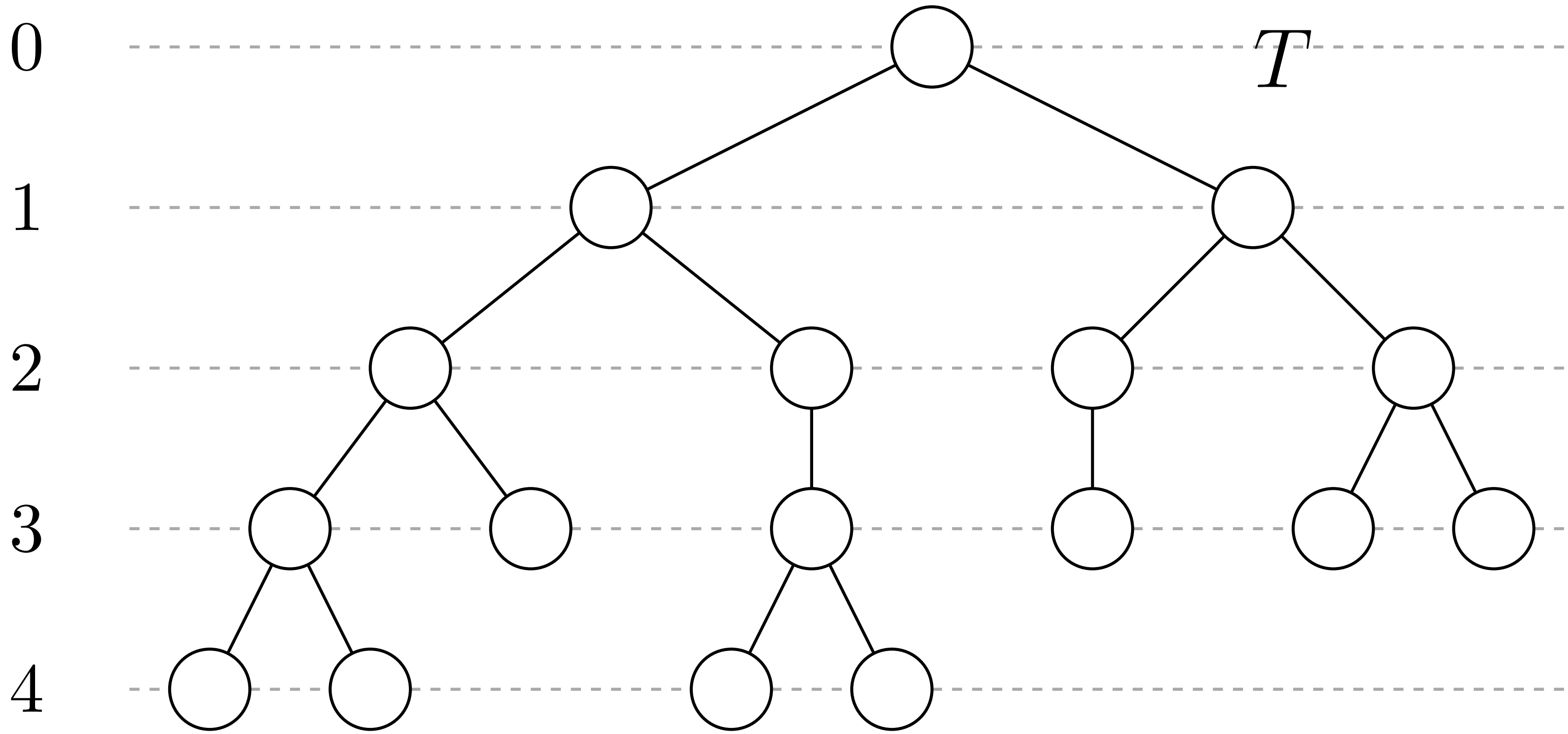
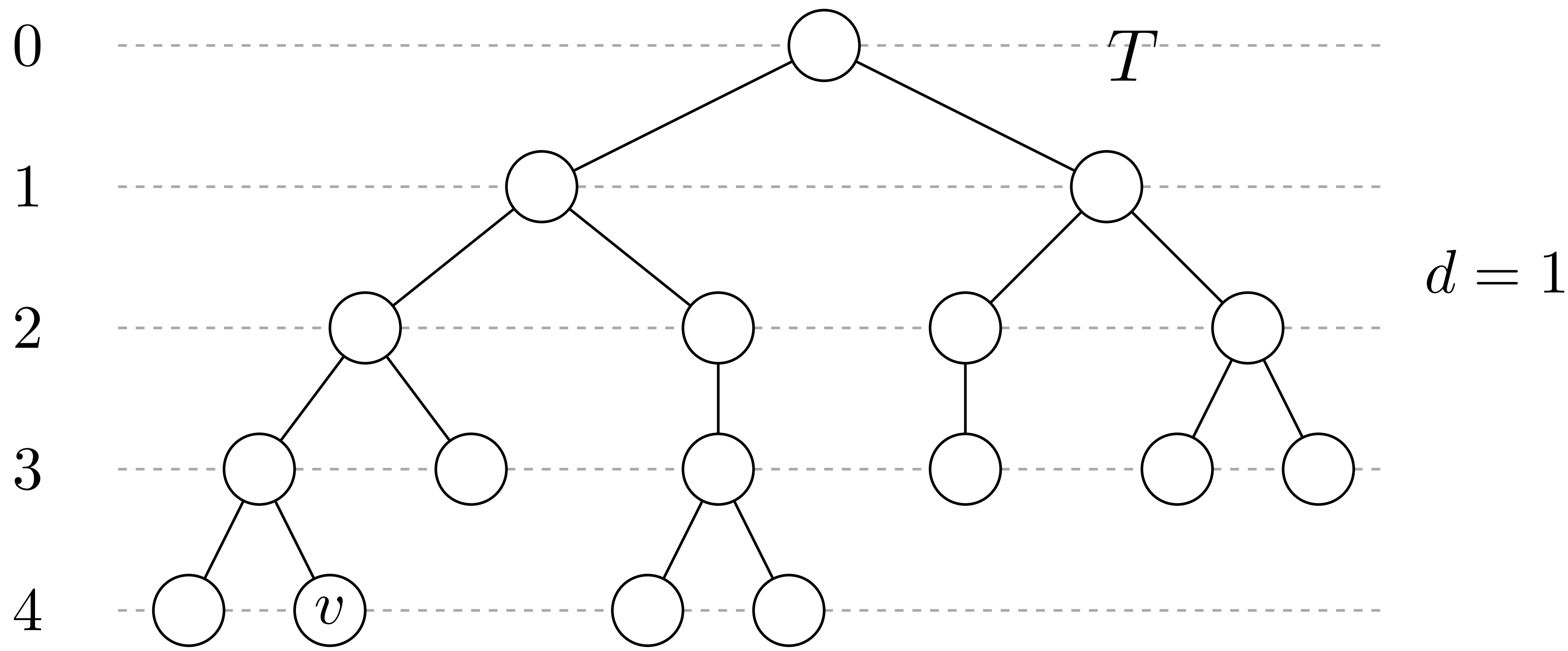
# Level Ancestors

# Level Ancestor Queries



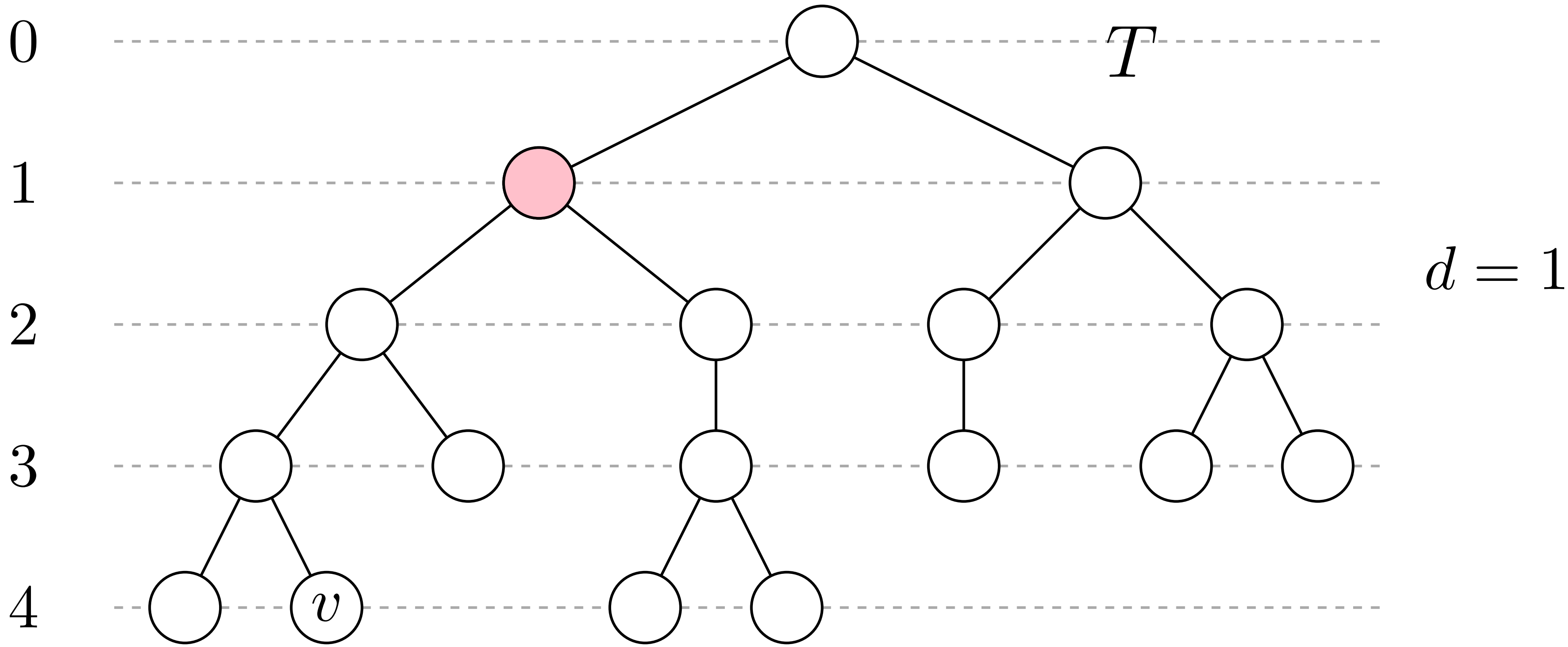

**Definition:** Let $v$ be a vertex at depth $d_v$ in $T$. For $d \leq d_v$, a *level ancestor query* LA$(v, d)$ on a vertex $v$ asks to report the ancestor of $v$ at depth $d$.

# Level Ancestor Queries



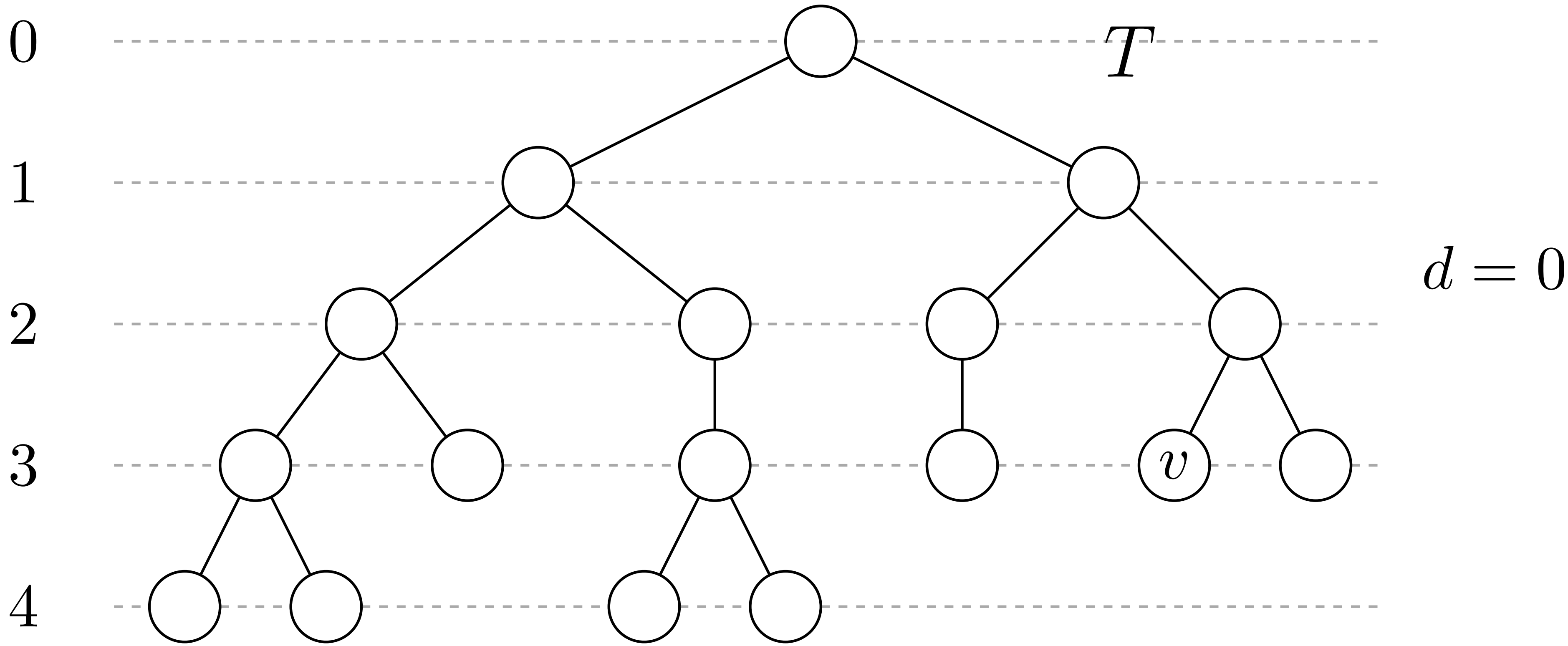

**Definition:** Let $v$ be a vertex at depth $d_v$ in $T$. For $d \leq d_v$, a *level ancestor query* LA$(v, d)$ on a vertex $v$ asks to report the ancestor of $v$ at depth $d$.

# Level Ancestor Queries



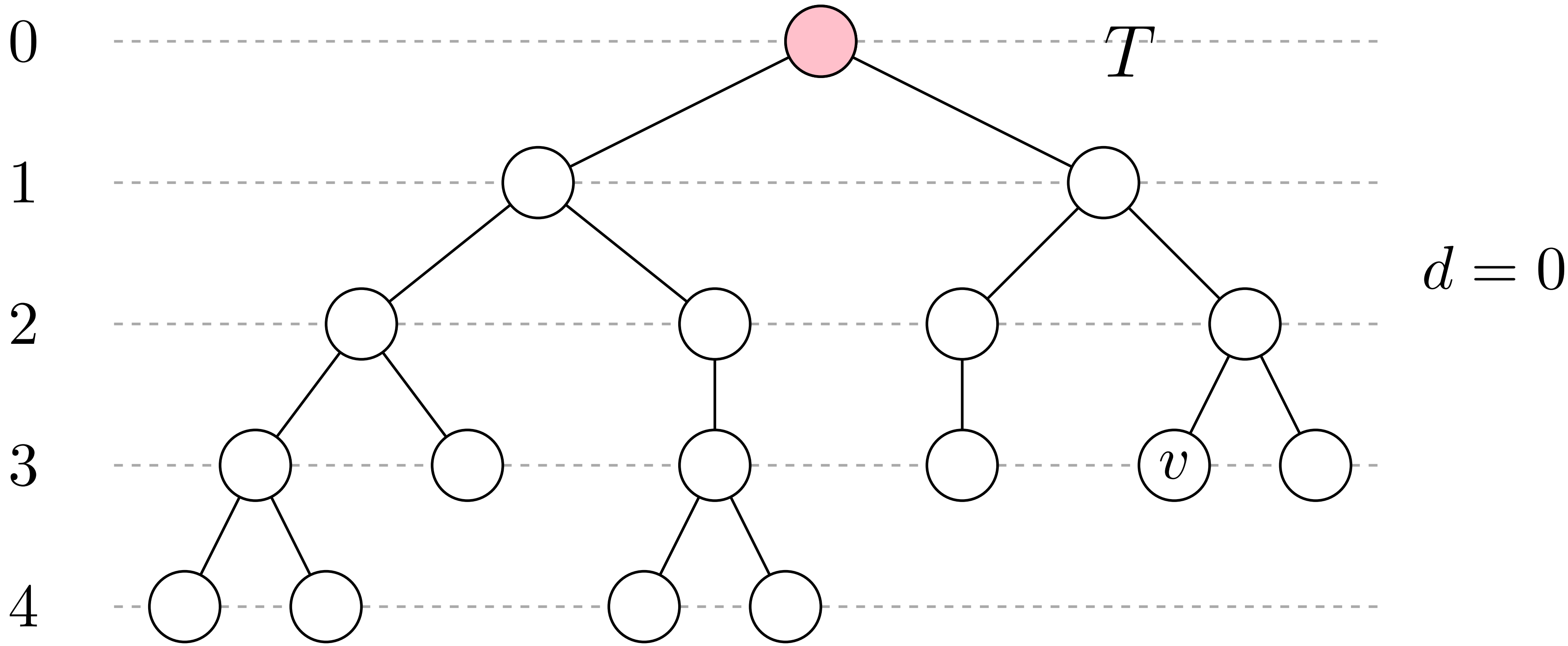

**Definition:** Let $v$ be a vertex at depth $d_v$ in $T$. For $d \leq d_v$, a *level ancestor query* LA$(v, d)$ on a vertex $v$ asks to report the ancestor of $v$ at depth $d$.

