Optimal Reachability and a Space-Time Tradeoff for Distance Queries in Constant-Treewidth Graphs

Krishnendu Chatterjee, Rasmus Ibsen-Jensen, **Andreas Pavlogiannis**

August 23, 2016

Reachability and distance oracles

Input: A (weighted) directed graph $G = (V, E, \mathrm{w} : E \to \mathbb{R})$

- Reachability
  - Pair query: Given $u, v \in V$, is there a path $u \rightsquigarrow v$?
  - Single source query: Given $u \in V$, determine those $v \in V$ for which $u \rightsquigarrow v$
- Distance
  - Pair Query: Given $u, v \in V$, determine the distance $d(u, v) = \inf_{P: u \rightsquigarrow v} \mathrm{w}(P)$
  - Single source query: Given $u \in V$, determine $d(u, v)$ for all $v \in V$

Reachability and distance oracles

Input: A (weighted) directed graph $G = (V, E, \mathrm{w} : E \to \mathbb{R})$

- Reachability
  - Pair query: Given $u, v \in V$, is there a path $u \rightsquigarrow v$?
  - Single source query: Given $u \in V$, determine those $v \in V$ for which $u \rightsquigarrow v$
- Distance
  - Pair Query: Given $u, v \in V$, determine the distance $d(u, v) = \inf_{P: u \rightsquigarrow v} \mathrm{w}(P)$
  - Single source query: Given $u \in V$, determine $d(u, v)$ for all $v \in V$

Reachability and distance oracles

Input: A (weighted) directed graph $G = (V, E, \mathrm{w} : E \to \mathbb{R})$

- Reachability
  - Pair query: Given $u, v \in V$, is there a path $u \rightsquigarrow v$?
  - Single source query: Given $u \in V$, determine those $v \in V$ for which $u \rightsquigarrow v$
- Distance
  - Pair Query: Given $u, v \in V$, determine the distance $d(u, v) = \inf_{P: u \rightsquigarrow v} \mathrm{w}(P)$
  - Single source query: Given $u \in V$, determine $d(u, v)$ for all $v \in V$

Treewidth (Informally)

The treewidth $\mathrm{tw}(G)$ of a graph G is a positive integer $\mathrm{tw}(G) \geq 1$

- Measure of how similar a graph is to a tree
  - $\mathrm{tw}(G) = 1$ precisely if G is a tree
- NP-complete to compute
- FPT: $O(n)$ for bounded treewidth graph families
- Many hard problems admit polynomial solutions in bounded treewidth families
  - Vertex cover, independent set, MSO ...

Families of bounded treewidth graphs

- Series-parallel graphs, outerplanar graphs
- Graphs arising in expert systems
- Natural language processing
- Control flow graphs of programs

This work

Reachability and distance oracles for constant treewidth graphs

- A preprocess and a query phase
- Phase 1: spend some resources (time, space) in building an oracle
- Phase 2: Query the oracle, expect fast answers

Results - Reachability queries

Given a graph $G = (V, E)$ of n nodes and bounded treewidth $\text{tw} = O(1)$

Preprocessing time	Pair query time	Single-source query time
–	$O(n)$	$O(n)$
$O(n \cdot \log n)$	$O(\log n)$	$O(n)$
$\mathbf{O(n)}$	$\mathbf{O(1)}$	$\mathbf{O\left(\frac{n}{\log n}\right)}$

- Optimal
- Faster than DFS/BFS after a constant number of queries

Results - Reachability queries

Given a graph $G = (V, E)$ of n nodes and bounded treewidth $\text{tw} = O(1)$

Preprocessing time	Pair query time	Single-source query time
–	$O(n)$	$O(n)$
$O(n \cdot \log n)$	$O(\log n)$	$O(n)$
$\mathbf{O(n)}$	$\mathbf{O(1)}$	$\mathbf{O\left(\frac{n}{\log n}\right)}$

- Optimal
- Faster than DFS/BFS after a constant number of queries

Results - Reachability queries

Given a graph $G = (V, E)$ of n nodes and bounded treewidth $\text{tw} = O(1)$

Preprocessing time	Pair query time	Single-source query time
–	$O(n)$	$O(n)$
$O(n \cdot \log n)$	$O(\log n)$	$O(n)$
$\mathbf{O(n)}$	$\mathbf{O(1)}$	$\mathbf{O\left(\frac{n}{\log n}\right)}$

- Optimal
- Faster than DFS/BFS after a constant number of queries

Results - Reachability queries

Given a graph $G = (V, E)$ of n nodes and bounded treewidth $\text{tw} = O(1)$

Preprocessing time	Pair query time	Single-source query time
–	$O(n)$	$O(n)$
$O(n \cdot \log n)$	$O(\log n)$	$O(n)$
$\mathbf{O(n)}$	$\mathbf{O(1)}$	$\mathbf{O\left(\frac{n}{\log n}\right)}$

- Optimal
- Faster than DFS/BFS after a constant number of queries

Results - Reachability queries

Given a graph $G = (V, E)$ of n nodes and bounded treewidth $\text{tw} = O(1)$

Preprocessing time	Pair query time	Single-source query time
–	$O(n)$	$O(n)$
$O(n \cdot \log n)$	$O(\log n)$	$O(n)$
$\mathbf{O(n)}$	$\mathbf{O(1)}$	$\mathbf{O\left(\frac{n}{\log n}\right)}$

- Optimal
- Faster than DFS/BFS after a constant number of queries

Results - Distance queries

Given a graph $G = (V, E, \mathrm{w})$ of n nodes and bounded treewidth $\mathrm{tw} = O(1)$, and fixed $\epsilon \in [1/2, 1]$

Preprocessing time	Space usage	Pair query time	Single-source query time
$O(n^2)$	$O(n^2)$	$O(1)$	$O(n)$
$O(n)$	$O(n)$	$O(\alpha(n))$	$O(n)$
Polynomial	$O(n^\epsilon \cdot \log^2 n)$	$O(n^{1-\epsilon} \cdot \log n)$	–
Polynomial	$\mathbf{O(n^\epsilon)}$	$\mathbf{O(n^{1-\epsilon} \cdot \alpha(n))}$	–

$\alpha(n)$ is the inverse Ackermann function

Results - Distance queries

Given a graph $G = (V, E, \mathrm{w})$ of n nodes and bounded treewidth $\mathrm{tw} = O(1)$, and fixed $\epsilon \in [1/2, 1]$

Preprocessing time	Space usage	Pair query time	Single-source query time
$O(n^2)$	$O(n^2)$	$O(1)$	$O(n)$
$O(n)$	$O(n)$	$O(\alpha(n))$	$O(n)$
Polynomial	$O(n^{\epsilon} \cdot \log^2 n)$	$O(n^{1-\epsilon} \cdot \log n)$	–
Polynomial	$\mathbf{O(n^{\epsilon})}$	$\mathbf{O(n^{1-\epsilon} \cdot \alpha(n))}$	–

$\alpha(n)$ is the inverse Ackermann function

Results - Distance queries

Given a graph $G = (V, E, \mathrm{w})$ of n nodes and bounded treewidth $\mathrm{tw} = O(1)$, and fixed $\epsilon \in [1/2, 1]$

Preprocessing time	Space usage	Pair query time	Single-source query time
$O(n^2)$	$O(n^2)$	$O(1)$	$O(n)$
$O(n)$	$O(n)$	$O(\alpha(n))$	$O(n)$
Polynomial	$O(n^{\epsilon} \cdot \log^2 n)$	$O(n^{1-\epsilon} \cdot \log n)$	–
Polynomial	$\mathbf{O(n^{\epsilon})}$	$\mathbf{O(n^{1-\epsilon} \cdot \alpha(n))}$	–

$\alpha(n)$ is the inverse Ackermann function

Results - Distance queries

Given a graph $G = (V, E, \mathrm{w})$ of n nodes and bounded treewidth $\mathrm{tw} = O(1)$, and fixed $\epsilon \in [1/2, 1]$

Preprocessing time	Space usage	Pair query time	Single-source query time
$O(n^2)$	$O(n^2)$	$O(1)$	$O(n)$
$O(n)$	$O(n)$	$O(\alpha(n))$	$O(n)$
Polynomial	$O(n^{\epsilon} \cdot \log^2 n)$	$O(n^{1-\epsilon} \cdot \log n)$	–
Polynomial	$\mathbf{O(n^{\epsilon})}$	$\mathbf{O(n^{1-\epsilon} \cdot \alpha(n))}$	–

$\alpha(n)$ is the inverse Ackermann function

Tree-Decompositions and Treewidth

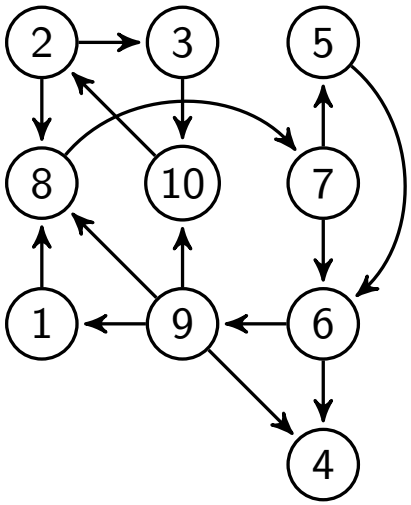
Tree Decomposition

Definition (Tree decomposition)

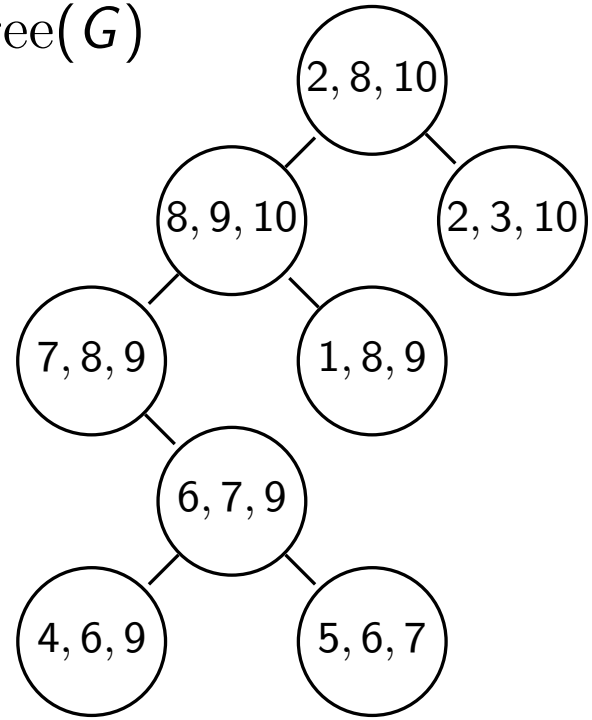
Given a graph $G = (V, E)$, a **tree-decomposition** $\mathrm{Tree}(G) = (V_T, E_T)$ is a tree of bags $B_i \subseteq V$ such that:

1. Every node of G is contained in a bag
2. Every edge of G is contained in a bag
3. Every node of G appears in a contiguous subtree of $\mathrm{Tree}(G)$.

