# Enumeration for MSO-Queries on Compressed Trees

Markus Lohrey[1], **Markus L. Schmid**[2]

[1] Universität Siegen, Germany
[2] HU Berlin, Germany

PODS 2024

# Enumeration of MSO-Queries over Trees

**Input**:

MSO-query $q(X)$.

A node-labelled tree $T$ with node set $V_T$.

**Task**:

Enumerate $q(T) = \{S \subseteq V_T \mid T \models q(S)\}$.

# Enumeration of MSO-Queries over Trees

**Input**:

MSO-query $q(X)$.

A node-labelled tree $T$ with node set $V_T$.

**Task**:

Enumerate $q(T) = \{S \subseteq V_T \mid T \models q(S)\}$.

Problem investigated in:

Bagan, CSL 2006
Courcelle, Discrete Applied Mathematics 2009
Kazana, Segoufin, ACM Trans. Comput. Log. 2013
Niewerth, LICS 2018
Amarilli, Bourhis, Mengel, Niewerth, PODS 2019

# Enumeration of MSO-Queries over Compressed Trees

**Input**:

MSO-query $q(X)$.

A compressed node-labelled tree $T$ with node set $V_T$.

**Task**:

Enumerate $q(T) = \{S \subseteq V_T \mid T \models q(S)\}$.

# The Compression Scheme: Straight-Line Programs (SLPs)

A lossless compression scheme most famous for strings.

Practically highly relevant: Fast compressors with good compression.

Lots of practical and theoretical work on SLPs.

Have been applied in database theory.

# SLPs – How Do They Work?

String: $\mathtt{aabbabaabaabbab} \in \Sigma^*$

# SLPs – How Do They Work?

String: $\mathtt{aabbabaabaabbab} \in \Sigma^*$

Formula over $(\Sigma^*, \odot)$, where "$\odot$" is string concatenation:

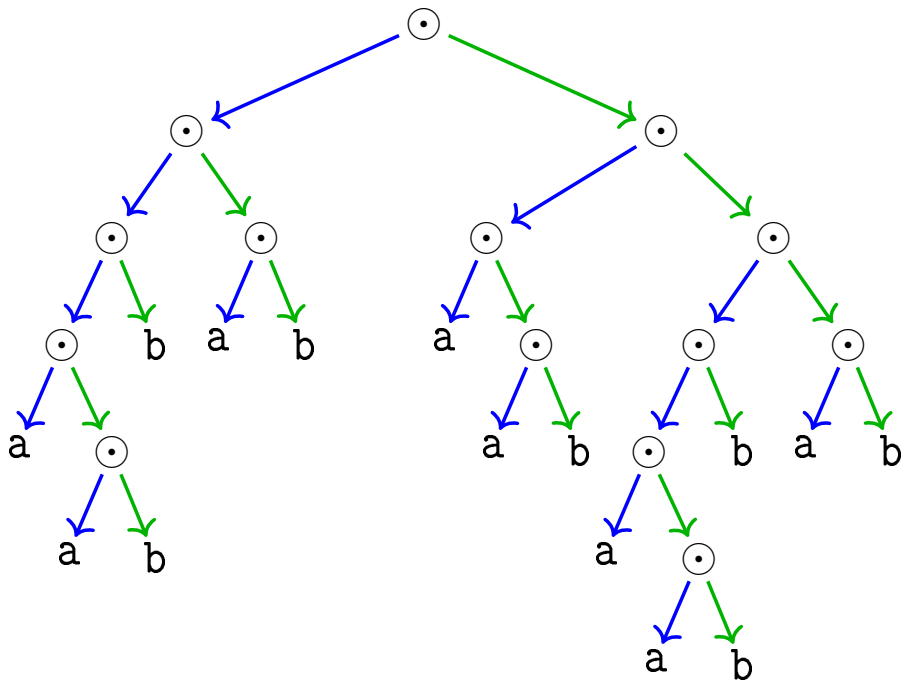
$((((\mathtt{a} \odot (\mathtt{a} \odot \mathtt{b})) \odot \mathtt{b})) \odot (\mathtt{a} \odot \mathtt{b})) \odot ((\mathtt{a} \odot (\mathtt{a} \odot \mathtt{b})) \odot (((\mathtt{a} \odot (\mathtt{a} \odot \mathtt{b})) \odot \mathtt{b}) \odot (\mathtt{a} \odot \mathtt{b})))$

# SLPs – How Do They Work?

String: $\mathtt{aabbabaabaabbbab} \in \Sigma^*$

Formula over $(\Sigma^*, \odot)$, where "$\odot$" is string concatenation:

$((((\mathtt{a} \odot (\mathtt{a} \odot \mathtt{b})) \odot \mathtt{b}) \odot (\mathtt{a} \odot \mathtt{b})) \odot ((\mathtt{a} \odot (\mathtt{a} \odot \mathtt{b})) \odot (((\mathtt{a} \odot (\mathtt{a} \odot \mathtt{b})) \odot \mathtt{b}) \odot (\mathtt{a} \odot \mathtt{b})))$



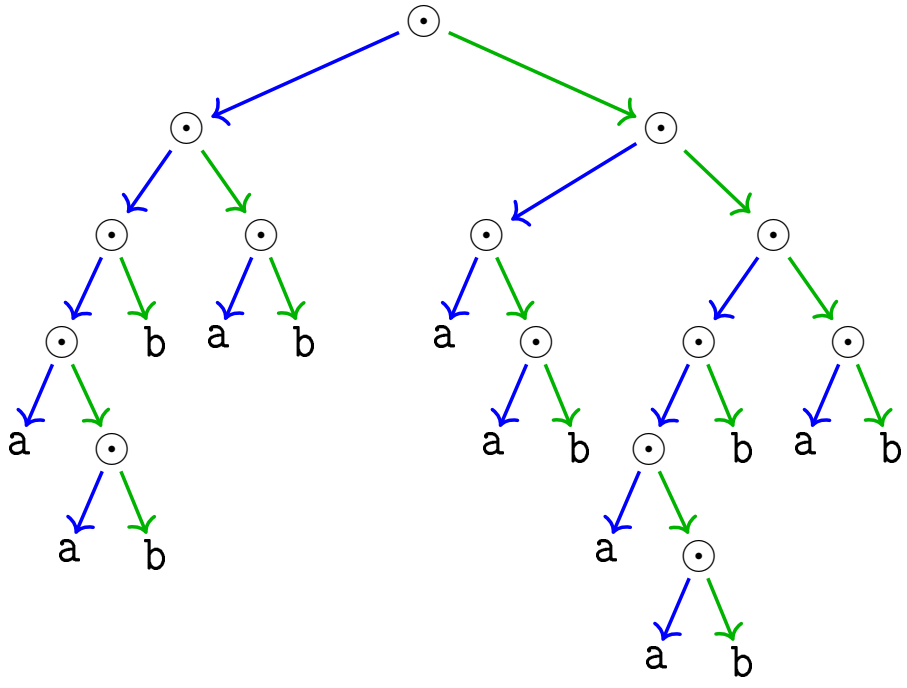

Formula as syntax tree.

# SLPs – How Do They Work?

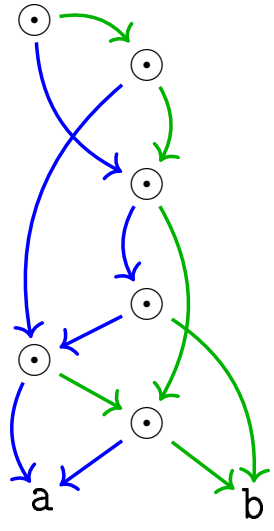
String: $\mathtt{aabbabaabaabbbab} \in \Sigma^*$

Formula over $(\Sigma^*, \odot)$, where "$\odot$" is string concatenation:

$((((\mathtt{a} \odot (\mathtt{a} \odot \mathtt{b})) \odot \mathtt{b})) \odot (\mathtt{a} \odot \mathtt{b})) \odot ((\mathtt{a} \odot (\mathtt{a} \odot \mathtt{b})) \odot (((\mathtt{a} \odot (\mathtt{a} \odot \mathtt{b})) \odot \mathtt{b}) \odot (\mathtt{a} \odot \mathtt{b})))$




Formula as syntax tree.




DAG-folding.
(SLP)

# SLPs for Trees and Forests

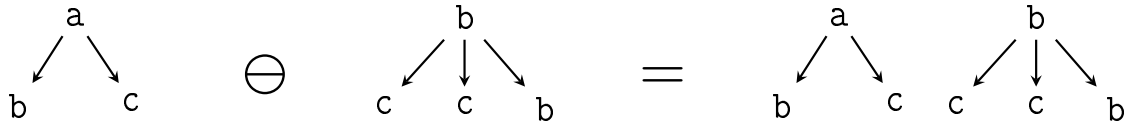
Instead of string concatenation, use an algebra for trees!

# SLPs for Trees and Forests

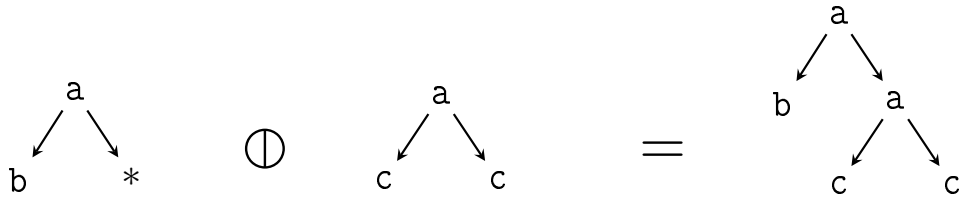
Instead of string concatenation, use an algebra for trees!

