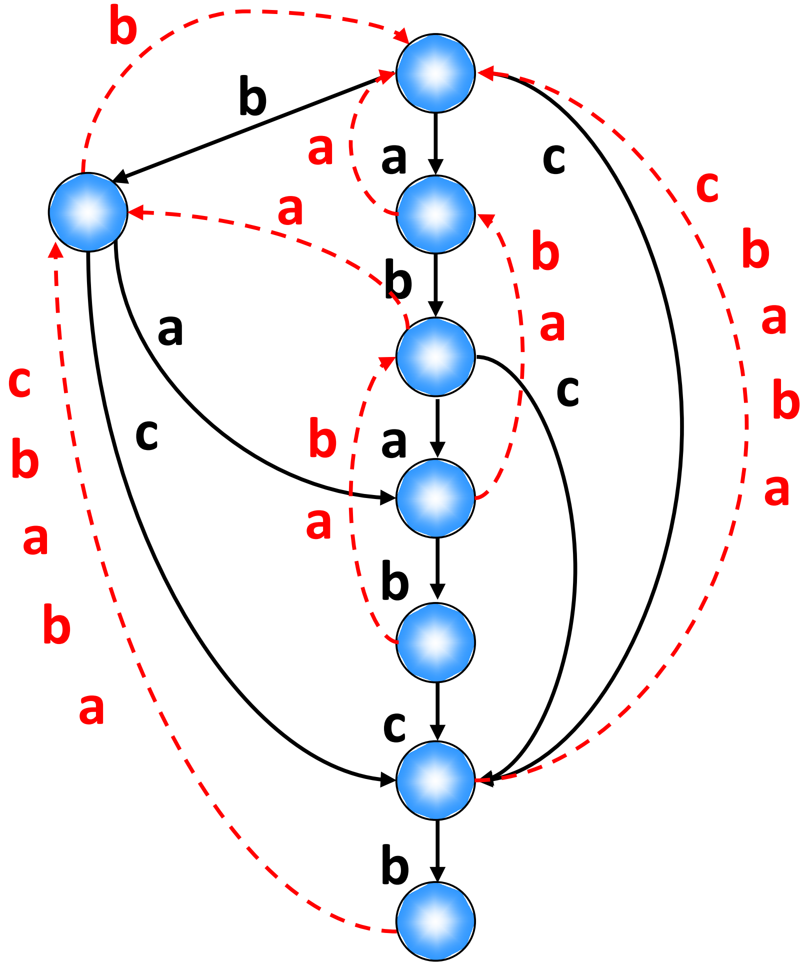


DAWG of **ababcb**

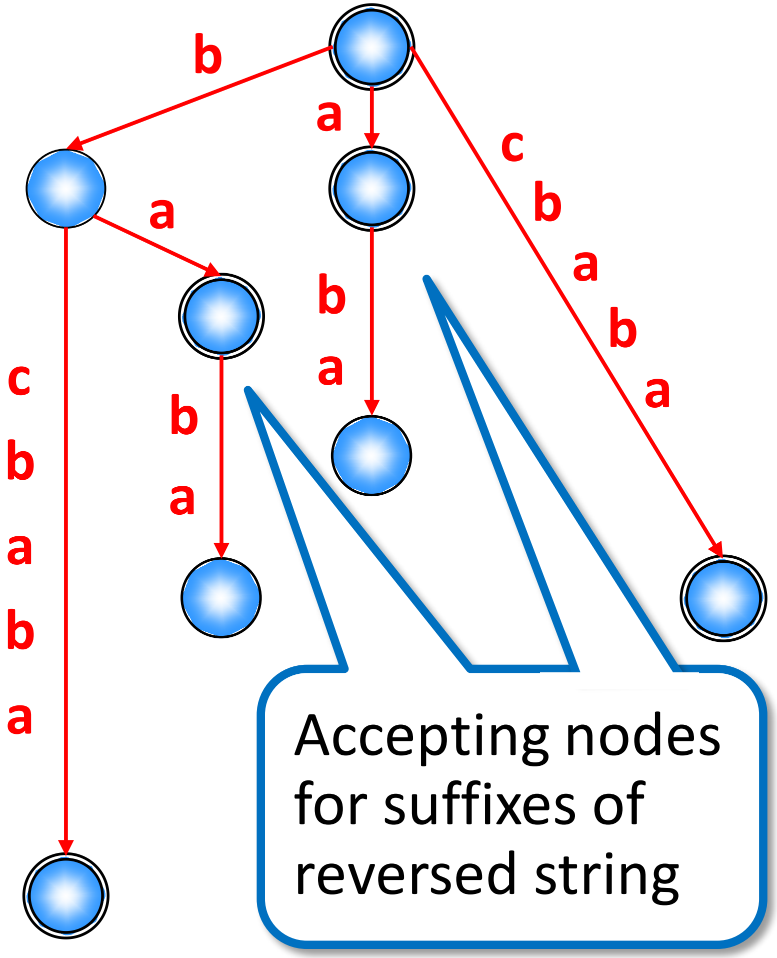


Suffix Tree of **bcbaba**

SLT of DAWG = Suffix Tree of reverse



DAWG of **ababcb**



Accepting nodes for suffixes of reversed string

Suffix Tree of **bcbaba**

SLT of DAWG = Suffix Tree of reverse

Theorem 2

The suffix link tree of the DAWG of string $\boldsymbol{w}$ forms the suffix tree of reversed string $\boldsymbol{w}^{\boldsymbol{R}}$.

- Contracting suffix links during node merges is equivalent to contracting non-branching paths of the suffix trie of $\boldsymbol{w}^{\boldsymbol{R}}$.

Number of Nodes of DAWG

Corollary 1

The number of nodes of the DAWG of any string of length $\boldsymbol{n}$ is at most $\mathbf{2}\boldsymbol{n}\mathbf{-1}$.

- Immediate from Theorem 2.

Number of Edges of DAWG

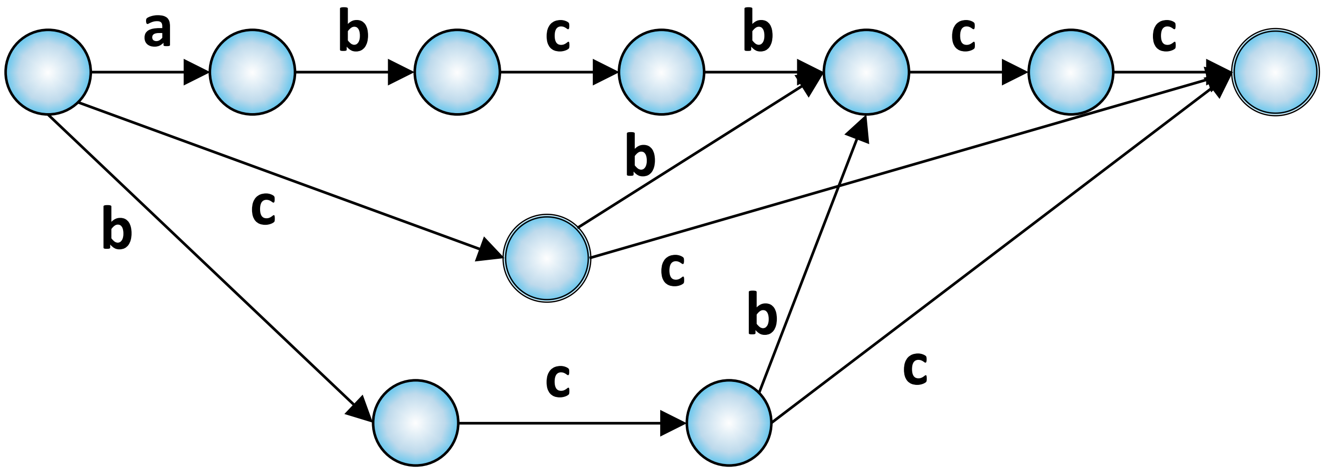
Lemma 2

The number of edges of the DAWG of any string of length n is at most $3n$-3.

Number of Edges of DAWG

Proof.

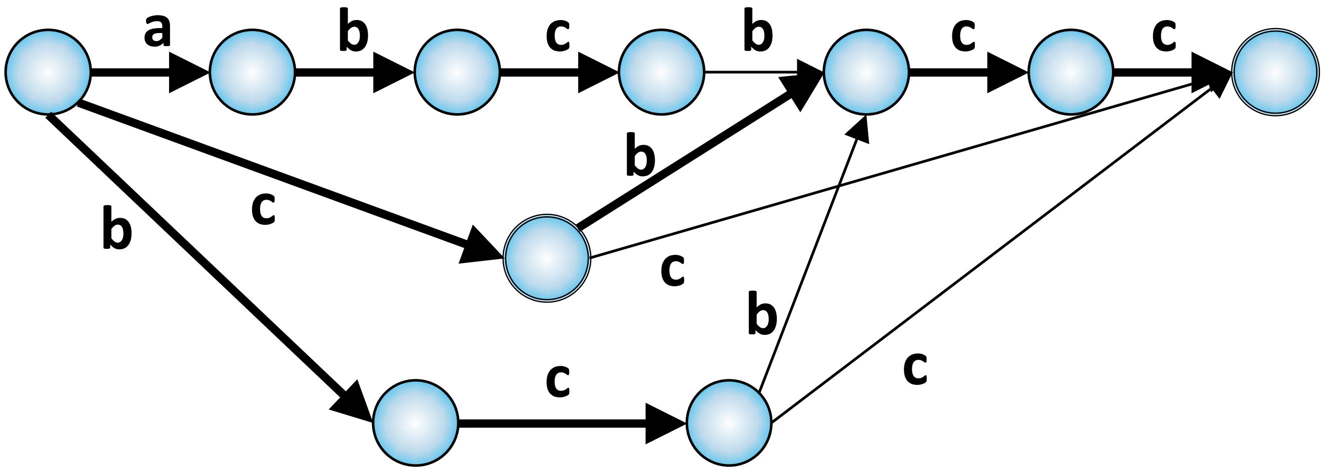
- Consider any spanning tree T of the DAWG.



Number of Edges of DAWG

Proof.

- Consider any spanning tree T of the DAWG.

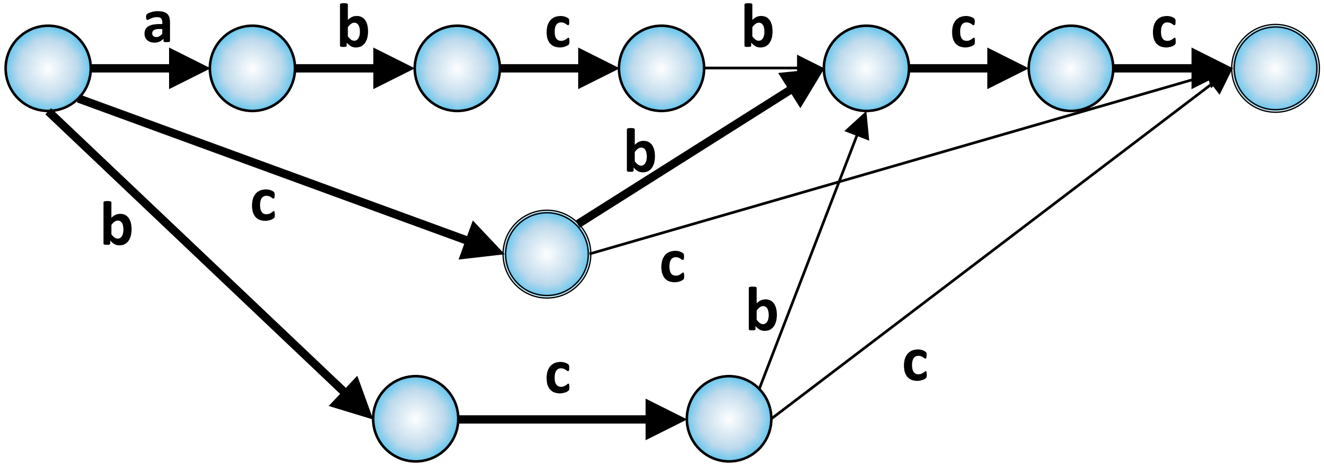


Number of Edges of DAWG

Proof.

- Because the DAWG has at most $2n$-1 nodes, the spanning tree T contains at most $2n$-2 edges.

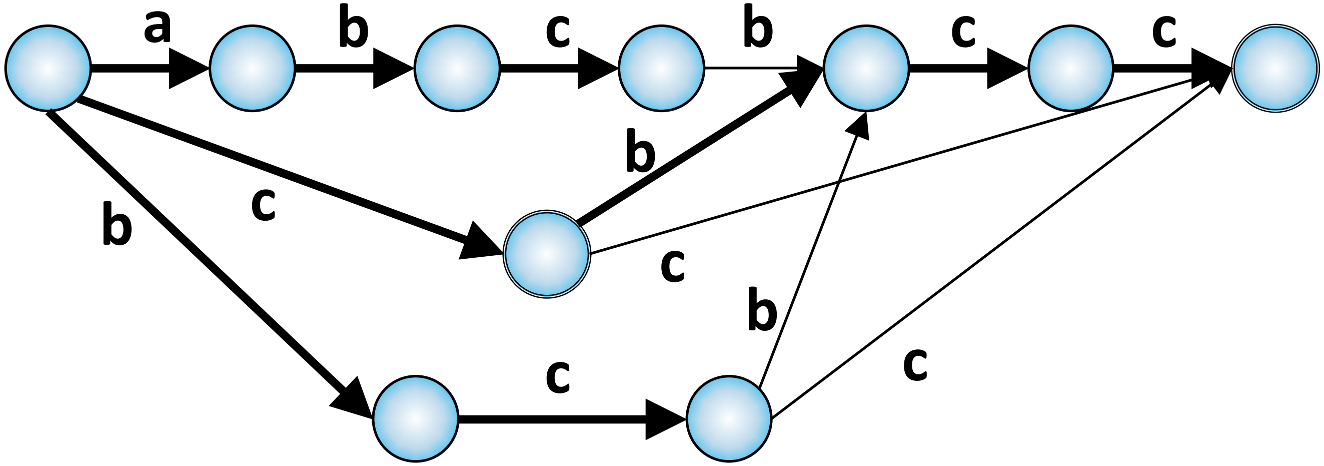
  Next, we count the number of edges outside T.



Number of Edges of DAWG

Proof.

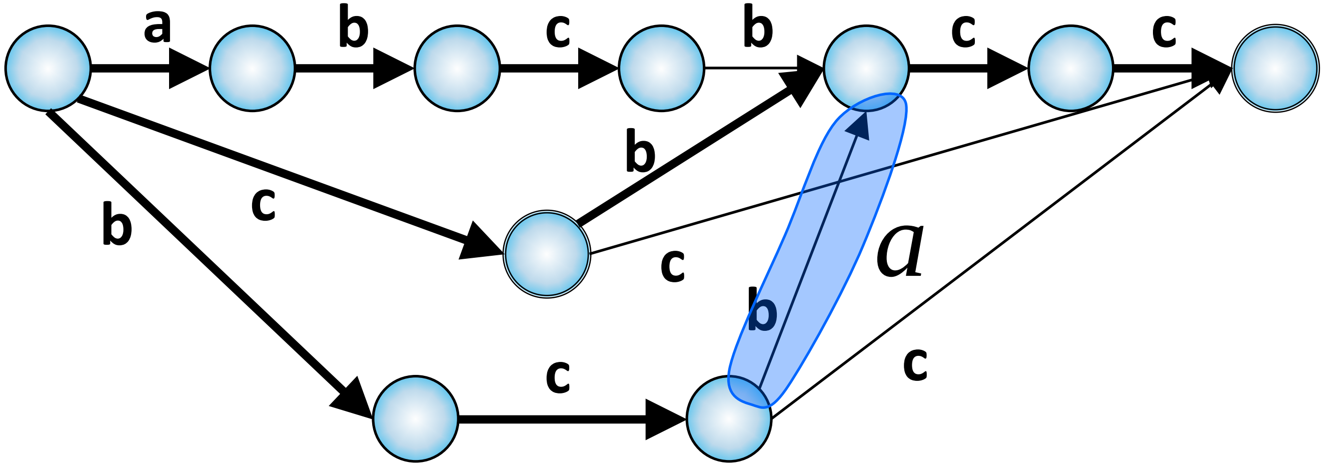
- For any edge e outside T, we consider path xay, where x is the path from the root to e, a is the label of e, and y is any path after e such that xay is a suffix of w.



Number of Edges of DAWG

Proof.

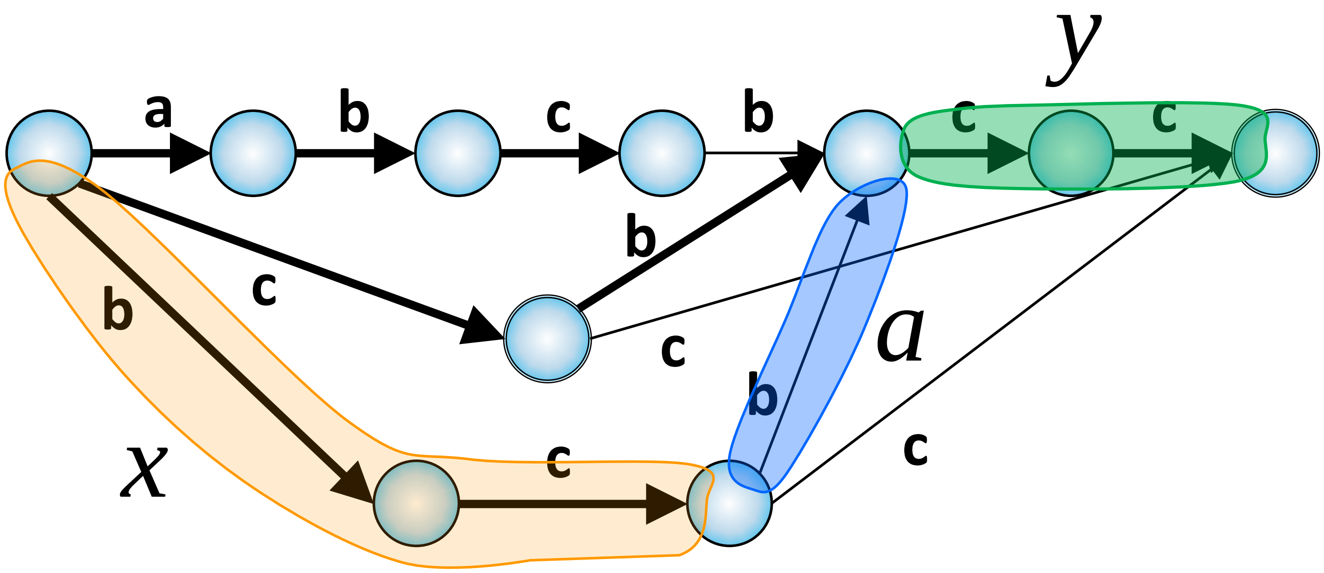
- For any edge e outside T, we consider path xay, where x is the path from the root to e, a is the label of e, and y is any path after e such that xay is a suffix of w.



Number of Edges of DAWG

Proof.