G



$\mathrm{Tree}(G)$



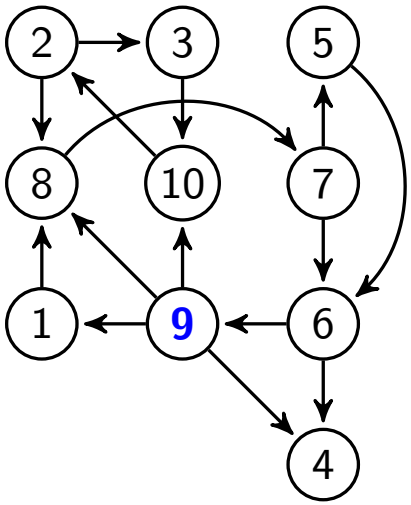
Tree Decomposition

Definition (Tree decomposition)

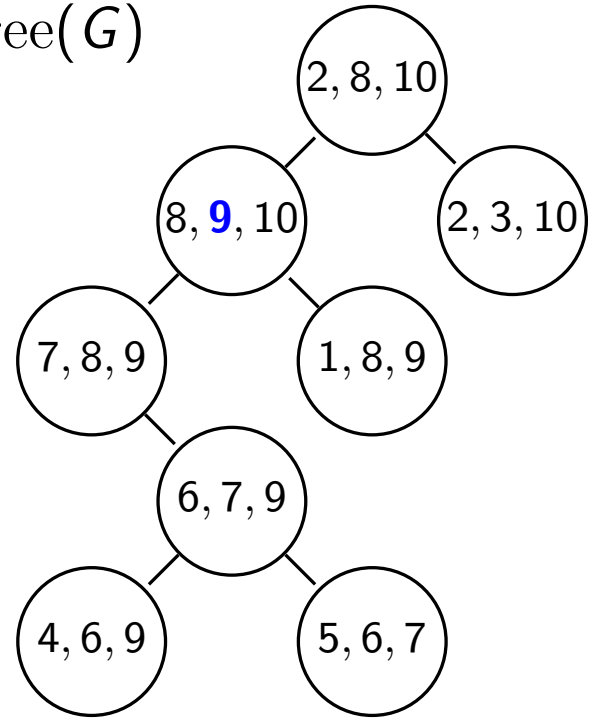
Given a graph $G = (V, E)$, a **tree-decomposition** $\mathrm{Tree}(G) = (V_T, E_T)$ is a tree of bags $B_i \subseteq V$ such that:

1. Every node of G is contained in a bag
2. Every edge of G is contained in a bag
3. Every node of G appears in a contiguous subtree of $\mathrm{Tree}(G)$.

G



$\mathrm{Tree}(G)$



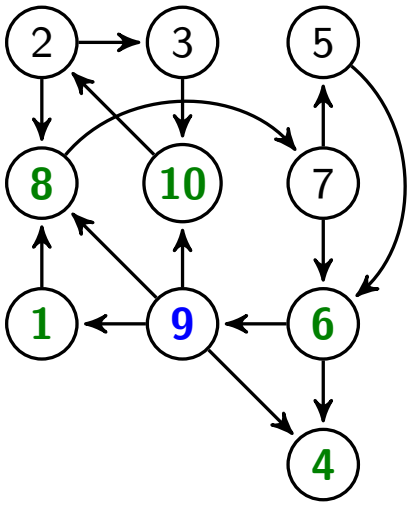
Tree Decomposition

Definition (Tree decomposition)

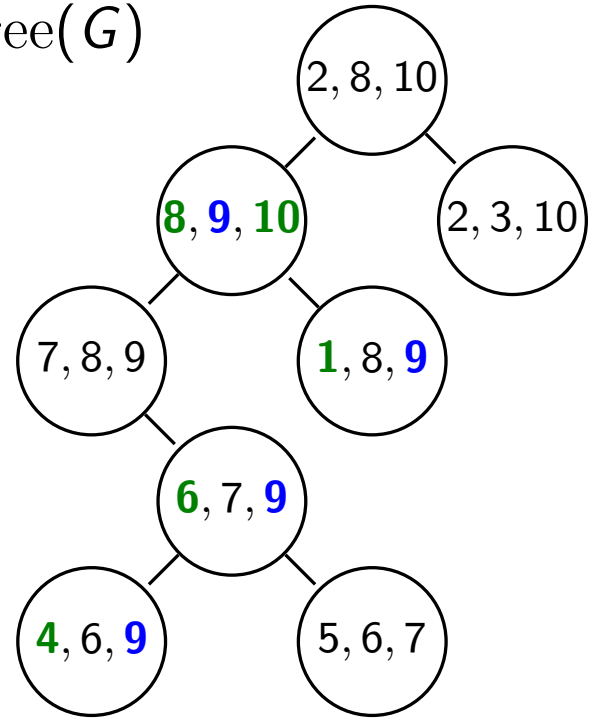
Given a graph $G = (V, E)$, a **tree-decomposition** $\mathrm{Tree}(G) = (V_T, E_T)$ is a tree of bags $B_i \subseteq V$ such that:

1. Every node of G is contained in a bag
2. Every edge of G is contained in a bag
3. Every node of G appears in a contiguous subtree of $\mathrm{Tree}(G)$.

G



$\mathrm{Tree}(G)$



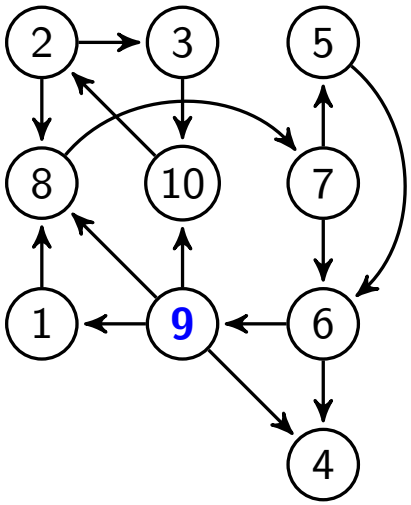
Tree Decomposition

Definition (Tree decomposition)

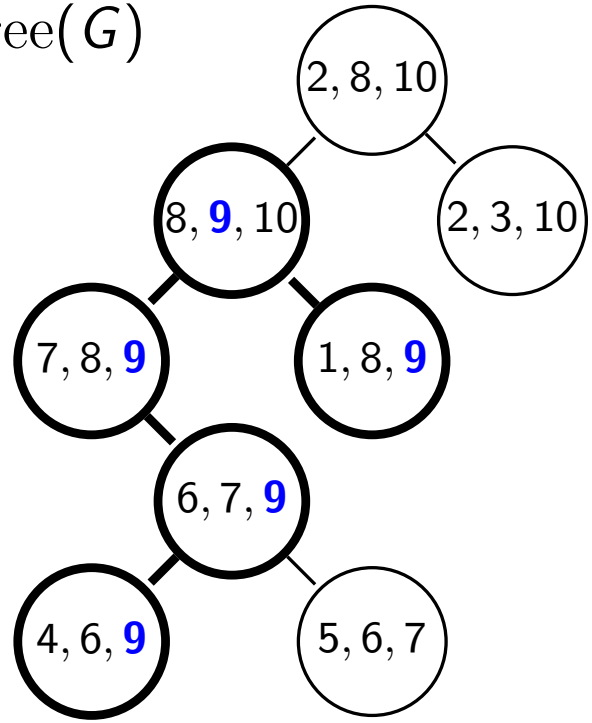
Given a graph $G = (V, E)$, a **tree-decomposition** $\mathrm{Tree}(G) = (V_T, E_T)$ is a tree of bags $B_i \subseteq V$ such that:

1. Every node of G is contained in a bag
2. Every edge of G is contained in a bag
3. Every node of G appears in a contiguous subtree of $\mathrm{Tree}(G)$.

G



$\mathrm{Tree}(G)$

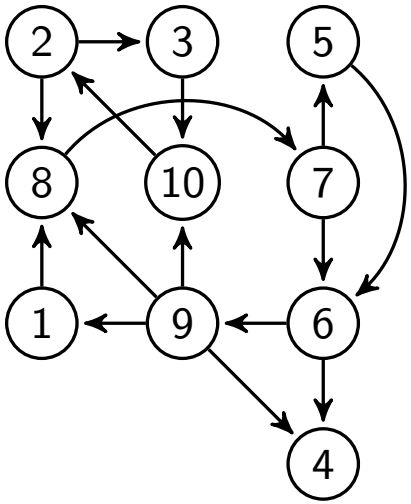


Treewidth

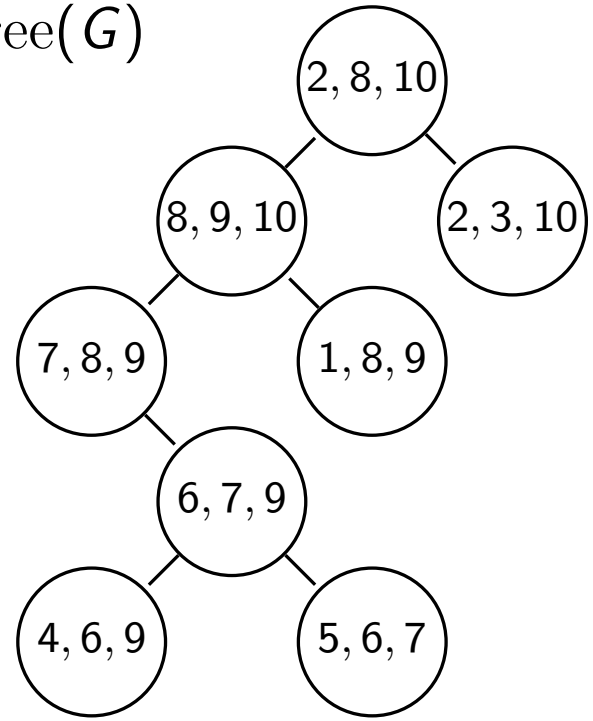
Definition

A family $\mathcal{G} = (G_i)_i$ has **constant treewidth** if we can always construct a tree decomposition $\mathrm{Tree}(G_i)$ in which every bag of has constant size.