Forest Algebra (Bojańczyk, Walukiewicz, 2008):

Horizontal concatenation:




Vertical concatenation:

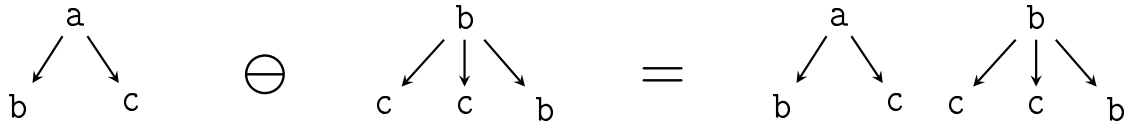


# SLPs for Trees and Forests

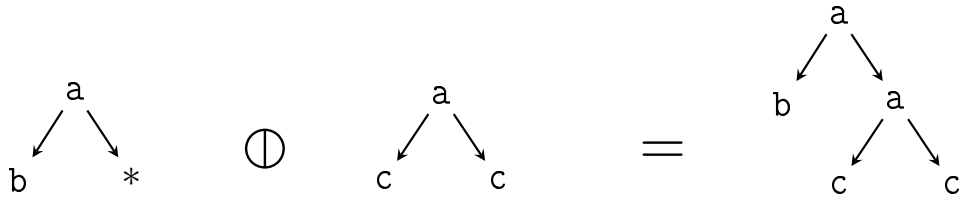
Instead of string concatenation, use an algebra for trees!

Forest Algebra (Bojańczyk, Walukiewicz, 2008):

Horizontal concatenation:




Vertical concatenation:




Atomic elements: For every $x \in \Sigma$,

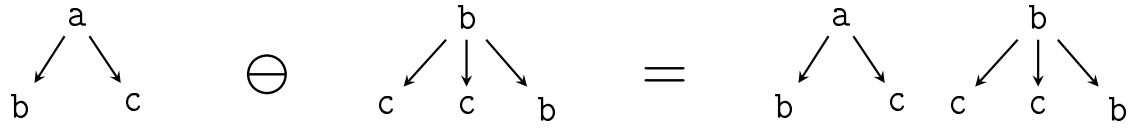
$x$ represents a single $x$-labelled node and $x_*$ represents $\overset{x}{\downarrow}{}^{*}$.

# SLPs for Trees and Forests

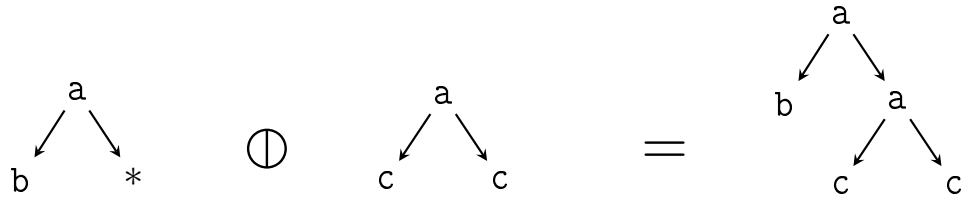
Instead of string concatenation, use an algebra for trees!

Forest Algebra (Bojańczyk, Walukiewicz, 2008):

Horizontal concatenation:



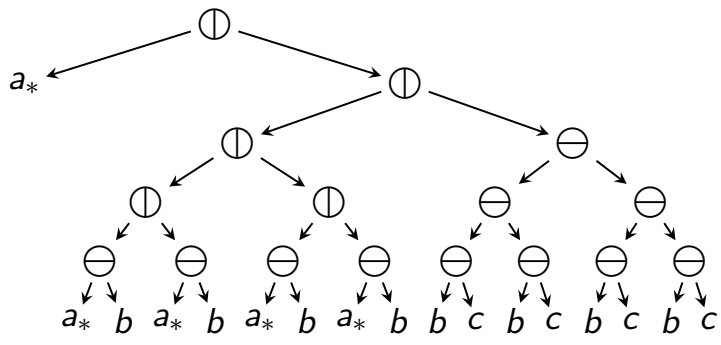

Vertical concatenation:



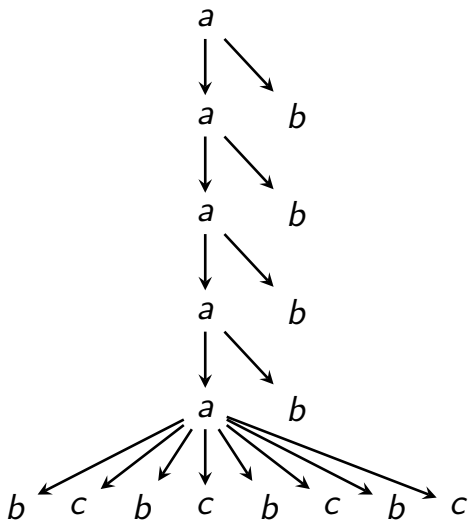

Atomic elements: For every $x \in \Sigma$,

$x$ represents a single $x$-labelled node and $x_*$ represents $\overset{x}{\downarrow}{*}$.

Example formula:



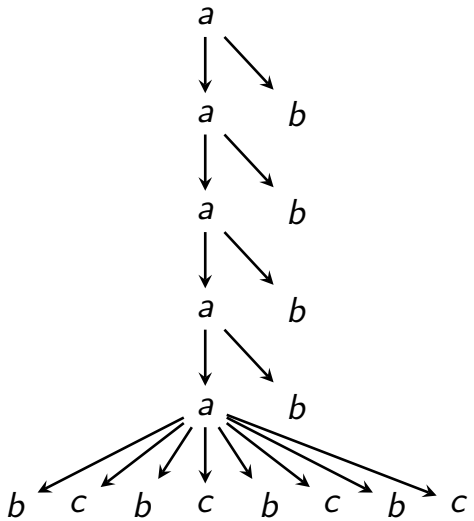
# Forest Straight-Line Programs (FSLPs)



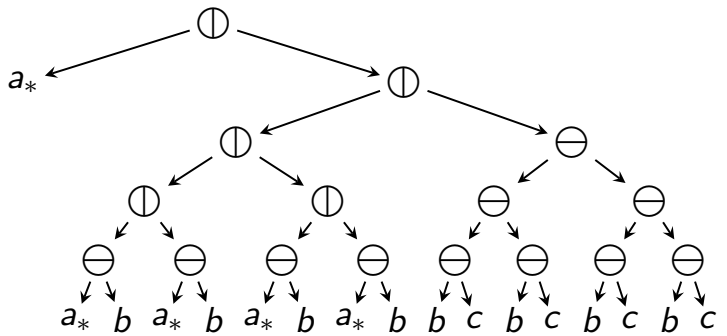

An unranked forest $F$.

Gascón, Lohrey, Maneth, Reh, Sieber, 2020

# Forest Straight-Line Programs (FSLPs)



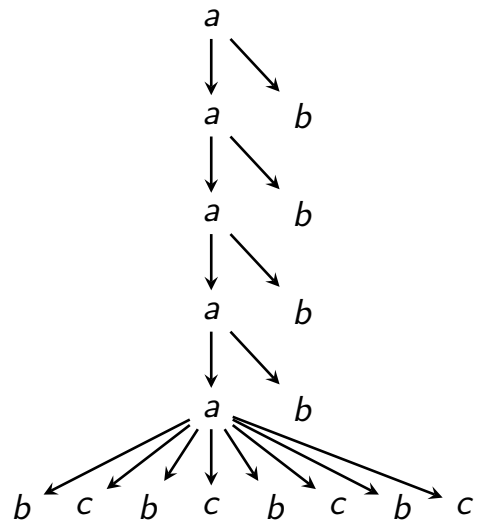

An unranked forest $F$.



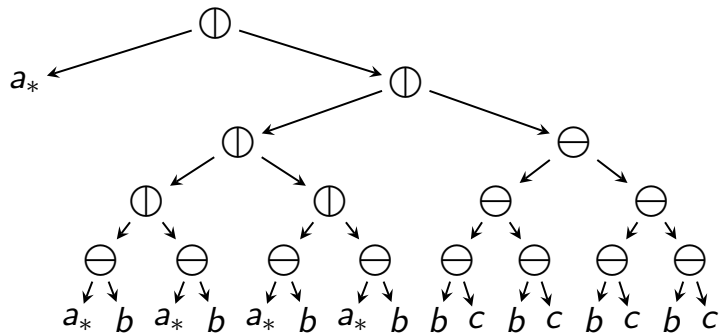

A forest algebra formula for $F$.

Gascón, Lohrey, Maneth, Reh, Sieber, 2020

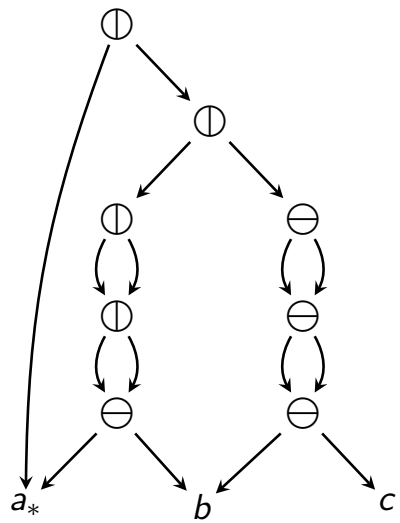
# Forest Straight-Line Programs (FSLPs)




An unranked forest $F$.




A forest algebra formula for $F$.




A forest-SLP for $F$.

Gascón, Lohrey, Maneth, Reh, Sieber, 2020

# Enumeration of MSO-Queries over SLP-Compressed Forests

**Input**:

MSO-query $q(X)$.

An FSLP $D$ that represents a node-labelled unranked forest $F$.

**Task**:

After preprocessing $O(|D|)$, enumerate $q(F)$ with output linear delay.

# Enumeration of MSO-Queries over SLP-Compressed Forests

**Input**:

MSO-query $q(X)$.

An FSLP $D$ that represents a node-labelled unranked forest $F$.

**Task**:

After preprocessing $O(|D|)$, enumerate $q(F)$ with output linear delay.

Practical relevance:

An experimental study shows that in linear time, we can compress a corpus of typical XML trees by FSLPs to approximately 3% of its actual size.

[Lohrey, Maneth, Mennicke, 2013]

# Reduction to dBUTA Enumeration on DAG-Foldings

By known techniques, we reduce the problem to:

**Input**:

Deterministic bottom-up tree automaton $\mathcal{A}$.