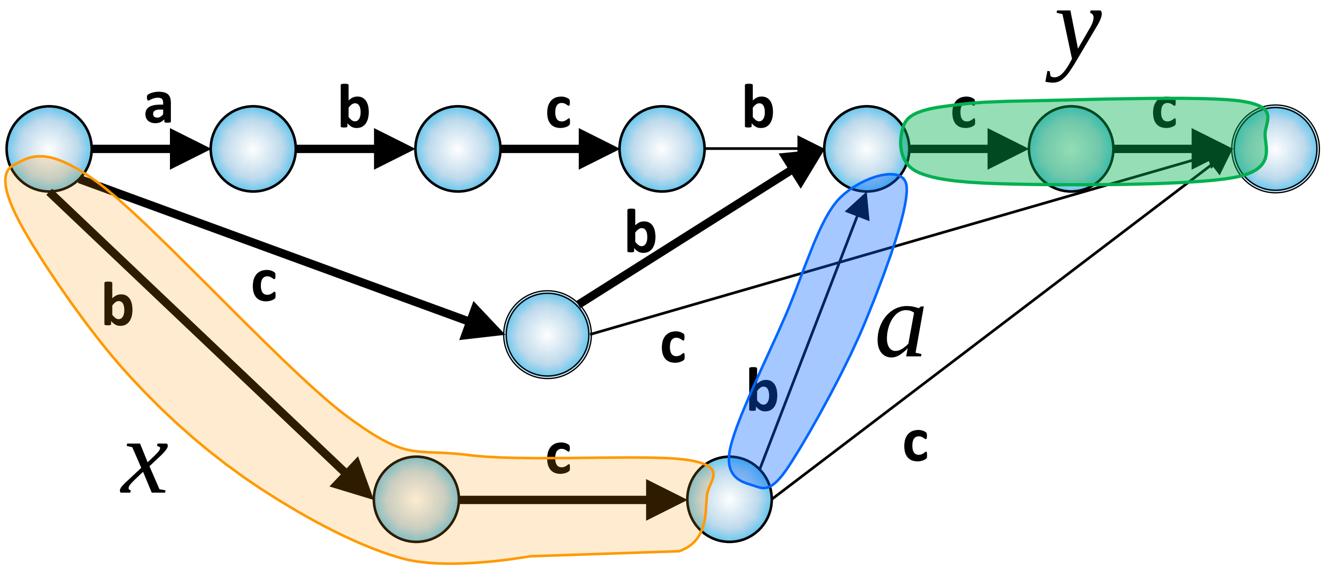
- For any edge e outside T, we consider path xay, where x is the path from the root to e, a is the label of e, and y is any path after e such that xay is a suffix of w.



Number of Edges of DAWG

Proof.

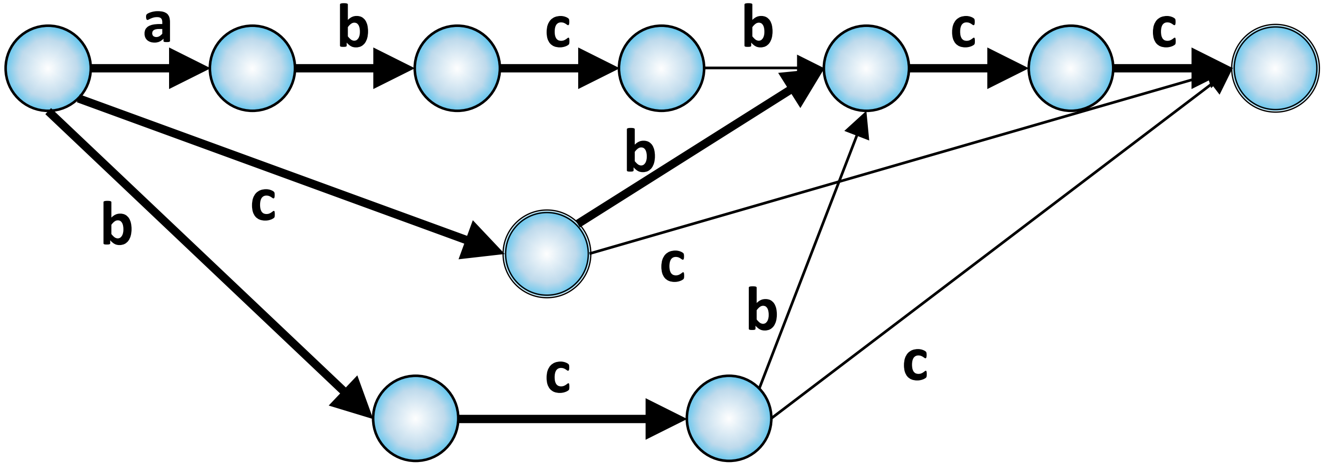
- This maps the edge e to the suffix xay of w. Moreover, any other edge outside the spanning tree T cannot be mapped to the same suffix xay.



Number of Edges of DAWG

Proof.

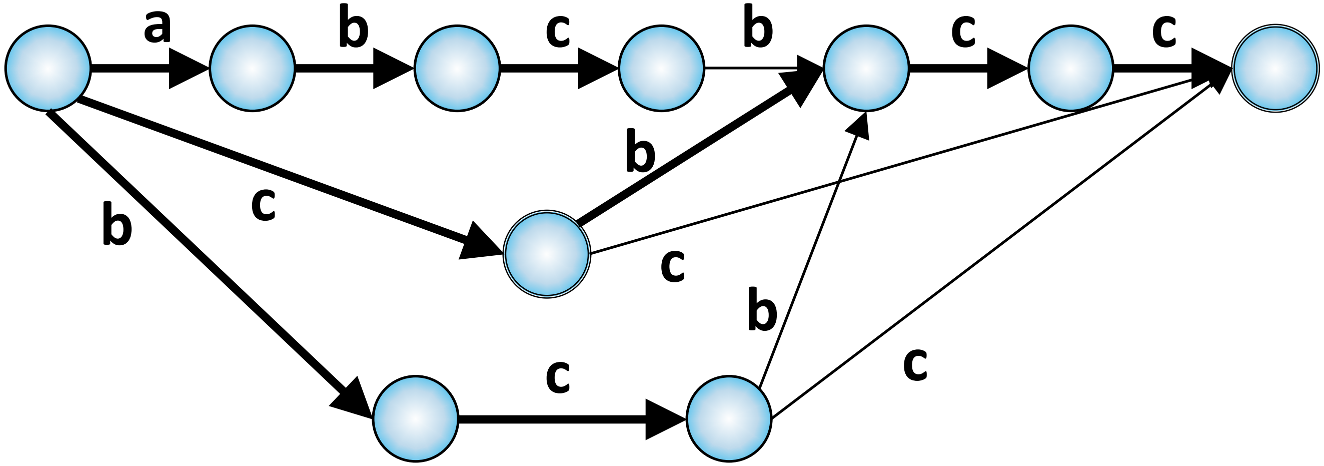
- Hence the mapping is injective.
  Since the spanning tree contains at least one suffix of w, there can be at most n-1 suffixes to which the edges outside T can be mapped.



Number of Edges of DAWG

Proof.

- Therefore, the number of edges outside T is at most n-1. Overall, DAWG has at most $(2n-2)+(n-1) = 3n-3$ edges.



The size of DAWG

Theorem 3

The DAWG of any string of length n has at most $2n$-1 nodes and $3n$-3 edges.

- Note: The bound for the number of edges can further be shaved to $3n$-4, and it is tight, i.e., the DAWG of string $ab^{n-2}c$ contains $3n$-4 edges.

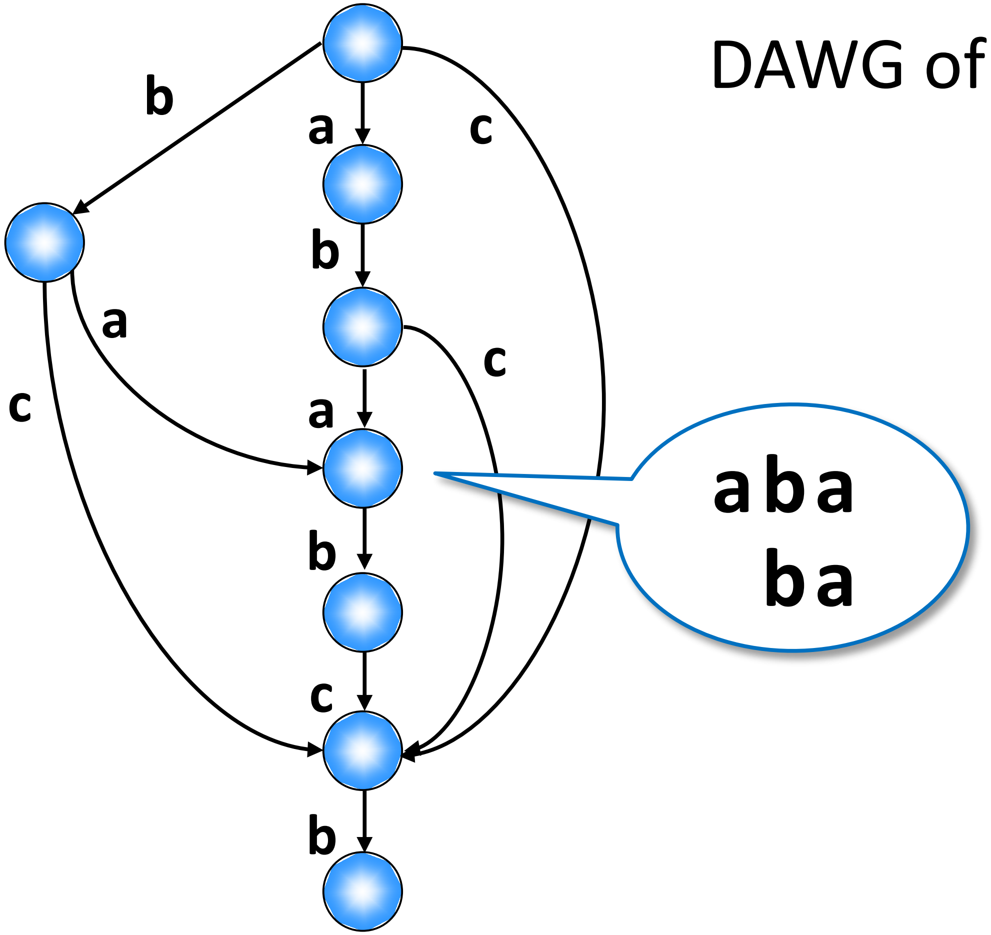
Properties of DAWG nodes

Lemma 3

Two strings x and y are represented by
the same node of the DAWG of w
iff x and y end at the same positions in w.

- This is true because we merged nodes of the suffix trie of w iff they have isomorphic subtrees (hence, the same sets of ending positions).

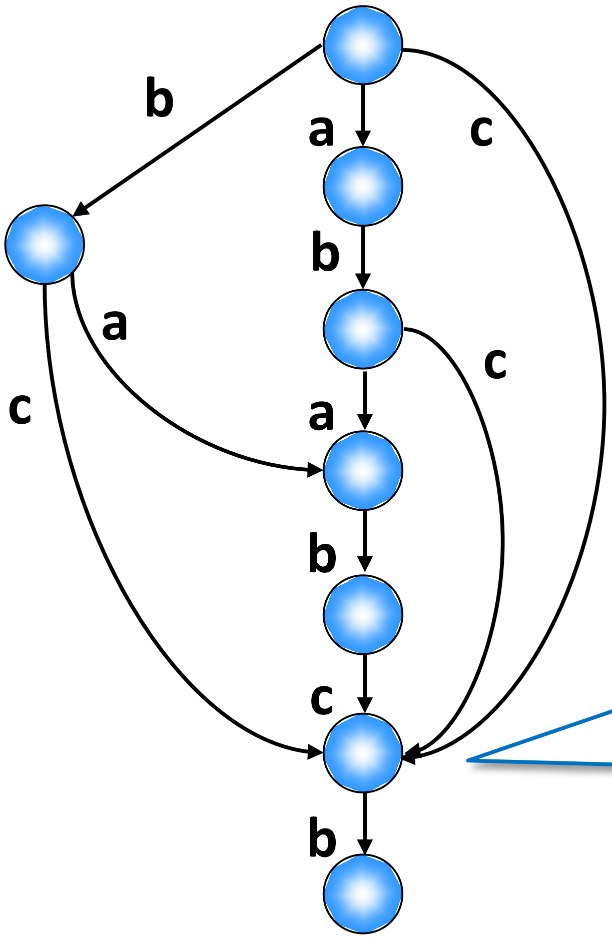
Properties of DAWG nodes



aba
ba

1 2 3 4 5 6
DAWG of **ababcb**

Properties of DAWG nodes



1 2 3 4 5 6

DAWG of **ababcb**

ababc

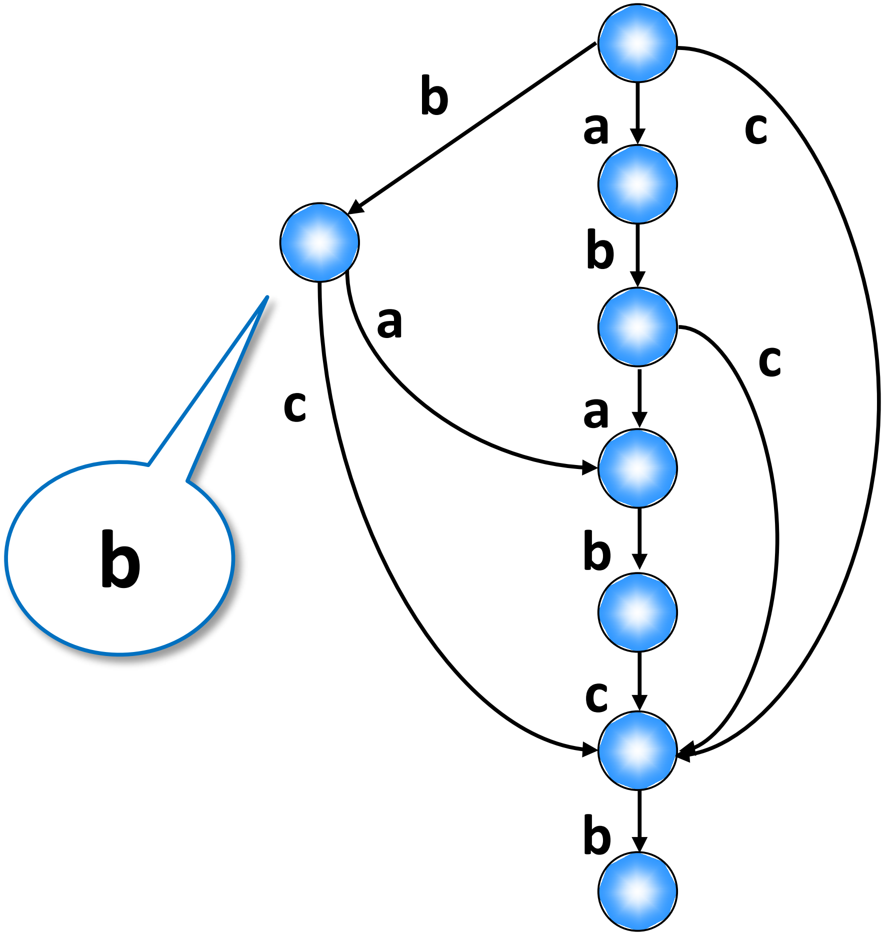
babc

abc

bc

c

Properties of DAWG nodes



1 2 3 4 5 6

DAWG of **ababcb**

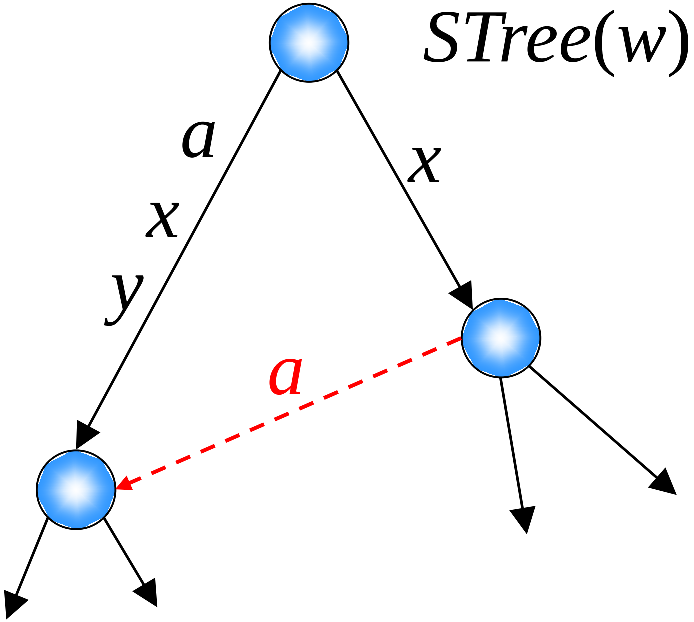
Connection with Weiner Links

Corollary 2

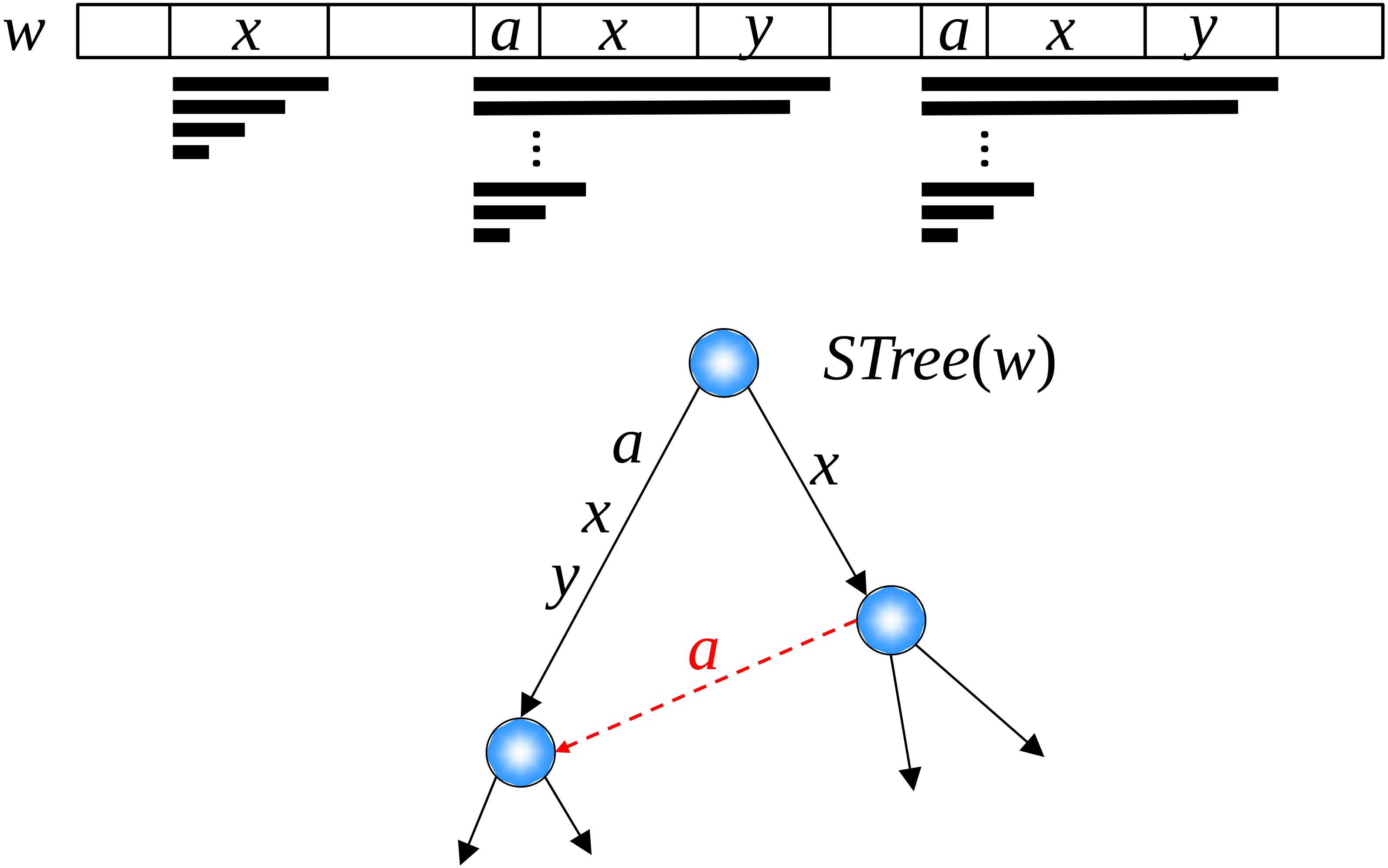
The DAG consisting of the explicit and implicit Weiner links of the suffix tree of string w is the DAWG of the reversed string $\boldsymbol{w^R}$.