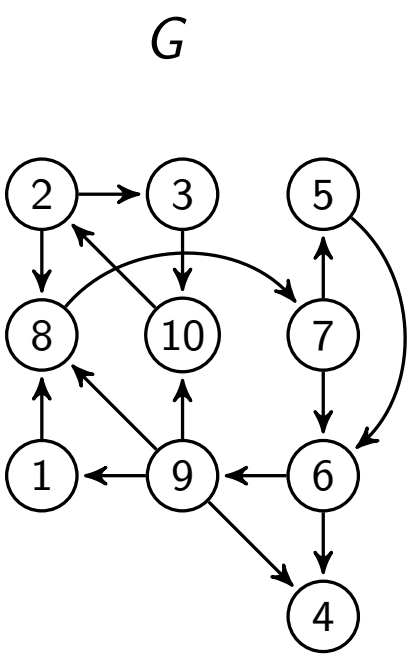
G

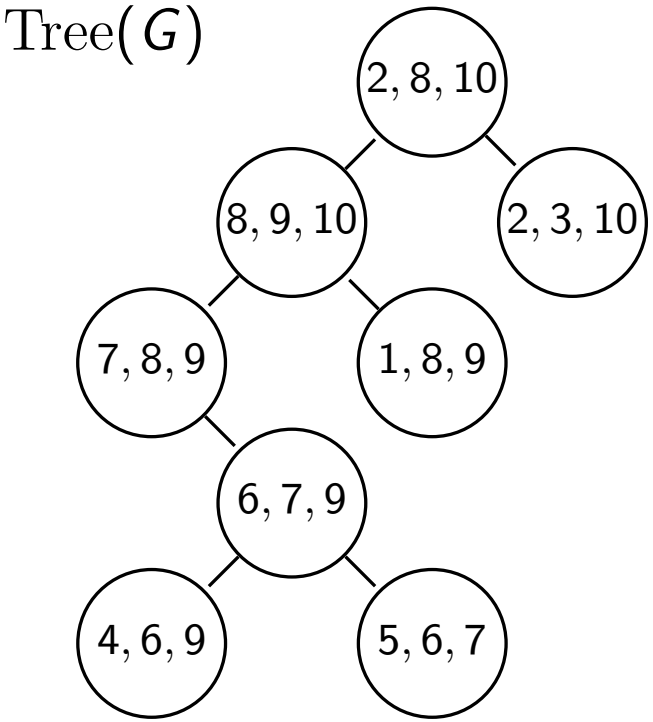


$\mathrm{Tree}(G)$



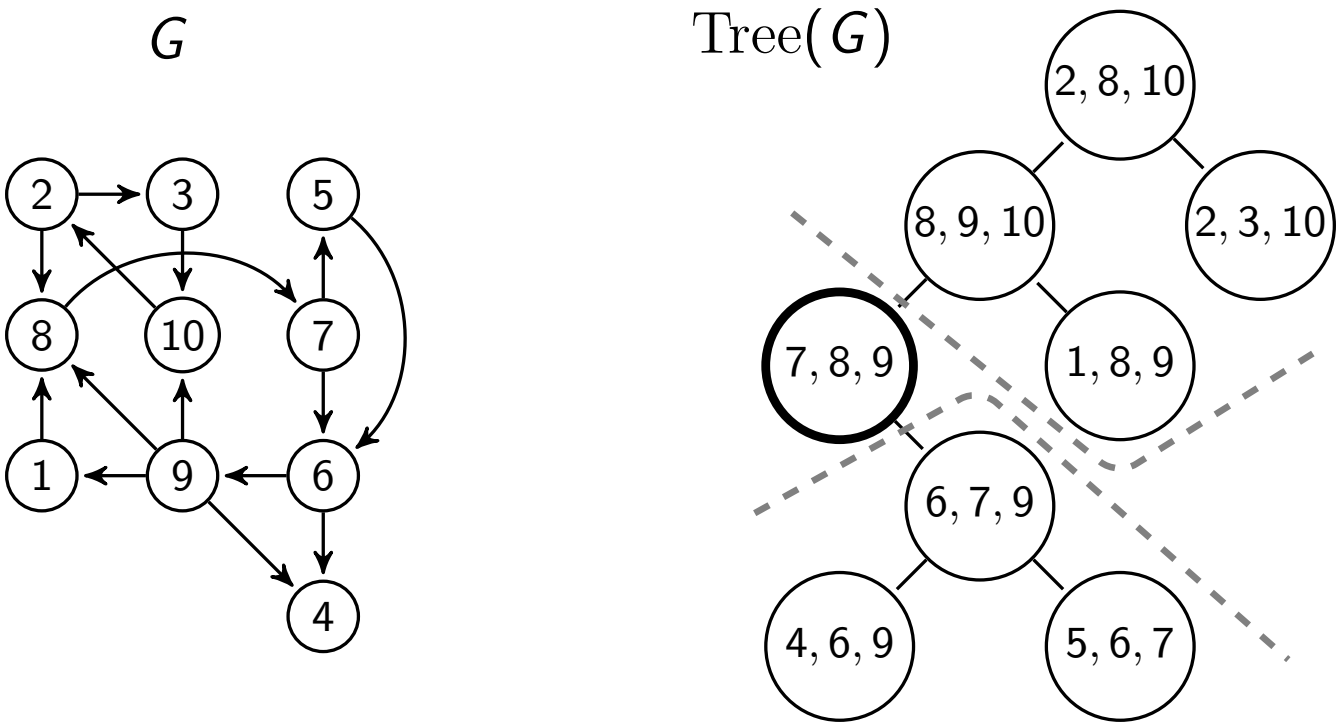
Tree Decompositions





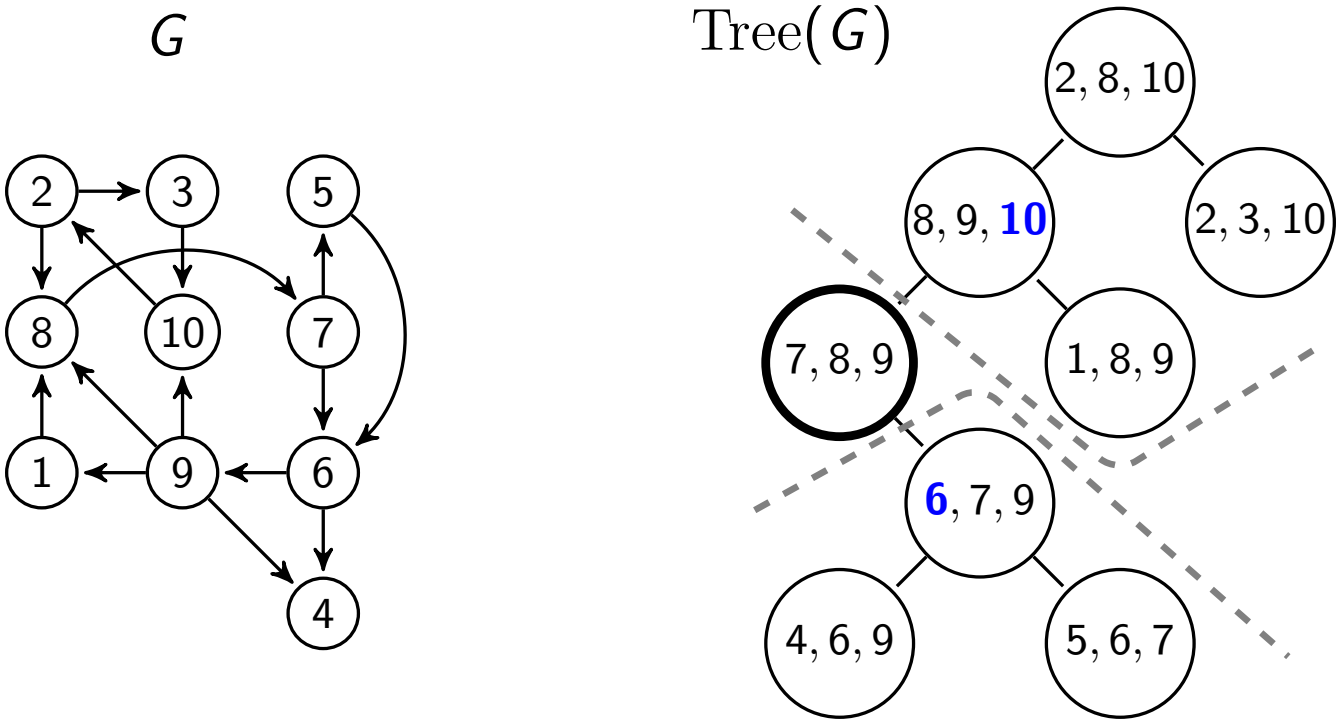
- There is a path $10 \rightsquigarrow 6$ iff there is a $u \in B$ and paths $10 \rightsquigarrow u$ and $u \rightsquigarrow 6$
- $$d(10,6) = \min_{u \in B}(d(10,u) + d(u,6))$$