The DAG-folding $D$ of a node-labelled binary tree $T$

# Reduction to dBUTA Enumeration on DAG-Foldings

By known techniques, we reduce the problem to:

**Input**:

Deterministic bottom-up tree automaton $\mathcal{A}$.

The DAG-folding $D$ of a node-labelled binary tree $T$

**Task**:

After preprocessing $O(|D|)$, enumerate

$$\text{select}(\mathcal{A}, T) := \{S \subseteq V_{\text{leafs}} \mid \text{mark}(T, S) \in L(\mathcal{A})\}$$

with output linear delay.

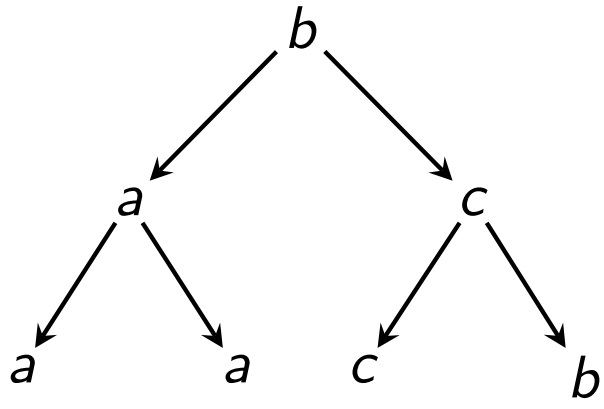
$\text{mark}(T, S)$: Tree $T$ with leaves from $S$ being marked.

# Bagan's Algorithm

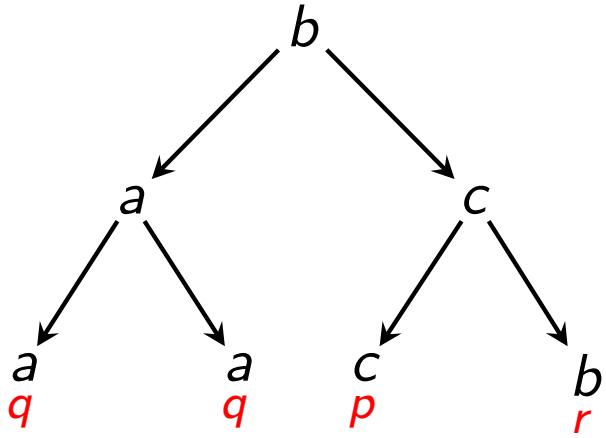
**Theorem** — Bagan 2006

Given leaf-selecting dBUTA $\mathcal{B}$ and a binary node-labelled tree $T$, after a preprocessing in time $O(|T|)$, we can enumerate select$(\mathcal{A}, T)$ with output linear delay.