Connection with Weiner Links

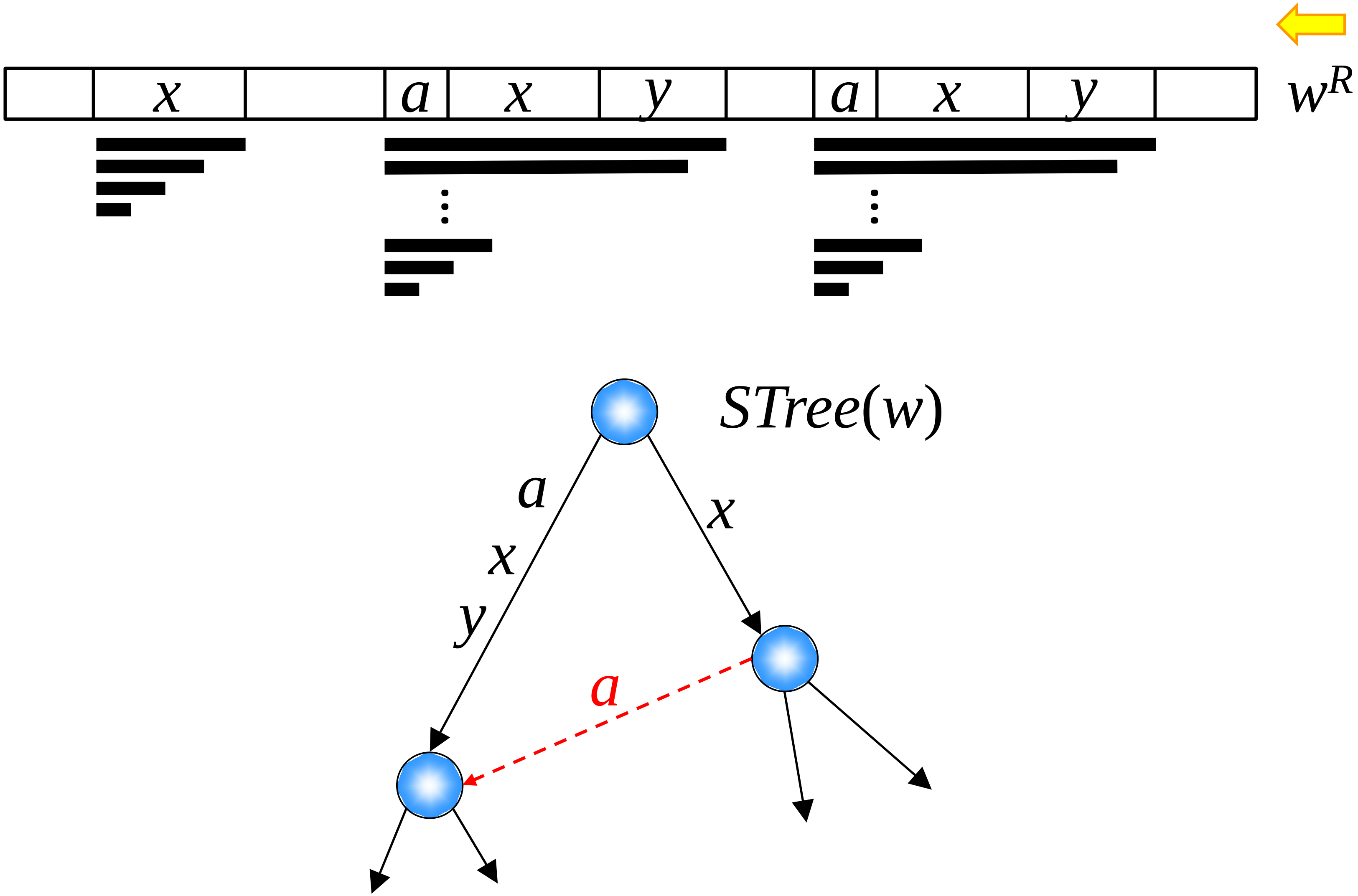
w		x		a	x	y		a	x	y	



Connection with Weiner Links



Connection with Weiner Links



Applications of DAWGs (Incomprehensive)

- Bidirectional pattern matching [Folklore]
- Approximate pattern matching
  [Ukkonen & Wood, 1993]
- Pattern matching with variable-length don't cares
  [Kucherov & Rusinowitch, 1997]
- Finding minimal absent words
  [Crochemore et al., 1998, 2015, Fujishige et al. 2016]
- Compact online Lempel Ziv 77 factorization
  [Yamamoto et al., 2014]
- Finding α-gapped repeats [Tanimura et al., 2015]
- Finding maximal-exponent substring in
  overlap-free string [Badkobeh & Crochemore, 2016]

Applications of DAWGs (Incomprehensive)

- Bidirectional pattern matching [Folklore]
- Approximate pattern matching [Ukkonen & Wood, 1993]
- Pattern matching with variable-length don’t cares [Kucherov & Rusinowitch, 1997]
- Finding minimal absent words [Crochemore et al., 1998, 2015, Fujishige et al. 2016]
- Compact online Lempel Ziv 77 factorization [Yamamoto et al., 2014]
- Finding α-gapped repeats [Tanimura et al., 2015]
- Finding maximal-exponent substring in overlap-free string [Badkobeh & Crochemore, 2016]

Minimal Absent Words

- A string y is said to be a **minimal absent word** (**MAW**) of a string w, if
  1. y does not occur in w, but
  2. proper substrings of y occur in w.

- E.g.) If w = **abaab** and Σ = {**a**, **b**}, then the MAWs of w are **aaa**, **aaba**, **bab**, **bb**.

- MAWs can be used to build phylogeny [Chairungsee & Crochemore, 2012].

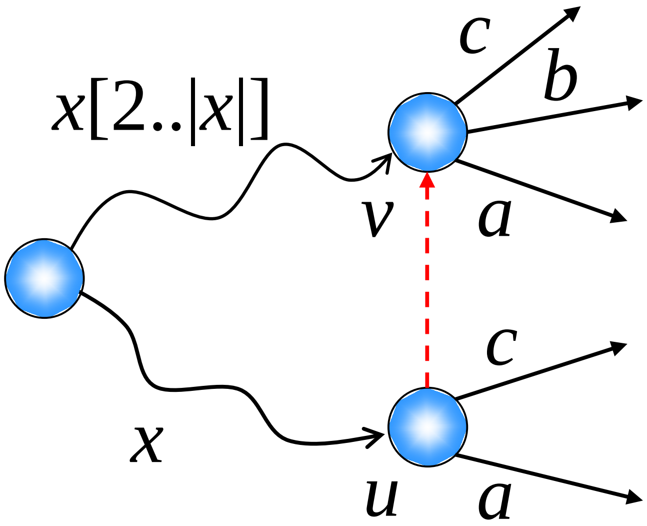
Minimal Absent Words

- For a node u of the DAWG of w, let x be the shortest string represented by u.

- Let $v = \mathrm{suf\_link}(u)$.

- Then, xb ($b \in \Sigma$) is a MAW of w iff
  1. there is no out-edge from u labeled b, and
  2. there is an out-edge from v labeled b.

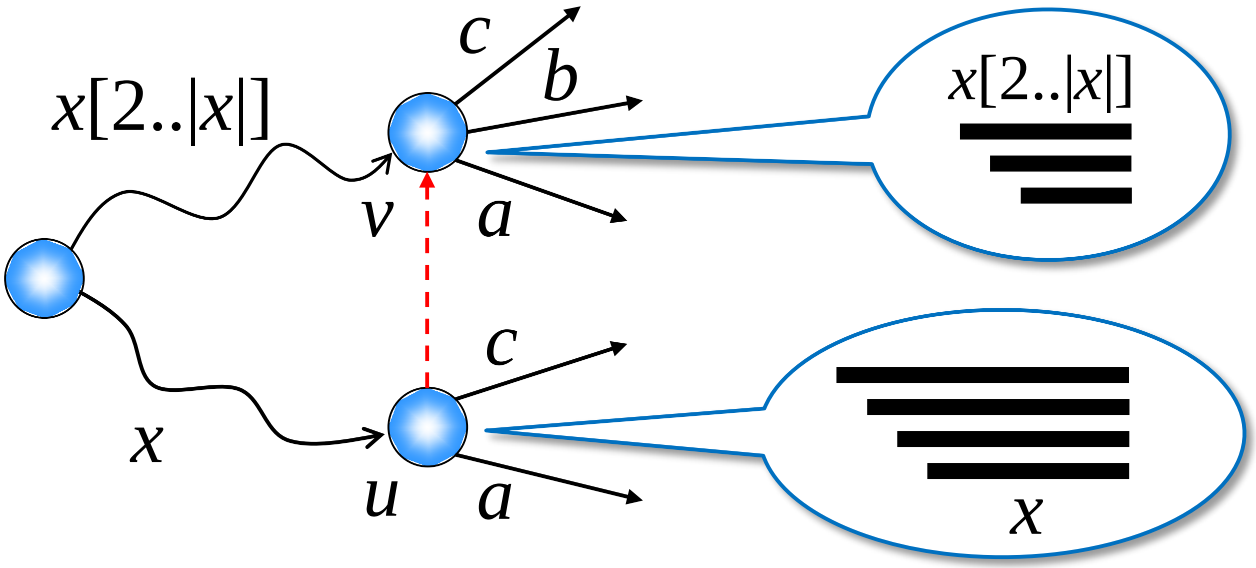
Minimal Absent Words



- xb does not occur in w, and
- both x and $x[2..|x|]b$ occur in w.

$\Leftrightarrow$ xb is a MAW of w.

Minimal Absent Words



- xb does not occur in w, and
- both x and $x[2..|x|]b$ occur in w.

$\Leftrightarrow$ xb is a MAW of w.

MAW Computation with DAWG

Theorem 4

Using the DAWG of string w, we can compute all MAWs of w in $O(\sigma n)$ time.

$\sigma = |\Sigma|, \ n = |w|$

- For each node of the DAWG of w, it is sufficient to test at most σ letters.
- The DAWG of w has $O(n)$ nodes.

Faster MAW Computation with DAWG

Theorem 5

Using the edge-sorted DAWG of string w, we can compute all MAWs of w in optimal $O(n + |MAW_w|)$ time.

- Testing out-edges of u and v can be charged to either existing edges or MAWs to output.

Direct Construction of DAWGs

- Since the suffix trie of string of length n can contain $\Omega(n^2)$ nodes, converting the suffix trie into the DAWG takes $O(n^2)$ time.
- Can we construct DAWGs directly?

Online Construction of DAWGs

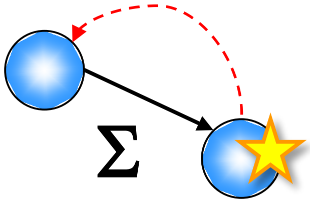
Theorem 6

The DAWG of a given string w of length n can be constructed online in $O(n \log \sigma)$ time, where σ is the alphabet size.

- We incrementally build the DAWG of $w[1..i]$ for increasing $i = 1, \ldots, n$ (left-to-right online).
- The DAWG is annotated with suffix links.
- The $\log \sigma$ factor is the cost to sort and search branching edges.

Online Construction of Suffix Tries

- Before going to online construction of DAWGs, we consider online construction of suffix tries.

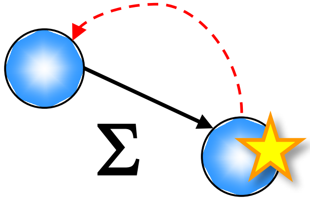


ababcb

When a new letter $w[i]$ arrives, we begin with the node which represents $w[1..i\text{-}1]$.

Online Construction of Suffix Tries

- Before going to online construction of DAWGs, we consider online construction of suffix tries.

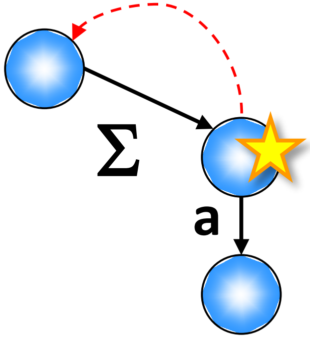


ababcb

If we cannot traverse from the star with new character $w[1] = \mathbf{a}$, create a new edge labeled **a**.

Online Construction of Suffix Tries

- Before going to online construction of DAWGs, we consider online construction of suffix tries.

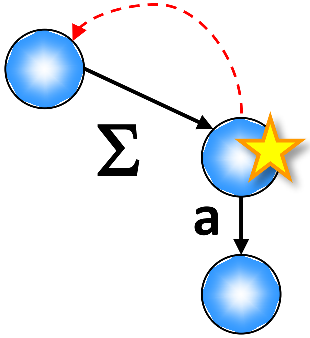


ababcb

If we cannot traverse from the star with new character $w[1] = \mathbf{a}$, create a new edge labeled **a**.

Online Construction of Suffix Tries

- Before going to online construction of DAWGs, we consider online construction of suffix tries.

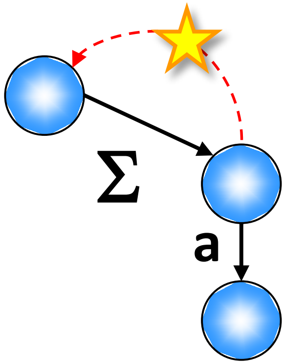


ababcb

Then, we move the star via the suffix link, and check whether we can traverse with **a**.

Online Construction of Suffix Tries

- Before going to online construction of DAWGs, we consider online construction of suffix tries.

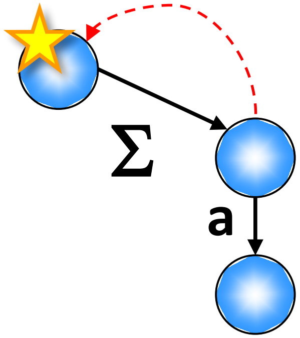


ababcb

Then, we move the star via the suffix link, and check whether we can traverse with **a**.

Online Construction of Suffix Tries

- Before going to online construction of DAWGs, we consider online construction of suffix tries.

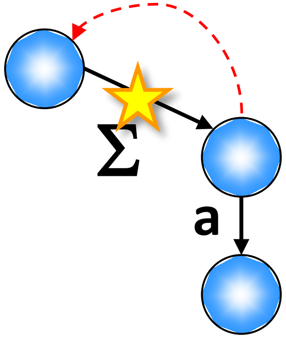


ababcb

Then, we move the star via the suffix link, and check whether we can traverse with **a**.

Online Construction of Suffix Tries

- Before going to online construction of DAWGs, we consider online construction of suffix tries.

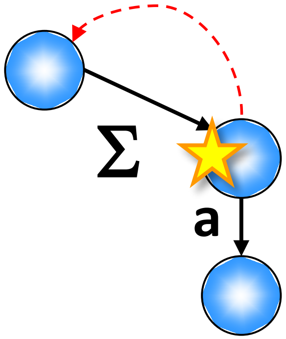


ababcb

Then, we move the star via the suffix link, and check whether we can traverse with **a**.

Online Construction of Suffix Tries

- Before going to online construction of DAWGs, we consider online construction of suffix tries.

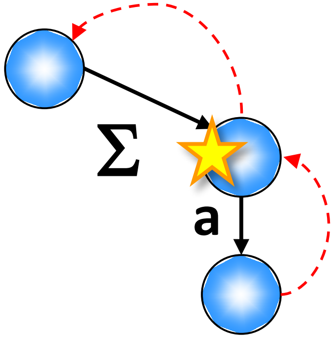


ababcb

Then, we move the star via the suffix link, and check whether we can traverse with **a**.

Online Construction of Suffix Tries

- Before going to online construction of DAWGs, we consider online construction of suffix tries.

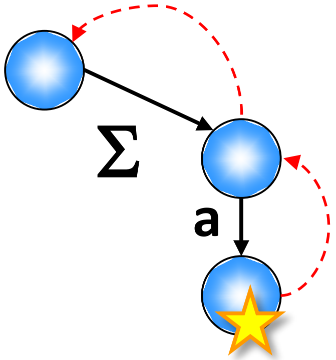


ababcb

After a successful move of the star, we create the suffix link from the new leaf to the node where the star has arrived.

Online Construction of Suffix Tries

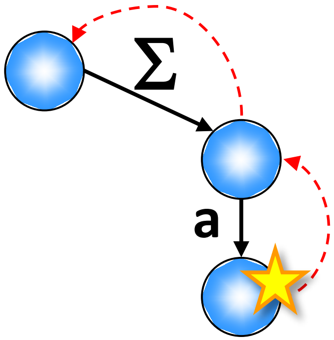
- Before going to online construction of DAWGs, we consider online construction of suffix tries.



ababcb

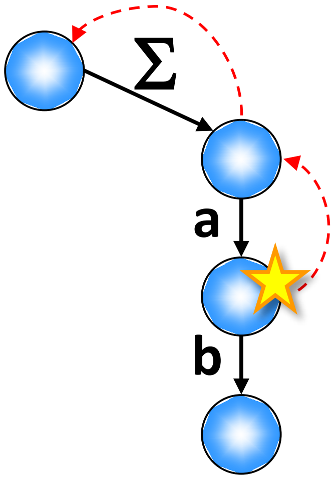
The suffix trie for $w[1..1]$ = **a** is complete.
We move the star to the leaf which represents $w[1..1]$ = **a**.