# Level Ancestor Queries



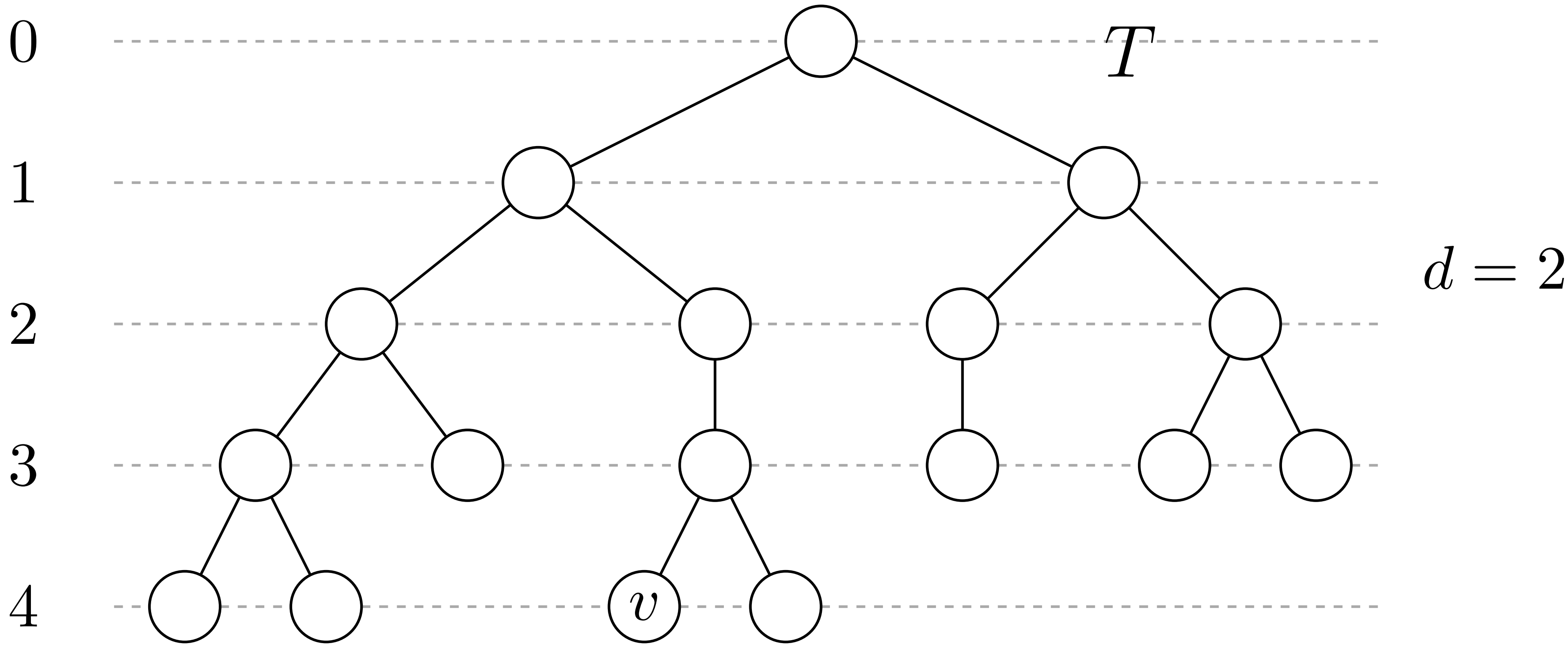

**Definition:** Let $v$ be a vertex at depth $d_v$ in $T$. For $d \leq d_v$, a *level ancestor query* LA$(v, d)$ on a vertex $v$ asks to report the ancestor of $v$ at depth $d$.

# Level Ancestor Queries



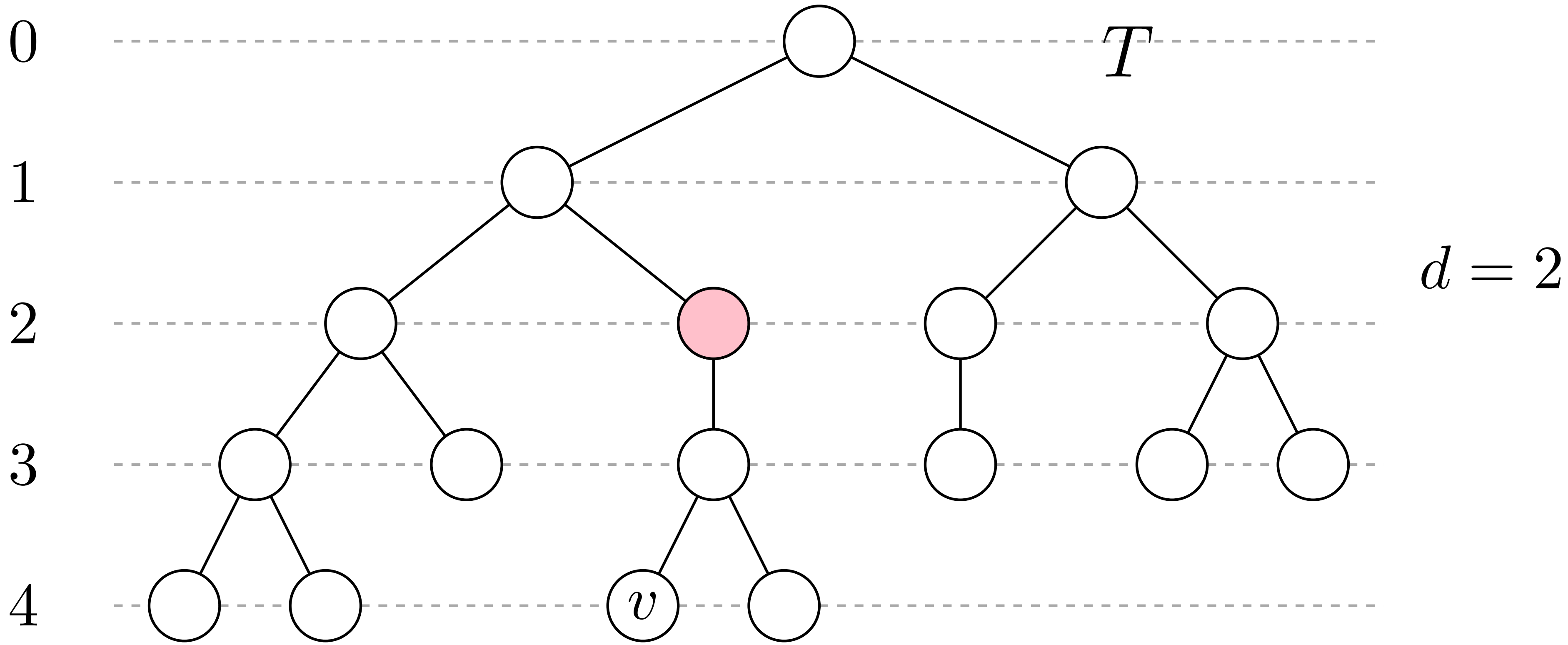

**Definition:** Let $v$ be a vertex at depth $d_v$ in $T$. For $d \leq d_v$, a *level ancestor query* LA$(v, d)$ on a vertex $v$ asks to report the ancestor of $v$ at depth $d$.

# Level Ancestor Queries




**Definition:** Let $v$ be a vertex at depth $d_v$ in $T$. For $d \leq d_v$, a *level ancestor query* LA$(v, d)$ on a vertex $v$ asks to report the ancestor of $v$ at depth $d$.

# Level Ancestor Queries




**Definition:** Let $v$ be a vertex at depth $d_v$ in $T$. For $d \leq d_v$, a *level ancestor query* LA$(v, d)$ on a vertex $v$ asks to report the ancestor of $v$ at depth $d$.

# The Problem

Given $T$, design a data structure that is able to preprocess $T$ to answer level ancestors queries.

# The Problem

Given $T$, design a data structure that is able to preprocess $T$ to answer level ancestors queries.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none  Size: $O(n)$  Query time: $O(n)$

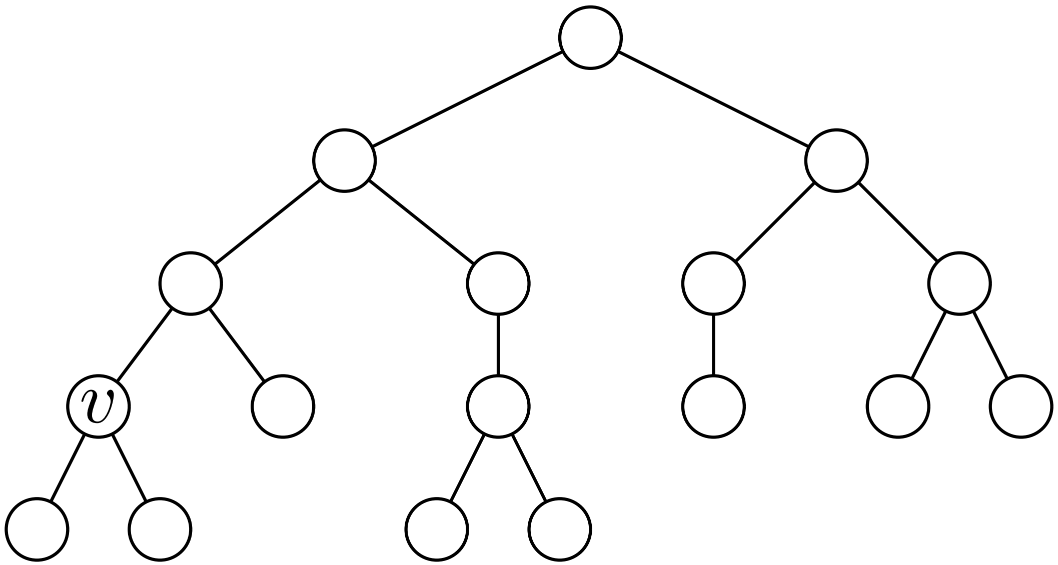
# The Problem

Given $T$, design a data structure that is able to preprocess $T$ to answer level ancestors queries.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$



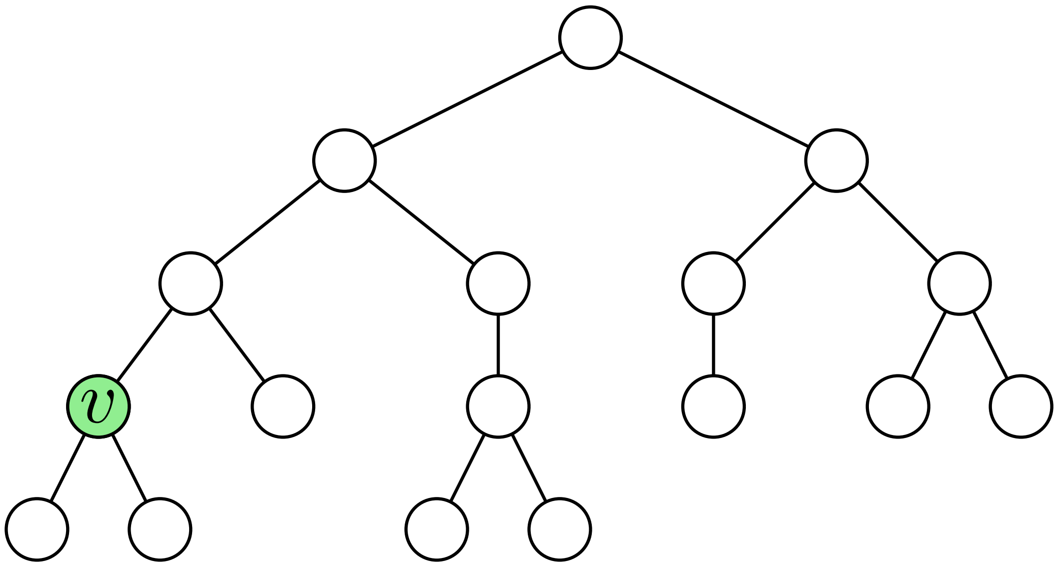
# The Problem

Given $T$, design a data structure that is able to preprocess $T$ to answer level ancestors queries.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$



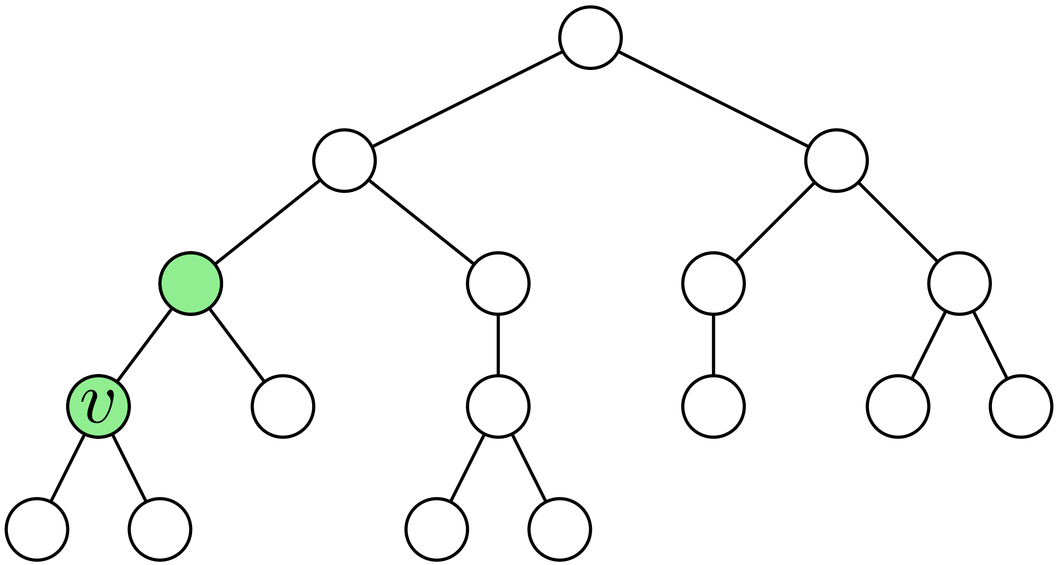
# The Problem

Given $T$, design a data structure that is able to preprocess $T$ to answer level ancestors queries.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none  Size: $O(n)$  Query time: $O(n)$



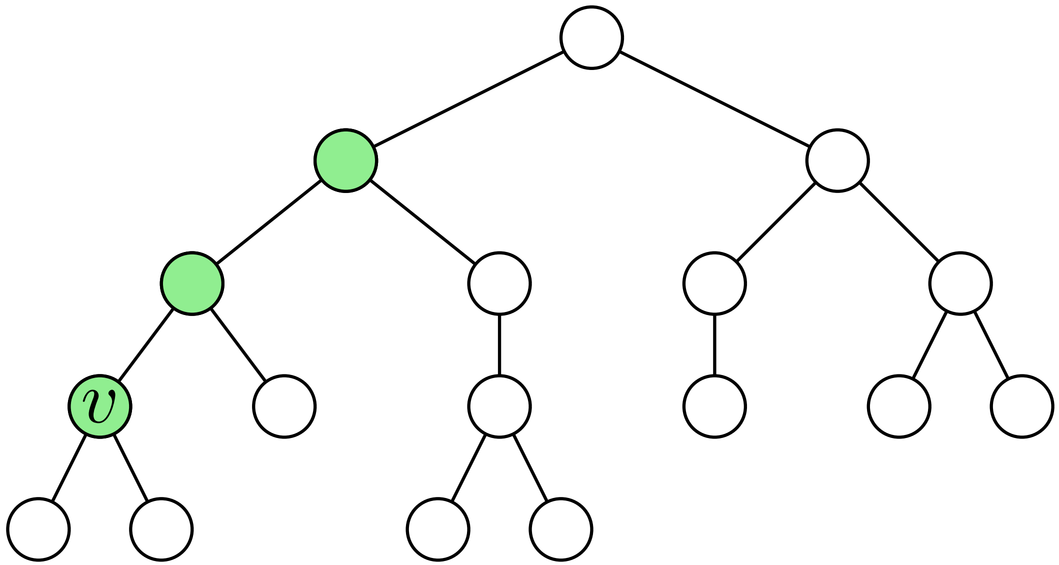
# The Problem

Given $T$, design a data structure that is able to preprocess $T$ to answer level ancestors queries.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$