# Deterministic Bottom-Up Tree Automata

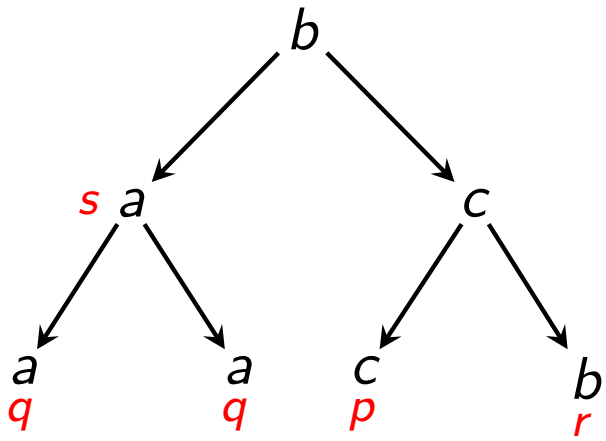


# Deterministic Bottom-Up Tree Automata




$$a \to q$$
$$b \to r$$
$$c \to p$$

# Deterministic Bottom-Up Tree Automata



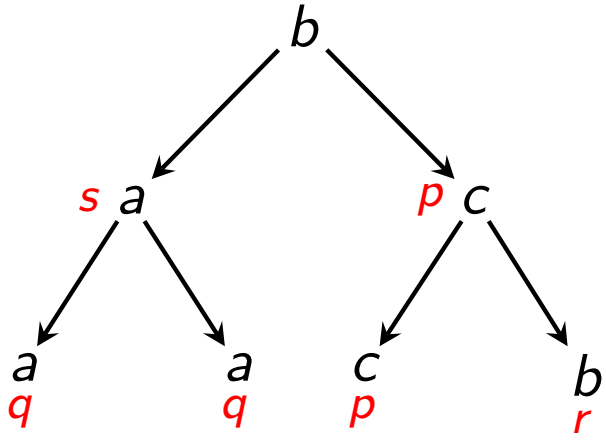

$a \to q$

$b \to r$

$c \to p$

$(q, q, a) \to s$

# Deterministic Bottom-Up Tree Automata




$a \to q$

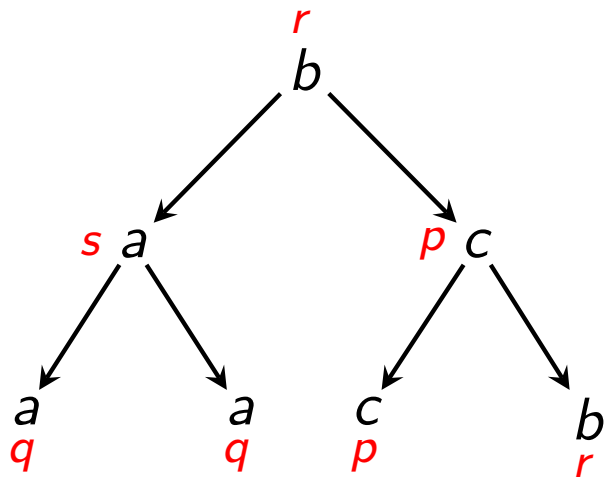
$b \to r$

$c \to p$

$(q, q, a) \to s$

$(p, r, c) \to p$

# Deterministic Bottom-Up Tree Automata




$a \to q$

$b \to r$

$c \to p$

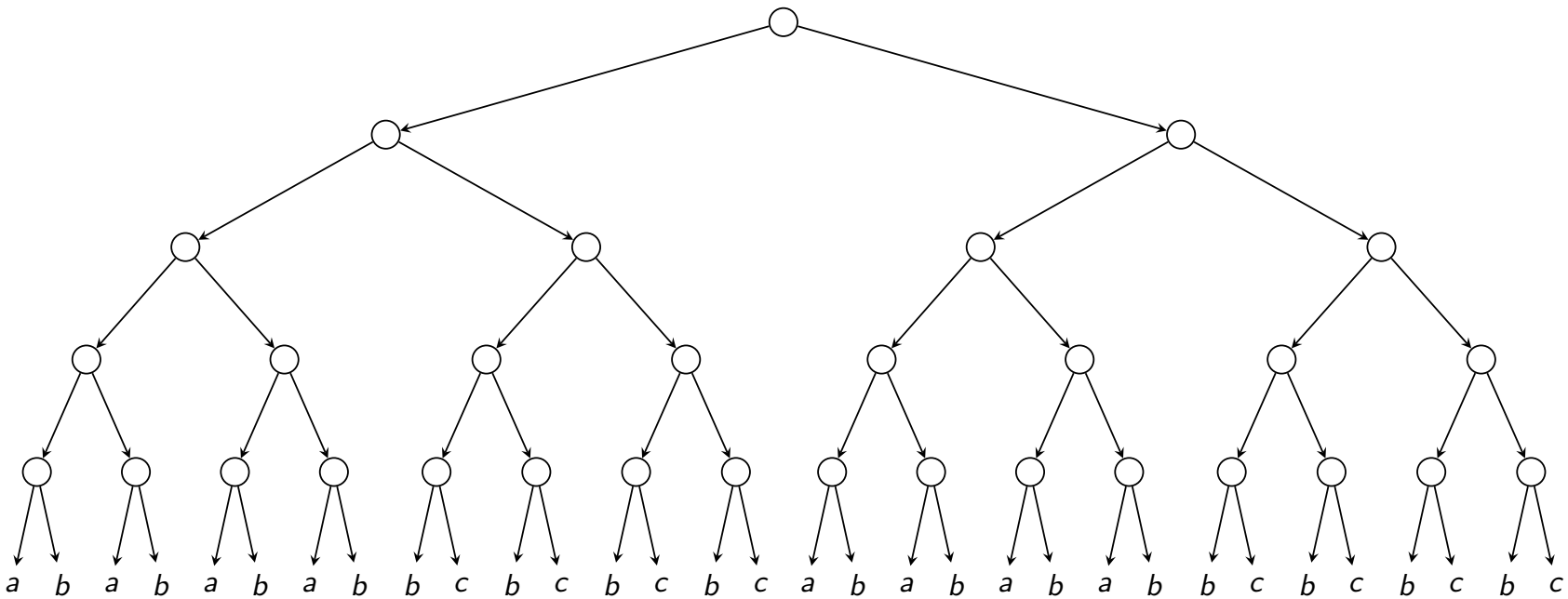
$(q, q, a) \to s$

$(p, r, c) \to p$

$(s, p, b) \to r$

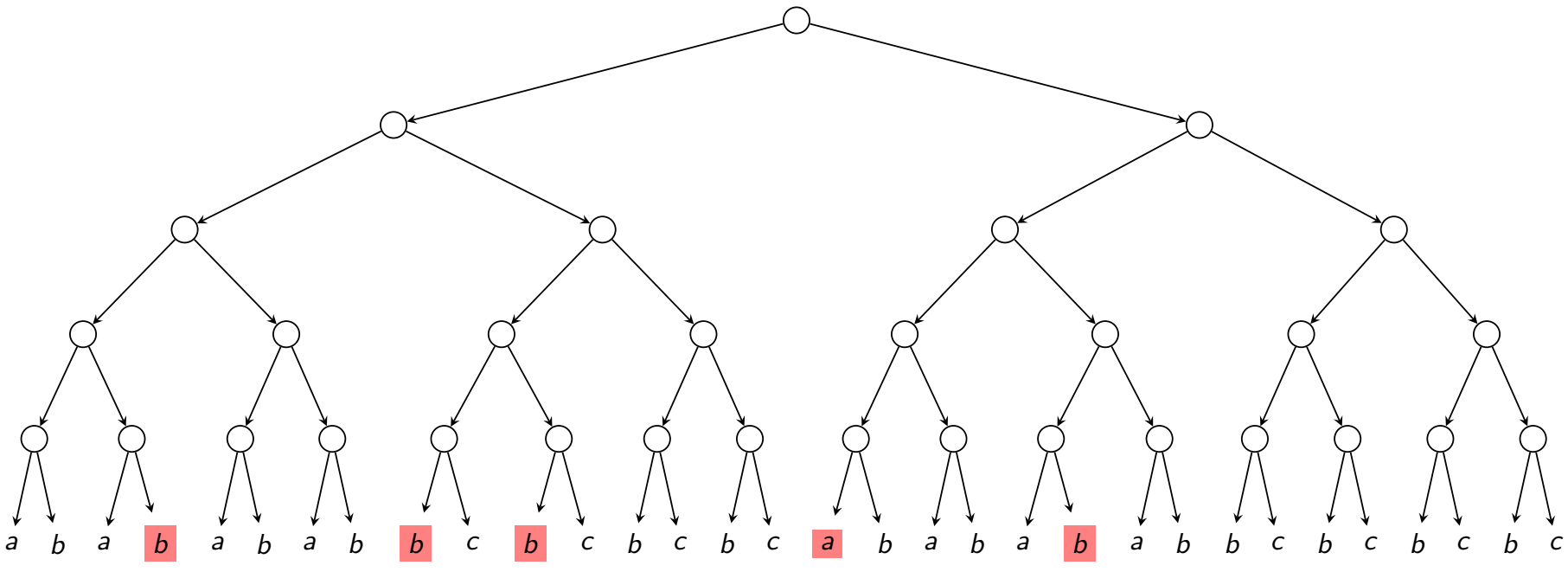
# Bagan's Algorithm

Leaf-labelled tree $T$:



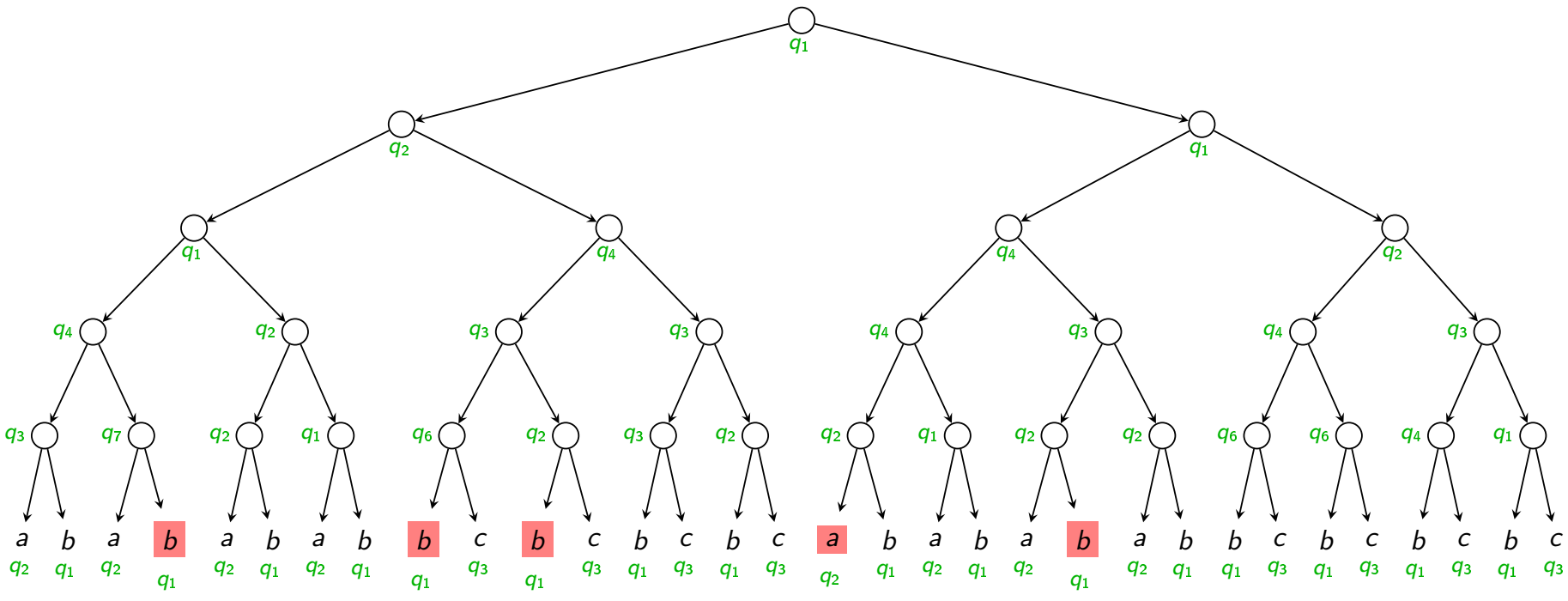
# Bagan's Algorithm

Marked tree mark($T, S$) for leaf-set $S$:



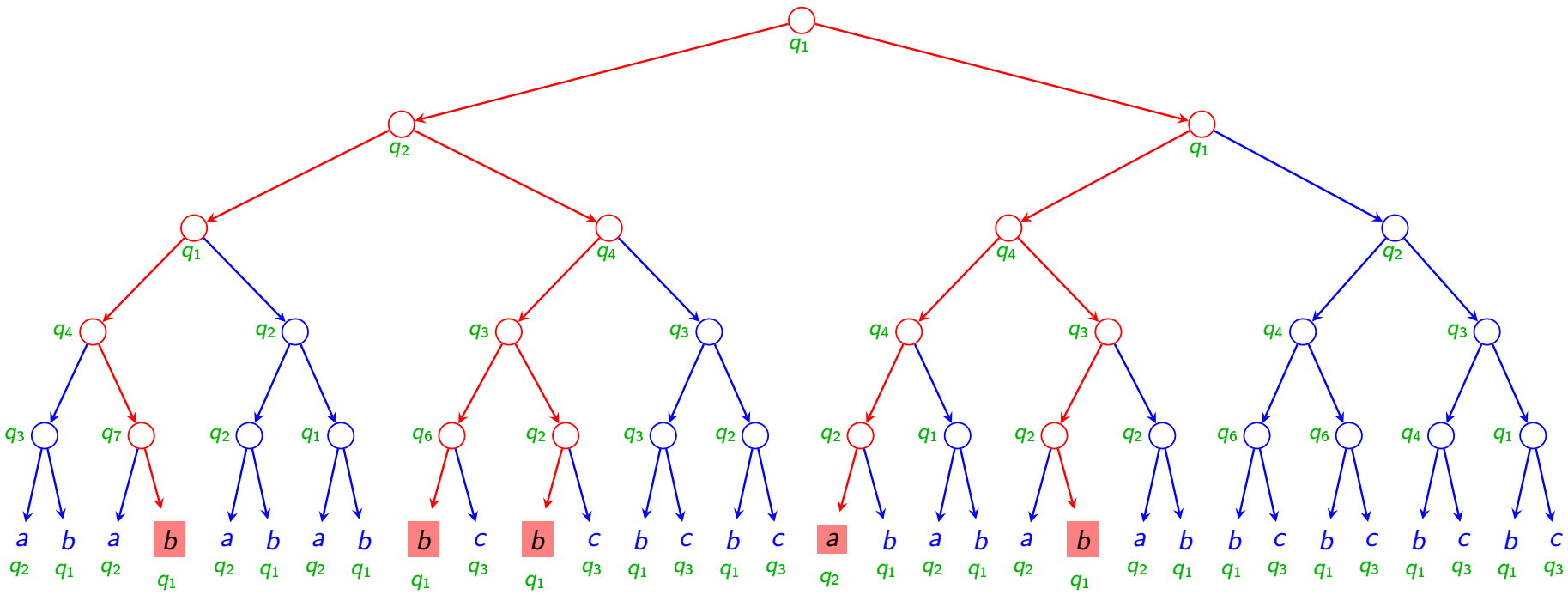
# Bagan's Algorithm

Run on mark($T, S$):