Online Construction of Suffix Tries



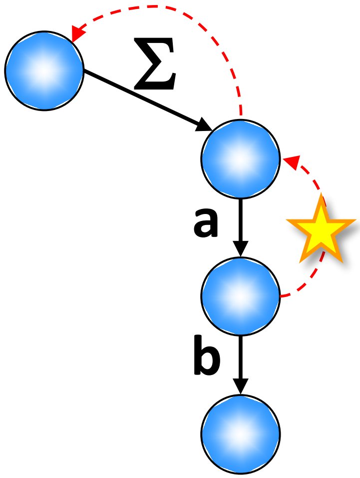
ababcb

Online Construction of Suffix Tries



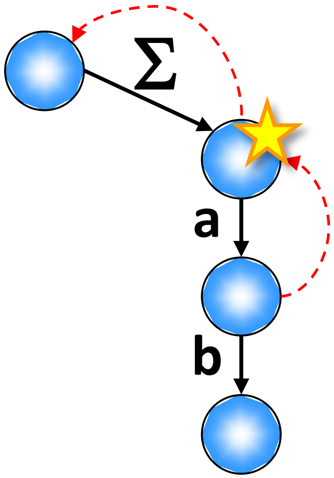
ababcb

Online Construction of Suffix Tries



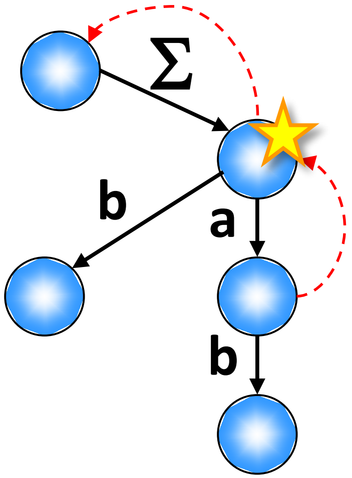
ababcb

Online Construction of Suffix Tries



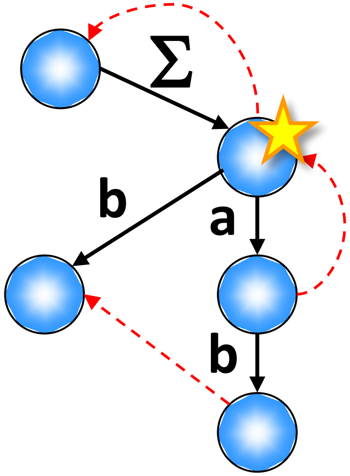
ababcb

Online Construction of Suffix Tries



ababcb

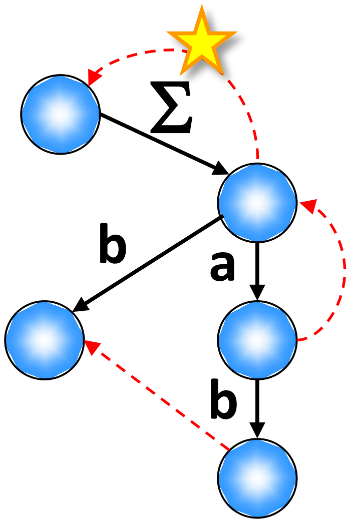
Online Construction of Suffix Tries



ababcb

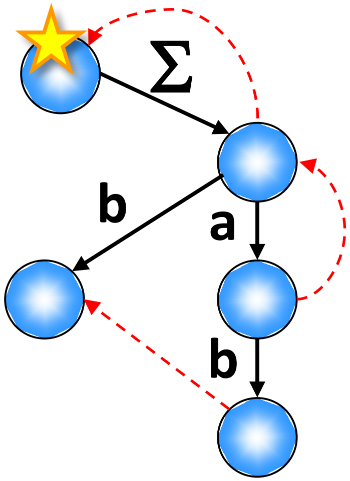
We have two new leaves for **ab** and **b**. We create the suffix link from **ab** to **b**.

Online Construction of Suffix Tries



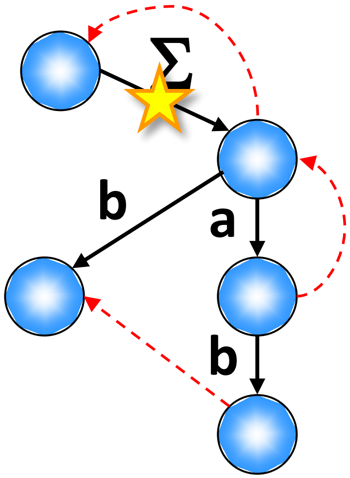
ababcb

Online Construction of Suffix Tries



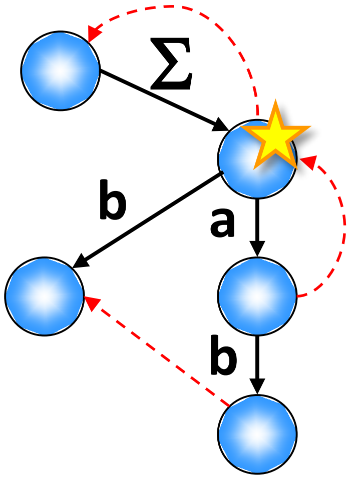
ababcb

Online Construction of Suffix Tries



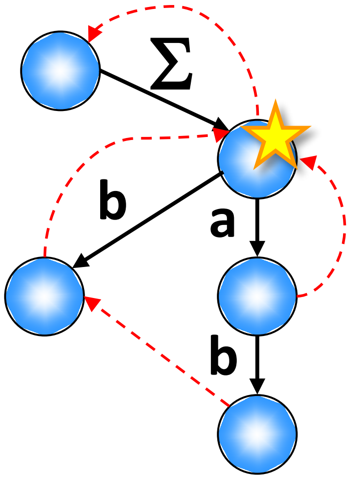
ababcb

Online Construction of Suffix Tries



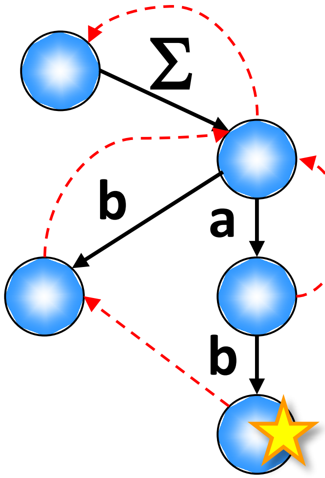
ababcb

Online Construction of Suffix Tries



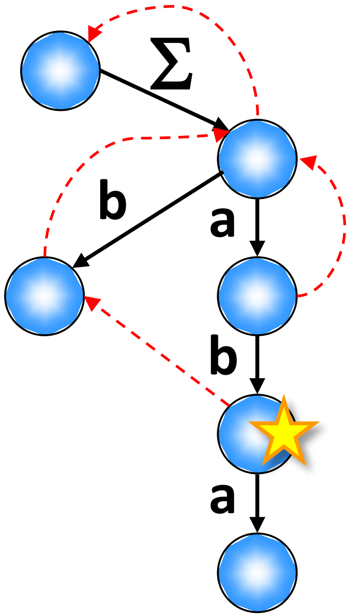
ababcb

Online Construction of Suffix Tries



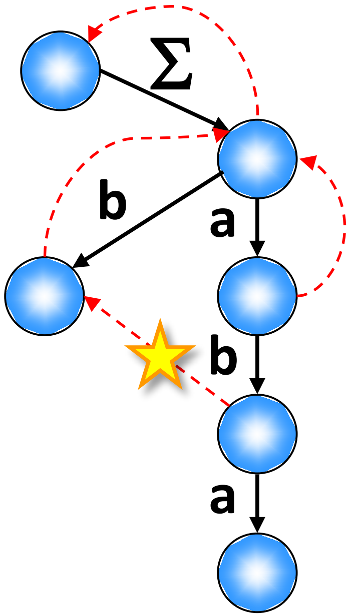
ababcb

Online Construction of Suffix Tries



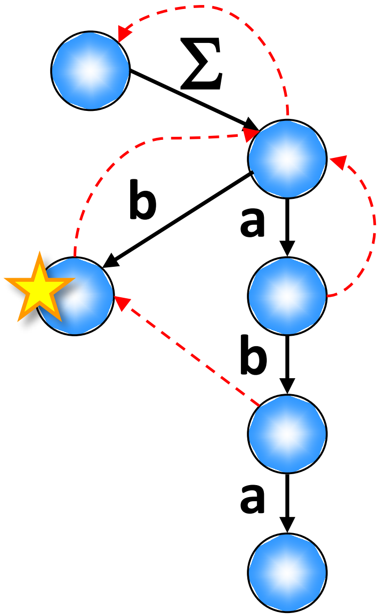
ababcb

Online Construction of Suffix Tries



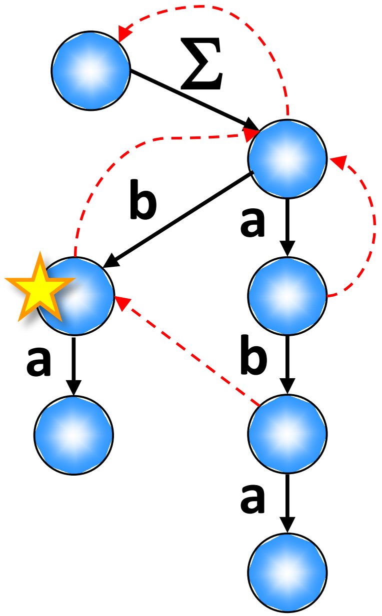
ababcb

Online Construction of Suffix Tries



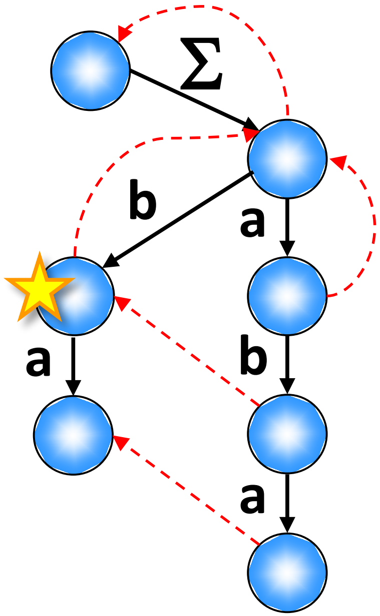
ababcb

Online Construction of Suffix Tries



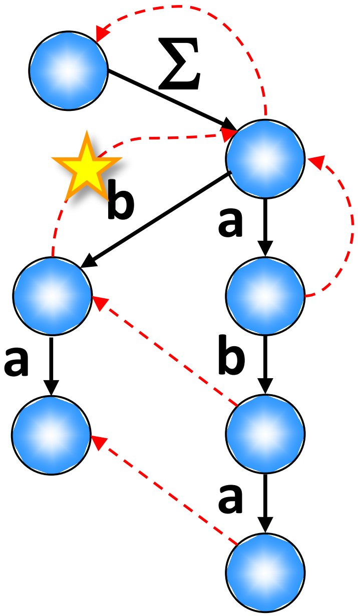
ababcb

Online Construction of Suffix Tries



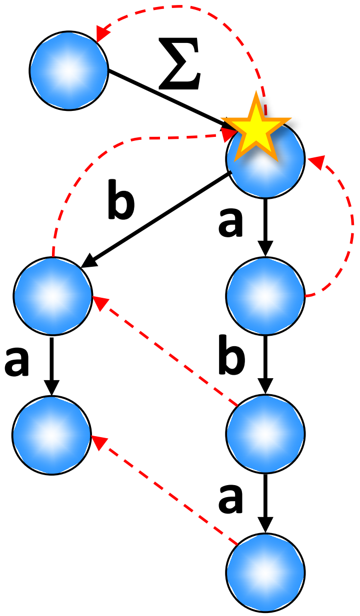
ababcb

Online Construction of Suffix Tries



ababcb

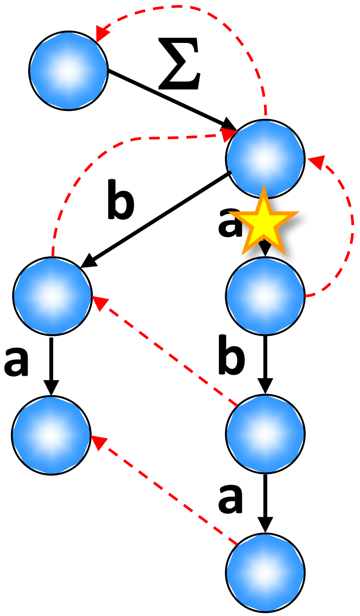
Online Construction of Suffix Tries



ababcb

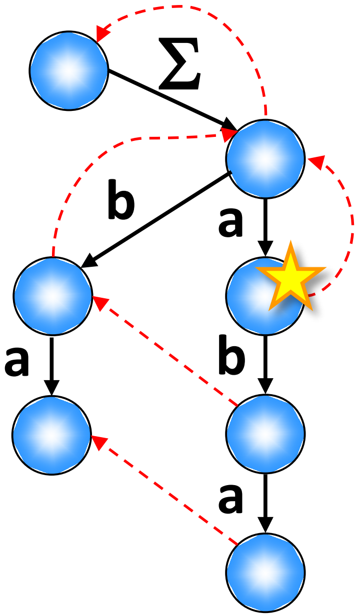
We are looking for an edge labeled $w[3]$ = **a**.
Using a BST for branching edges, we can find it in $O(\log\ \sigma)$ time.

Online Construction of Suffix Tries



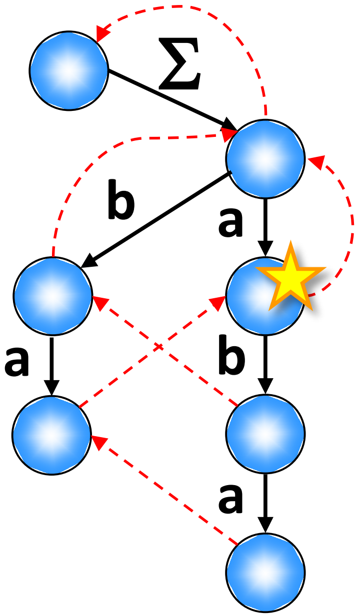
ababcb

Online Construction of Suffix Tries



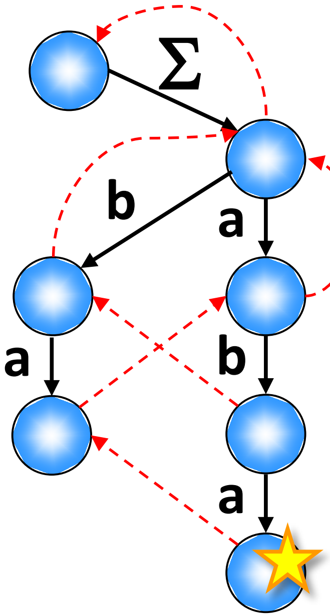
ababcb

Online Construction of Suffix Tries



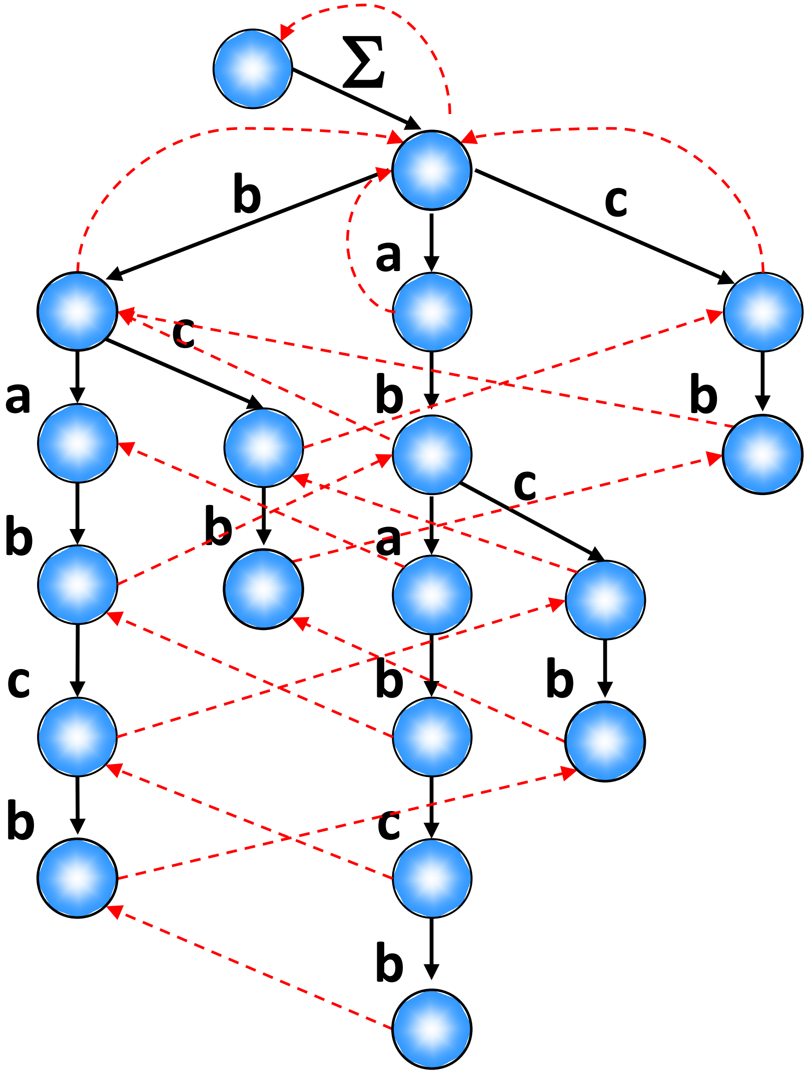
ababcb

Online Construction of Suffix Tries



ababcb

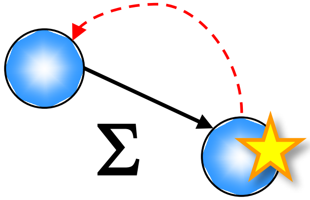
Online Construction of Suffix Tries



ababcb

Online Construction of DAWGs

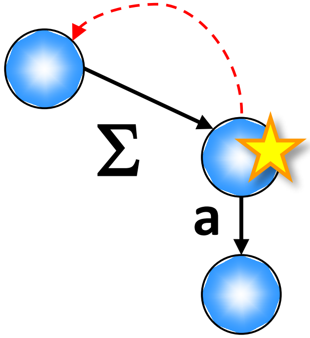
- Online DAWG construction follows the approach of online suffix trie construction.



ababcb

Online Construction of DAWGs

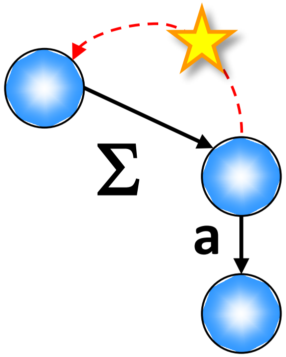
- Online DAWG construction follows the approach of online suffix trie construction.



ababcb

Online Construction of DAWGs

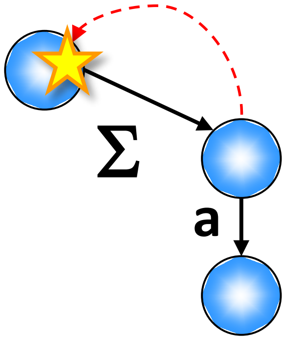
- Online DAWG construction follows the approach of online suffix trie construction.



ababcb

Online Construction of DAWGs

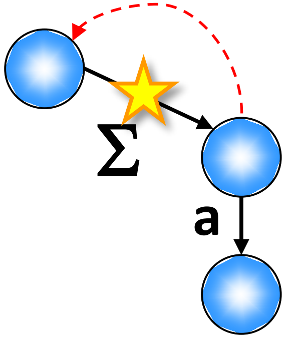
- Online DAWG construction follows the approach of online suffix trie construction.



ababcb

Online Construction of DAWGs

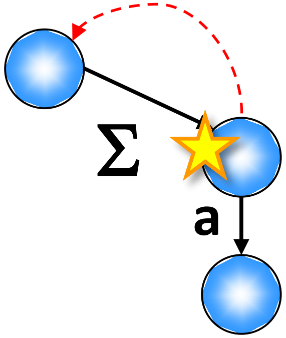
- Online DAWG construction follows
  the approach of online suffix trie construction.



ababcb

Online Construction of DAWGs

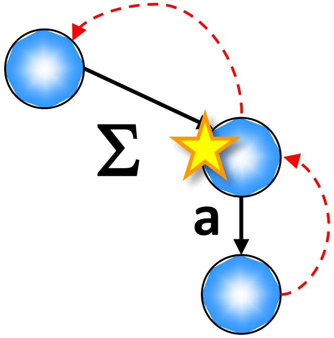
- Online DAWG construction follows the approach of online suffix trie construction.



ababcb

Online Construction of DAWGs

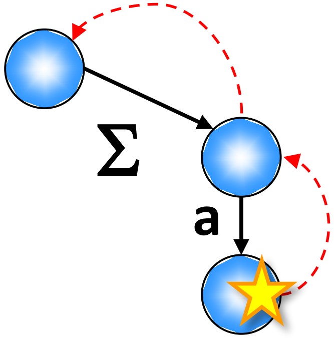
- Online DAWG construction follows the approach of online suffix trie construction.



ababcb

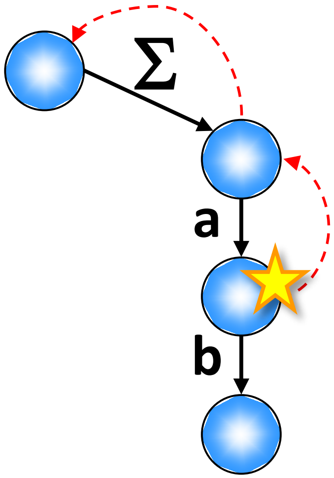
Online Construction of DAWGs

- Online DAWG construction follows the approach of online suffix trie construction.