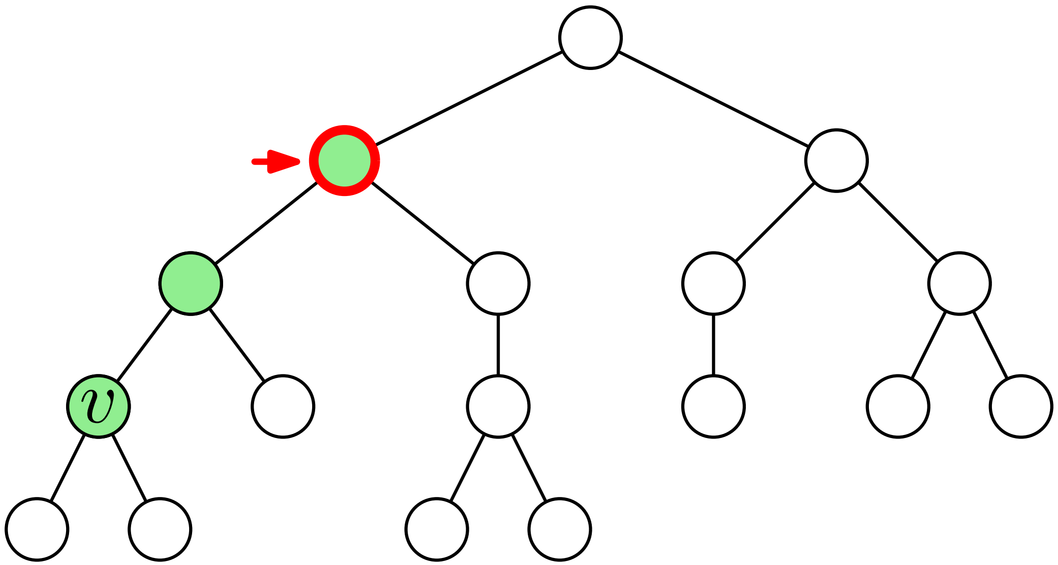
# The Problem

Given $T$, design a data structure that is able to preprocess $T$ to answer level ancestors queries.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$



# The Problem

Given $T$, design a data structure that is able to preprocess $T$ to answer level ancestors queries.

Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none Size: $O(n)$ Query time: $O(n)$
- Preprocessing time: $O(n^3)$ Size: $O(n^2)$ Query time: $O(1)$

(precompute the answer to all possible queries)

# The Problem

Given $T$, design a data structure that is able to preprocess $T$ to answer level ancestors queries.

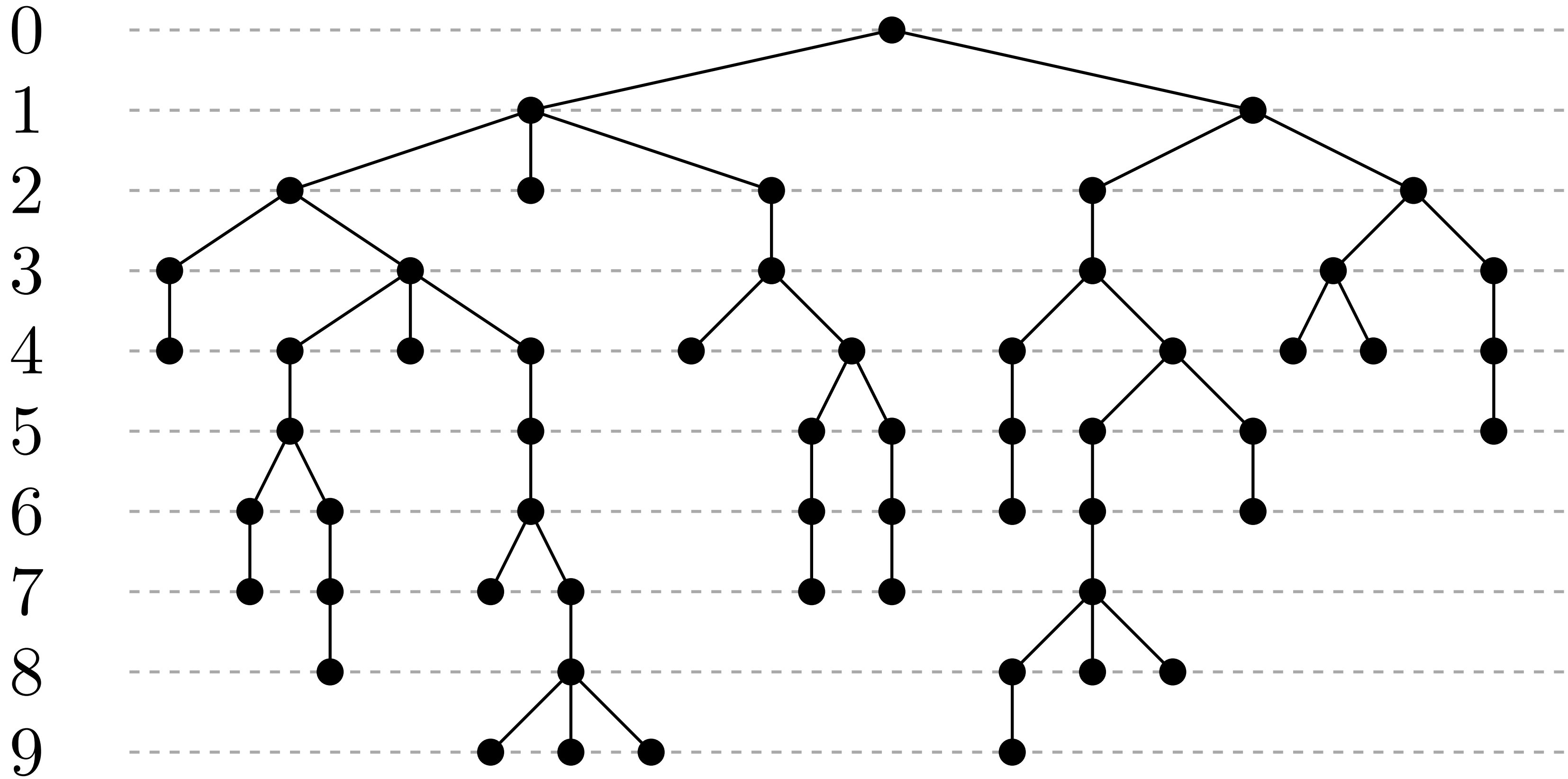
Trivial solutions:

$n = \#$ of nodes

- Preprocessing time: none   Size: $O(n)$   Query time: $O(n)$
- Preprocessing time: $O(n^3)$   Size: $O(n^2)$   Query time: $O(1)$
- Preprocessing time: $O(n^2)$   Size: $O(n^2)$   Query time: $O(1)$

$$\text{LA}(v, d) = \begin{cases} v & \text{if } d = d_v \\ \text{LA}(\text{parent}(v),\ d) & \text{if } d < d_v \end{cases}$$

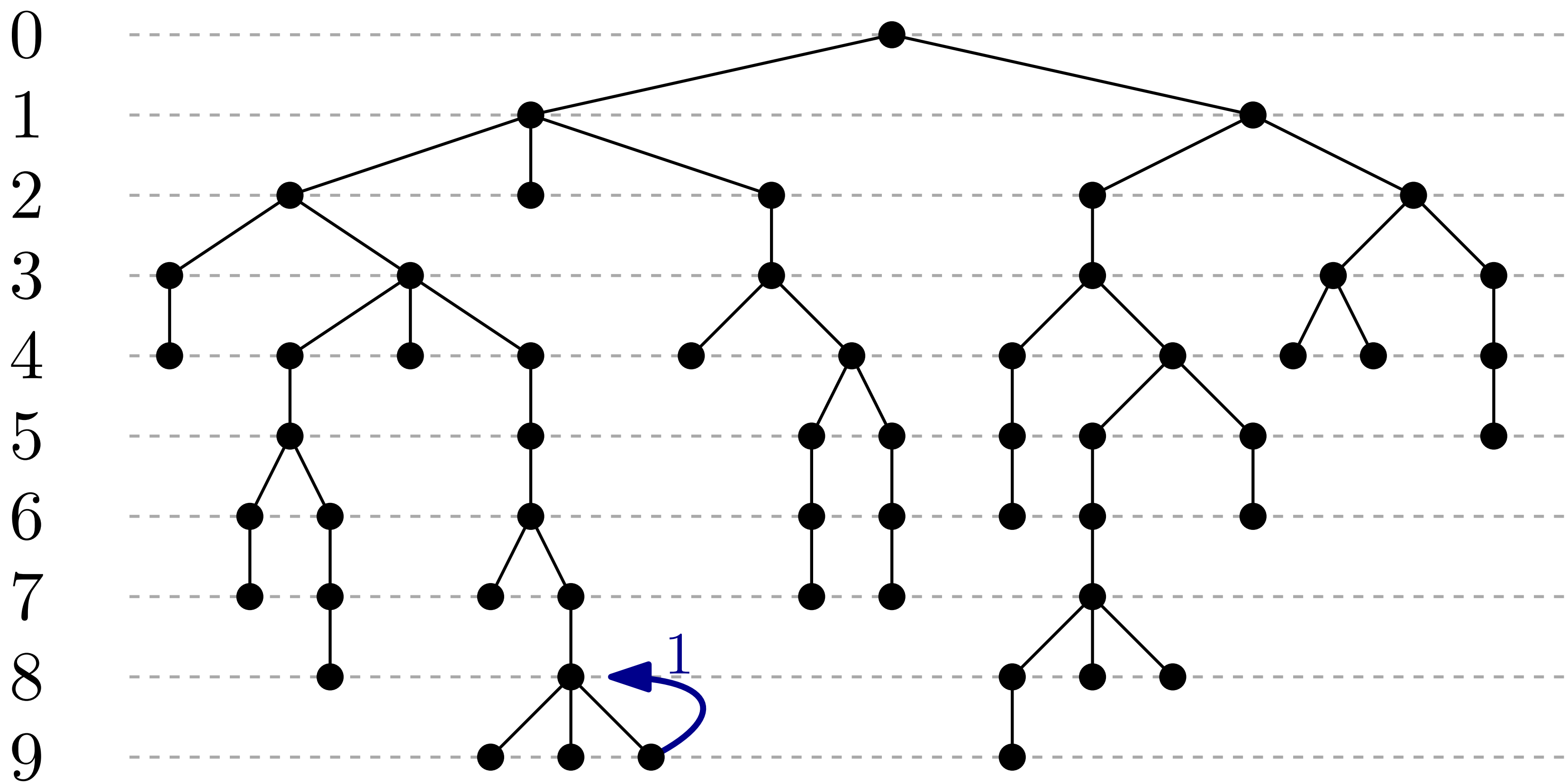
# Jump Pointers: Idea




For each vertex $v$ and $\ell = 0, 1, \ldots, \lfloor \log d_v \rfloor$, store:

$$J(v, \ell) = \mathsf{LA}(v, d_v - 2^{\ell})$$

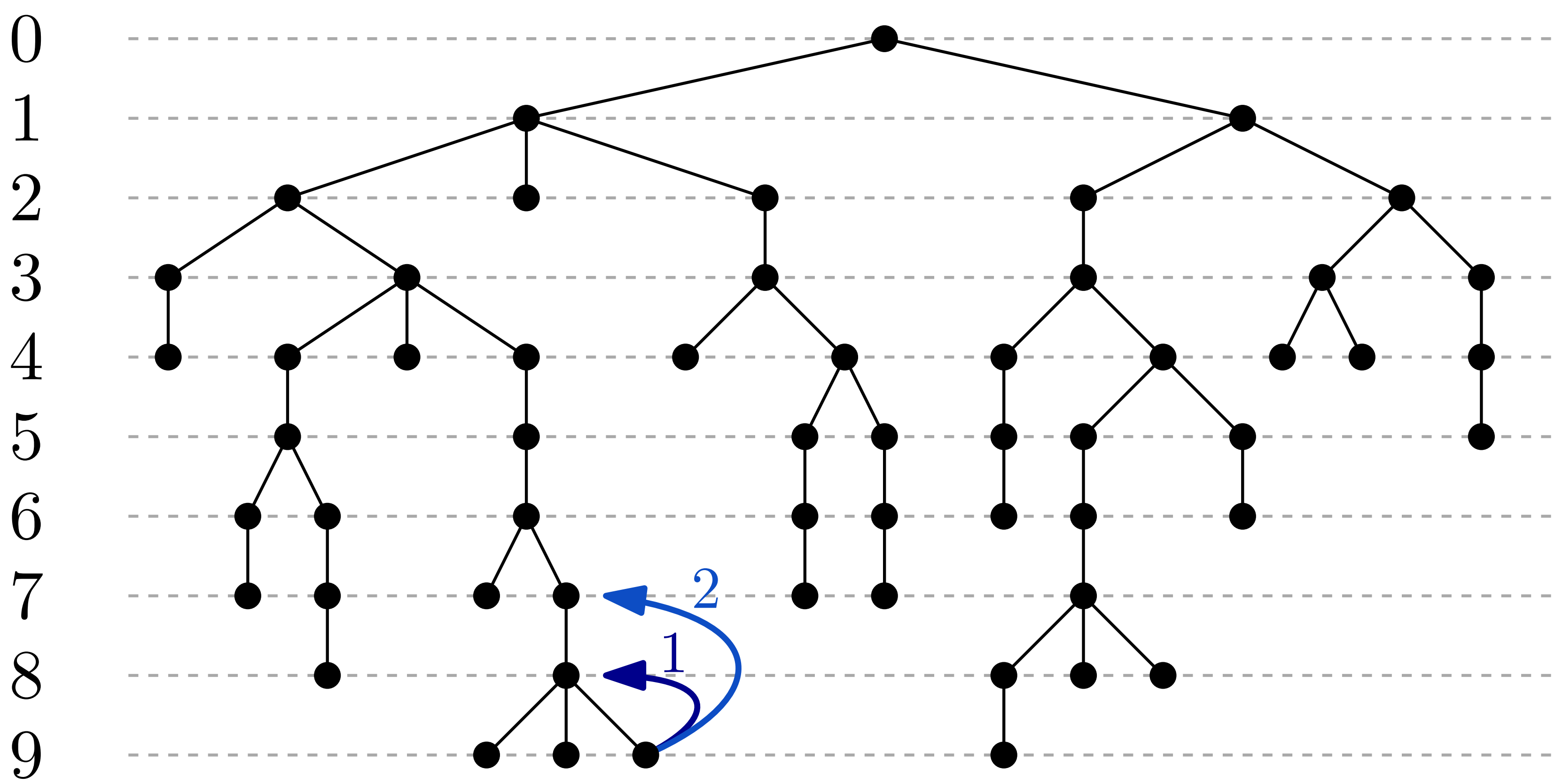
# Jump Pointers: Idea




For each vertex $v$ and $\ell = 0, 1, \ldots, \lfloor \log d_v \rfloor$, store:

$$\color{blue}{J(v,\ell)} = \mathsf{LA}(v, d_v - 2^\ell)$$

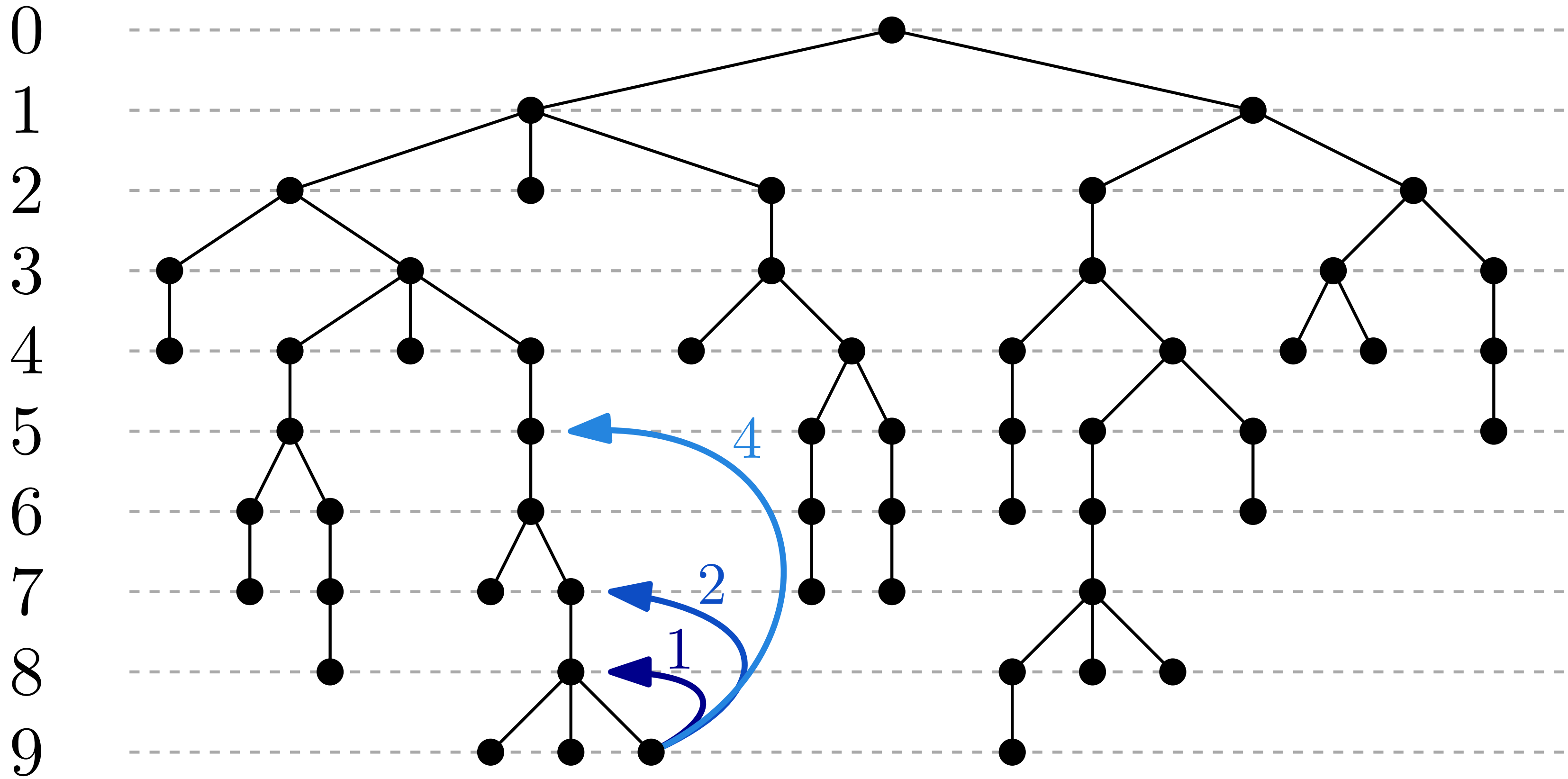
# Jump Pointers: Idea




For each vertex $v$ and $\ell = 0, 1, \ldots, \lfloor \log d_v \rfloor$, store:

$$J(v, \ell) = \mathsf{LA}(v, d_v - 2^\ell)$$

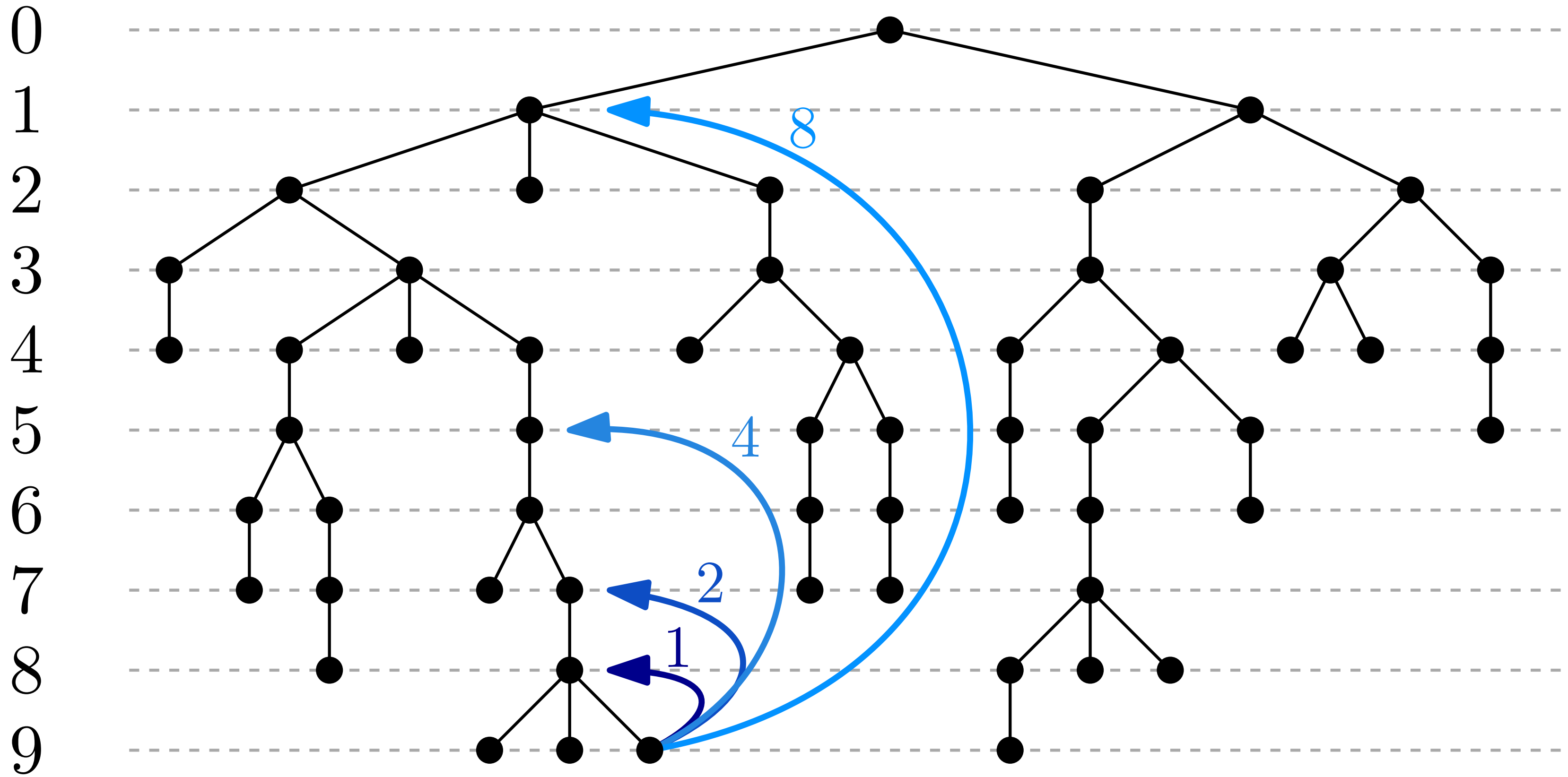
# Jump Pointers: Idea




For each vertex $v$ and $\ell = 0, 1, \ldots, \lfloor \log d_v \rfloor$, store:

$$J(v, \ell) = \mathsf{LA}(v, d_v - 2^\ell)$$

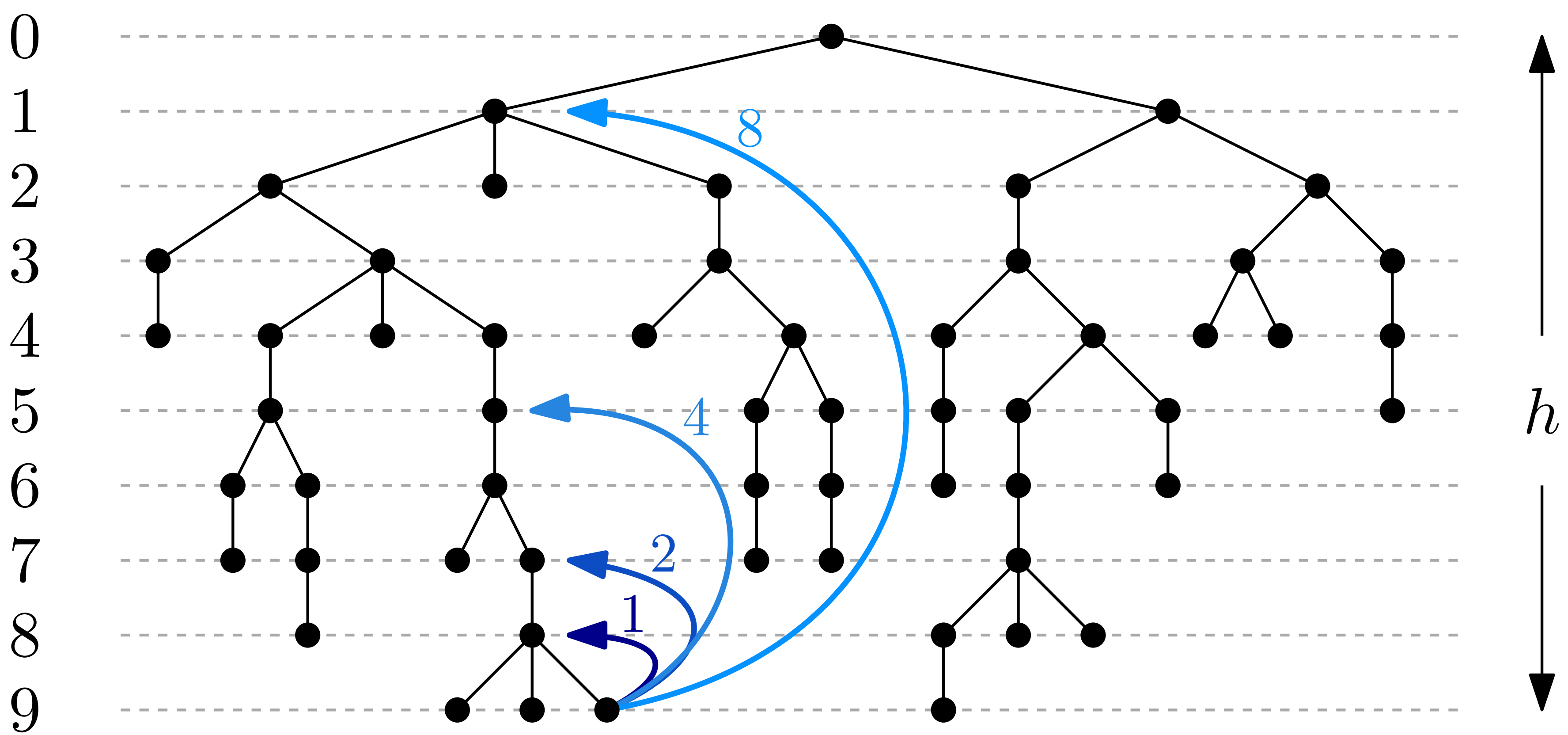
# Jump Pointers: Idea




For each vertex $v$ and $\ell = 0, 1, \ldots, \lfloor \log d_v \rfloor$, store:

$$J(v, \ell) = \mathsf{LA}(v, d_v - 2^\ell)$$

# Jump Pointers: Idea

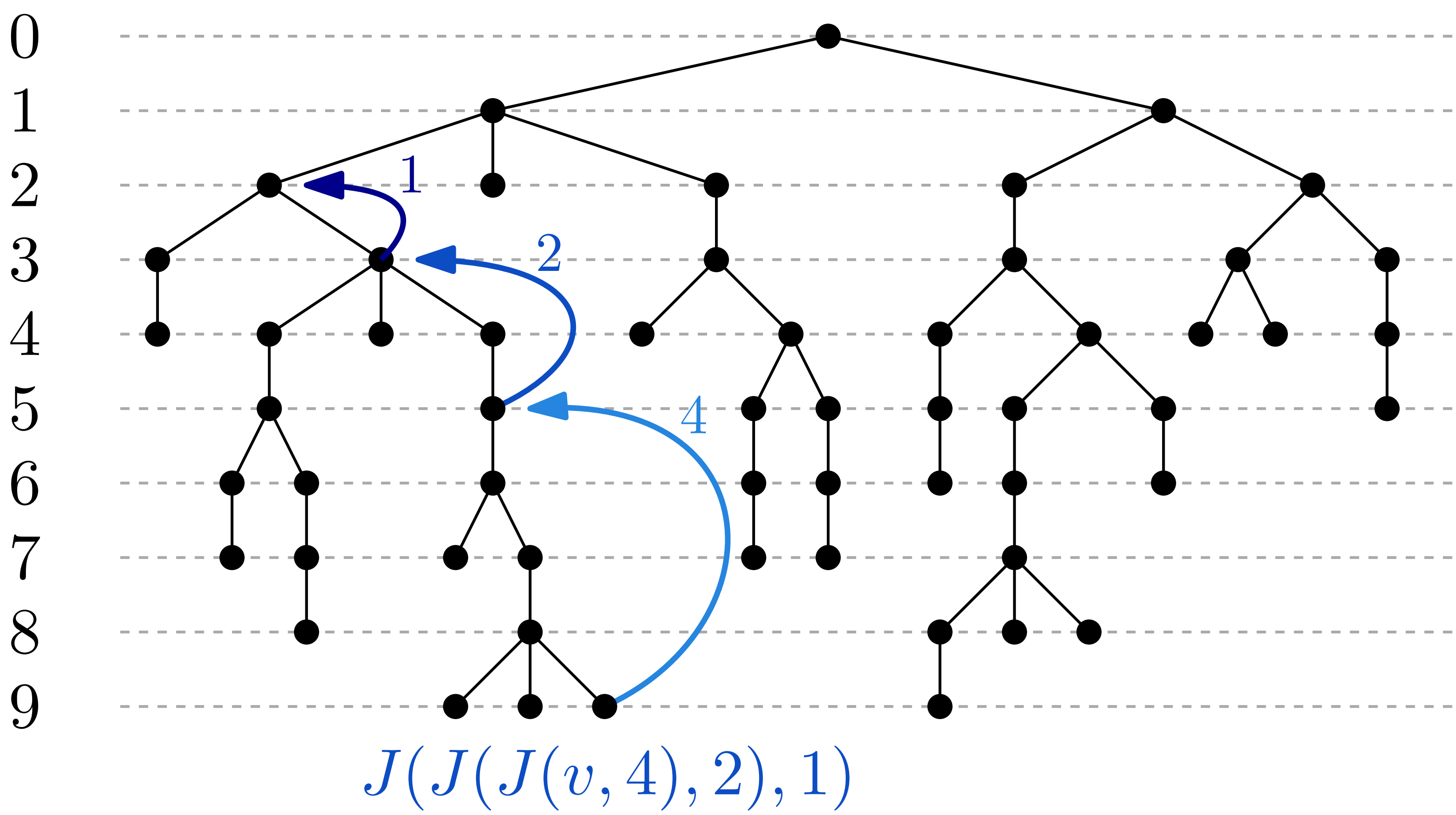



For each vertex $v$ and $\ell = 0, 1, \ldots, \lfloor \log d_v \rfloor$, store:

$$J(v, \ell) = \mathsf{LA}(v, d_v - 2^\ell)$$

Total size: $O(n \log h) = O(n \log n)$

# Jump Pointers: Query




$$0 < d_v - d = 2^{\ell_k} + 2^{\ell_{k-1}} + \cdots + 2^{\ell_1} \qquad \ell_{i+1} > \ell_i$$

$$\mathsf{LA}(v,d) = J(\ldots J(J(v,\ell_k),\ell_{k-1}),\ldots,\ell_1)$$

Number of accessed pointers: $O(\log h) = O(\log n)$

# Jump Pointers: Construction

**With a DFS visit of $T$:**

- Maintain a stack $S$ that stores all the ancestors of the current vertex $v$ of the visit
- $S$ can be updated in $O(1)$ per traversed edge
- When vertex $v$ is visited, its ancestor at depth $d$ in $T$ is the $(d_v - d)$-th vertex from the top of the stack