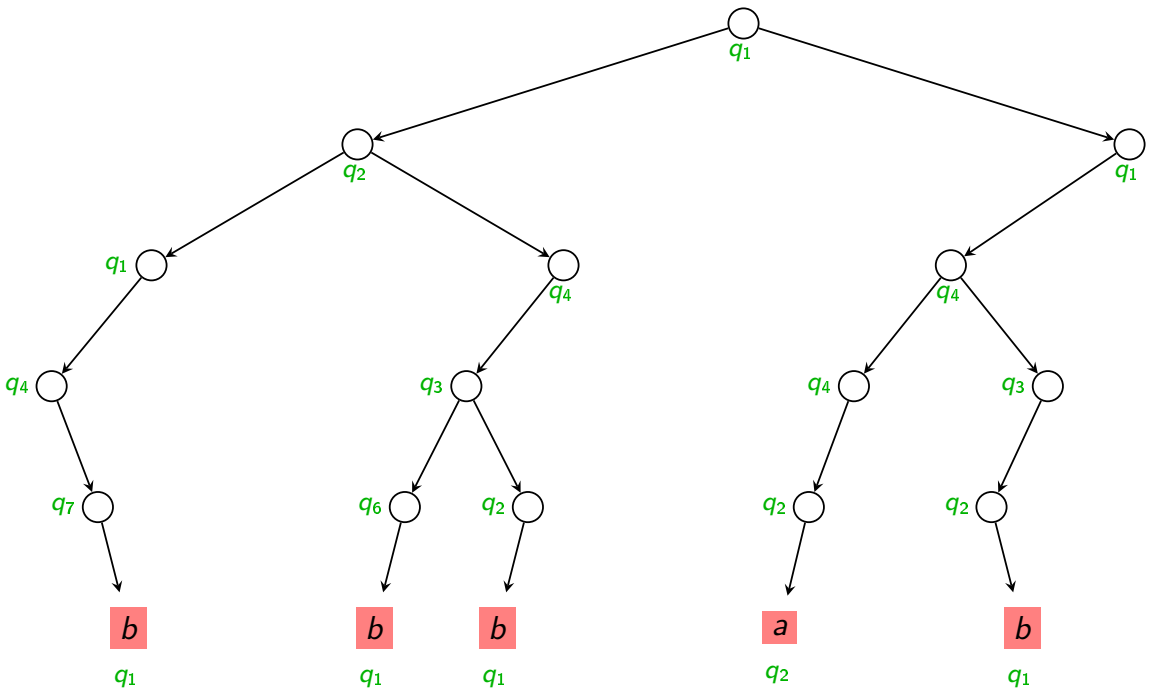


# Bagan's Algorithm

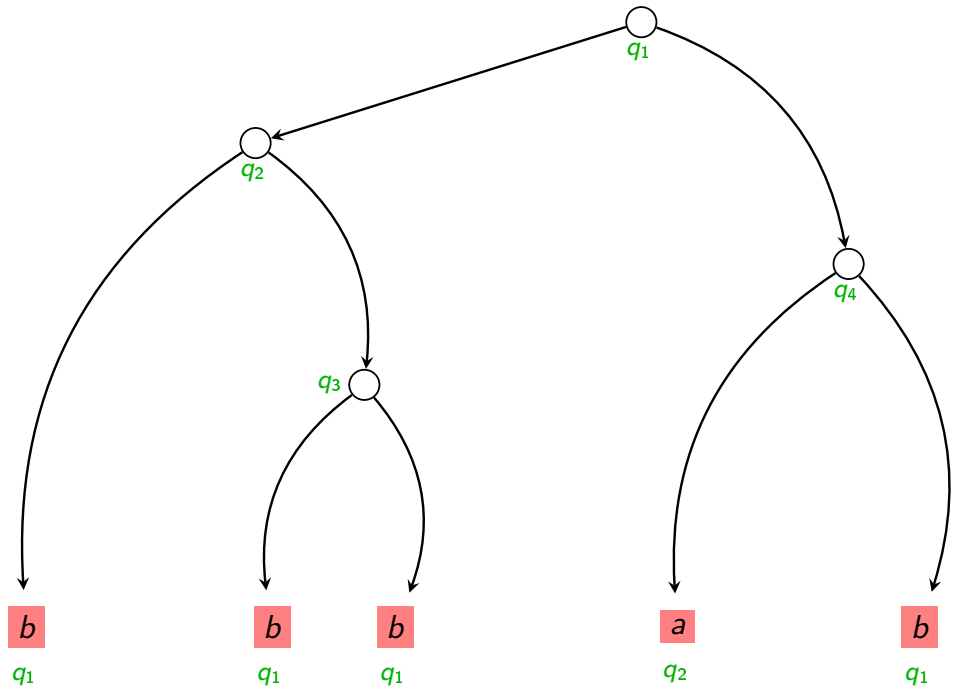
Run on mark($T, S$):



# Bagan's Algorithm

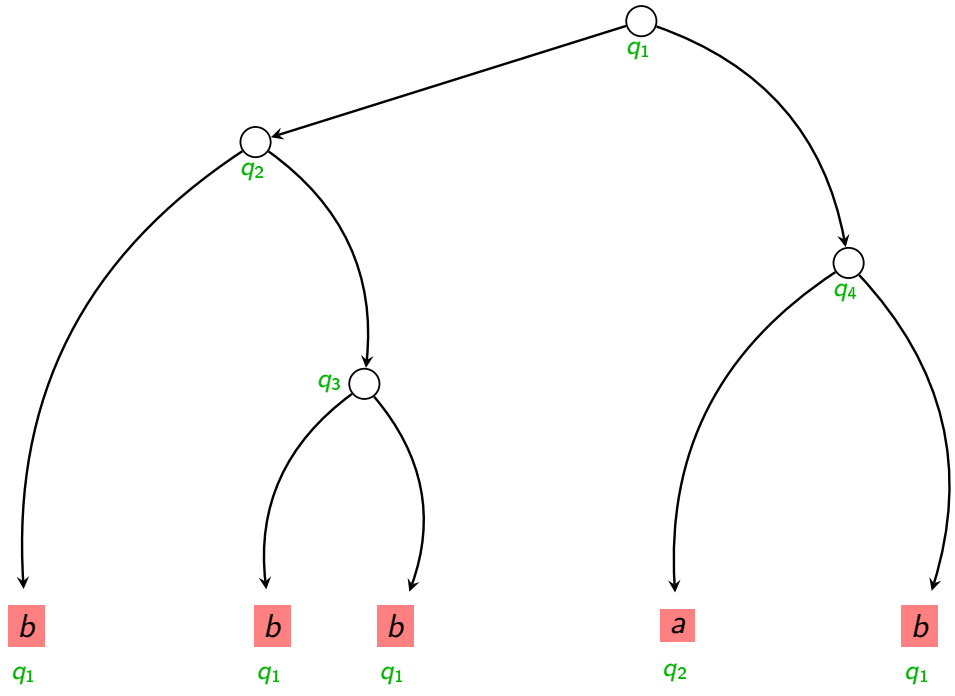


# Bagan's Algorithm



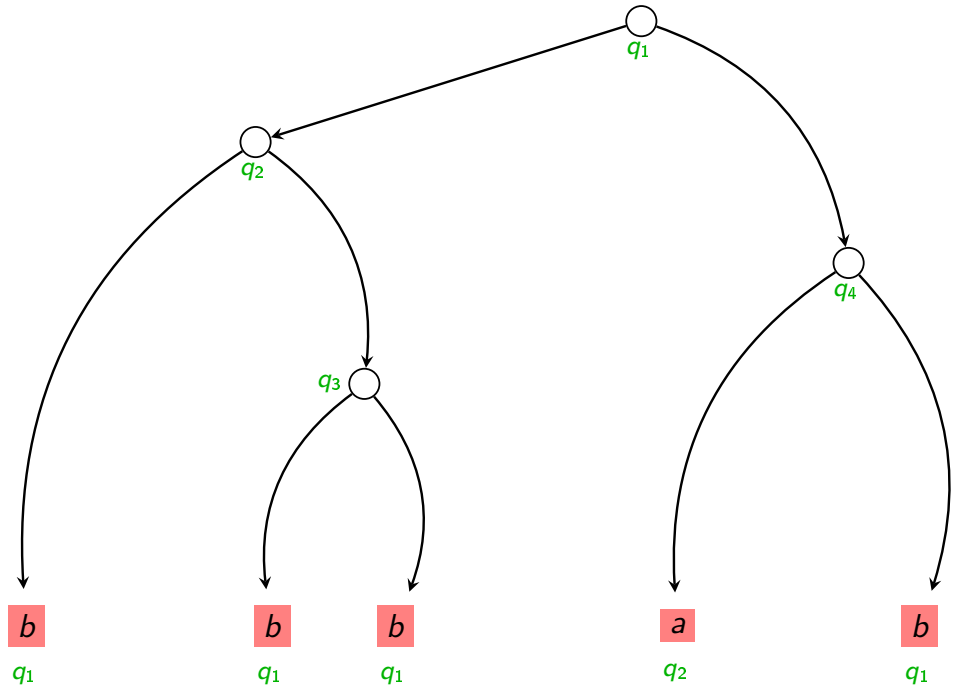
# Bagan's Algorithm

**Witness tree** for leaf-set $S$:



# Bagan's Algorithm

**Witness tree** for leaf-set $S$:




Every $(v, q) \in V \times Q$ is a **configuration**.

A configuration if **useful** if it is the node of some witness tree.

Arcs $(v, q) \to (u, p)$ of witness trees are called **shortcuts**, and $(u, p)$ is then a **shortcut successor** of $(v, q)$.

# Bagan's Algorithm

Compute the **shortcut forest**:

**Nodes**:

$(v, q)$ with $v \in V$ and $q \in Q$ is a node.

# Bagan's Algorithm

Compute the **shortcut forest**:

**Nodes**:

$(v, q)$ with $v \in V$ and $q \in Q$ is a node.

**Edges**:

Let $(v, q)$ and $(u, p)$ be useful.

$$\text{There is a path from } (v, q) \text{ to } (u, p)$$
$$\Longleftrightarrow$$
$$(u, p) \text{ is a possible shortcut successor of } (v, q).$$

# Bagan's Algorithm

Compute the **shortcut forest**:

**Nodes**:

$(v, q)$ with $v \in V$ and $q \in Q$ is a node.

**Edges**:

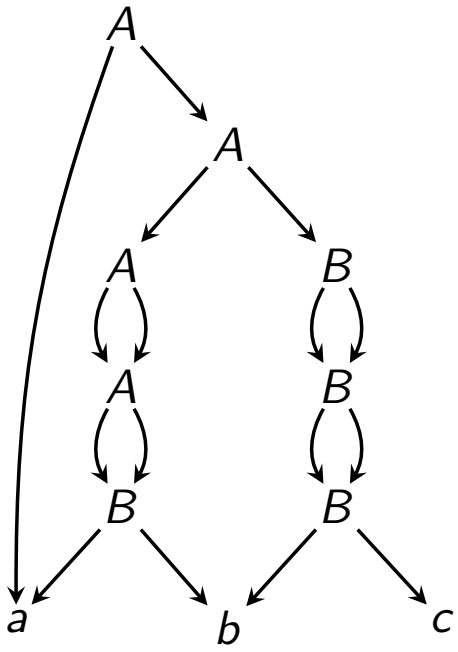
Let $(v, q)$ and $(u, p)$ be useful.

$$\text{There is a path from } (v, q) \text{ to } (u, p)$$
$$\Longleftrightarrow$$
$$(u, p) \text{ is a possible shortcut successor of } (v, q).$$

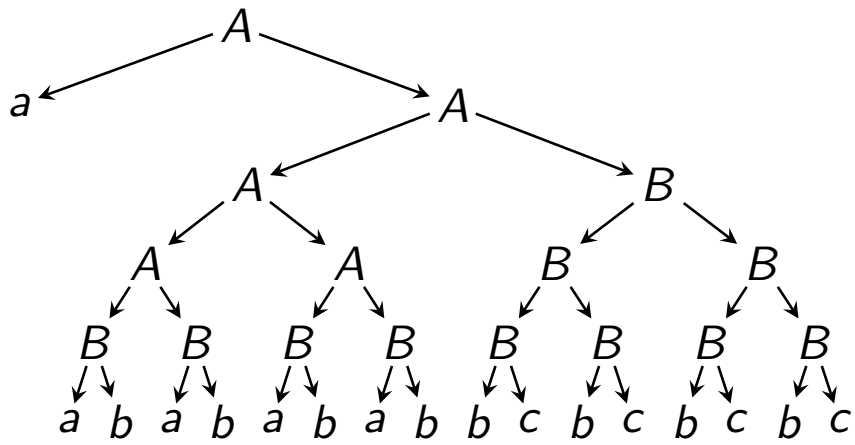
The Shortcut forest allows us to compute all possible shortcut successors of a given $(v, q)$, which is enough to efficiently enumerate the witness trees.