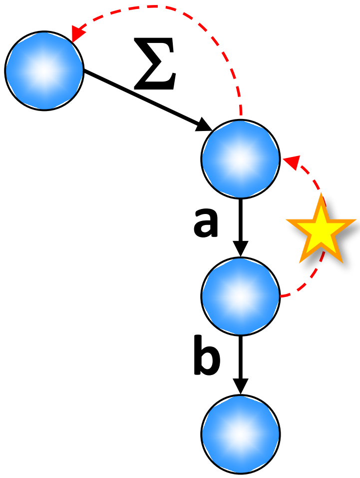
ababcb

Online Construction of DAWGs



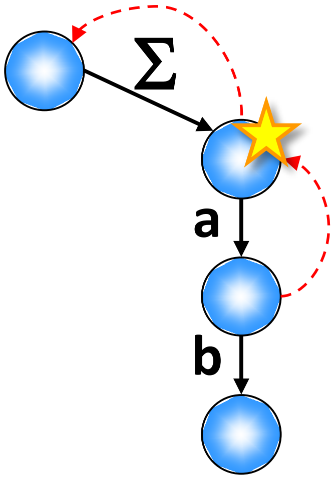
ababcb

Online Construction of DAWGs



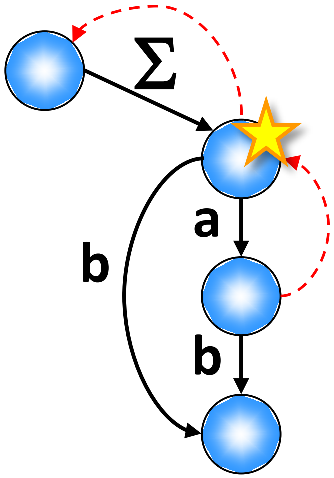
ababcb

Online Construction of DAWGs



ababcb

Online Construction of DAWGs

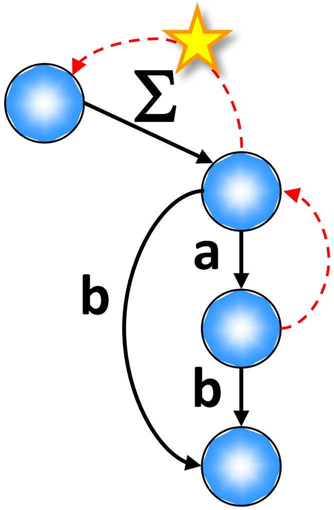


ababcb

Instead of making a new edge with a new leaf, we create a new edge which leads to the sink node.

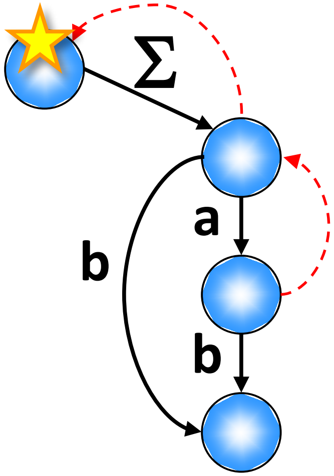
This corresponds to merging nodes of the suffix trie!