# Jump Pointers: Construction

**With a DFS visit of $T$:**

- Maintain a stack $S$ that stores all the ancestors of the current vertex $v$ of the visit
- $S$ can be updated in $O(1)$ per traversed edge
- When vertex $v$ is visited, its ancestor at depth $d$ in $T$ is the $(d_v - d)$-th vertex from the top of the stack

**Or using dynamic programming:**

$$J(v, \ell) = \begin{cases} \text{parent}(v) & \text{if } \ell = 0 \\ J(J(v, \ell - 1), \ell - 1) & \text{if } \ell > 0 \end{cases}$$

# Jump Pointers: Construction

**With a DFS visit of $T$:**

- Maintain a stack $S$ that stores all the ancestors of the current vertex $v$ of the visit
- $S$ can be updated in $O(1)$ per traversed edge
- When vertex $v$ is visited, its ancestor at depth $d$ in $T$ is the $(d_v - d)$-th vertex from the top of the stack

**Or using dynamic programming:**

$$J(v, \ell) = \begin{cases} \text{parent}(v) & \text{if } \ell = 0 \\ J(J(v, \ell - 1), \ell - 1) & \text{if } \ell > 0 \end{cases}$$

**Time complexity:** $O(n + n \log h) = O(n \log n)$

# Solutions so far

| Size | Preprocessing Time | Query Time | Notes |
|---|---|---|---|
| $O(n)$ | – | $O(n)$ | |
| $O(n^2)$ | $O(n^3)$ | $O(1)$ | |
| $O(n^2)$ | $O(n^2)$ | $O(1)$ | |

# Solutions so far

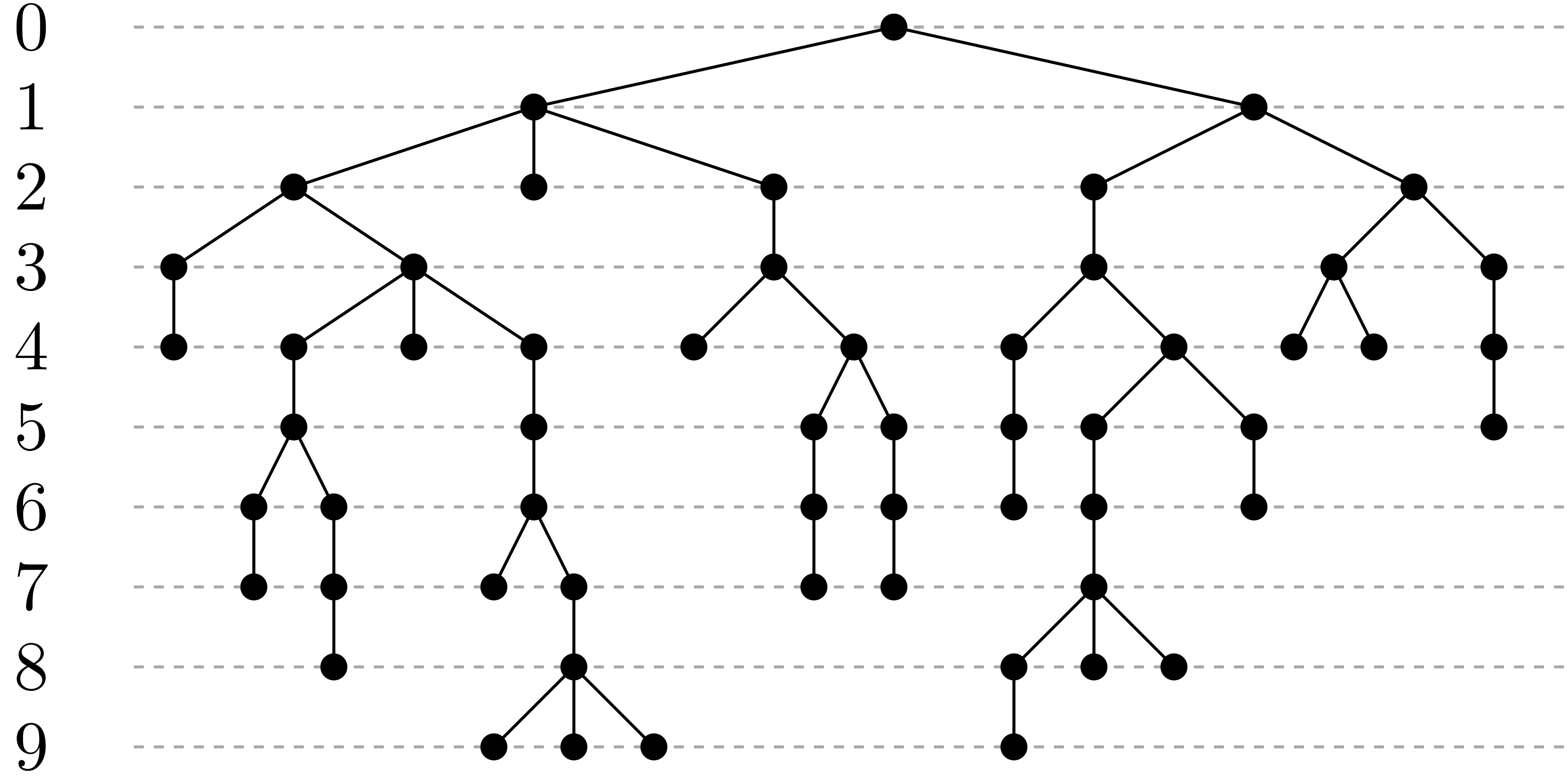
| Size | Preprocessing Time | Query Time | Notes |
| --- | --- | --- | --- |
| $O(n)$ | – | $O(n)$ | |
| $O(n^2)$ | $O(n^3)$ | $O(1)$ | |
| $O(n^2)$ | $O(n^2)$ | $O(1)$ | |
| $O(n \log n)$ | $O(n \log n)$ | $O(\log n)$ | Jump Pointers |

# Solutions so far

| Size | Preprocessing Time | Query Time | Notes |
|---|---|---|---|
| $O(n)$ | – | $O(n)$ | |
| $O(n^2)$ | $O(n^3)$ | $O(1)$ | |
| $O(n^2)$ | $O(n^2)$ | $O(1)$ | |
| $O(n \underline{\log n})$ | $O(n \underline{\log n})$ | $O(\underline{\log n})$ | Jump Pointers |

We want to get rid of the $\log n$ factors!

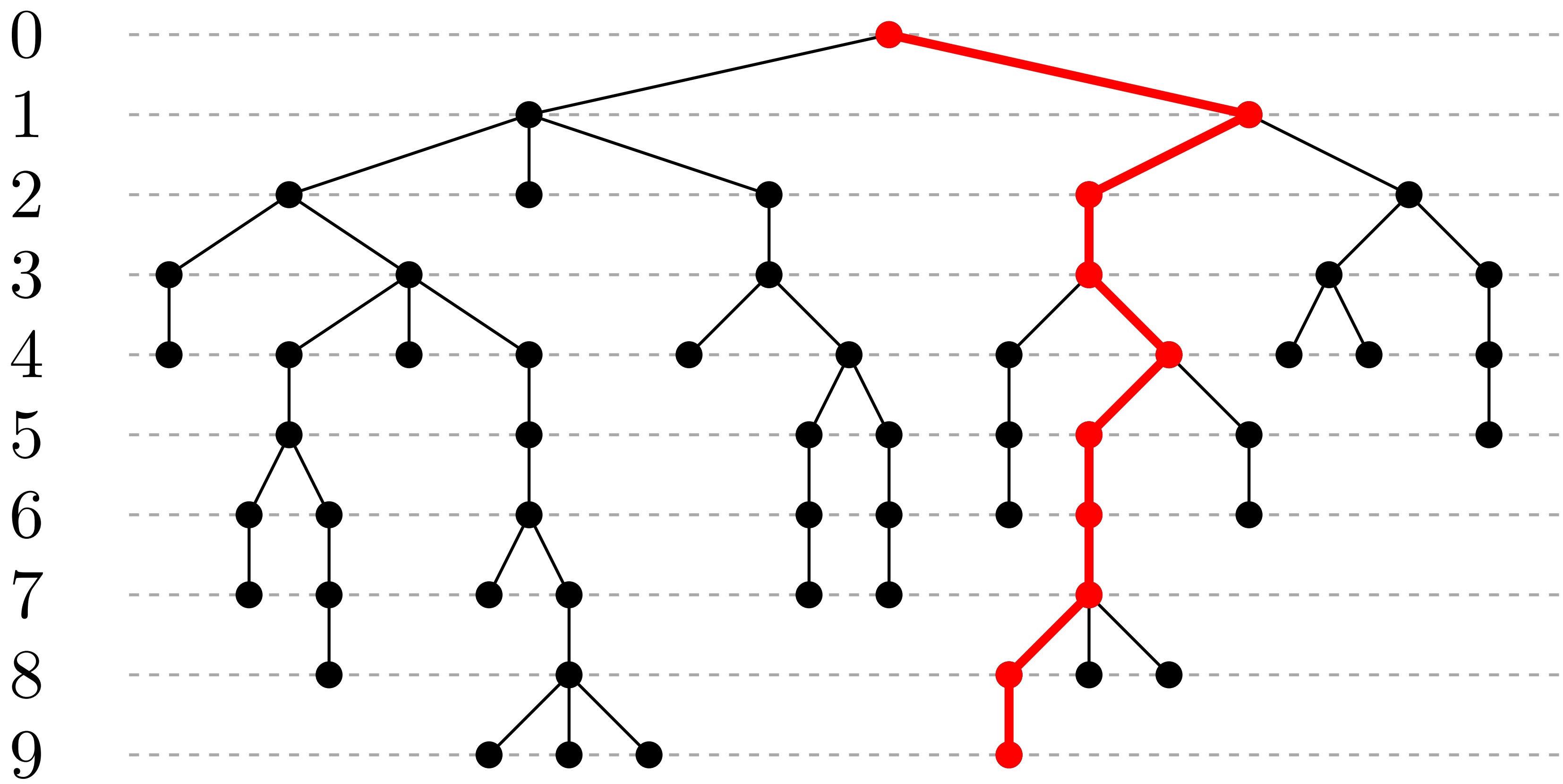
# Long Path Decomposition




Partitions $T$ into a collection of paths $\mathcal{D}$. Recursively defined:

- Select one of the longest root-to-leaf paths $P$ in $T$
- Select paths recursively from each the tree of the forest $T \setminus P$

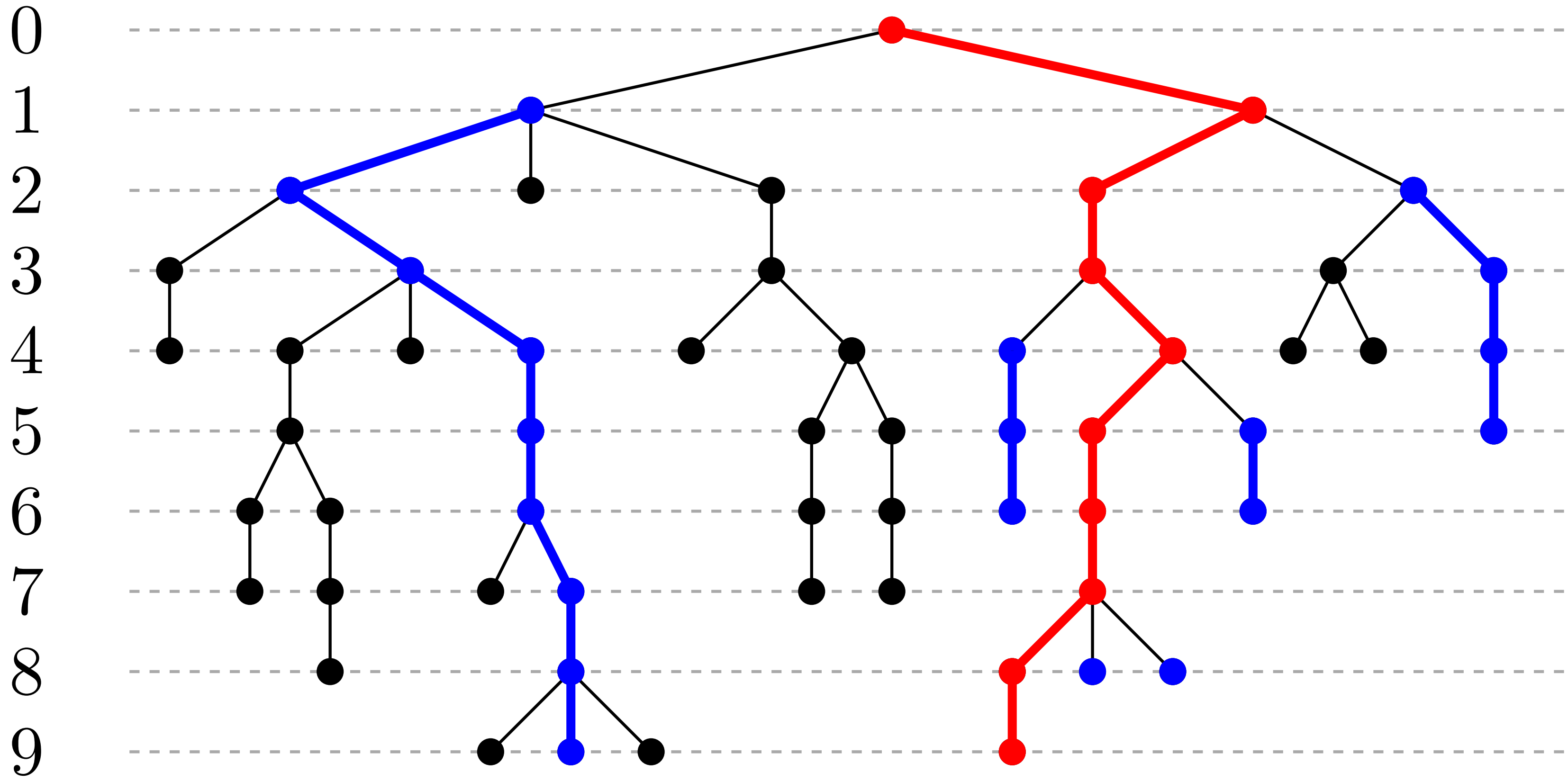
# Long Path Decomposition




Partitions $T$ into a collection of paths $\mathcal{D}$. Recursively defined:

- Select one of the longest root-to-leaf paths $P$ in $T$
- Select paths recursively from each the tree of the forest $T \setminus P$

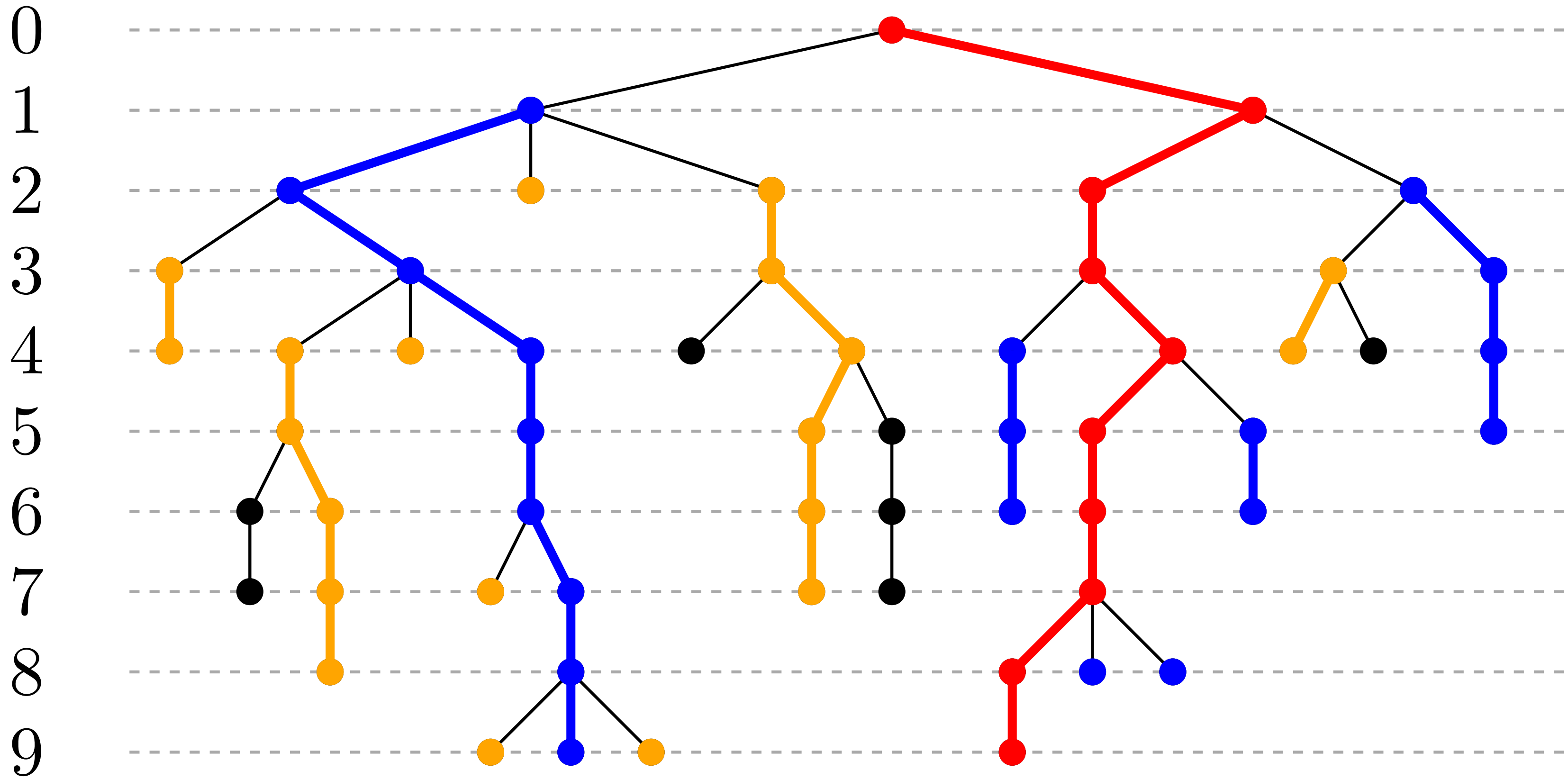
# Long Path Decomposition




Partitions $T$ into a collection of paths $\mathcal{D}$. Recursively defined:

- Select one of the longest root-to-leaf paths $P$ in $T$
- Select paths recursively from each the tree of the forest $T \setminus P$

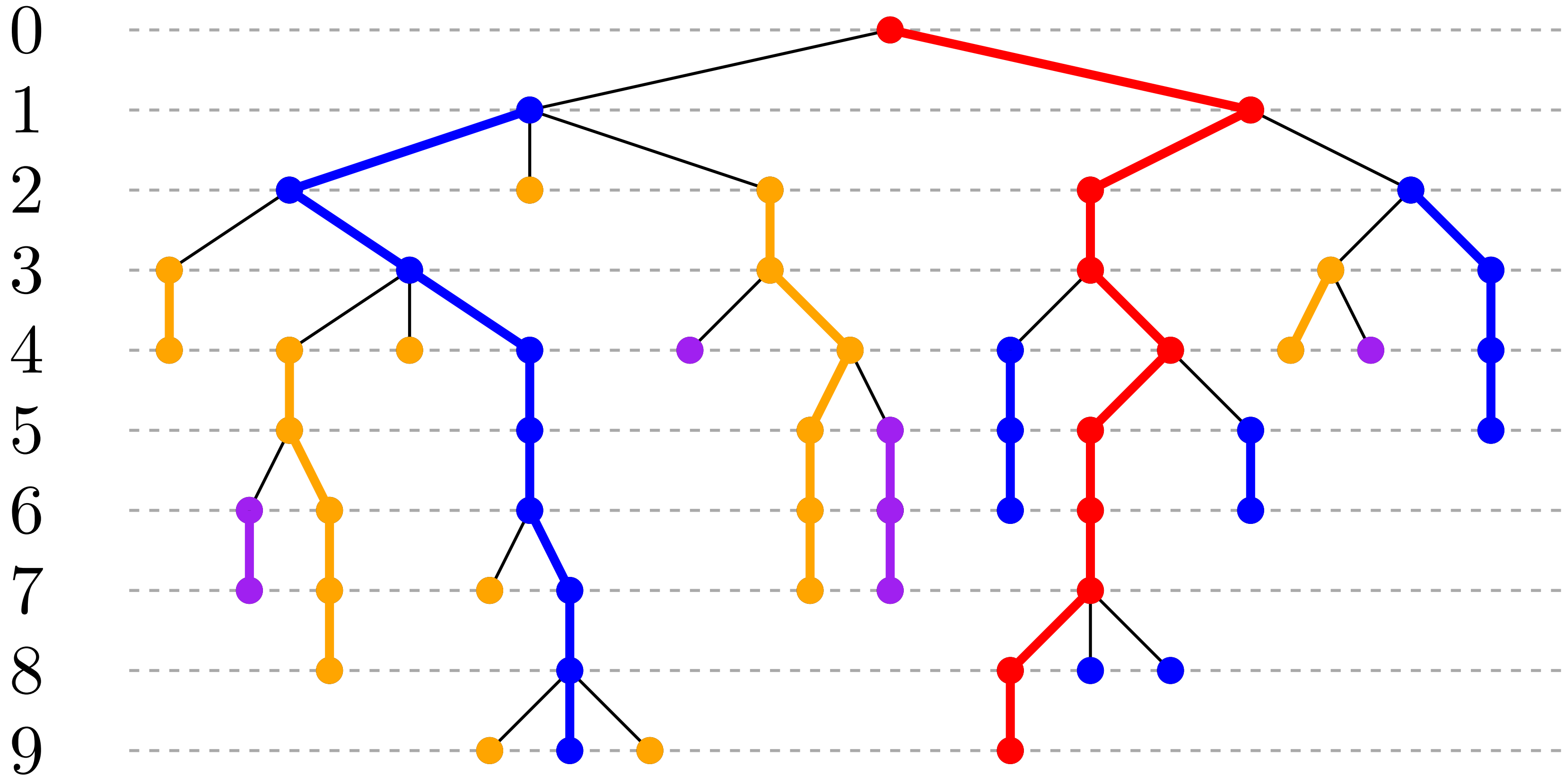
# Long Path Decomposition




Partitions $T$ into a collection of paths $\mathcal{D}$. Recursively defined:

- Select one of the longest root-to-leaf paths $P$ in $T$
- Select paths recursively from each the tree of the forest $T \setminus P$

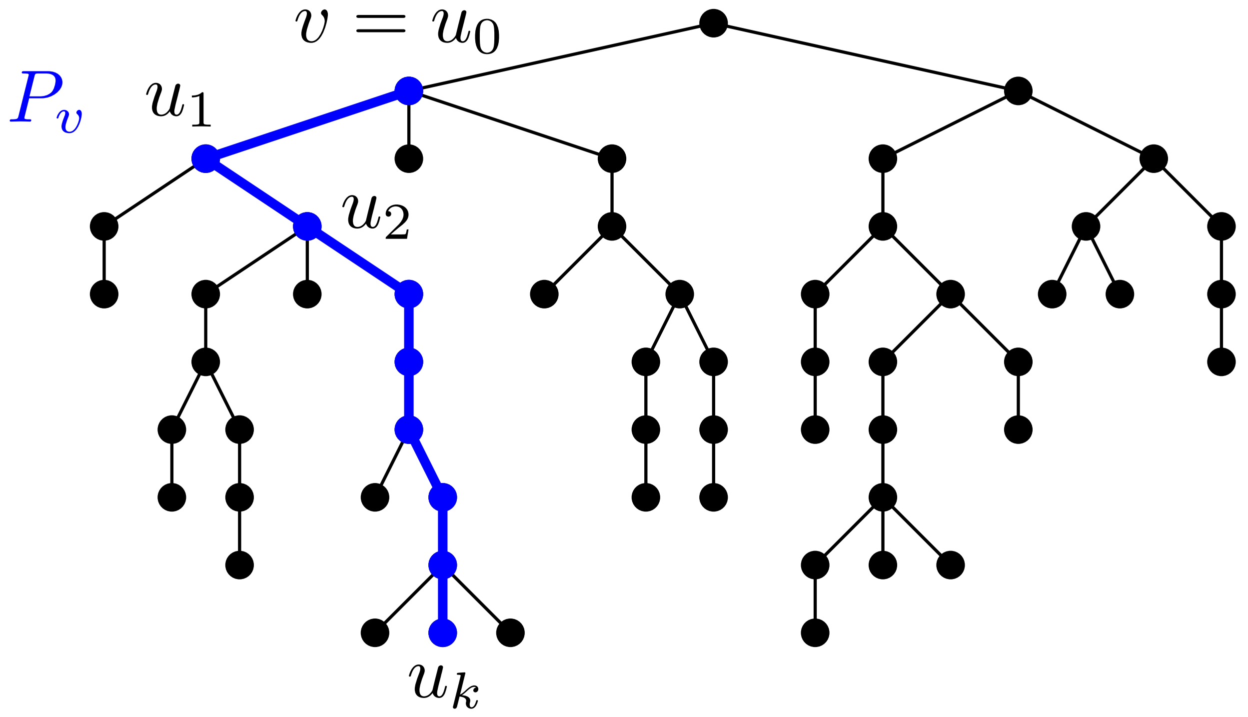
# Long Path Decomposition




Partitions $T$ into a collection of paths $\mathcal{D}$. Recursively defined:

- Select one of the longest root-to-leaf paths $P$ in $T$
- Select paths recursively from each the tree of the forest $T \setminus P$

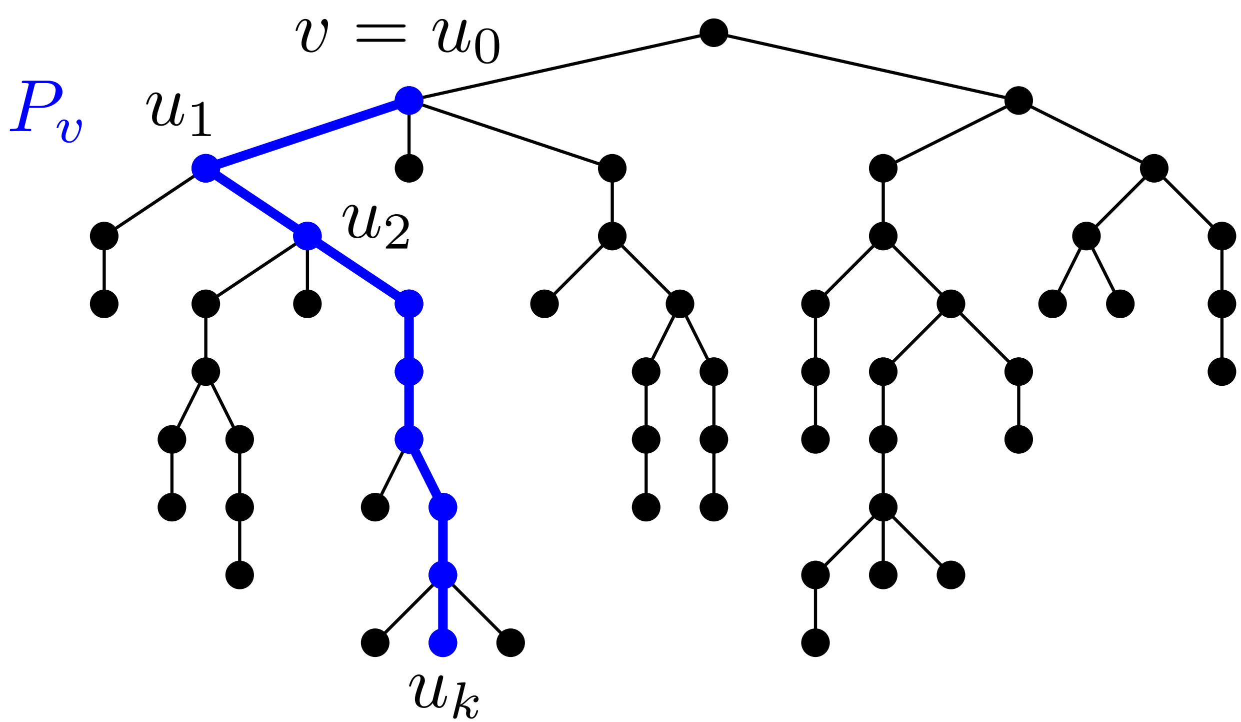
# Long Path Decomposition




For each path $P_v = \langle v = u_0, \ldots, u_k \rangle \in \mathcal{D}$:

- Store an array $A_v$ of length $k + 1$ where $A_v[i]$, $i = 0, \ldots, k$, contains (a reference to) $u_i$
- Each $u_i$ stores a reference $\tau(u_i)$ to $v$.

# Long Path Decomposition

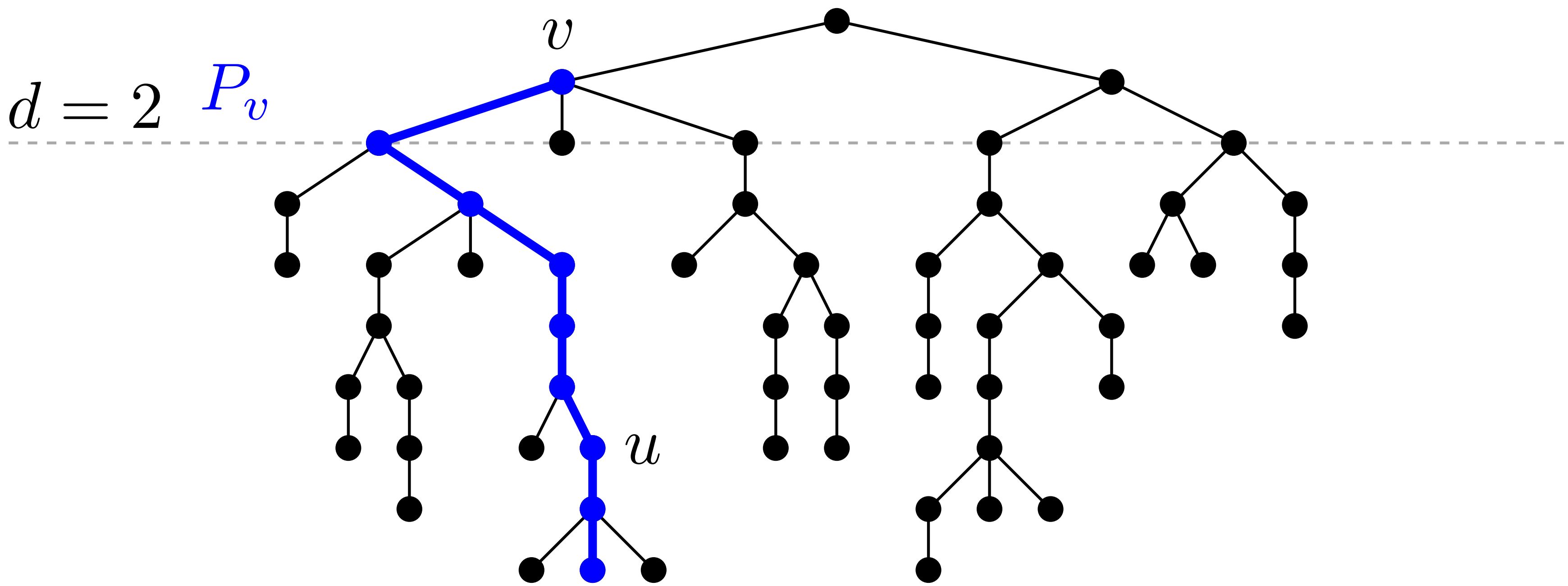



For each path $P_v = \langle v = u_0, \ldots, u_k \rangle \in \mathcal{D}$:

- Store an array $A_v$ of length $k+1$ where $A_v[i]$, $i = 0, \ldots, k$, contains (a reference to) $u_i$
- Each $u_i$ stores a reference $\tau(u_i)$ to $v$.