# Bagan's Algorithm for DAG-Folded Binary Trees

We want to carry out Bagan's algorithm, but the binary tree is given as its DAG-folding.



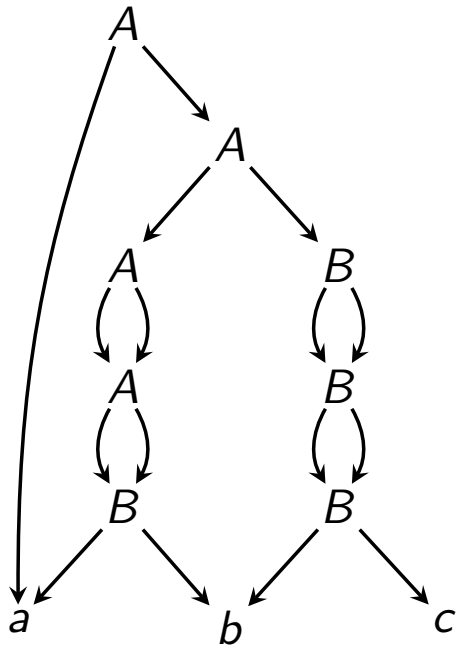

DAG-folding of the tree.



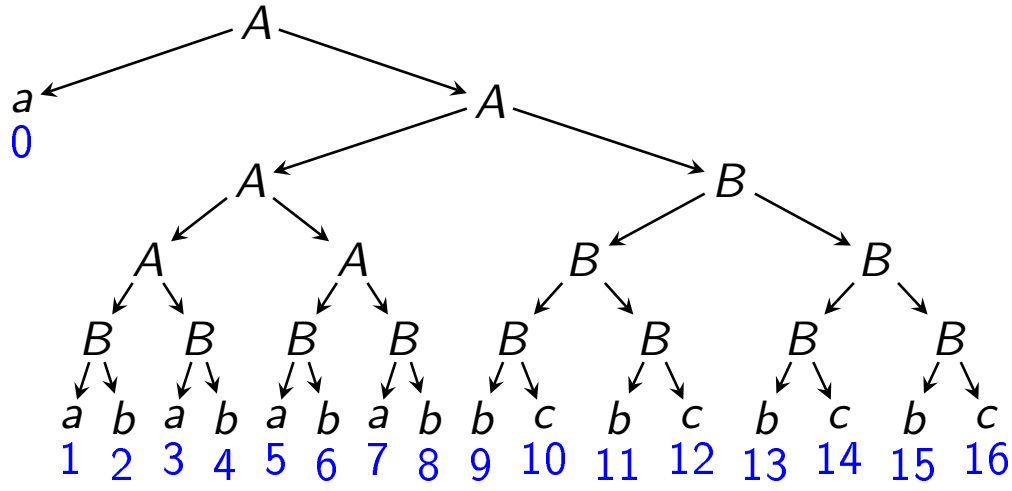

Binary tree.

# Bagan's Algorithm for DAG-Folded Binary Trees

We want to carry out Bagan's algorithm, but the binary tree is given as its DAG-folding.



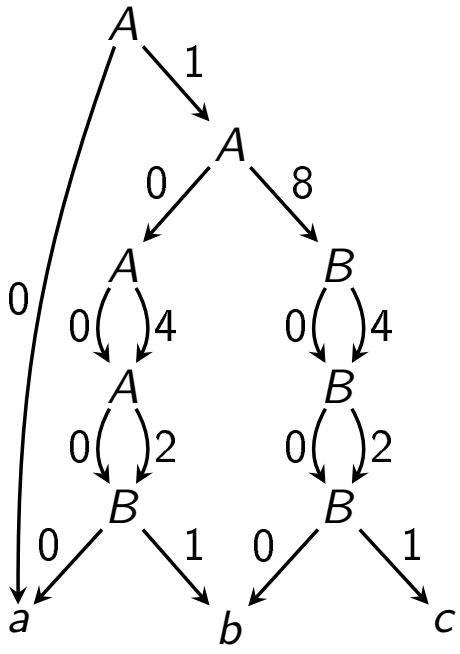

DAG-folding of the tree.



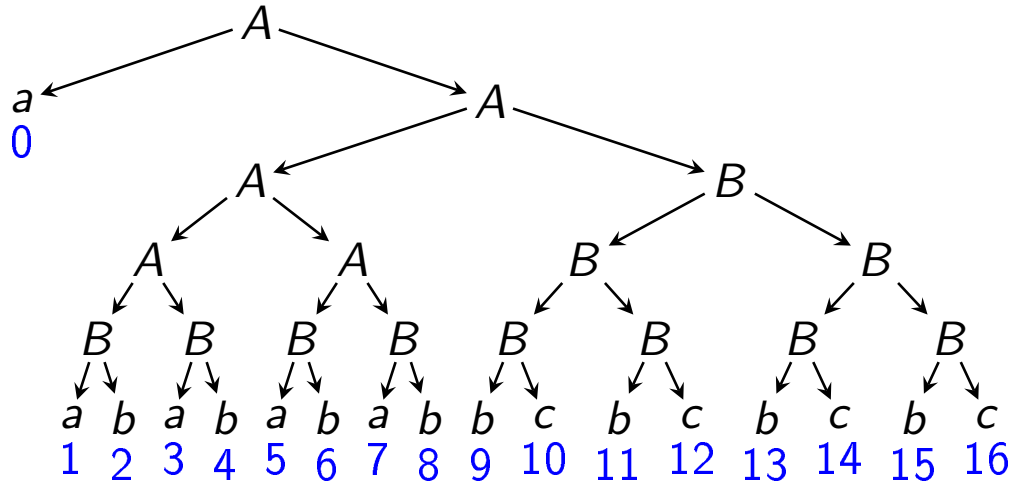

Binary tree.

# Bagan's Algorithm for DAG-Folded Binary Trees

We want to carry out Bagan's algorithm, but the binary tree is given as its DAG-folding.



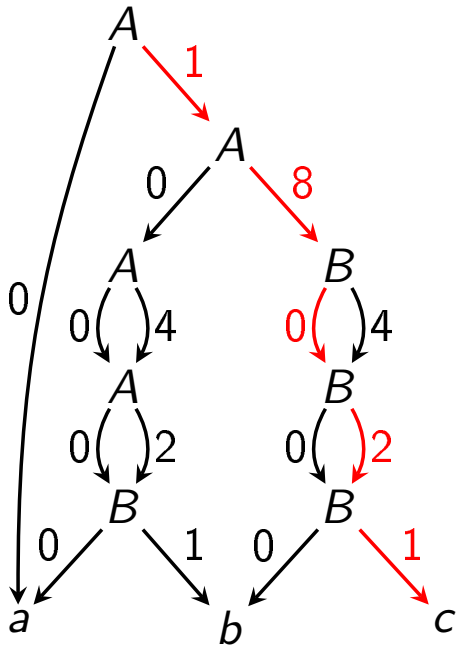

DAG-folding of the tree.



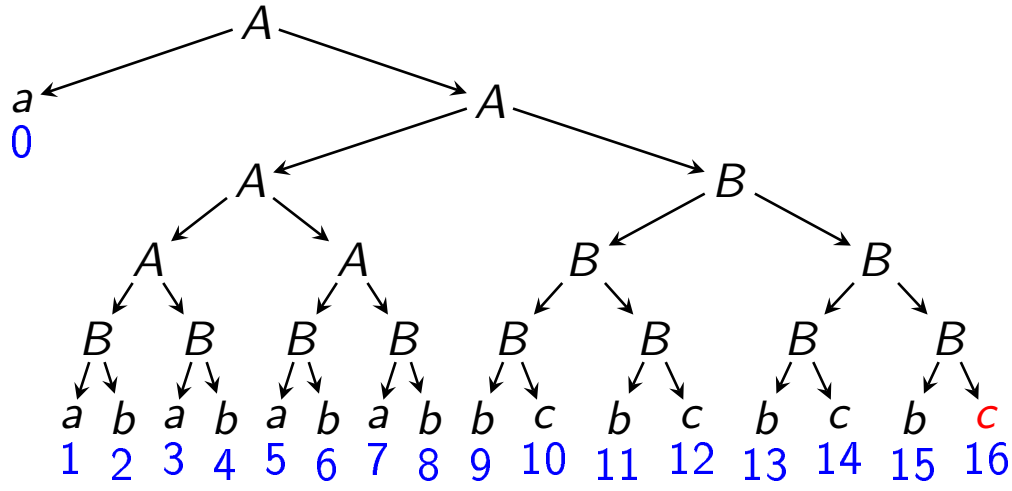

Binary tree.

# Bagan's Algorithm for DAG-Folded Binary Trees

We want to carry out Bagan's algorithm, but the binary tree is given as its DAG-folding.