Online Construction of DAWGs



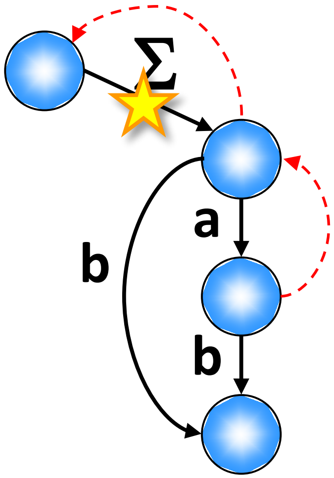
ababcb

Online Construction of DAWGs



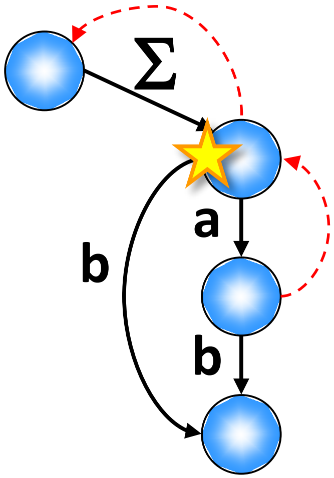
ababcb

Online Construction of DAWGs



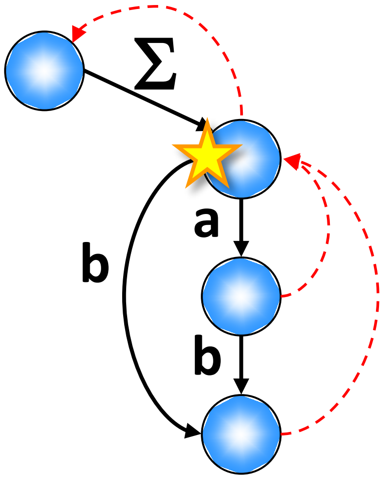
ababcb

Online Construction of DAWGs



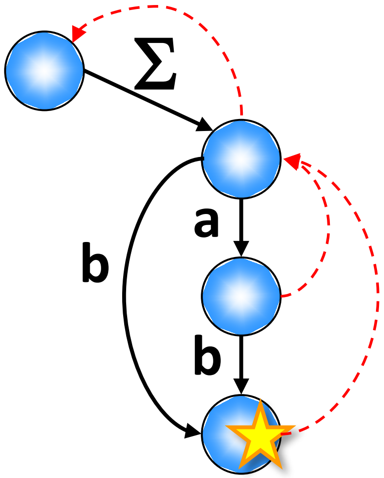
ababcb

Online Construction of DAWGs



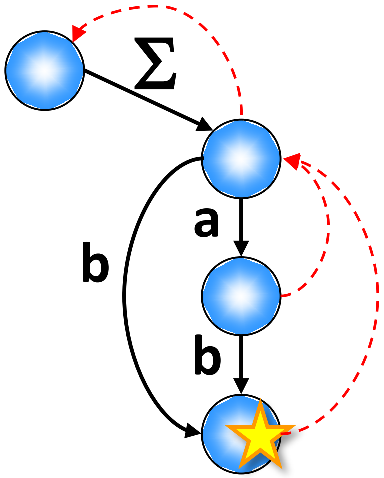
ababcb

Online Construction of DAWGs



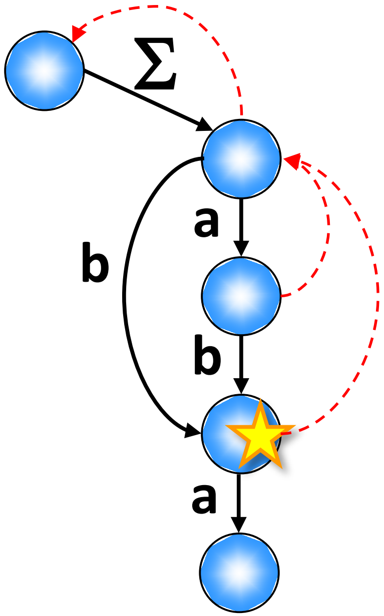
ababcb

Online Construction of DAWGs



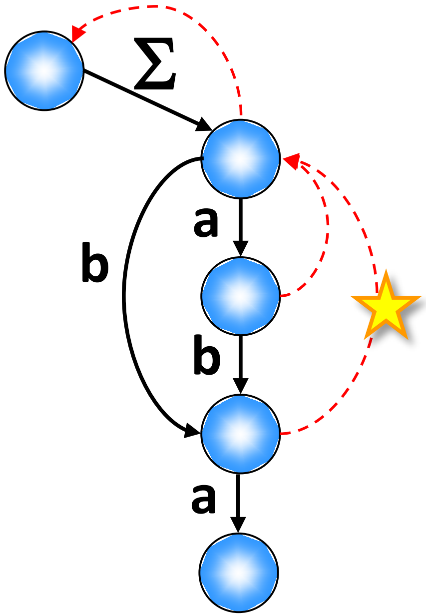
ababcb

Online Construction of DAWGs



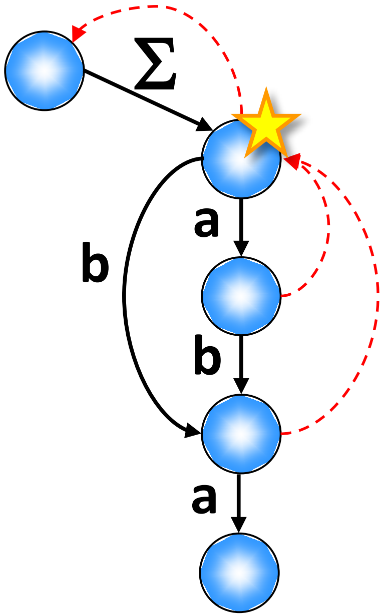
ababcb

Online Construction of DAWGs



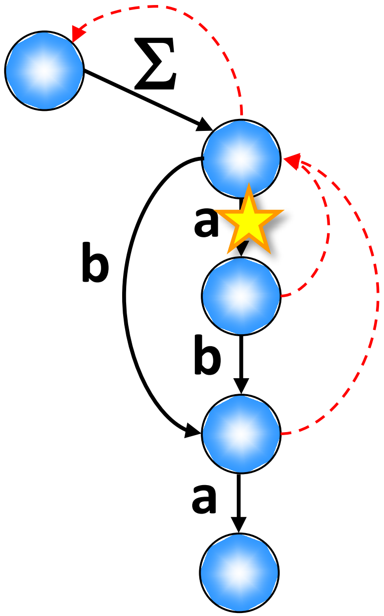
ababcb

Online Construction of DAWGs



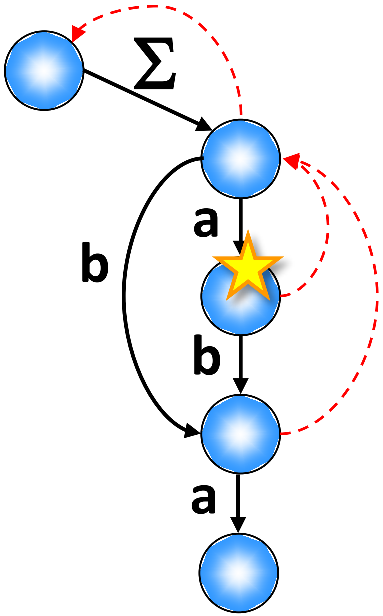
abababcb

Online Construction of DAWGs



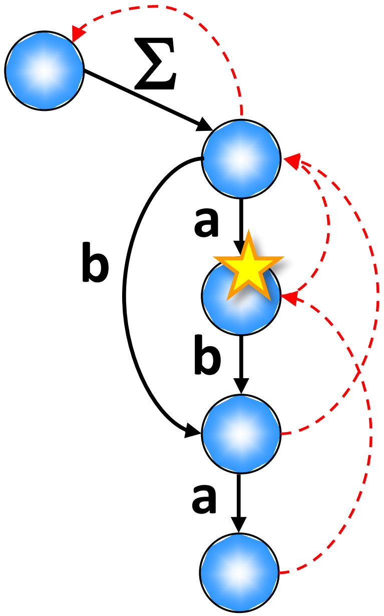
ababcb

Online Construction of DAWGs



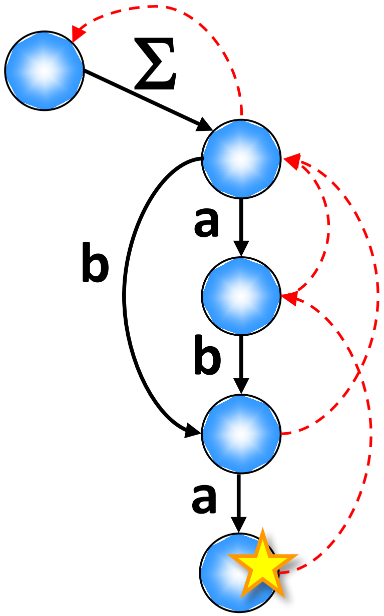
ababcb

Online Construction of DAWGs



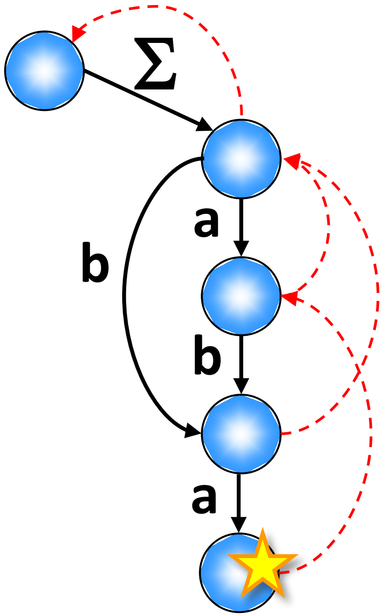
abababcb

Online Construction of DAWGs



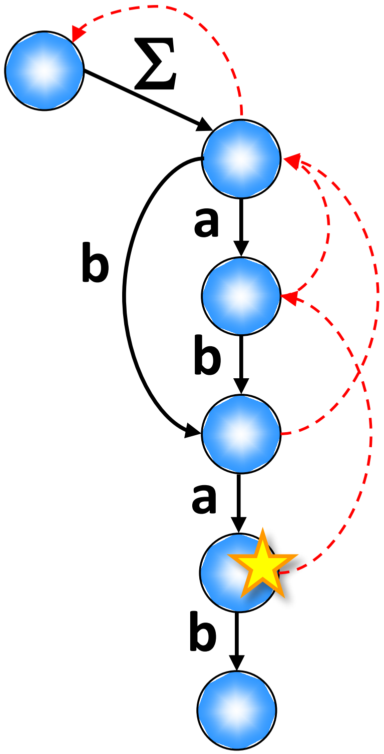
ababcb

Online Construction of DAWGs



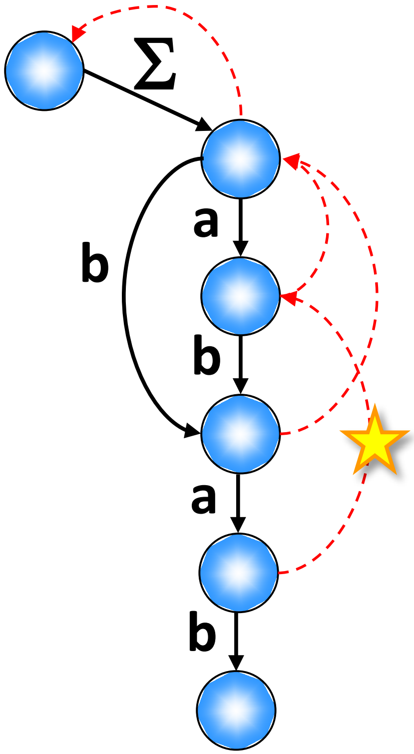
ababcb

Online Construction of DAWGs



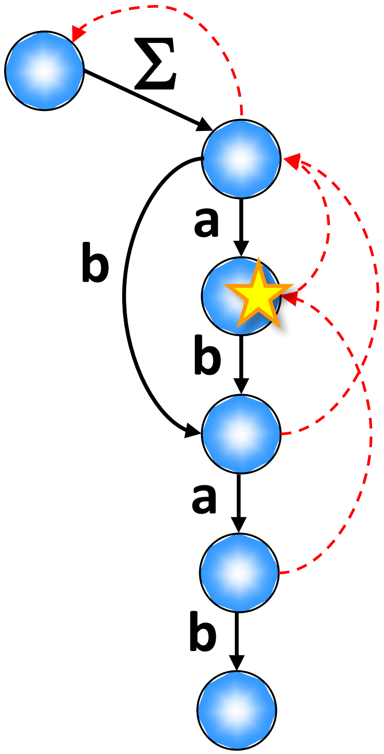
ababcb

Online Construction of DAWGs



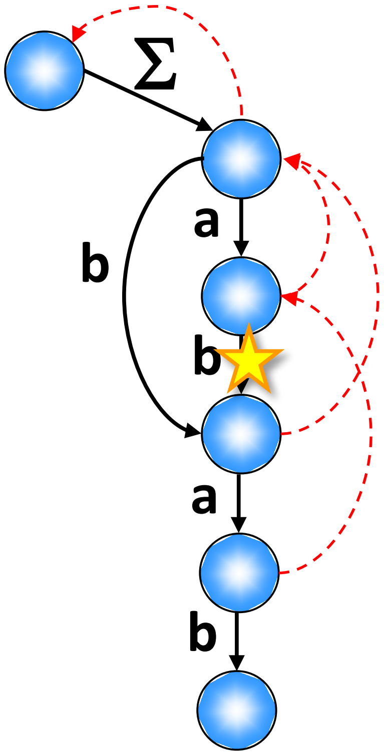
ababcb

Online Construction of DAWGs



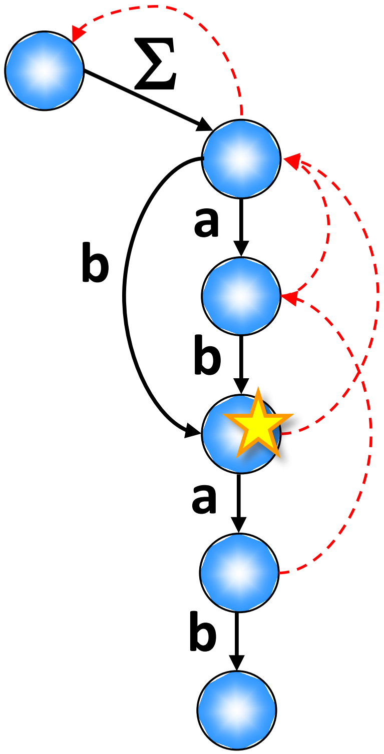
ababcb

Online Construction of DAWGs



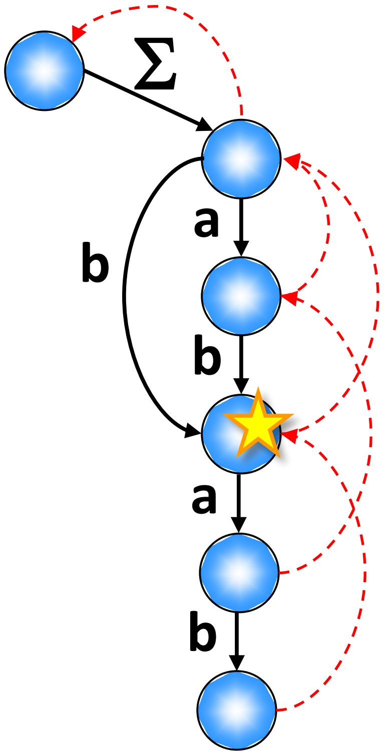
ababcb

Online Construction of DAWGs



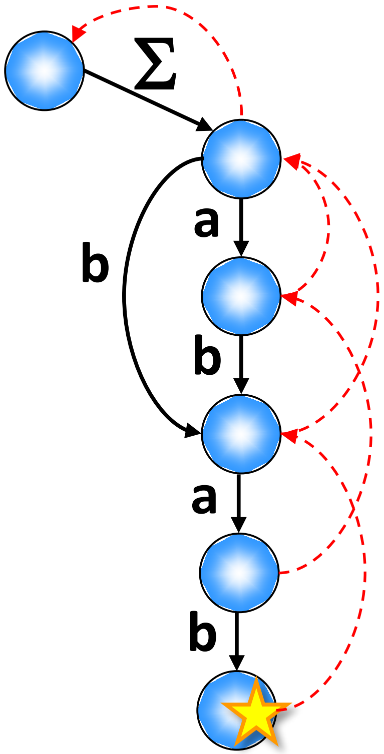
ababcb

Online Construction of DAWGs



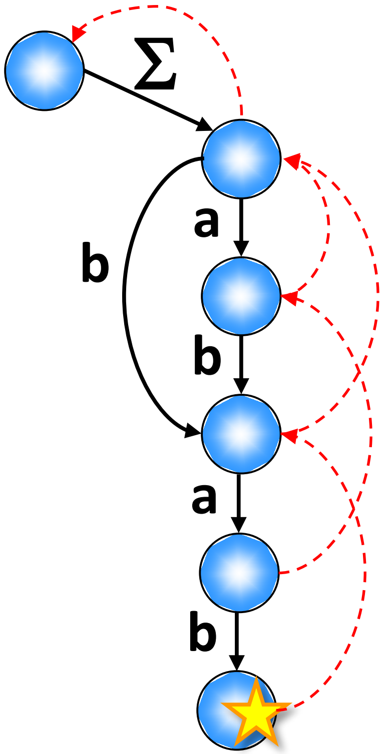
ababcb

Online Construction of DAWGs



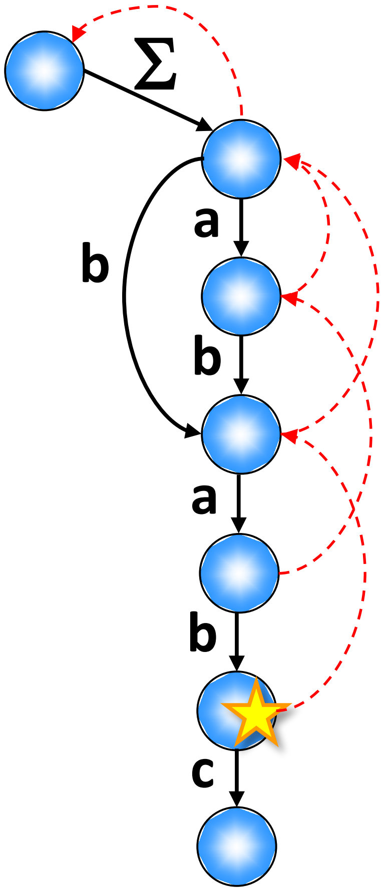