Tree Decompositions



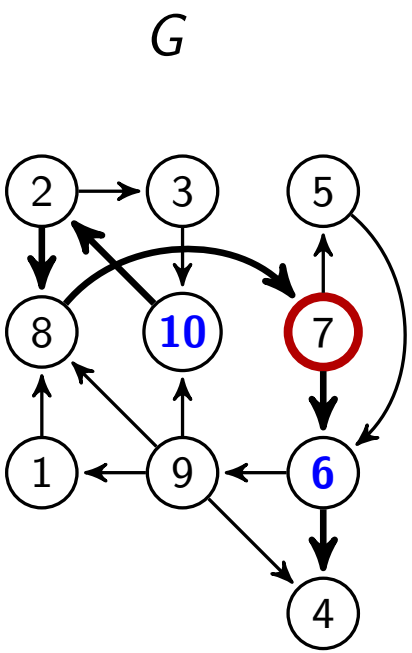
- There is a path $10 \rightsquigarrow 6$ iff there is a $u \in B$ and paths $10 \rightsquigarrow u$ and $u \rightsquigarrow 6$
- $$d(10,6) = \min_{u \in B}(d(10,u) + d(u,6))$$

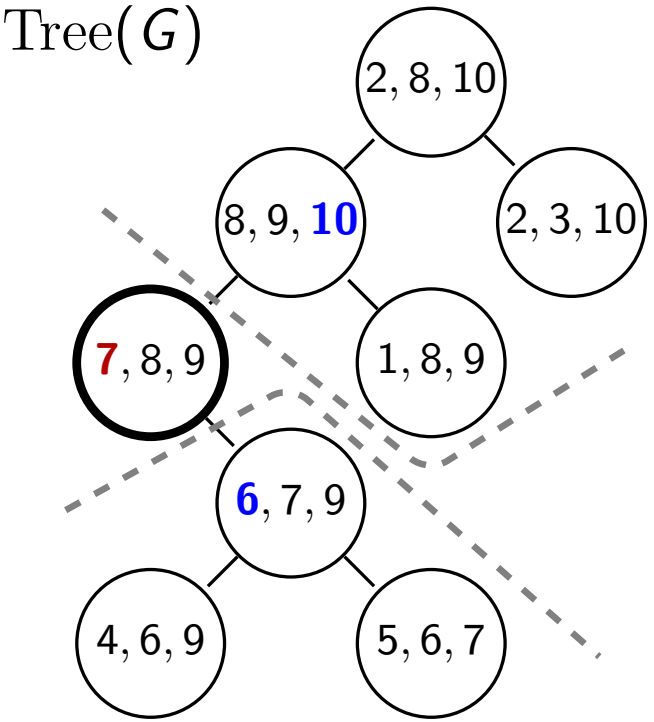
Tree Decompositions



- There is a path $10 \rightsquigarrow 6$ iff there is a $u \in B$ and paths $10 \rightsquigarrow u$ and $u \rightsquigarrow 6$
- $$d(10,6) = \min_{u \in B}(d(10,u) + d(u,6))$$

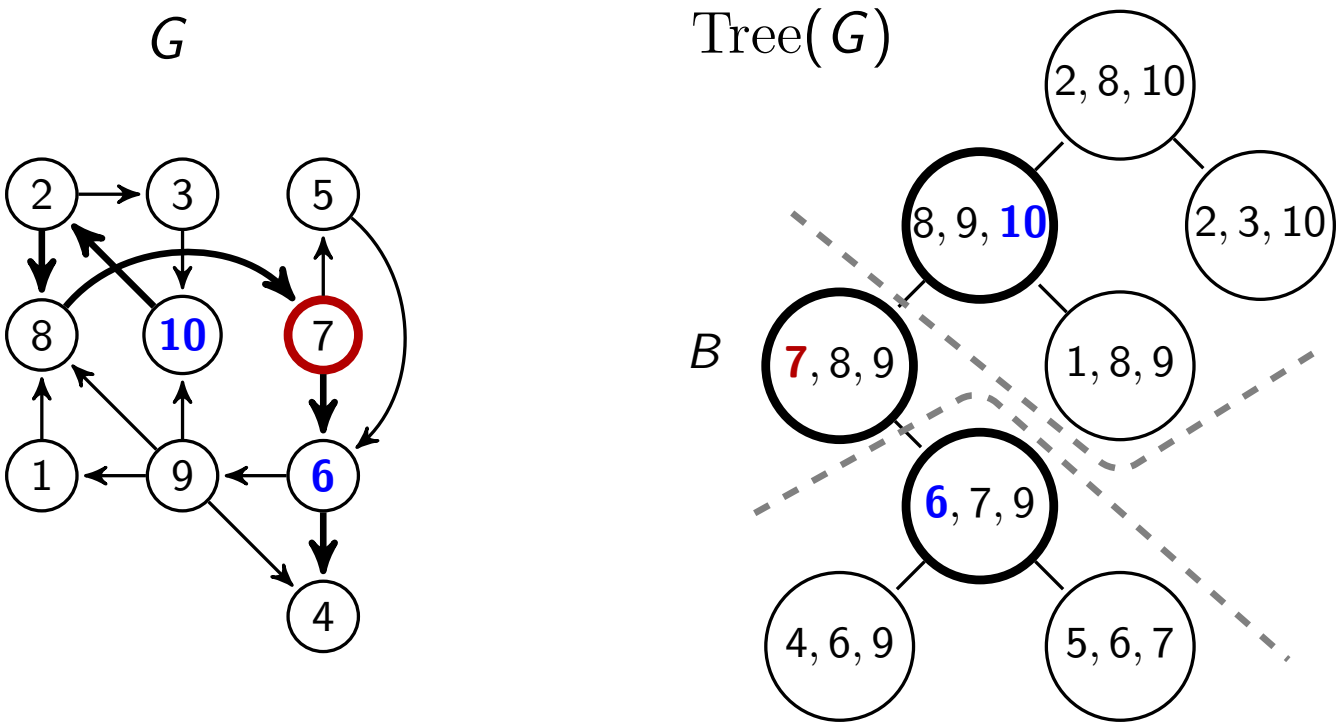
Tree Decompositions





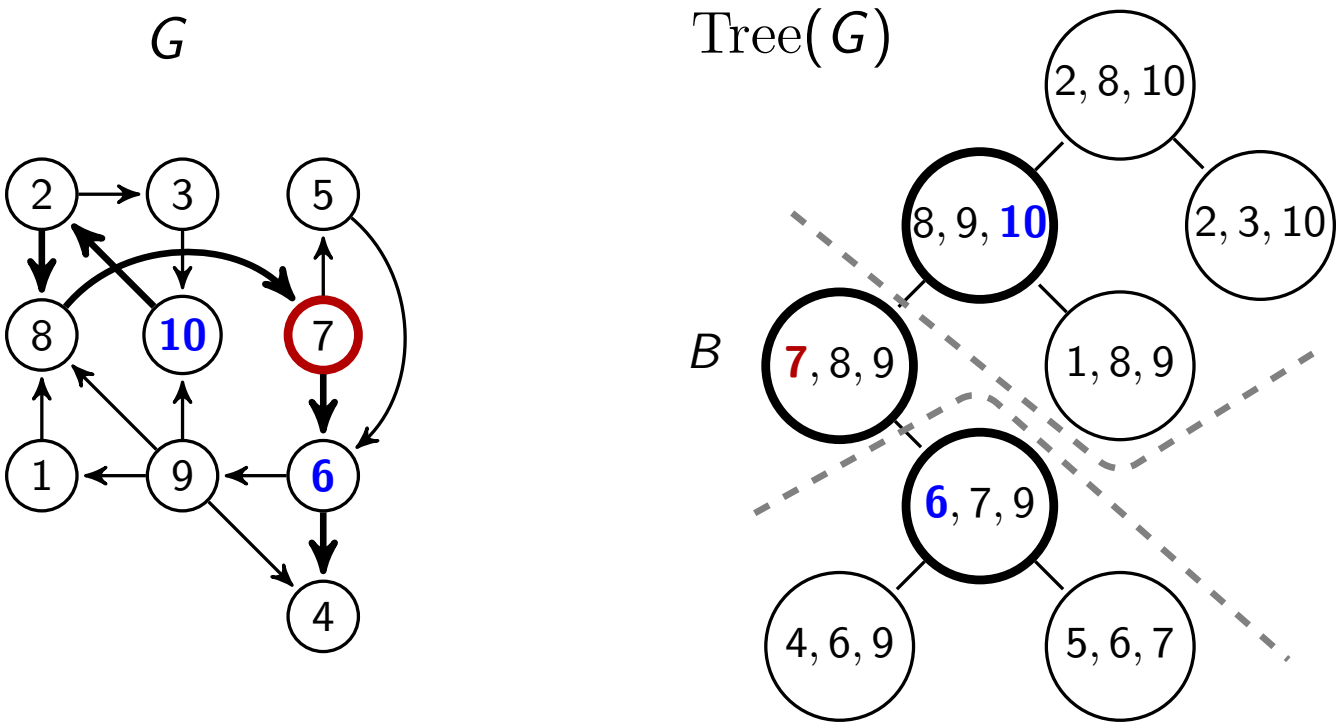
- There is a path $10 \rightsquigarrow 6$ iff there is a $u \in B$ and paths $10 \rightsquigarrow u$ and $u \rightsquigarrow 6$
- $$d(10,6) = \min_{u \in B}(d(10,u) + d(u,6))$$

Tree Decompositions



- There is a path $10 \rightsquigarrow 6$ iff there is a $u \in B$ and paths $10 \rightsquigarrow u$ and $u \rightsquigarrow 6$
- $$d(10,6) = \min_{u \in B}(d(10,u) + d(u,6))$$