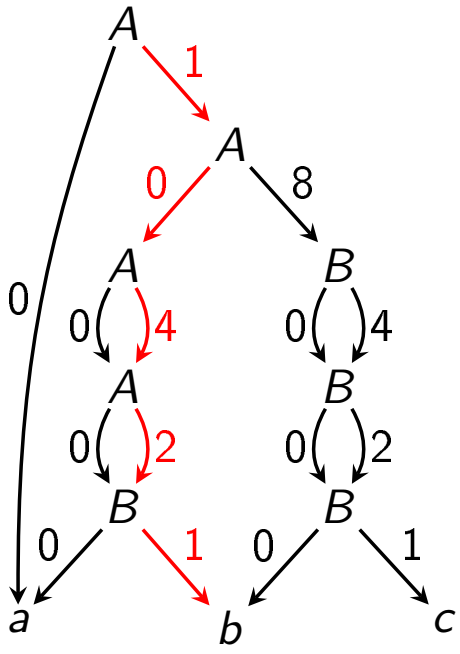

DAG-folding of the tree.



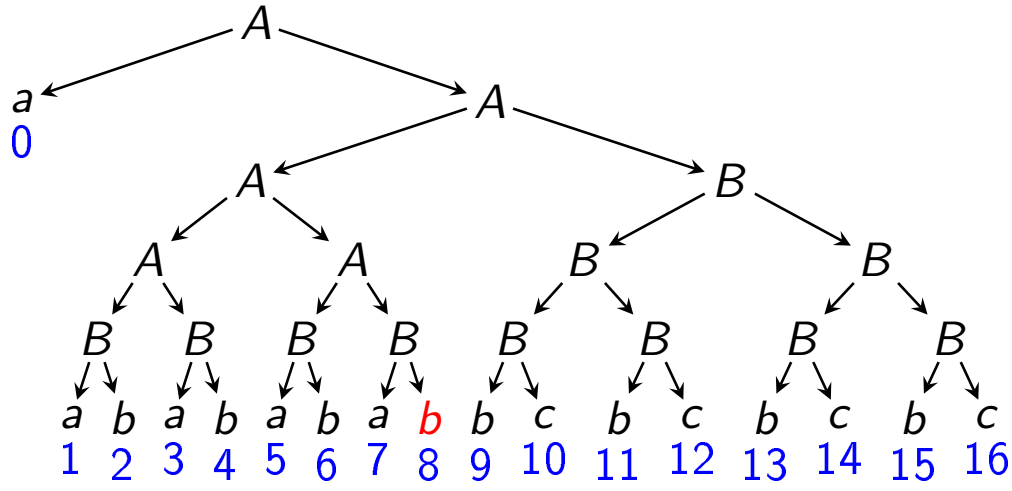

Binary tree.

# Bagan's Algorithm for DAG-Folded Binary Trees

We want to carry out Bagan's algorithm, but the binary tree is given as its DAG-folding.




DAG-folding of the tree.




Binary tree.

# Bagan's Algorithm for DAG-Foldings

Problem: We cannot afford to compute the full shortcut forest

# Bagan's Algorithm for DAG-Foldings

Problem: We cannot afford to compute the full shortcut forest

Solution: We can compute the DAG-folding of the shortcut forest.

# Bagan's Algorithm for DAG-Foldings

Problem: We cannot afford to compute the full shortcut forest

Solution: We can compute the DAG-folding of the shortcut forest.

Problem: We cannot afford to explicitly compute all shortcut successors for given $(v, q)$.

# Bagan's Algorithm for DAG-Foldings

Problem: We cannot afford to compute the full shortcut forest

Solution: We can compute the DAG-folding of the shortcut forest.

Problem: We cannot afford to explicitly compute all shortcut successors for given $(v, q)$.

Solution: For given $(v, q)$, we can **enumerate** all shortcut successors with linear preprocessing and constant delay.

# Bagan's Algorithm for DAG-Foldings

Problem: We cannot afford to compute the full shortcut forest

Solution: We can compute the DAG-folding of the shortcut forest.

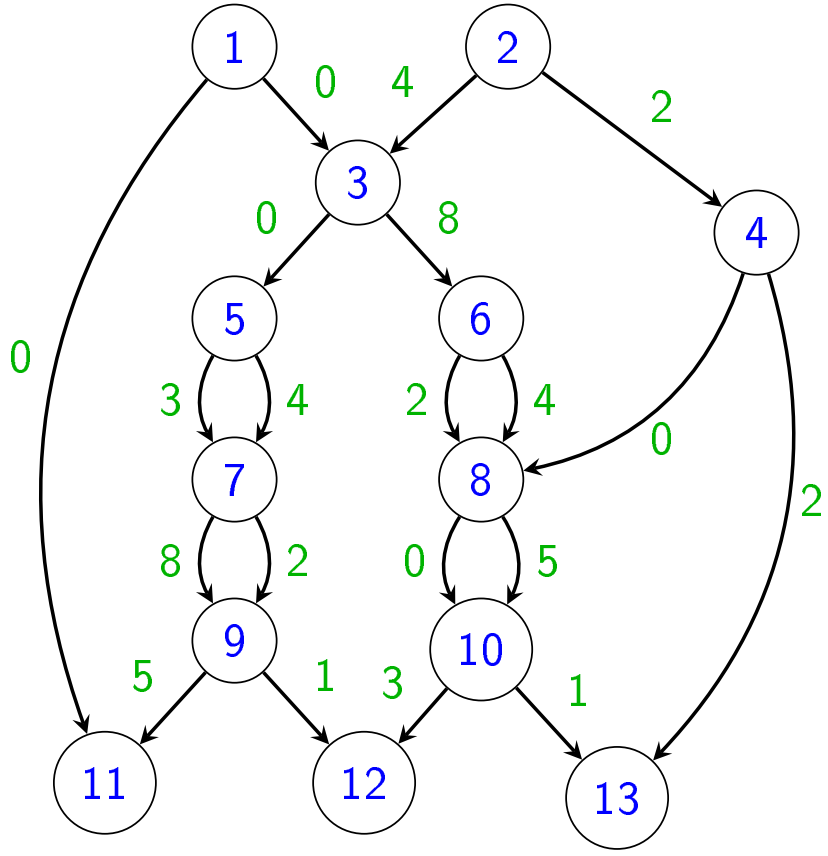
Problem: We cannot afford to explicitly compute all shortcut successors for given $(v, q)$.

Solution: For given $(v, q)$, we can **enumerate** all shortcut successors with linear preprocessing and constant delay.

This boils down to the following path enumeration problem in DAGs.

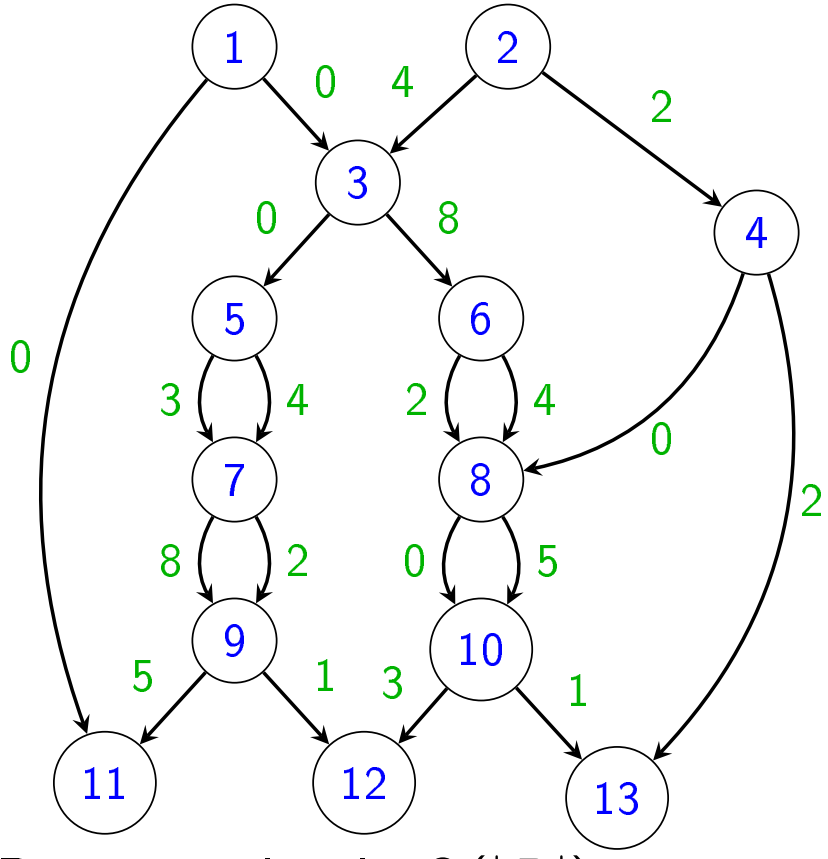
# Bagan's Algorithm for DAG-Foldings

Let $D = (V, E)$ be a binary DAG with weight function $\gamma : E \to M$.



# Bagan's Algorithm for DAG-Foldings

Let $D = (V, E)$ be a binary DAG with weight function $\gamma : E \to M$.