ababcb

Online Construction of DAWGs



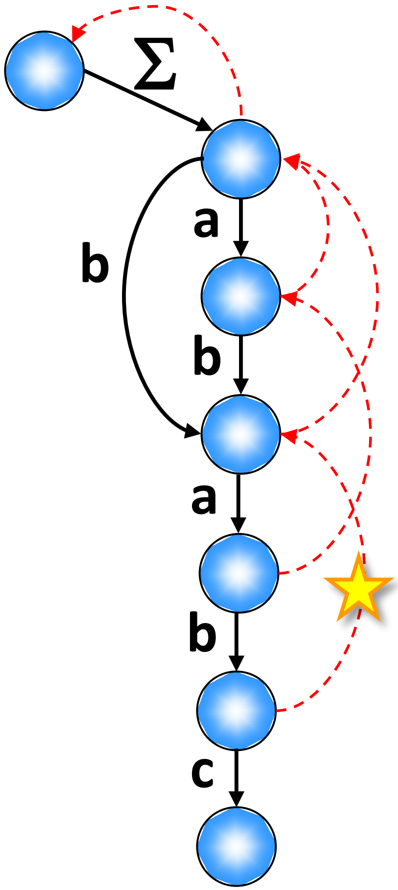
ababcb

Online Construction of DAWGs



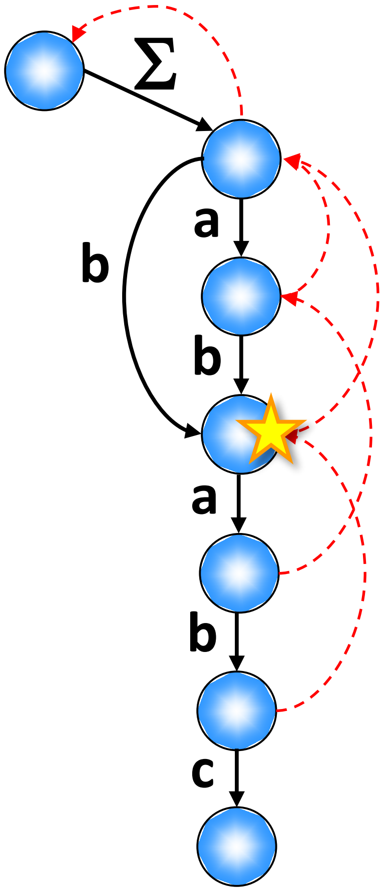
ababcb

Online Construction of DAWGs



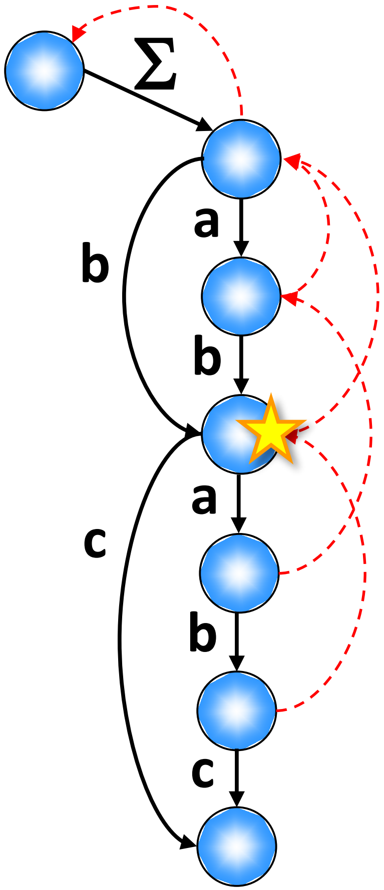
ababcb

Online Construction of DAWGs



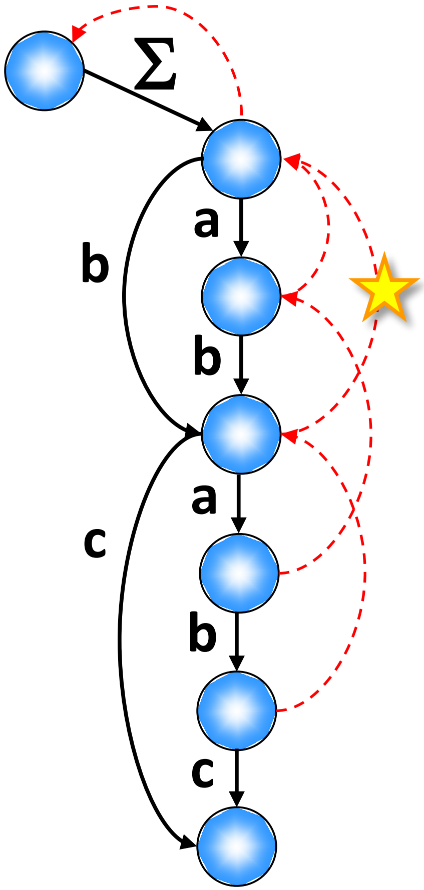
ababcb

Online Construction of DAWGs



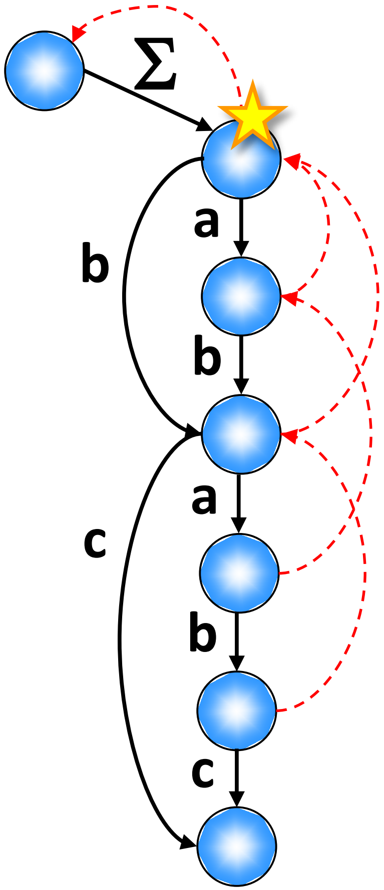
ababcb

Online Construction of DAWGs



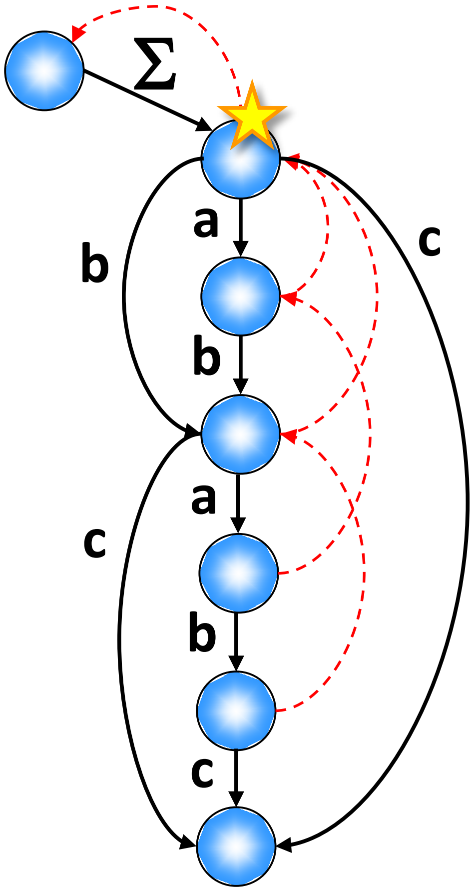
ababcb

Online Construction of DAWGs



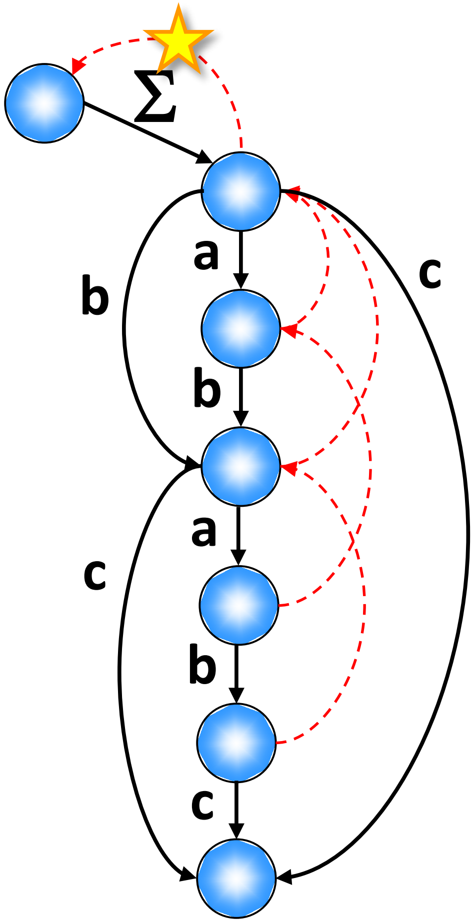
ababcb

Online Construction of DAWGs



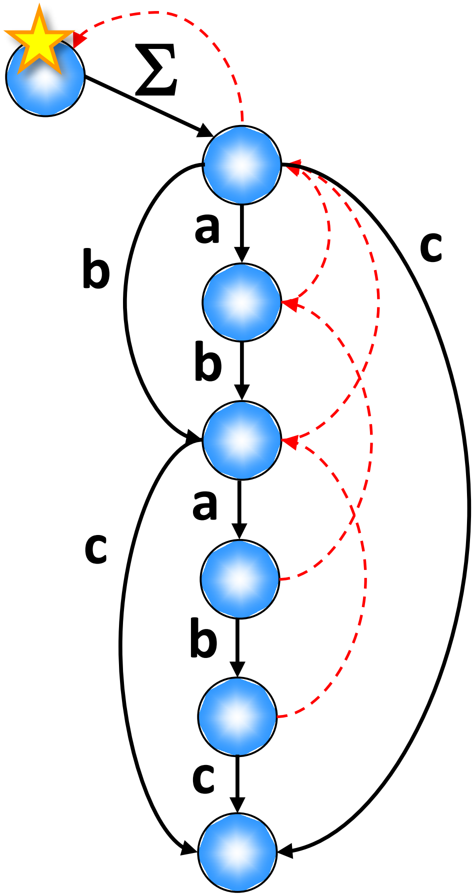
ababcb

Online Construction of DAWGs



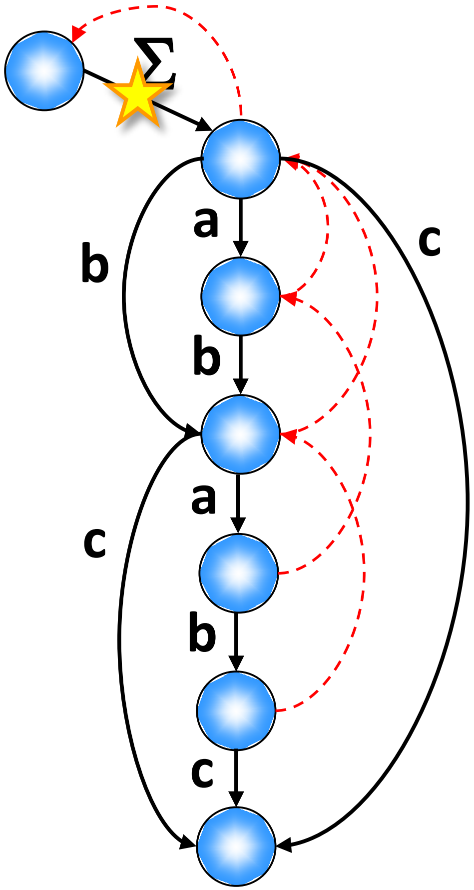
ababcb

Online Construction of DAWGs



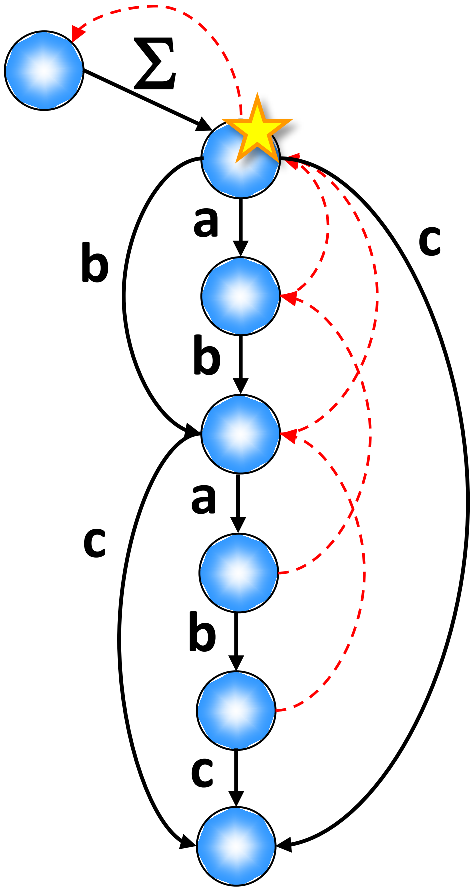
ababcb

Online Construction of DAWGs



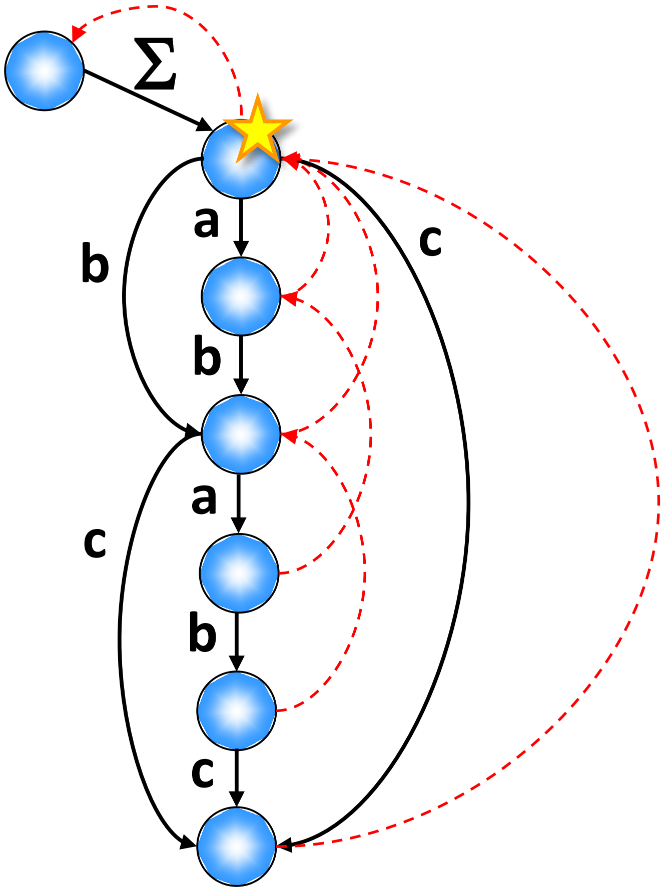
ababcb

Online Construction of DAWGs



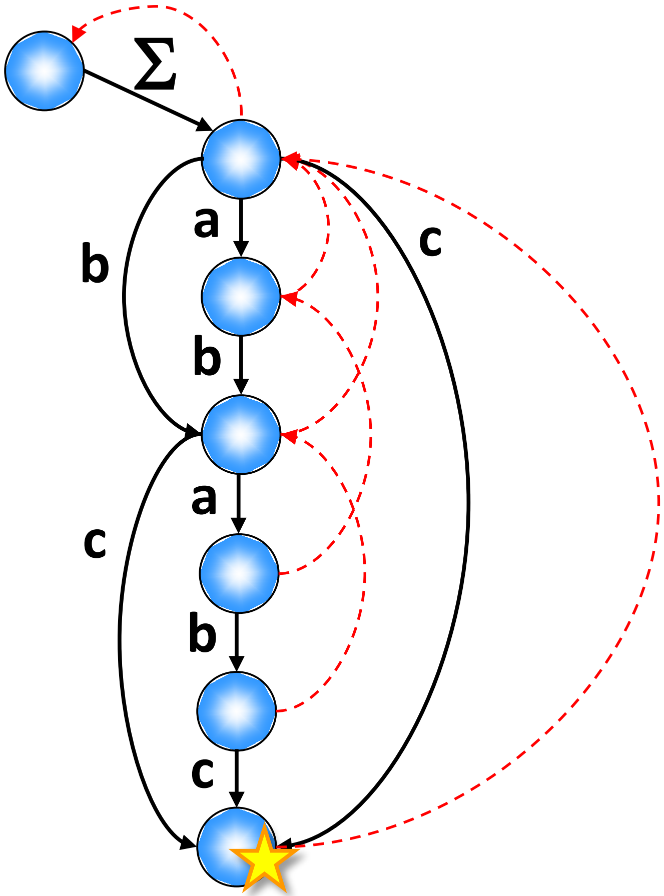
ababcb

Online Construction of DAWGs



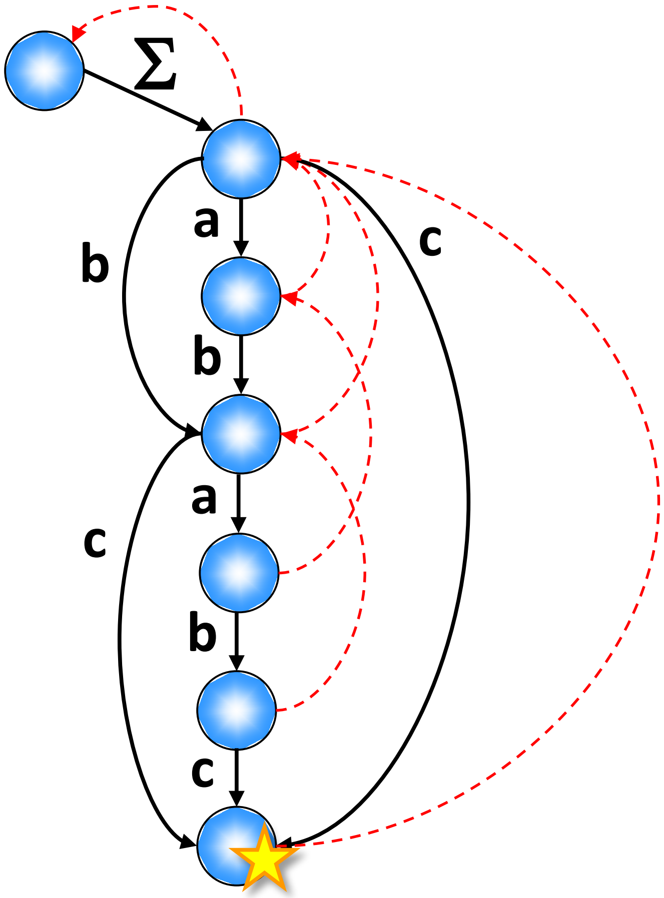
ababcb

Online Construction of DAWGs



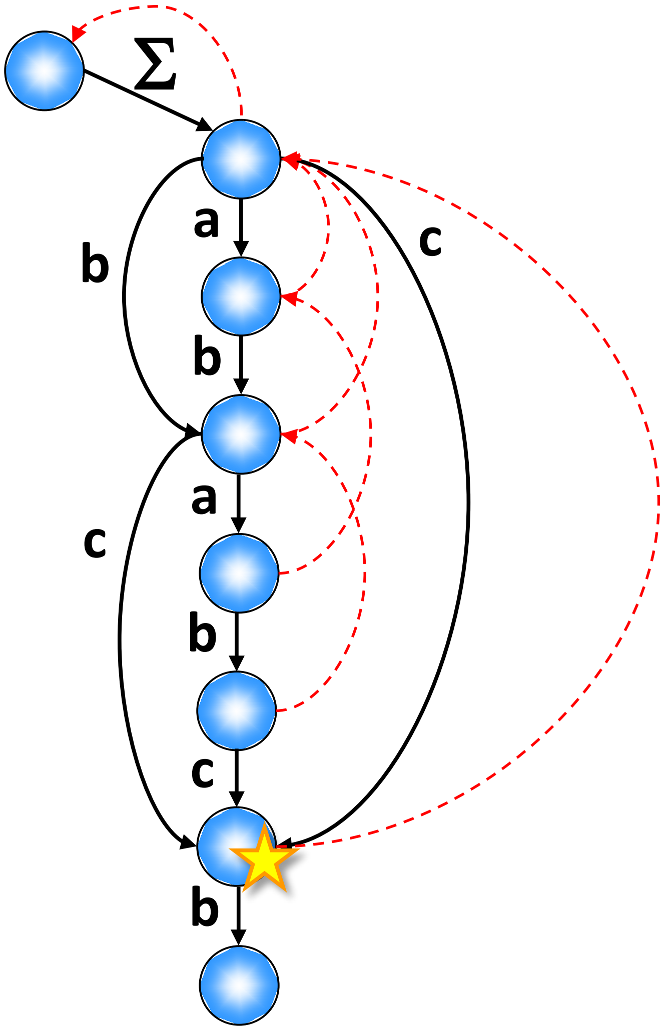
ababcb

Online Construction of DAWGs



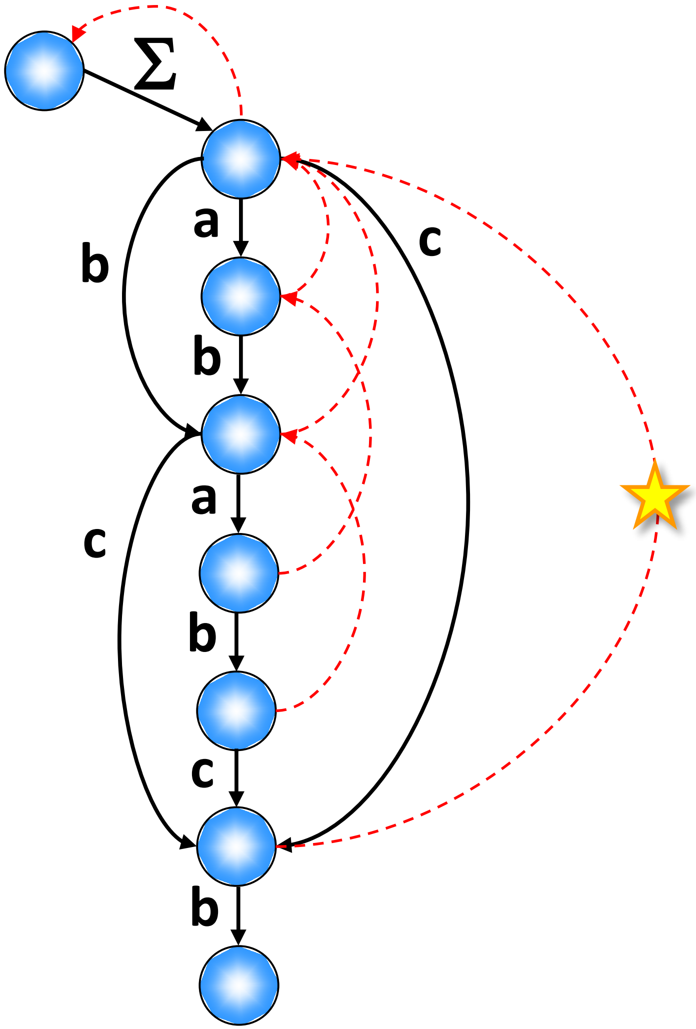
ababcb

Online Construction of DAWGs



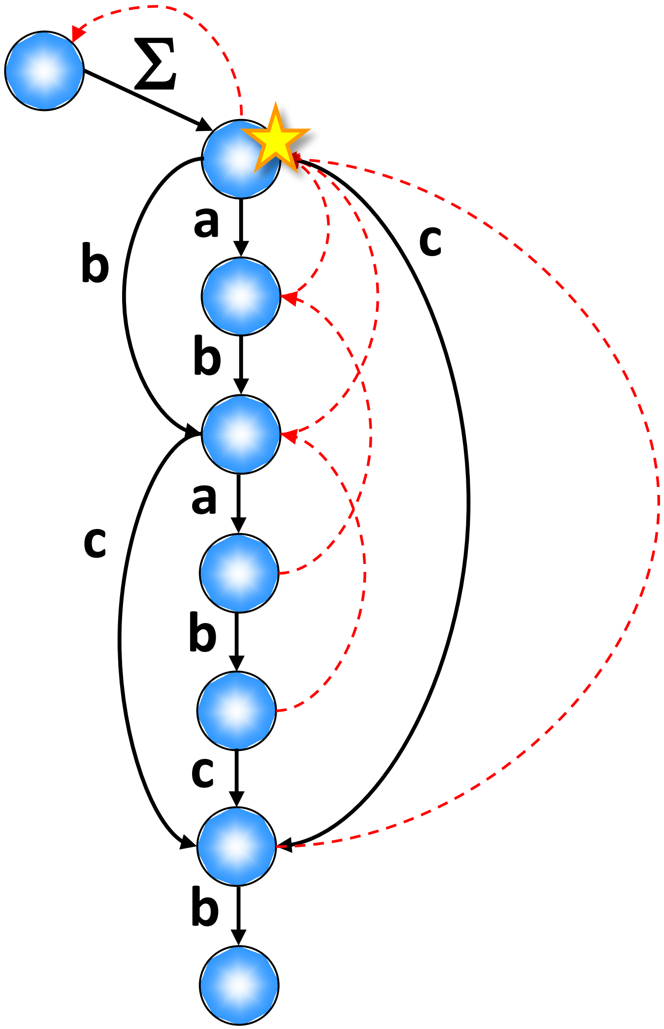
ababcb

Online Construction of DAWGs



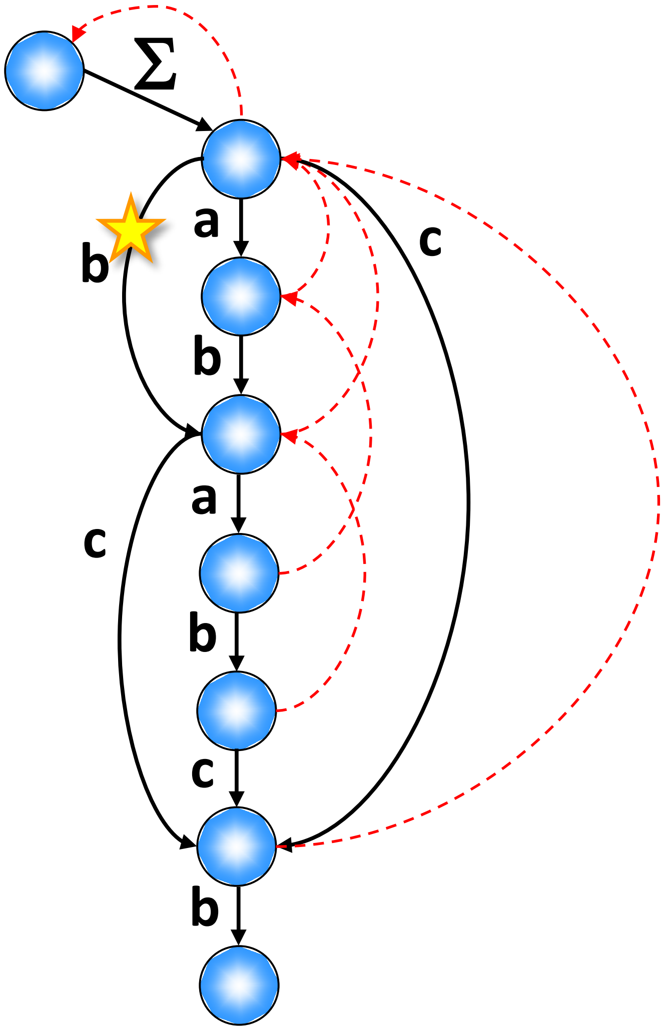
ababcb

Online Construction of DAWGs



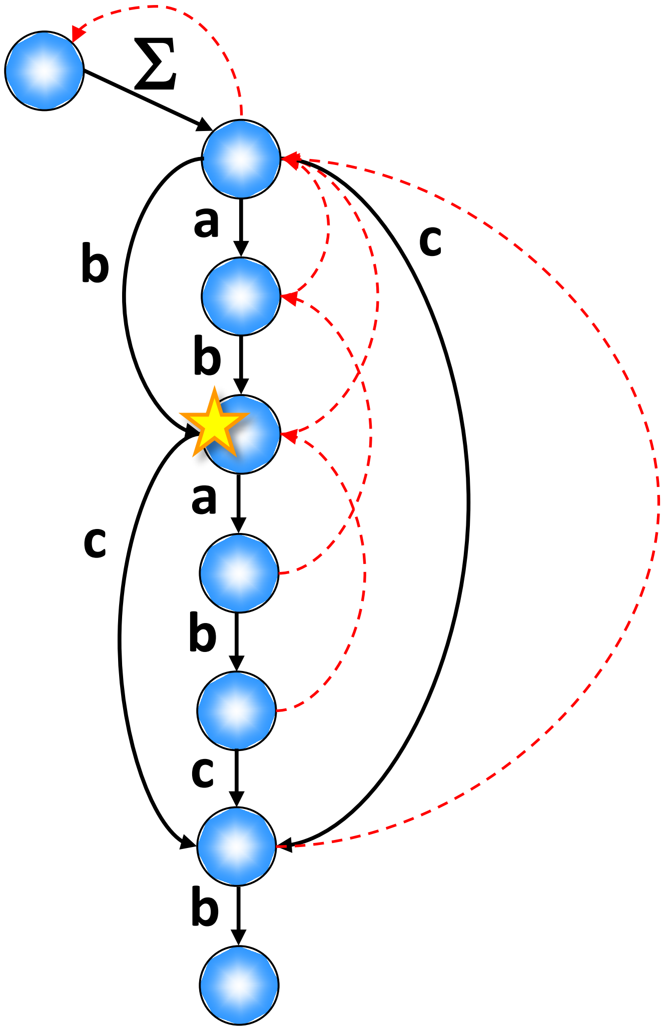
ababcb

Online Construction of DAWGs



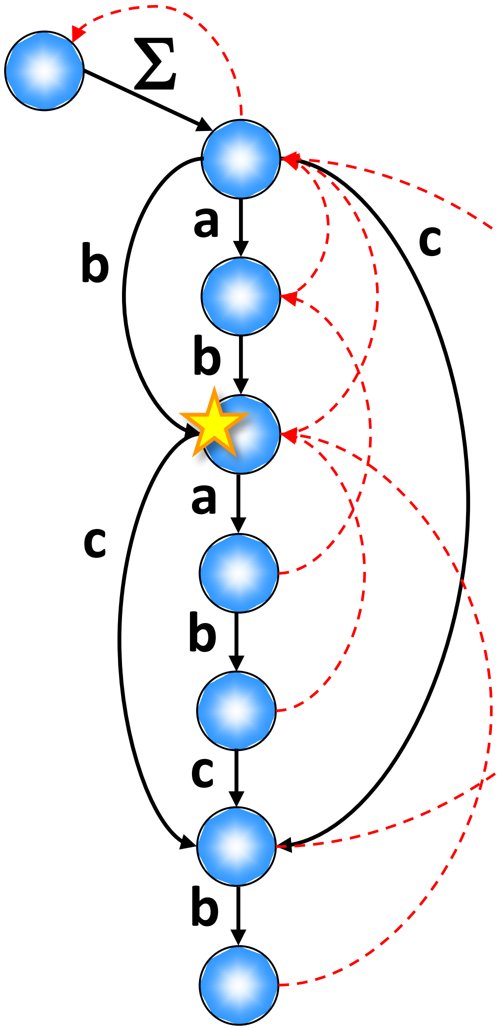
ababcb

Online Construction of DAWGs

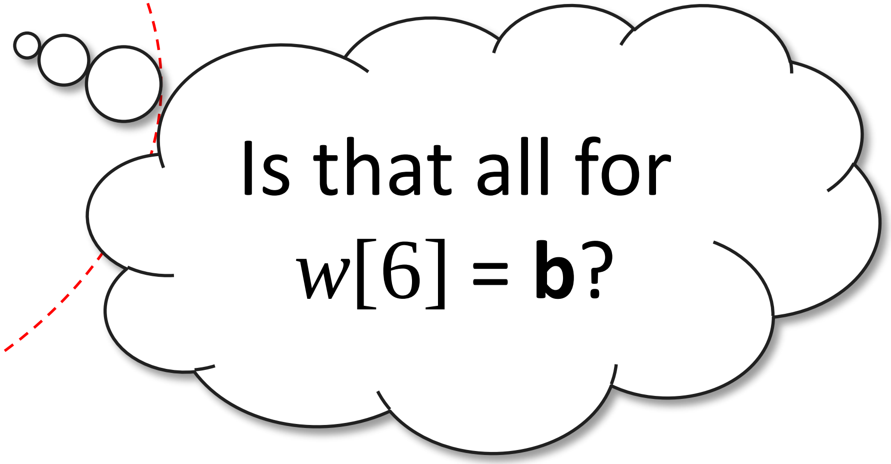


ababcb

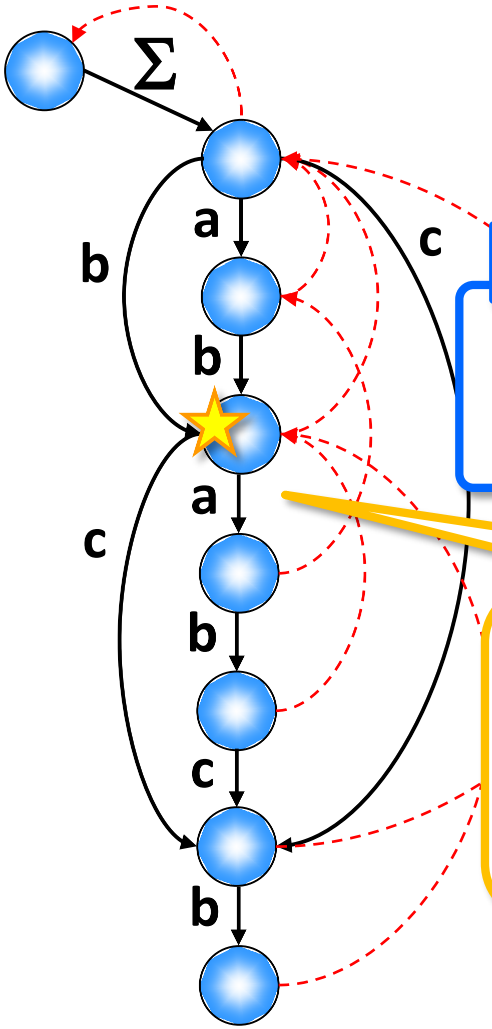
Online Construction of DAWGs



ababcb


Is that all for $w[6] = \mathbf{b}$?