Tree Decompositions



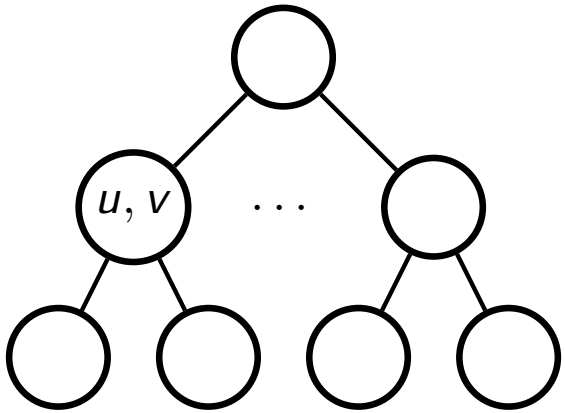
- There is a path $10 \rightsquigarrow 6$ iff there is a $u \in B$ and paths $10 \rightsquigarrow u$ and $u \rightsquigarrow 6$
- $$d(10,6) = \min_{u \in B}(d(10,u) + d(u,6))$$

Reachability

Local reachability

For every two nodes u, v appearing in a bag, compute whether $u \rightsquigarrow v$

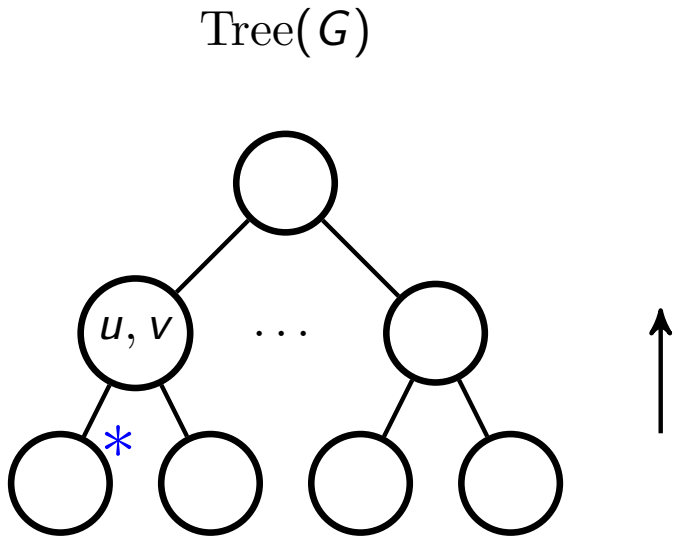
$\mathrm{Tree}(G)$



$O(n)$ time

Local reachability

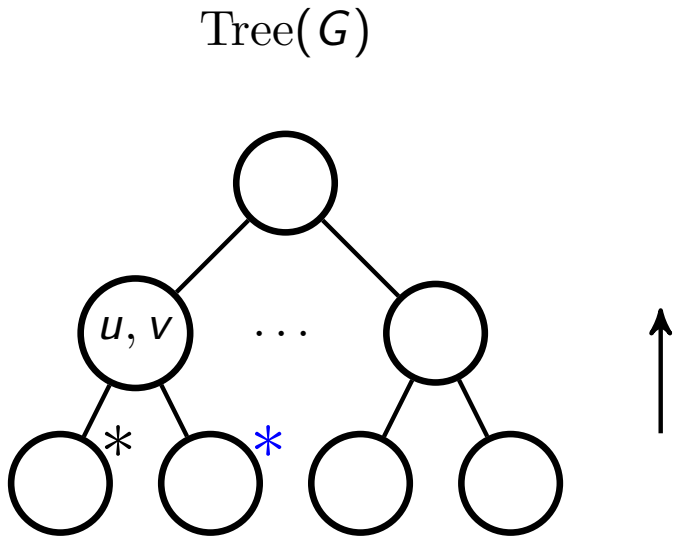
For every two nodes u, v appearing in a bag, compute whether $u \rightsquigarrow v$



$O(n)$ time

Local reachability

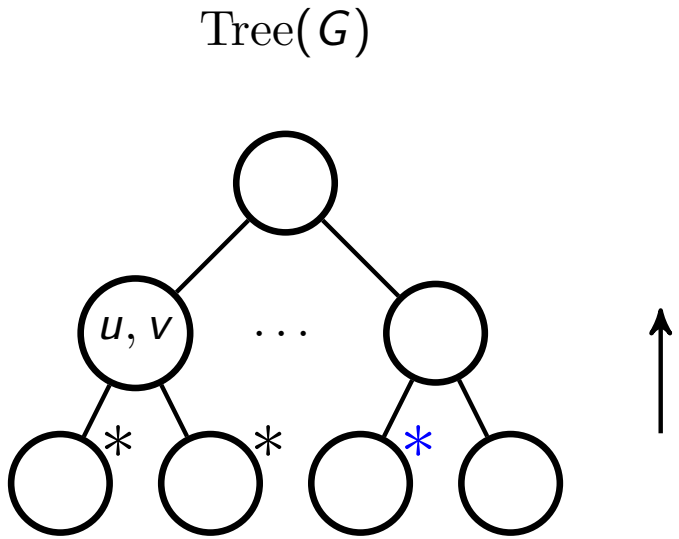
For every two nodes u, v appearing in a bag, compute whether $u \rightsquigarrow v$



$O(n)$ time

Local reachability

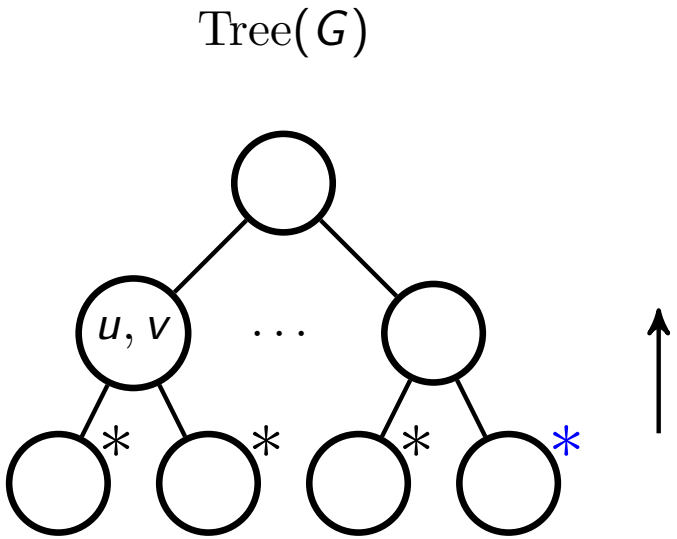
For every two nodes u, v appearing in a bag, compute whether $u \rightsquigarrow v$



$O(n)$ time

Local reachability

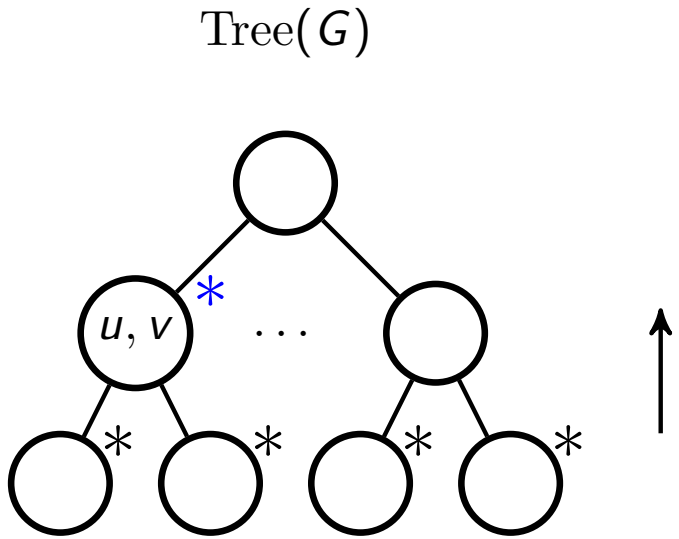
For every two nodes u, v appearing in a bag, compute whether $u \rightsquigarrow v$



$O(n)$ time

Local reachability

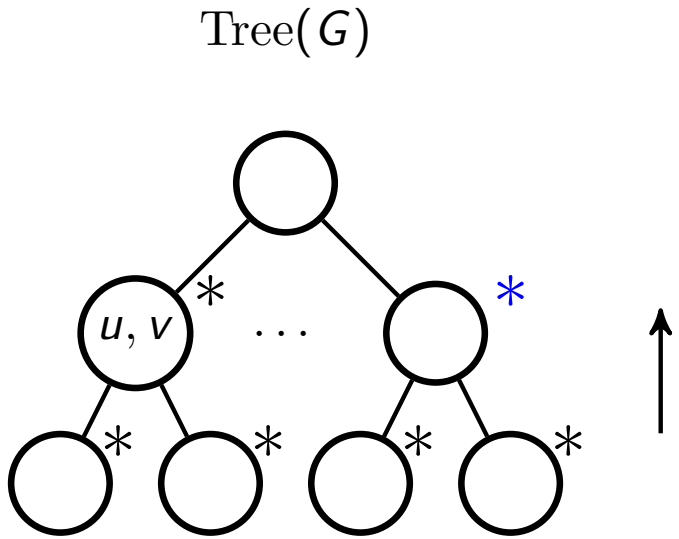
For every two nodes u, v appearing in a bag, compute whether $u \rightsquigarrow v$



$O(n)$ time

Local reachability

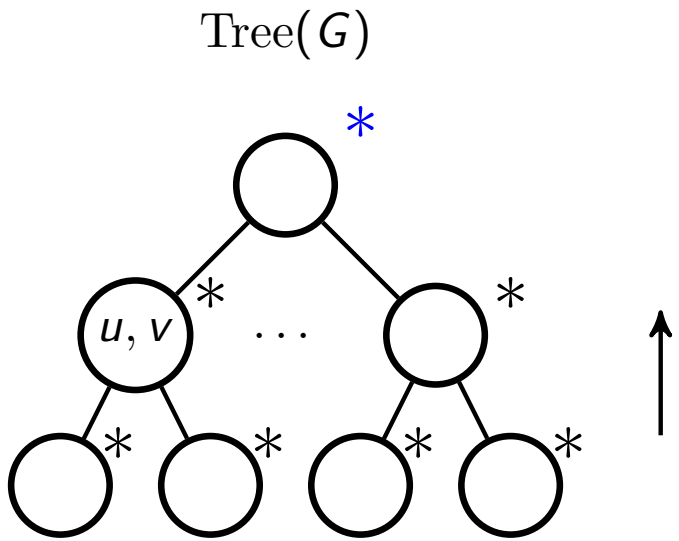
For every two nodes u, v appearing in a bag, compute whether $u \rightsquigarrow v$