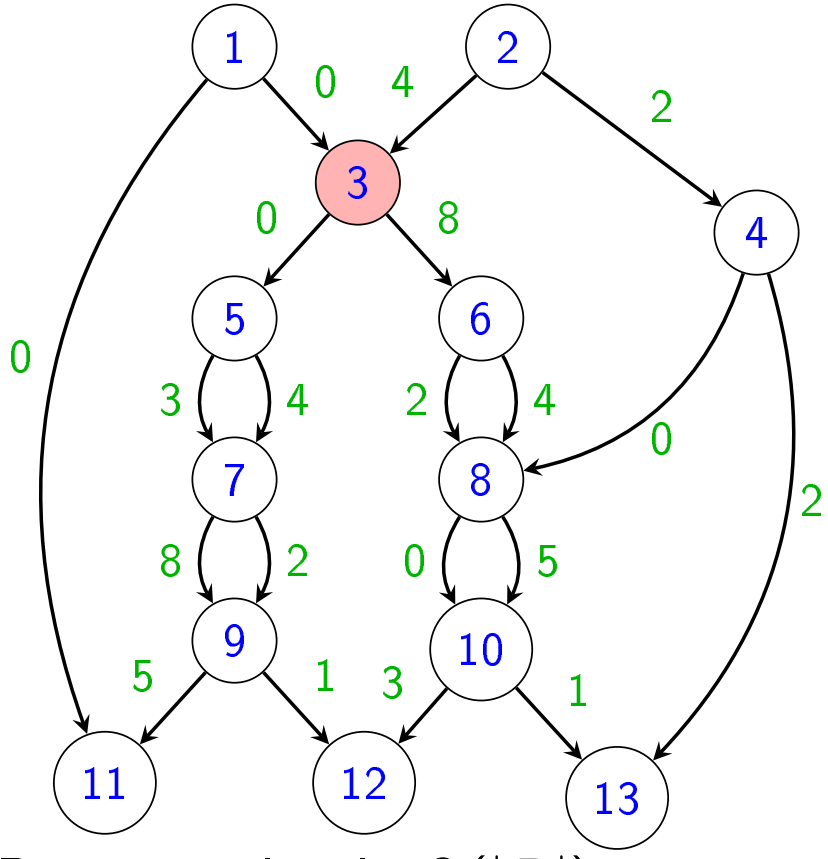



Preprocessing in $O(|D|)$.

For any $s \in V$, enumerate with constant delay all paths $\pi$ from $s$ to some leaf $u$ represented by $(u, \gamma(\pi))$.

# Bagan's Algorithm for DAG-Foldings

Let $D = (V, E)$ be a binary DAG with weight function $\gamma : E \to M$.



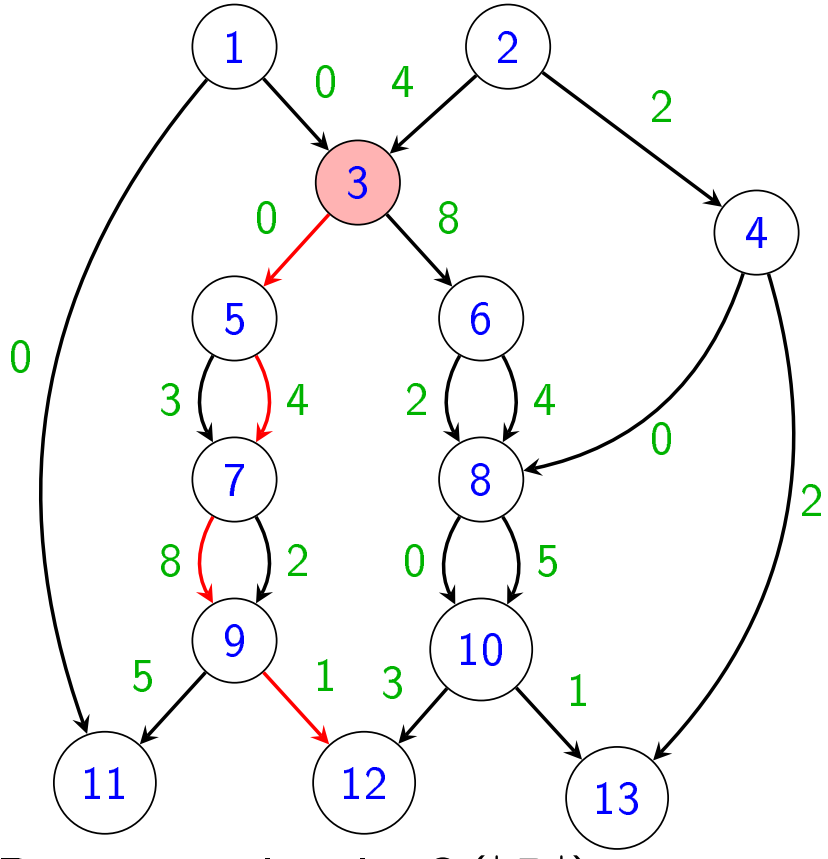

Enumeration for start node 3:

Preprocessing in $O(|D|)$.

For any $s \in V$, enumerate with constant delay all paths $\pi$ from $s$ to some leaf $u$ represented by $(u, \gamma(\pi))$.

# Bagan's Algorithm for DAG-Foldings

Let $D = (V, E)$ be a binary DAG with weight function $\gamma : E \to M$.

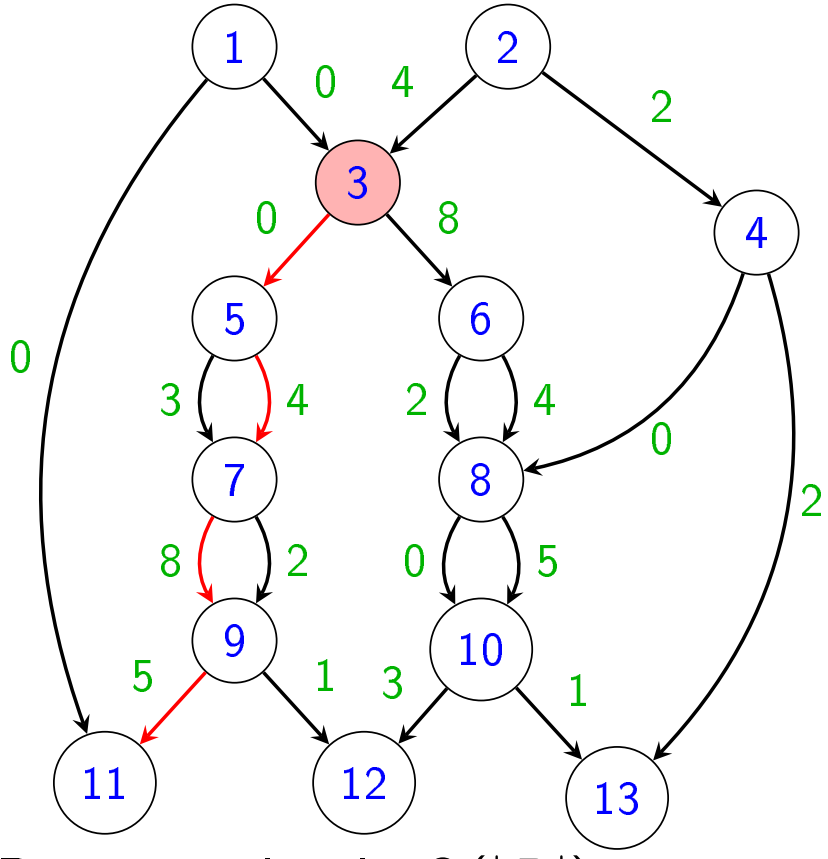



Enumeration for start node 3:
$(12, 13)$

Preprocessing in $O(|D|)$.
For any $s \in V$, enumerate with constant delay all paths $\pi$ from $s$ to some leaf $u$ represented by $(u, \gamma(\pi))$.

# Bagan's Algorithm for DAG-Foldings

Let $D = (V, E)$ be a binary DAG with weight function $\gamma : E \to M$.

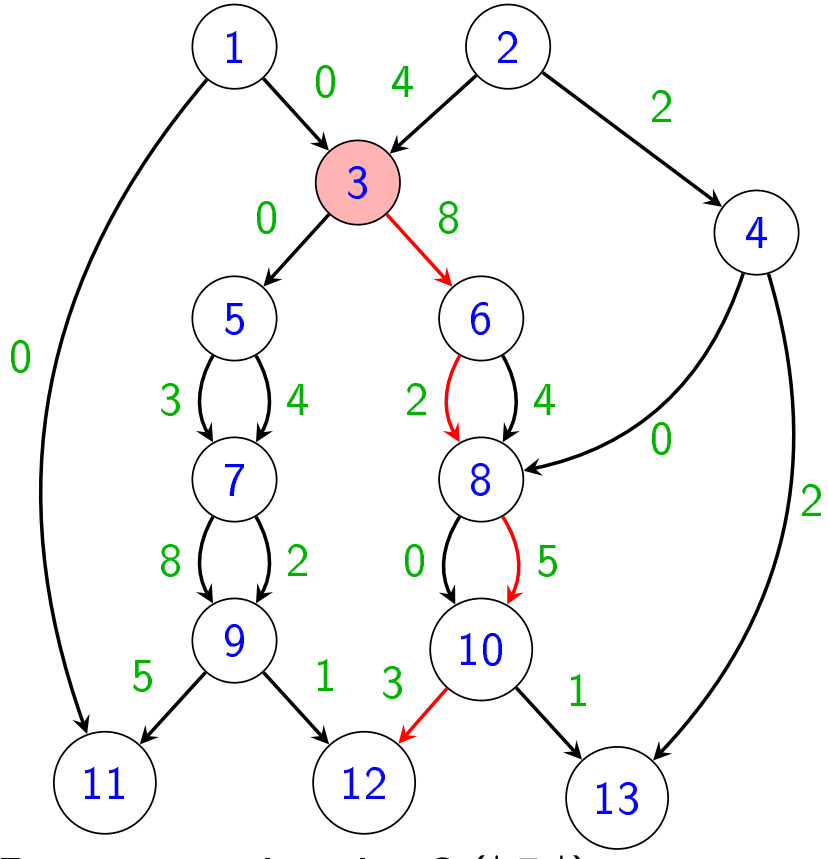



Enumeration for start node 3:
$(12, 13), (11, 17)$

Preprocessing in $O(|D|)$.
For any $s \in V$, enumerate with constant delay all paths $\pi$ from $s$ to some leaf $u$ represented by $(u, \gamma(\pi))$.

# Bagan's Algorithm for DAG-Foldings

Let $D = (V, E)$ be a binary DAG with weight function $\gamma : E \to M$.




Enumeration for start node 3:
$(12, 13), (11, 17), (12, 18), \ldots$

Preprocessing in $O(|D|)$.
For any $s \in V$, enumerate with constant delay all paths $\pi$ from $s$ to some leaf $u$ represented by $(u, \gamma(\pi))$.

Thank you very much for your attention.