Total space: $\sum_{P_v \in \mathcal{D}} O(1 + |P_v|) = O(n)$

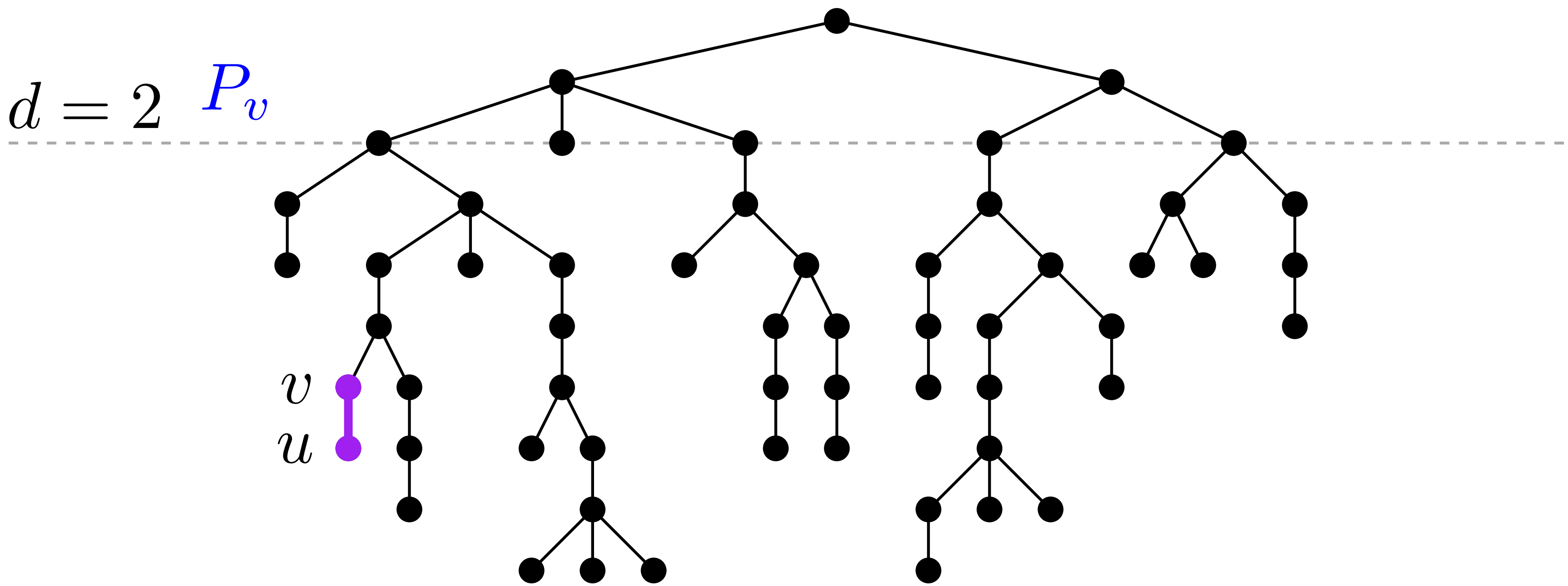
# Long Path Decomposition




To report LA$(u, d)$:

- Let $v = \tau(u)$
- If $d \geq d_v$: return $A_v[d - d_v]$.

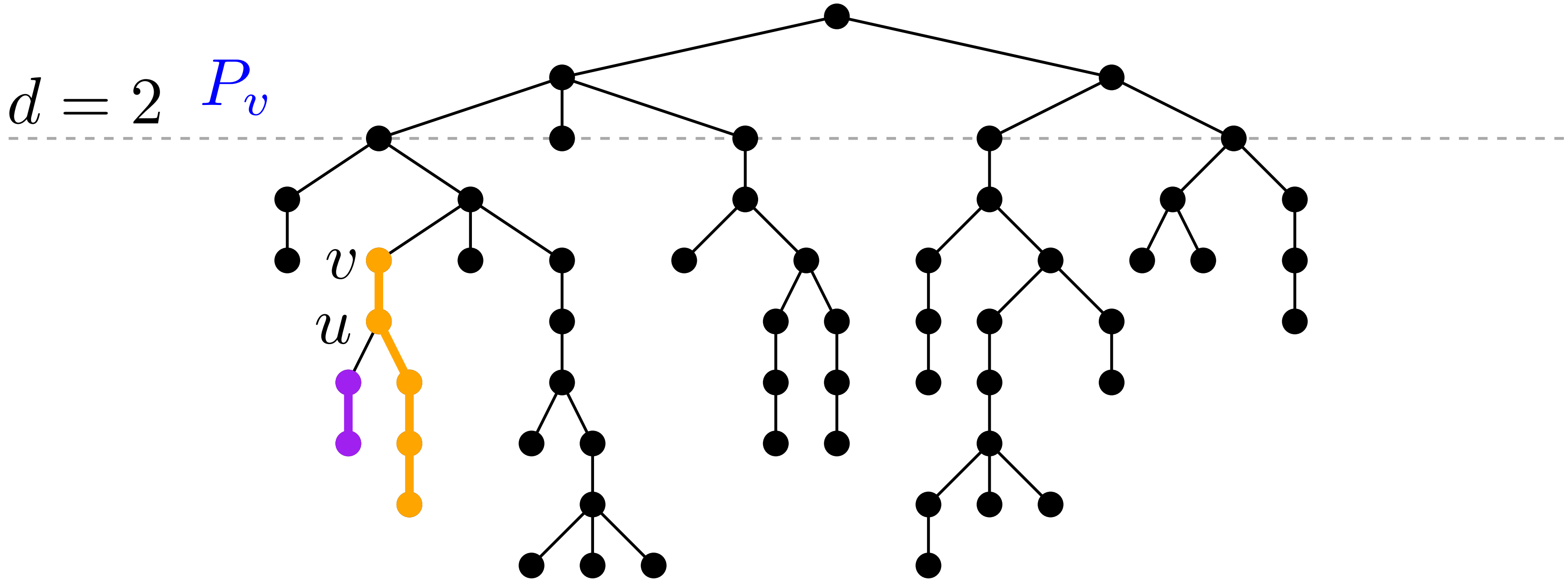
# Long Path Decomposition




To report $\mathrm{LA}(u, d)$:

- Let $v = \tau(u)$
- If $d \geq d_v$: return $A_v[d - d_v]$.
- If $d < d_v$: return $\mathrm{LA}(\text{parent}(v), d)$. (recursively)

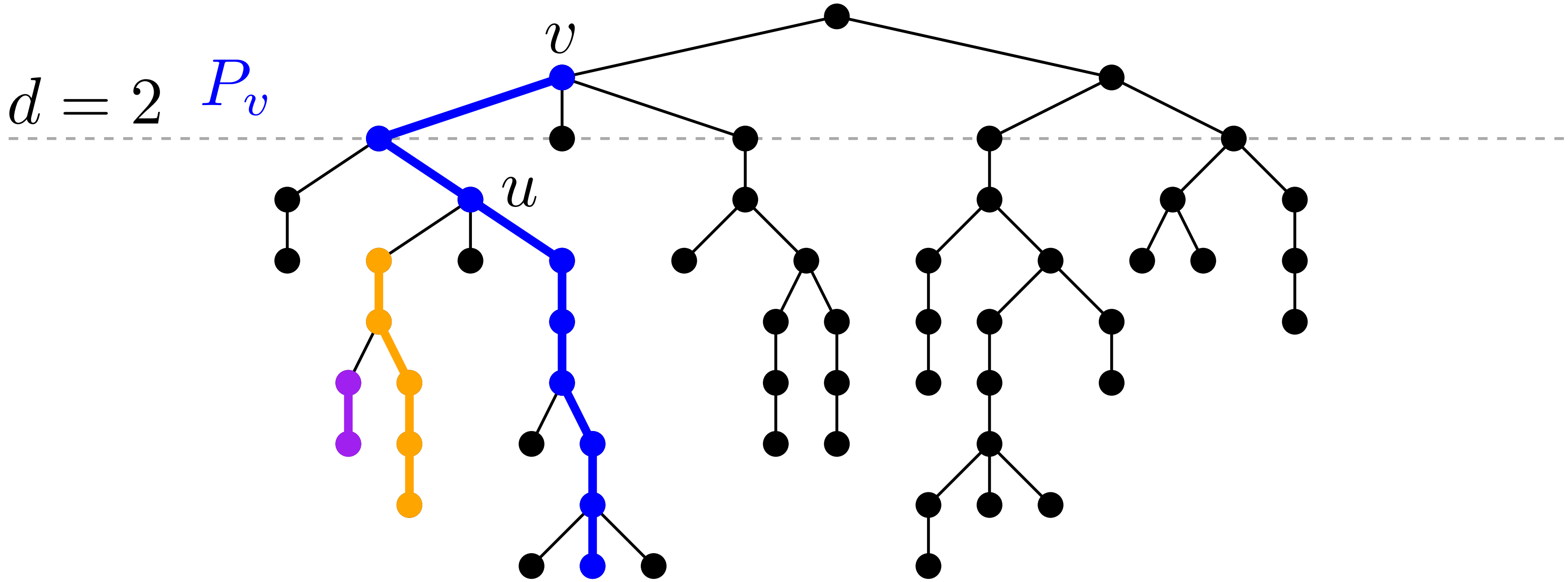
# Long Path Decomposition




To report $\mathsf{LA}(u, d)$:

- Let $v = \tau(u)$
- If $d \geq d_v$: return $A_v[d - d_v]$.
- If $d < d_v$: return $\mathsf{LA}(\mathsf{parent}(v), d)$. (recursively)

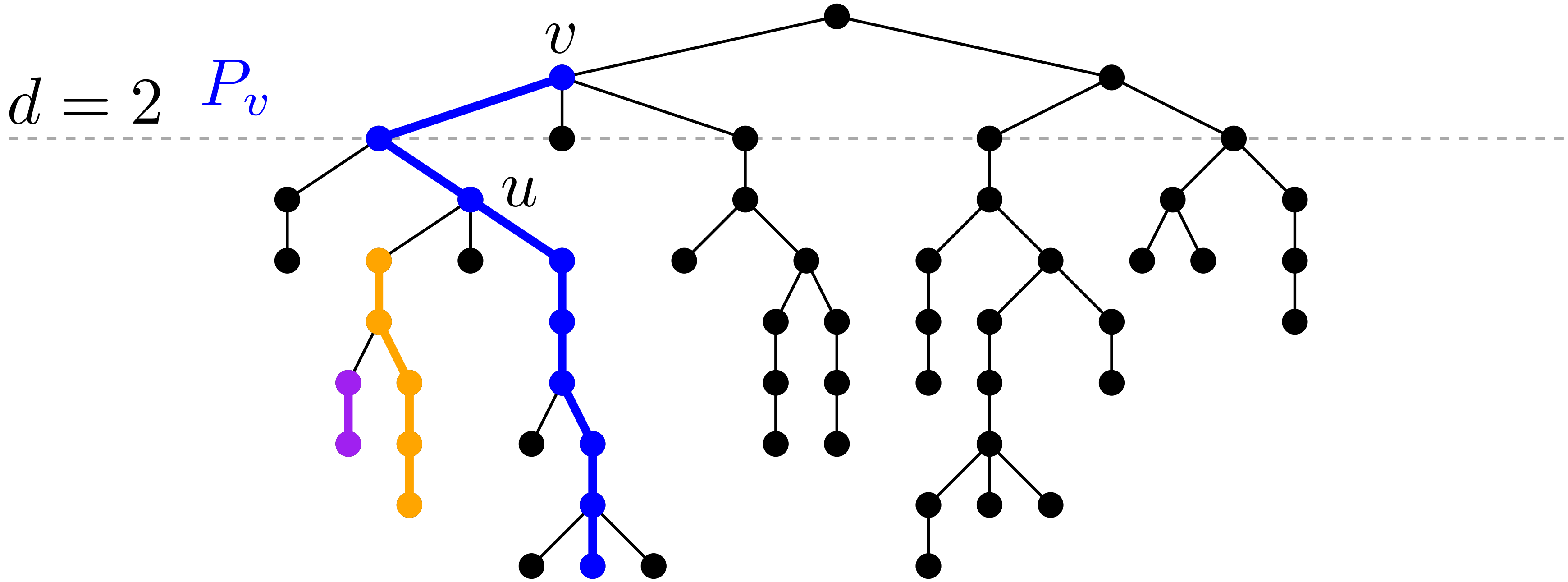
# Long Path Decomposition




To report LA$(u, d)$:

- Let $v = \tau(u)$
- If $d \geq d_v$: return $A_v[d - d_v]$.
- If $d < d_v$: return LA(parent$(v), d)$. (recursively)

# Long Path Decomposition




To report LA$(u, d)$:

- Let $v = \tau(u)$
- If $d \geq d_v$: return $A_v[d - d_v]$.
- If $d < d_v$: return LA$($parent$(v), d)$. (recursively)

Time: $O($#recursive calls$) = O($#paths in $\mathcal{D}$ from $v$ to the root$)$.

# Long Path Decomposition

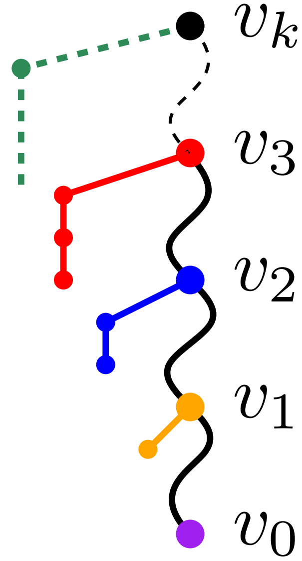
**Claim:** The number of distinct paths in $\mathcal{D}$ encountered in the path $P$ from $v$ to the root in $T$ is $O(\sqrt{n})$.

# Long Path Decomposition

**Claim:** The number of distinct paths in $\mathcal{D}$ encountered in the path $P$ from $v$ to the root in $T$ is $O(\sqrt{n})$.

**Proof:** Let $v = v_0, v_1, \ldots, v_k$ be the vertices at which a new path of $\mathcal{D}$ is encountered while traversing $P$.

- Let $P_i$ be the path of $\mathcal{D}$ that contains $v_i$.

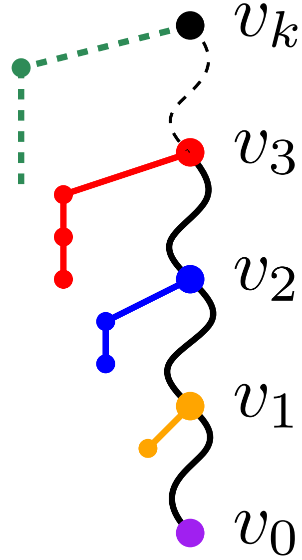


# Long Path Decomposition

**Claim:** The number of distinct paths in $\mathcal{D}$ encountered in the path $P$ from $v$ to the root in $T$ is $O(\sqrt{n})$.

**Proof:** Let $v = v_0, v_1, \ldots, v_k$ be the vertices at which a new path of $\mathcal{D}$ is encountered while traversing $P$.

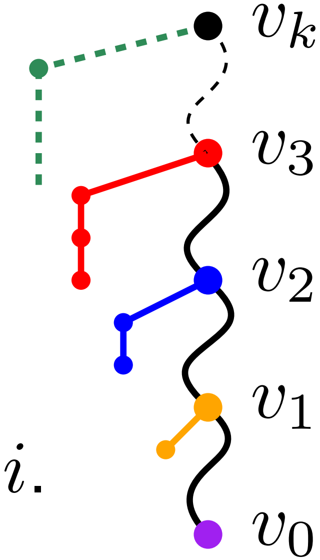
- Let $P_i$ be the path of $\mathcal{D}$ that contains $v_i$.
- Let $h(v)$ be the height $v$ in $T$ (i.e., the length of the longest path from $v$ to a leaf of $T$).



# Long Path Decomposition

**Claim:** The number of distinct paths in $\mathcal{D}$ encountered in the path $P$ from $v$ to the root in $T$ is $O(\sqrt{n})$.

**Proof:** Let $v = v_0, v_1, \ldots, v_k$ be the vertices at which a new path of $\mathcal{D}$ is encountered while traversing $P$.

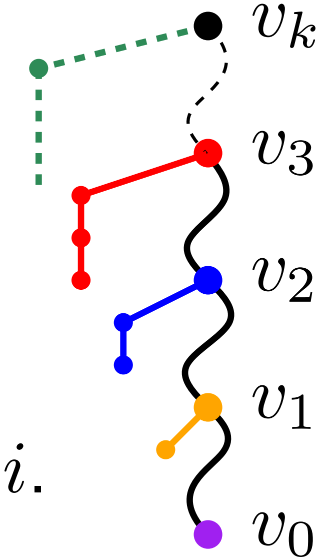



- Let $P_i$ be the path of $\mathcal{D}$ that contains $v_i$.
- Let $h(v)$ be the height $v$ in $T$ (i.e., the length of the longest path from $v$ to a leaf of $T$).
- By the long-path decomposition, $|P_i| \geq h(v_i) \geq i$.

# Long Path Decomposition

**Claim:** The number of distinct paths in $\mathcal{D}$ encountered in the path $P$ from $v$ to the root in $T$ is $O(\sqrt{n})$.

**Proof:** Let $v = v_0, v_1, \ldots, v_k$ be the vertices at which a new path of $\mathcal{D}$ is encountered while traversing $P$.