$O(n)$ time

Local reachability

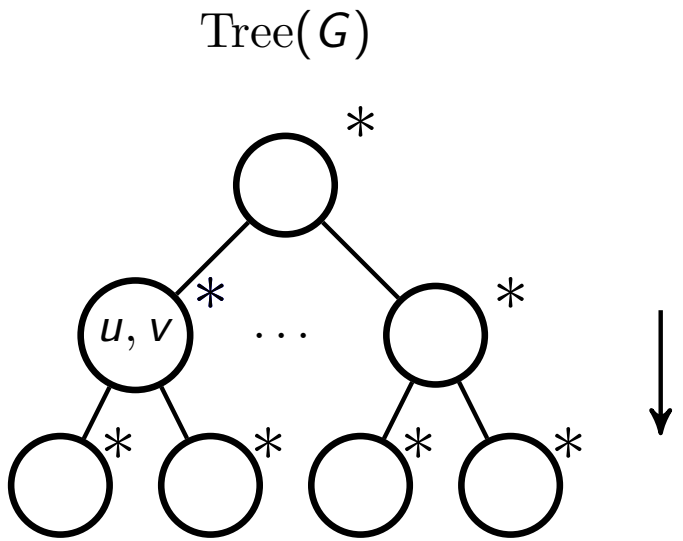
For every two nodes u, v appearing in a bag, compute whether $u \rightsquigarrow v$



$O(n)$ time

Local reachability

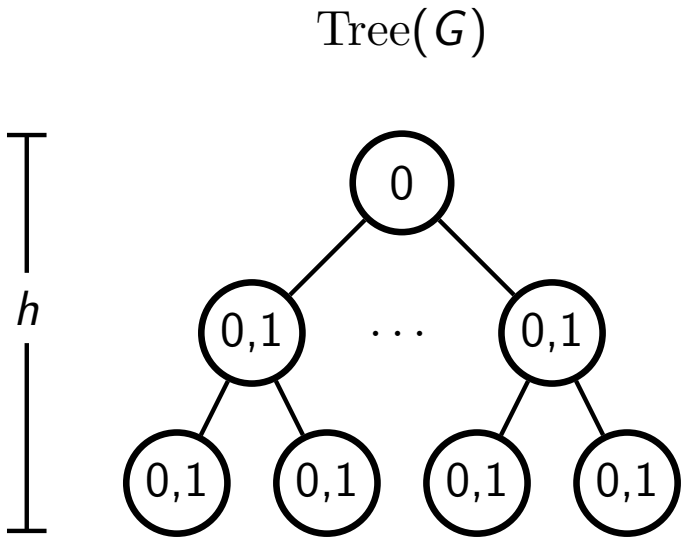
For every two nodes u, v appearing in a bag, compute whether $u \rightsquigarrow v$



$O(n)$ time

Reachability - pair query

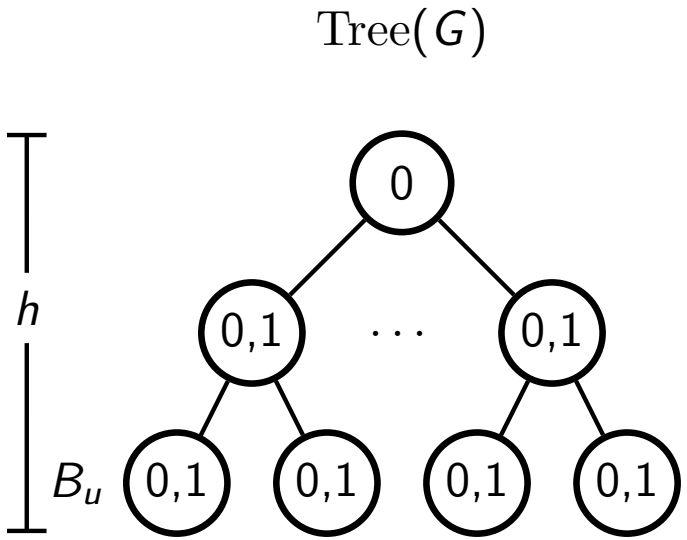
Give each node u in each bag B a local index I_u^B



- For each node u, identify the highest bag B_u containing u
- Store bit sets F_u^i and T_u^i of nodes contained in ancestor bag B^i of B_u at level i
  - The bit at position $I_v^{B^i}$ of F_u^i (T_u^i) is 1 iff $u \rightsquigarrow v$ ($v \rightsquigarrow u$)
- Pack bit sets to machine words of size $W = \Theta(\log n)$ each
- Each set has size $\frac{O(h)}{W} = O(1)$ - total space bounded by $O(n)$
- Same bound for time, by an amortization argument

Reachability - pair query

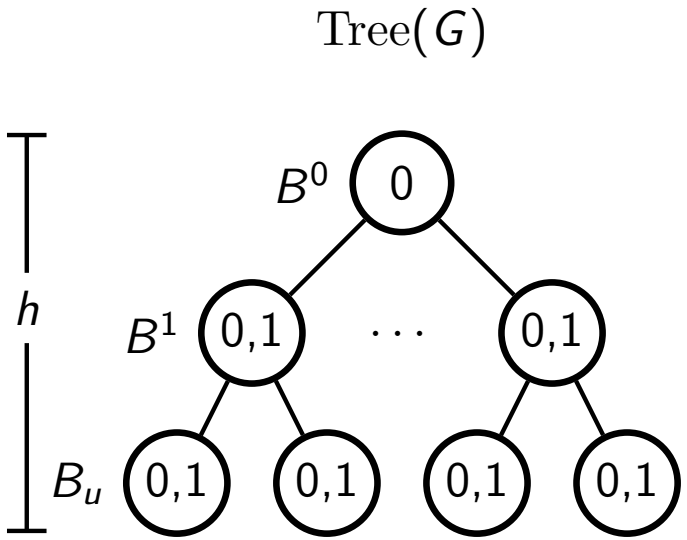
Give each node u in each bag B a local index I_u^B



- For each node u, identify the highest bag B_u containing u
- Store bit sets F_u^i and T_u^i of nodes contained in ancestor bag B^i of B_u at level i
  - The bit at position $I_v^{B^i}$ of F_u^i (T_u^i) is 1 iff $u \rightsquigarrow v$ ($v \rightsquigarrow u$)
- Pack bit sets to machine words of size $W = \Theta(\log n)$ each
- Each set has size $\frac{O(h)}{W} = O(1)$ - total space bounded by $O(n)$
- Same bound for time, by an amortization argument

Reachability - pair query

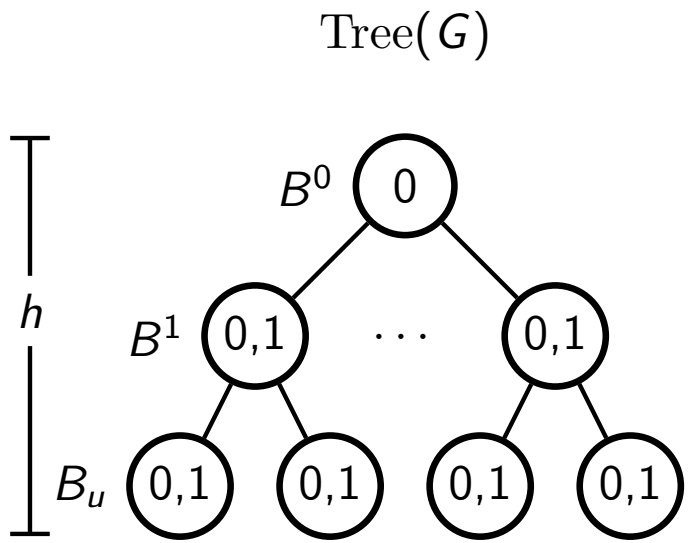
Give each node u in each bag B a local index I_u^B



- For each node u, identify the highest bag B_u containing u
- Store bit sets F_u^i and T_u^i of nodes contained in ancestor bag B^i of B_u at level i
  - The bit at position $I_v^{B^i}$ of F_u^i (T_u^i) is 1 iff $u \rightsquigarrow v$ ($v \rightsquigarrow u$)
- Pack bit sets to machine words of size $W = \Theta(\log n)$ each
- Each set has size $\frac{O(h)}{W} = O(1)$ - total space bounded by $O(n)$
- Same bound for time, by an amortization argument

Reachability - pair query

Give each node u in each bag B a local index I_u^B

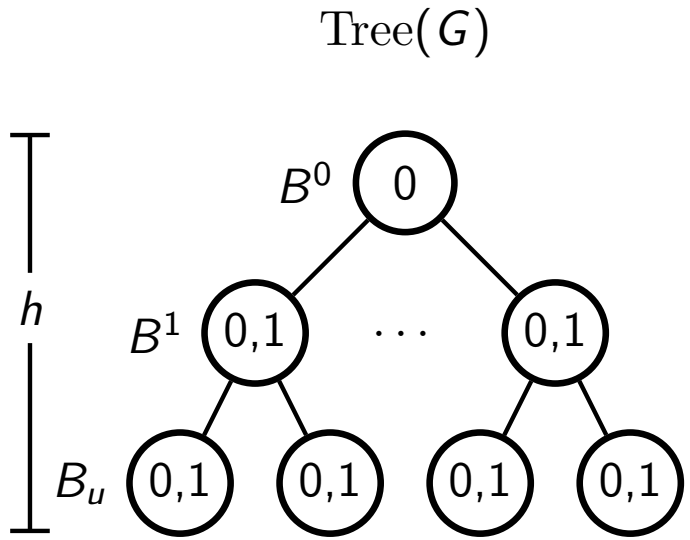


Top-down pass + Local reachability

- For each node u, identify the highest bag B_u containing u
- Store bit sets F_u^i and T_u^i of nodes contained in ancestor bag B^i of B_u at level i
  - The bit at position $I_v^{B^i}$ of F_u^i (T_u^i) is 1 iff $u \rightsquigarrow v$ ($v \rightsquigarrow u$)
- Pack bit sets to machine words of size $W = \Theta(\log n)$ each
- Each set has size $\frac{O(h)}{W} = O(1)$ - total space bounded by $O(n)$
- Same bound for time, by an amortization argument