Online Construction of DAWGs



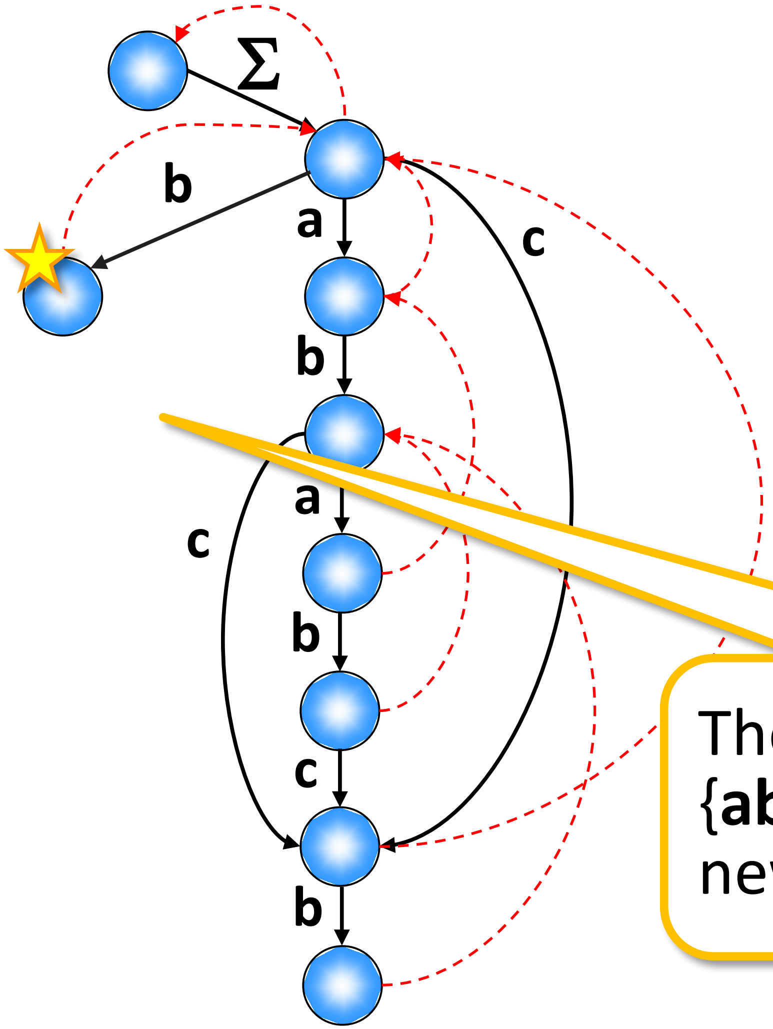
ababcb

Theorem 1

The DAWG of string w is the smallest automaton that **recognizes all suffixes** of w.

b is a suffix of $w[1..6]$ = **ababcb**, but **ab** isn't.
So, node {**ab**, **b**} must be split.

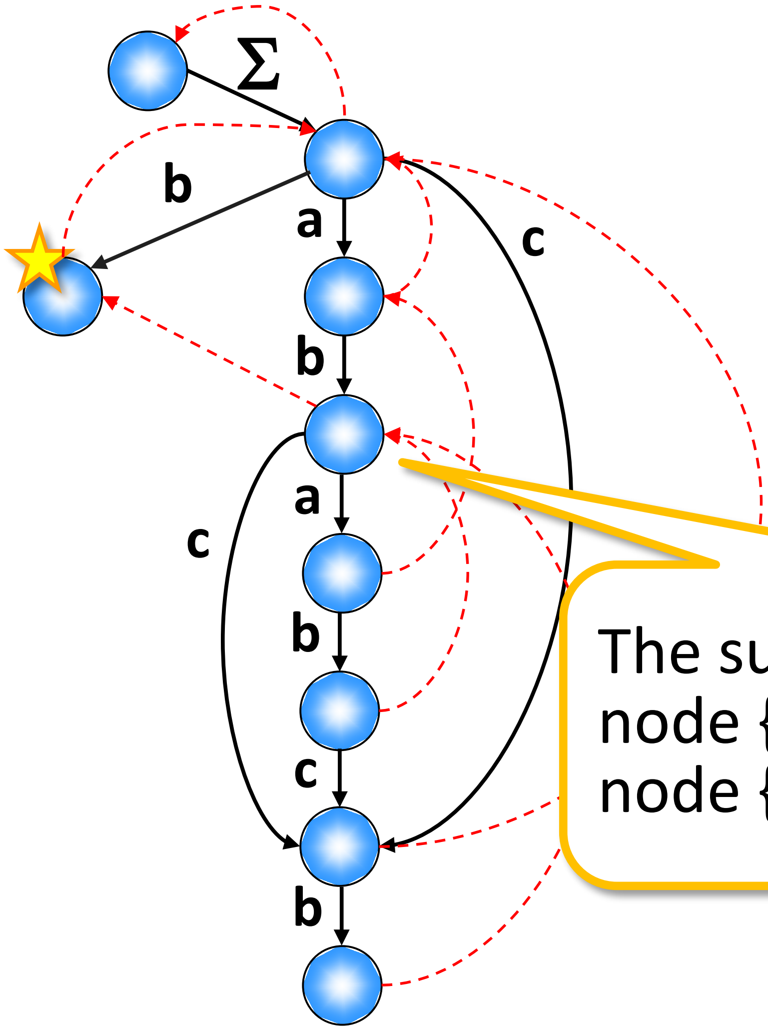
Online Construction of DAWGs



ababcb

The suffix link of old node {**ab**, **b**} is inherited by the new node {**b**}.

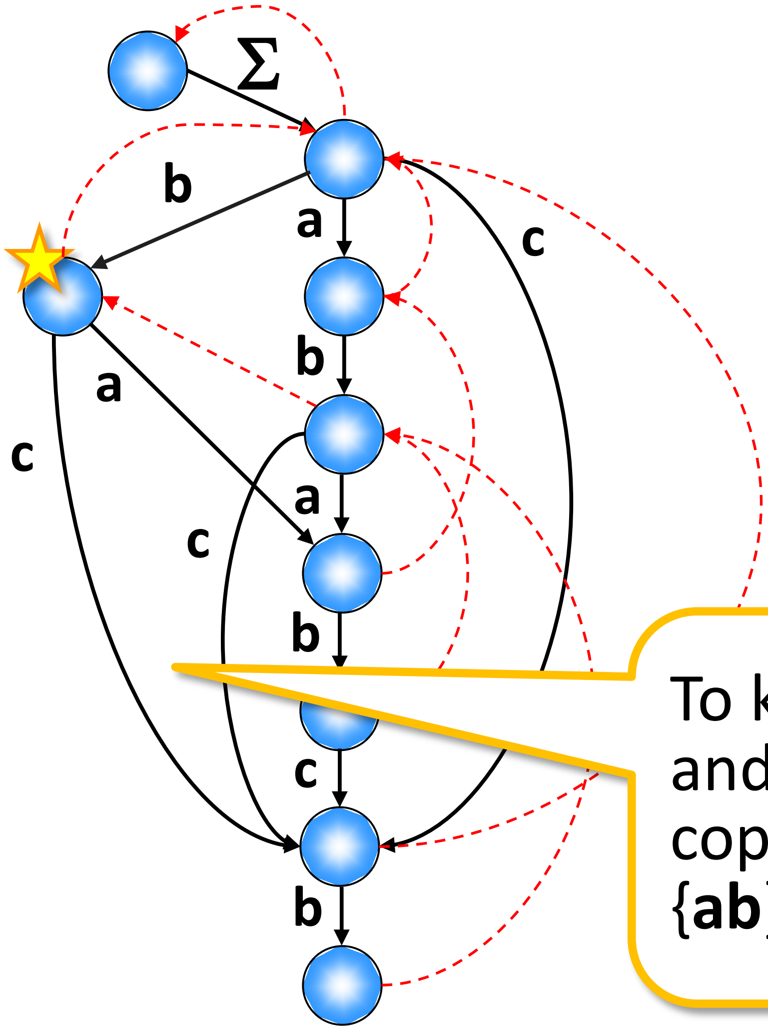
Online Construction of DAWGs



ababcb

The suffix link from the other node {**ab**} points to the new node {**b**}.

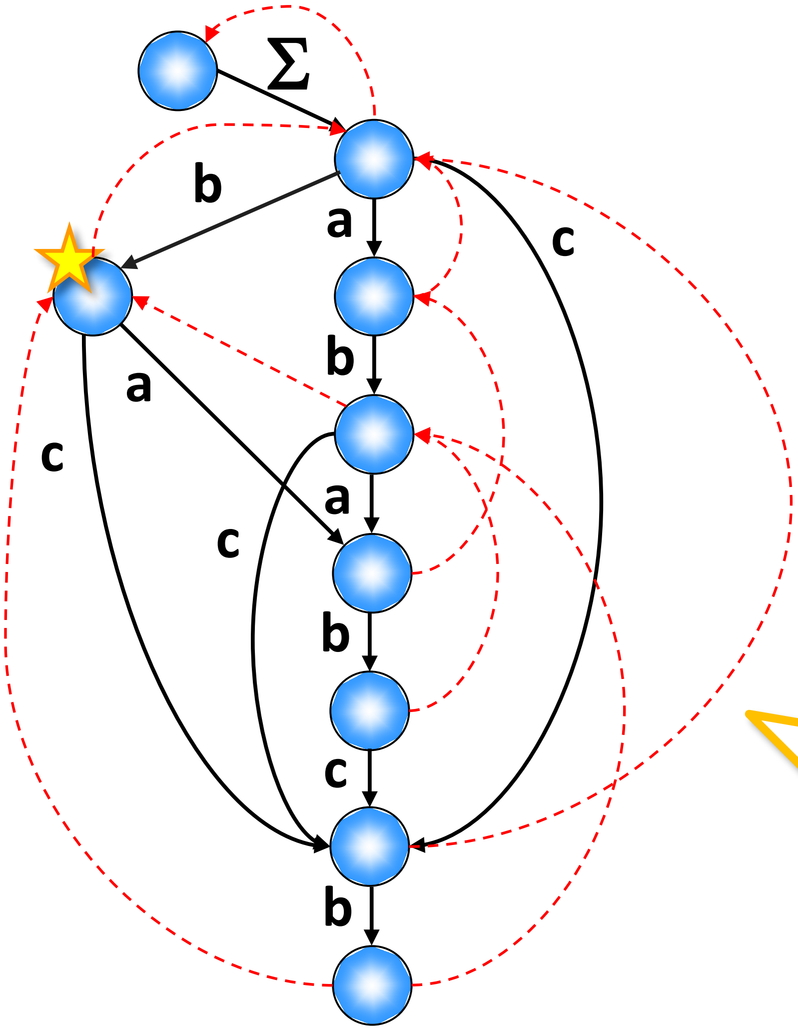
Online Construction of DAWGs



ababcb

To keep the path for **babcb** and **bcb** from node {**b**}, we copy the out-edges of node {**ab**}.

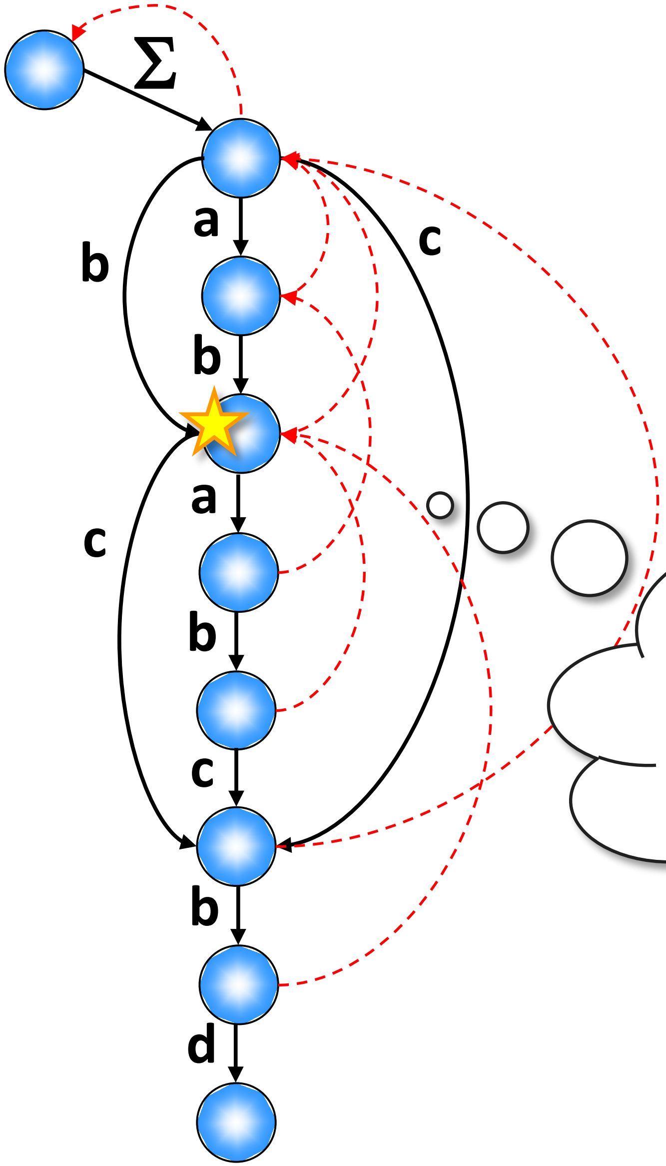
Online Construction of DAWGs



ababcb

The DAWG for string **ababcb** is complete.

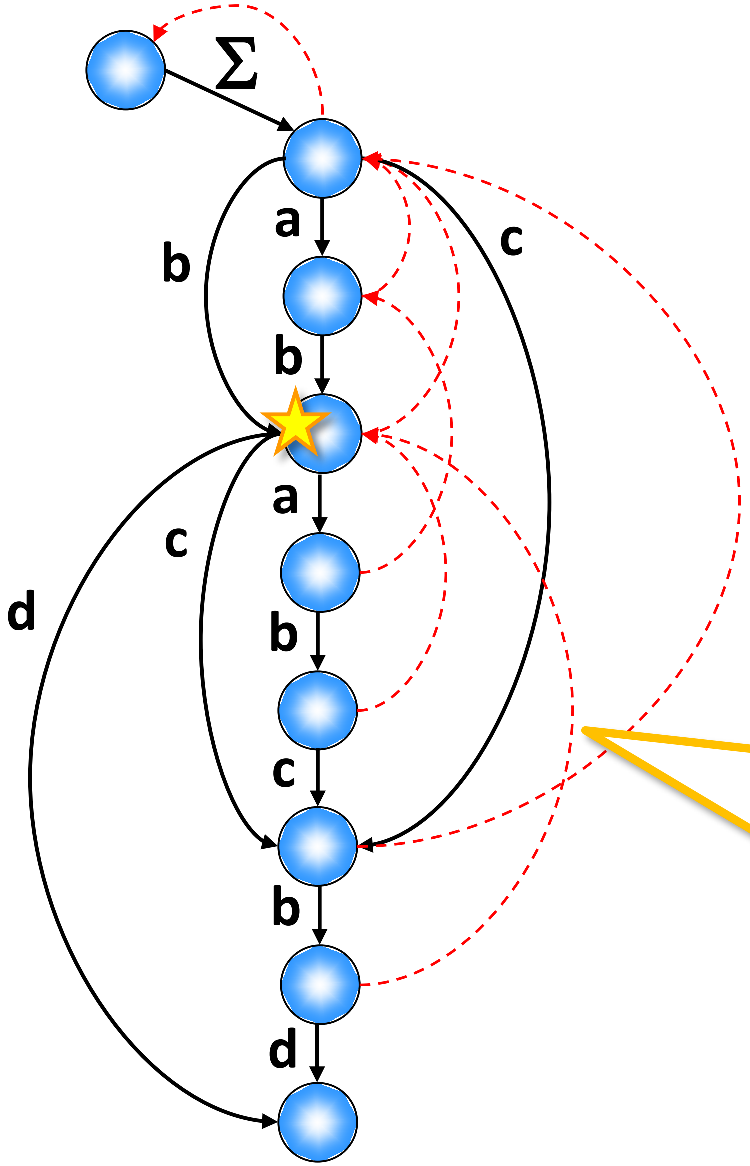
Node Split is Essential



ababcbd

What would happen if we did not split the node {**ab**, **b**} ?

Node Split is Essential



ababcbd

Now the DAG has a path for **abd**, but this does not appear in string **ababcbd**!

Online Construction of DAWGs

Theorem 6

The DAWG of a given string of length n can be constructed online (left-to-right) in $O(n \log \sigma)$ time, where $\sigma = |\Sigma|$.

- Clearly, the amount of work is proportional to the numbers of nodes, edges, and suffix links in the DAWG, each of which is $O(n)$.
- The $\log \sigma$ factor is to maintain BSTs for searching branches.

Left-to-right Construction of DAWG ⇒ Right-to-left Construction of Suffix Tree

Corollary 3

The suffix tree of a given string of length n can be constructed online (right-to-left) in $O(n \log \sigma)$ time.

- Immediate from Theorem 6.
- This corollary generalizes Weiner's right-to-left suffix tree construction.

DAWG Construction for Integer Alphabets

Theorem 7

The edge-sorted DAWG of a given string w of length n over an integer alphabet $\Sigma = \{1, \ldots, n^{O(1)}\}$ can be constructed in $O(n)$ time.

- Build the suffix tree of w with suffix links in $O(n)$ time [Farach-Colton et al., 2000].
- Build DAWG with suffix links from suffix tree.
- Edges can be sorted in $O(n)$ time by bucket sort.

Recommended Reading (1/3)

- "The Smallest Automaton Recognizing the Subwords of a Text", Blumer et al., *TCS*, 1985.
  - Introduced DAWGs.
  - Duality with suffix trees.
  - Online $O(n \log \sigma)$-time DAWG construction algorithm.
- *Text Algorithms*, Crochemore & Rytter, Oxford University Press, 1994.
  - Text book. Chapter 6 is devoted for DAWGs.
  - Free(!) copy is available online at http://www.mimuw.edu.pl/~rytter/BOOKS/text-algorithms.pdf

Recommended Reading (2/3)

- "Automata and Forbidden Words", Crochemore et al., *IPL*, 1998.
  - DAWG-based $O(\sigma n)$-time algorithm for finding all MAWs.
- "Linear-Time Sequence Comparison Using Minimal Absent Words & Applications", Crochemore et al., LATIN, 2016.
  - String similarity measure based on MAWs.

Recommended Reading (3/3)

- “Computing DAWGs and Minimal Absent Words in Linear Time for Integer Alphabets”, Fujishige et al., *MFCS*, 2016 (to appear).
  - Offline $O(n)$-time DAWG construction algorithm for integer alphabets of size $n^{O(1)}$.
  - $O(n + |MAW_w|)$-time algorithm for finding all MAWs.
- “Fully-online Construction of Suffix Trees for Multiple Texts”, Takagi et al., *CPM*, 2016.
  - Fully-online $O(n \log \sigma)$ -time DAWG construction algorithm for multiple strings.