Reachability - pair query

Give each node u in each bag B a local index I_u^B

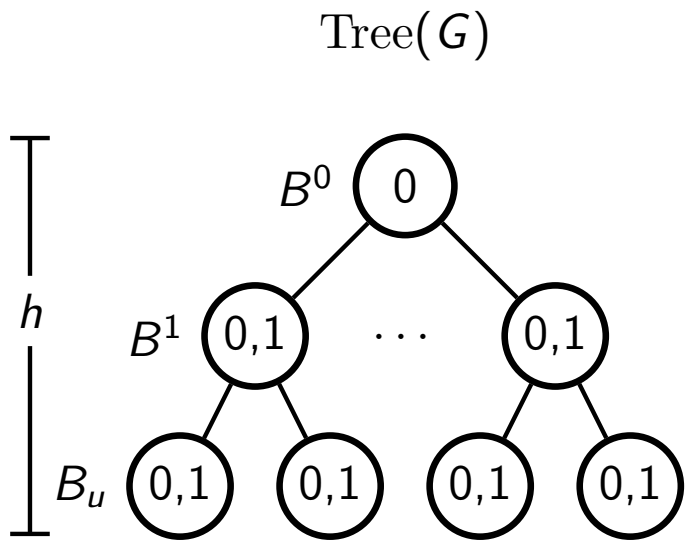


Top-down pass + Local reachability

- For each node u, identify the highest bag B_u containing u
- Store bit sets F_u^i and T_u^i of nodes contained in ancestor bag B^i of B_u at level i
  - The bit at position $I_v^{B^i}$ of F_u^i (T_u^i) is 1 iff $u \rightsquigarrow v$ ($v \rightsquigarrow u$)
- Pack bit sets to machine words of size $W = \Theta(\log n)$ each
- Each set has size $\frac{O(h)}{W} = O(1)$ - total space bounded by $O(n)$
- Same bound for time, by an amortization argument

Reachability - pair query

Give each node u in each bag B a local index I_u^B

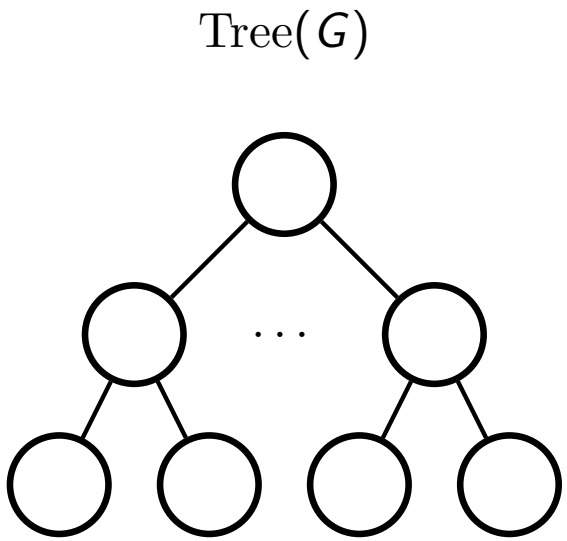


Top-down pass + Local reachability

- For each node u, identify the highest bag B_u containing u
- Store bit sets F_u^i and T_u^i of nodes contained in ancestor bag B^i of B_u at level i
  - The bit at position $I_v^{B^i}$ of F_u^i (T_u^i) is 1 iff $u \rightsquigarrow v$ ($v \rightsquigarrow u$)
- Pack bit sets to machine words of size $W = \Theta(\log n)$ each
- Each set has size $\frac{O(h)}{W} = O(1)$ - total space bounded by $O(n)$
- Same bound for time, by an amortization argument

Reachability - pair query

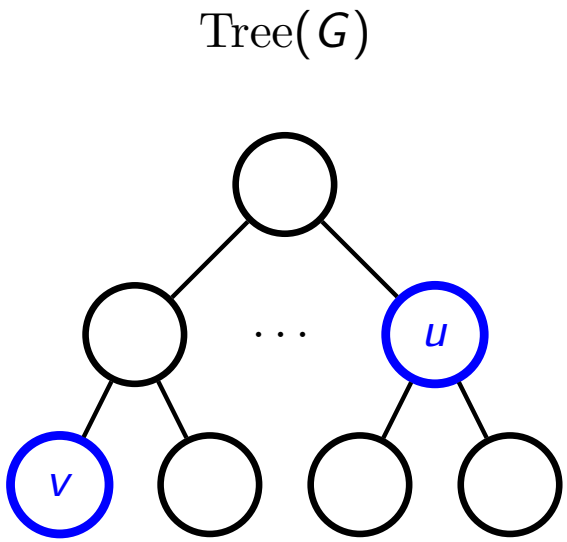
Query: does u reach v?



- Identify the highest bags B_u and B_v containing u and v
- Identify the LCA B^i of B_u and B_v at level i
- u reaches v iff F^i_u and T^i_v contain a 1 in the same position

Reachability - pair query

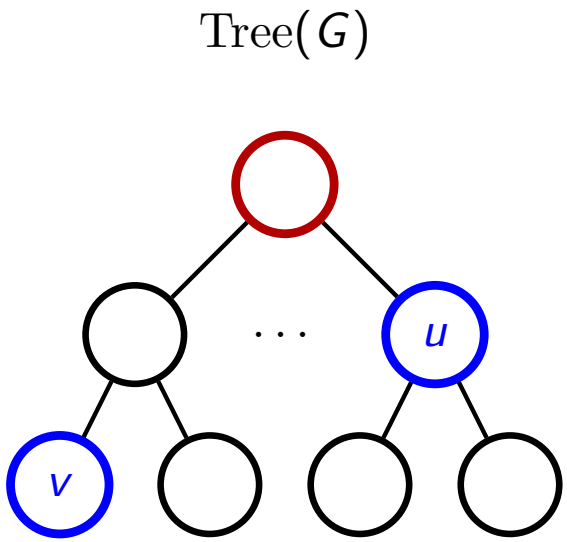
Query: does u reach v?



- Identify the highest bags B_u and B_v containing u and v
- Identify the LCA B^i of B_u and B_v at level i
- u reaches v iff F^i_u and T^i_v contain a 1 in the same position

Reachability - pair query

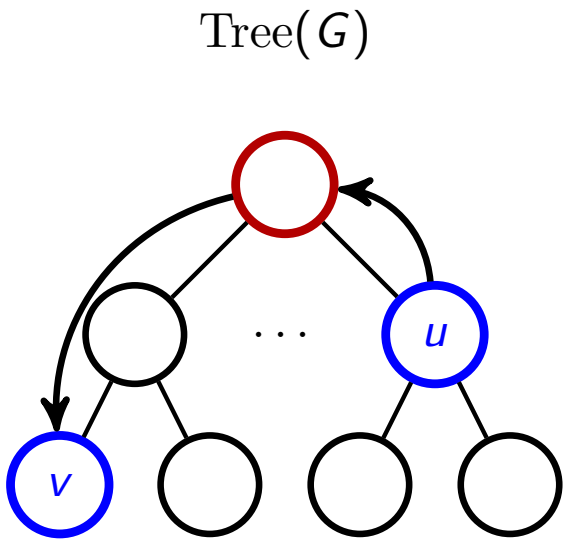
Query: does u reach v?



- Identify the highest bags B_u and B_v containing u and v
- Identify the LCA B^i of B_u and B_v at level i
- u reaches v iff F_u^i and T_v^i contain a 1 in the same position

Reachability - pair query

Query: does u reach v?



- Identify the highest bags B_u and B_v containing u and v
- Identify the LCA B^i of B_u and B_v at level i
- u reaches v iff F^i_u and T^i_v contain a 1 in the same position

Results

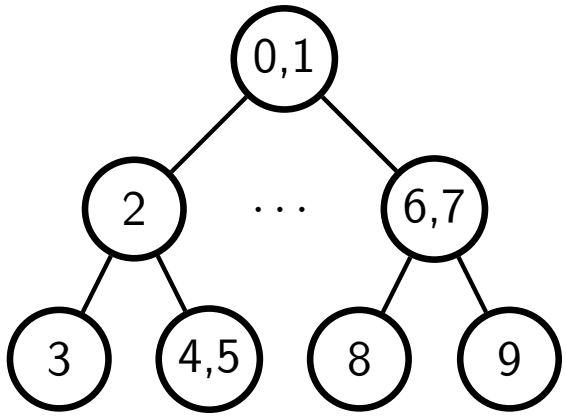
Preprocessing time	Pair query time	Single-source query time
–	$O(n)$	$O(n)$
$O(n \cdot \log n)$	$O(\log n)$	$O(n)$
$\mathbf{O(n)}$	$\mathbf{O(1)}$	$\mathbf{O\left(\frac{n}{\log n}\right)}$

Thank you!
Questions?

Reachability - single source query

Give each node u an index I_u, such that the indexes of nodes which are contained in each subtree form a contiguous interval

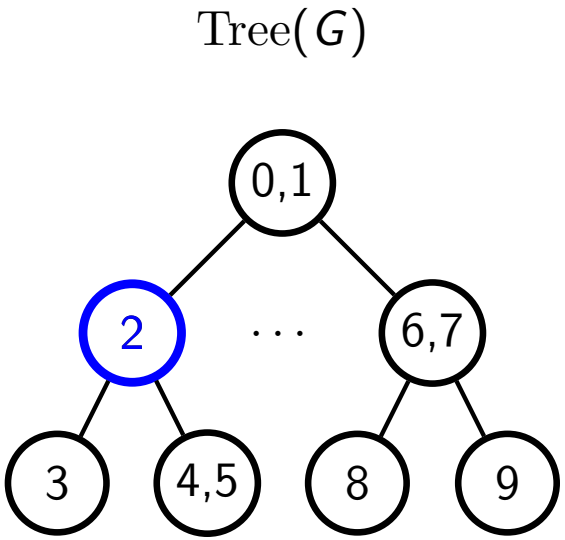
$\mathrm{Tree}(G)$



- For each node u, identify the highest bag B_u containing u
- Store a bit set F_u of nodes contained in the subtree of B_u
  - The bit at position $I_v - I_u$ of F_u is 1 iff $u \rightsquigarrow v$
- Pack bit sets to machine words of size $W = \Theta(\log n)$ each
- Each node appears in $O(h)$ sets
- Total space bounded by $\frac{n \cdot O(h)}{W} = O(n)$
- Same bound for time, by an amortization argument

Reachability - single source query

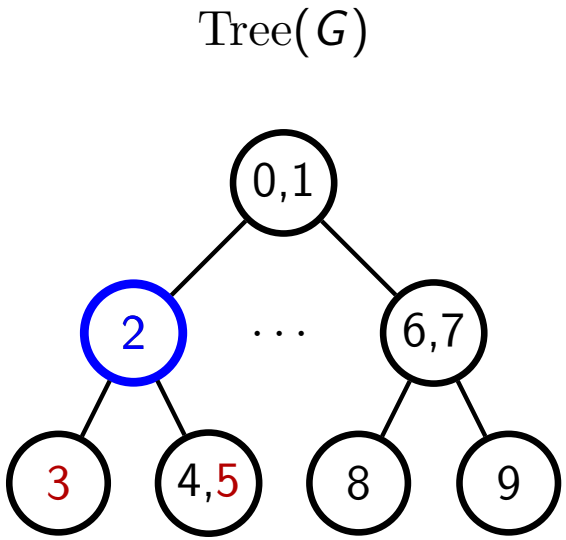
Give each node u an index I_u, such that the indexes of nodes which are contained in each subtree form a contiguous interval



- For each node u, identify the highest bag B_u containing u
- Store a bit set F_u of nodes contained in the subtree of B_u
  - The bit at position $I_v - I_u$ of F_u is 1 iff $u \rightsquigarrow v$
- Pack bit sets to machine words of size $W = \Theta(\log n)$ each
- Each node appears in $O(h)$ sets
- Total space bounded by $\frac{n \cdot O(h)}{W} = O(n)$
- Same bound for time, by an amortization argument

Reachability - single source query

Give each node u an index I_u, such that the indexes of nodes which are contained in each subtree form a contiguous interval

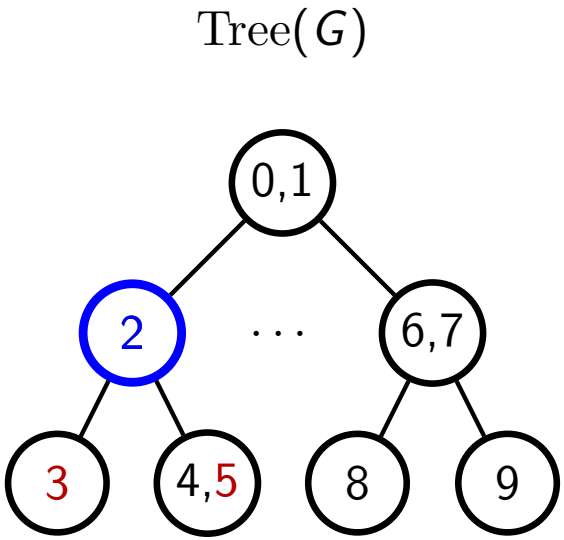


I_v	2	3	4	5
F_u	1	1	0	1

- For each node u, identify the highest bag B_u containing u
- Store a bit set F_u of nodes contained in the subtree of B_u
  - The bit at position $I_v - I_u$ of F_u is 1 iff $u \rightsquigarrow v$
- Pack bit sets to machine words of size $W = \Theta(\log n)$ each
- Each node appears in $O(h)$ sets
- Total space bounded by $\frac{n \cdot O(h)}{W} = O(n)$
- Same bound for time, by an amortization argument

Reachability - single source query

Give each node u an index I_u, such that the indexes of nodes which are contained in each subtree form a contiguous interval

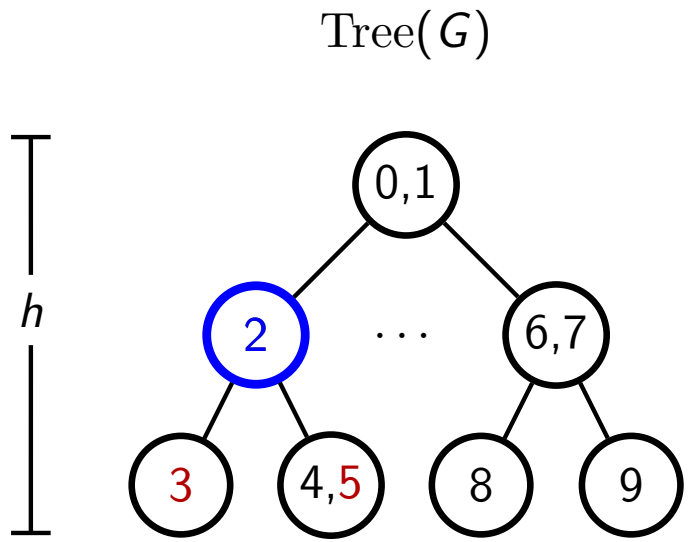


Bottom-up pass + Local reachability

- For each node u, identify the highest bag B_u containing u
- Store a bit set F_u of nodes contained in the subtree of B_u
  - The bit at position $I_v - I_u$ of F_u is 1 iff $u \rightsquigarrow v$
- Pack bit sets to machine words of size $W = \Theta(\log n)$ each
- Each node appears in $O(h)$ sets
- Total space bounded by $\frac{n \cdot O(h)}{W} = O(n)$
- Same bound for time, by an amortization argument

Reachability - single source query

Give each node u an index I_u, such that the indexes of nodes which are contained in each subtree form a contiguous interval

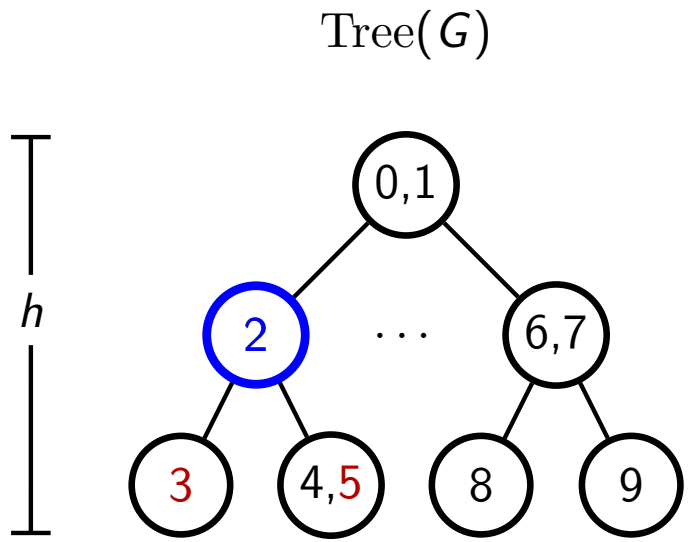


Bottom-up pass + Local reachability

- For each node u, identify the highest bag B_u containing u
- Store a bit set F_u of nodes contained in the subtree of B_u
  - The bit at position $I_v - I_u$ of F_u is 1 iff $u \rightsquigarrow v$
- Pack bit sets to machine words of size $W = \Theta(\log n)$ each
- Each node appears in $O(h)$ sets
- Total space bounded by $\frac{n \cdot O(h)}{W} = O(n)$
- Same bound for time, by an amortization argument

Reachability - single source query

Give each node u an index I_u, such that the indexes of nodes which are contained in each subtree form a contiguous interval

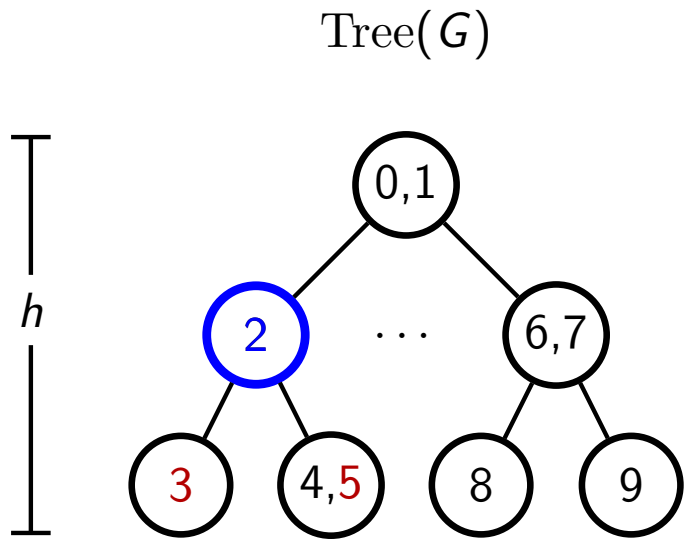


Bottom-up pass + Local reachability

- For each node u, identify the highest bag B_u containing u
- Store a bit set F_u of nodes contained in the subtree of B_u
  - The bit at position $I_v - I_u$ of F_u is 1 iff $u \rightsquigarrow v$
- Pack bit sets to machine words of size $W = \Theta(\log n)$ each
- Each node appears in $O(h)$ sets
- Total space bounded by $\frac{n \cdot O(h)}{W} = O(n)$
- Same bound for time, by an amortization argument

Reachability - single source query

Give each node u an index I_u, such that the indexes of nodes which are contained in each subtree form a contiguous interval



Bottom-up pass + Local reachability

- For each node u, identify the highest bag B_u containing u
- Store a bit set F_u of nodes contained in the subtree of B_u
  - The bit at position $I_v - I_u$ of F_u is 1 iff $u \rightsquigarrow v$
- Pack bit sets to machine words of size $W = \Theta(\log n)$ each
- Each node appears in $O(h)$ sets
- Total space bounded by $\frac{n \cdot O(h)}{W} = O(n)$
- Same bound for time, by an amortization argument