- Let $P_i$ be the path of $\mathcal{D}$ that contains $v_i$.
- Let $h(v)$ be the height $v$ in $T$ (i.e., the length of the longest path from $v$ to a leaf of $T$).
- By the long-path decomposition, $|P_i| \geq h(v_i) \geq i$.

$$n \geq \left| \bigcup_{i=1}^{k} P_i \right| \geq \sum_{i=1}^{k} i \geq \frac{k^2}{2} \implies \sqrt{2n} \geq k.$$

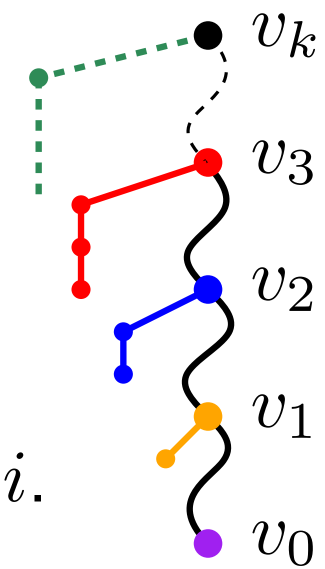
$\square$

# Long Path Decomposition

**Claim:** The number of distinct paths in $\mathcal{D}$ encountered in the path $P$ from $v$ to the root in $T$ is $O(\sqrt{n})$.

**Proof:** Let $v = v_0, v_1, \ldots, v_k$ be the vertices at which a new path of $\mathcal{D}$ is encountered while traversing $P$.

- Let $P_i$ be the path of $\mathcal{D}$ that contains $v_i$.
- Let $h(v)$ be the height $v$ in $T$ (i.e., the length of the longest path from $v$ to a leaf of $T$).
- By the long-path decomposition, $|P_i| \geq h(v_i) \geq i$.



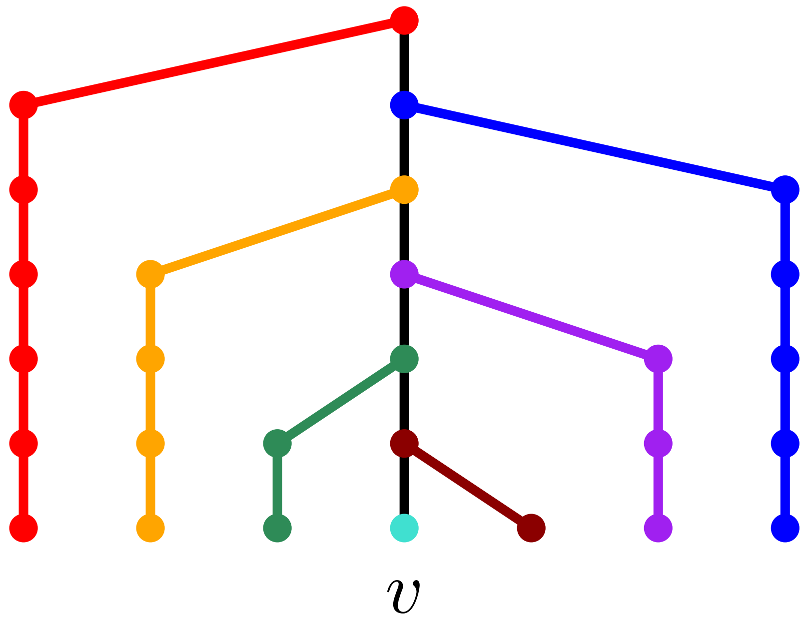

$$n \geq \left| \bigcup_{i=1}^{k} P_i \right| \geq \sum_{i=1}^{k} i \geq \frac{k^2}{2} \implies \sqrt{2n} \geq k.$$

$\square$

Time: $O(\sqrt{n})$

Is this tight?

# Long Path Decomposition




Time: $\Omega(\sqrt{n})$

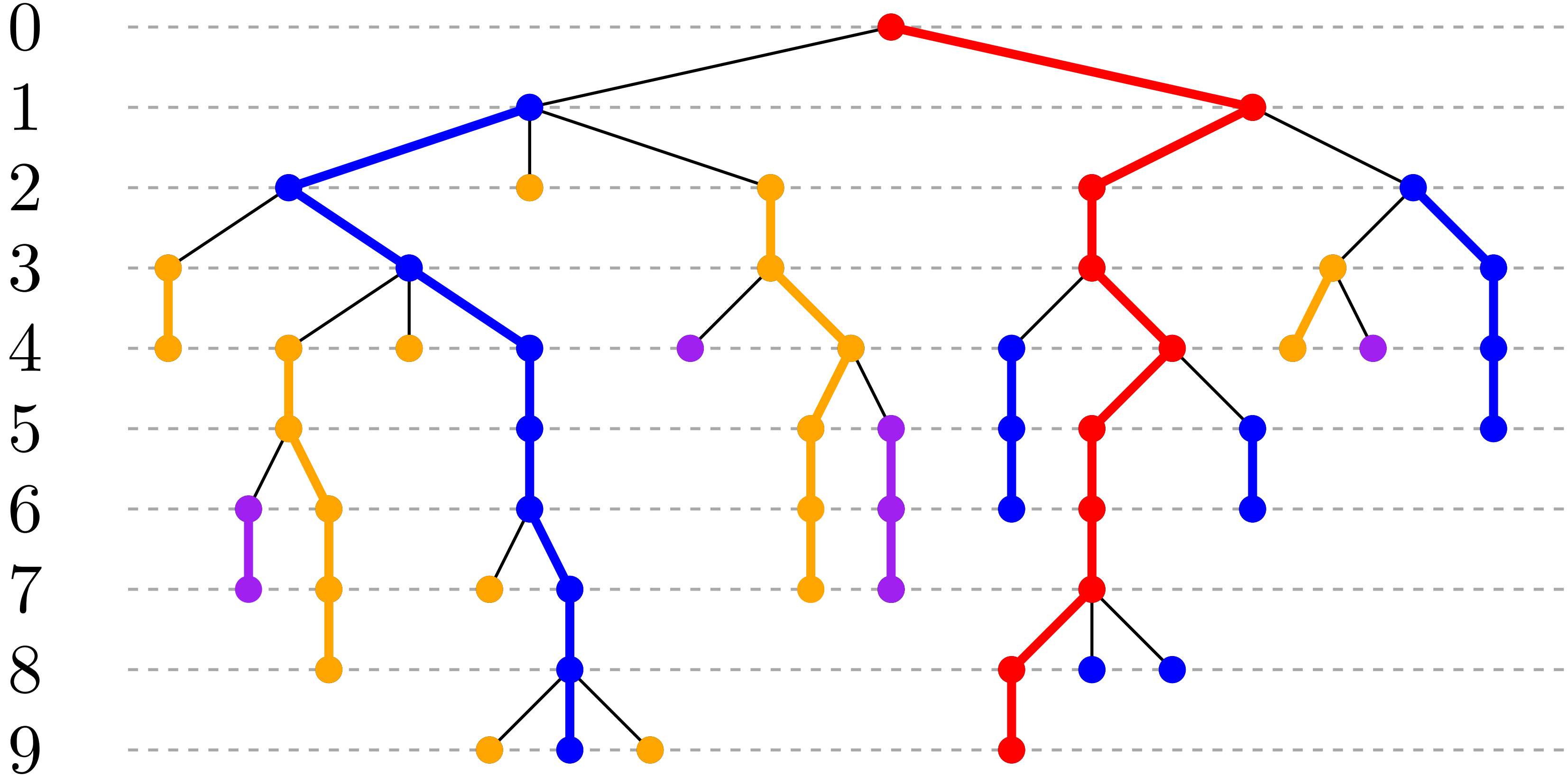
# Solutions so far

| Size | Preprocessing Time | Query Time | Notes |
|---|---|---|---|
| $O(n)$ | – | $O(n)$ | |
| $O(n^2)$ | $O(n^3)$ | $O(1)$ | |
| $O(n^2)$ | $O(n^2)$ | $O(1)$ | |
| $O(n \log n)$ | $O(n \log n)$ | $O(\log n)$ | Jump Pointers |

# Solutions so far

| Size | Preprocessing Time | Query Time | Notes |
|---|---|---|---|
| $O(n)$ | – | $O(n)$ | |
| $O(n^2)$ | $O(n^3)$ | $O(1)$ | |
| $O(n^2)$ | $O(n^2)$ | $O(1)$ | |
| $O(n \log n)$ | $O(n \log n)$ | $O(\log n)$ | Jump Pointers |
| $O(n)$ | $O(n)$ | $O(\sqrt{n})$ | Long Path Dec. |

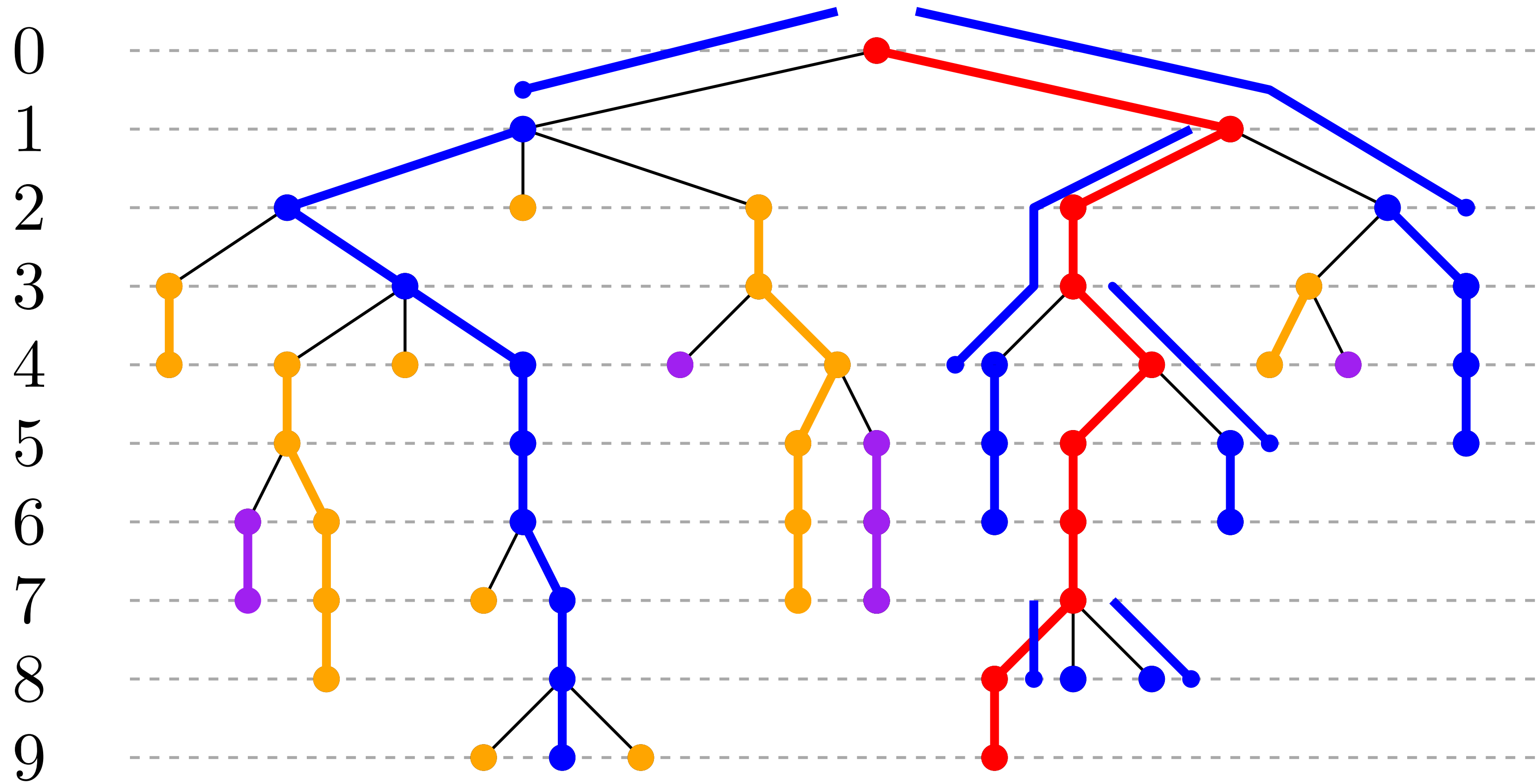
# Long Path Decomposition + Ladders




Let $\eta(P_v)$ be the number of vertices of the path $P_v \in \mathcal{D}$.

Extend each path $P_v \in \mathcal{D}$ into a *ladder* $L_v$ with $\eta(P_v)$ more vertices towards the root (if they exist).

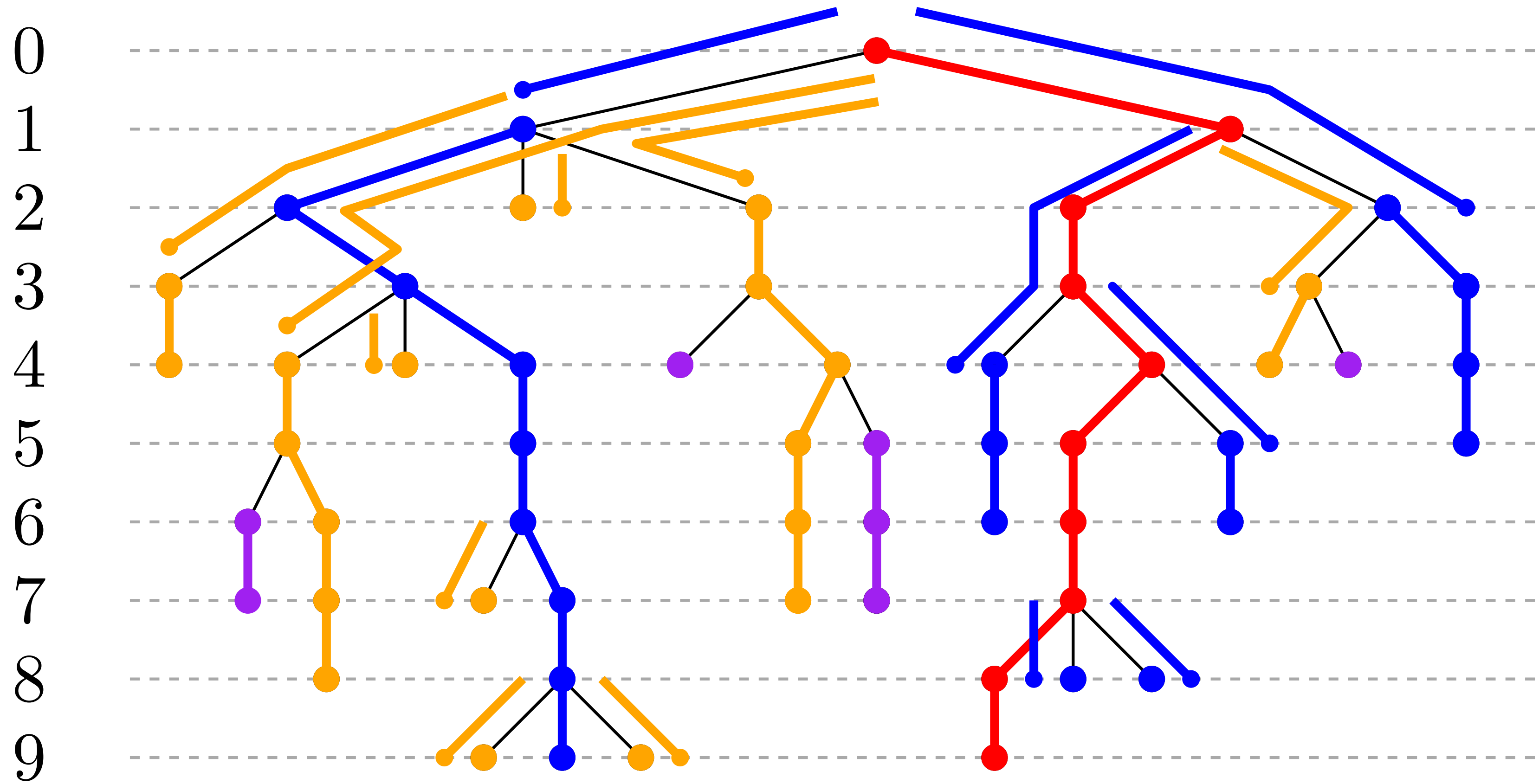
# Long Path Decomposition + Ladders




Let $\eta(P_v)$ be the number of vertices of the path $P_v \in \mathcal{D}$.

Extend each path $P_v \in \mathcal{D}$ into a *ladder* $L_v$ with $\eta(P_v)$ more vertices towards the root (if they exist).

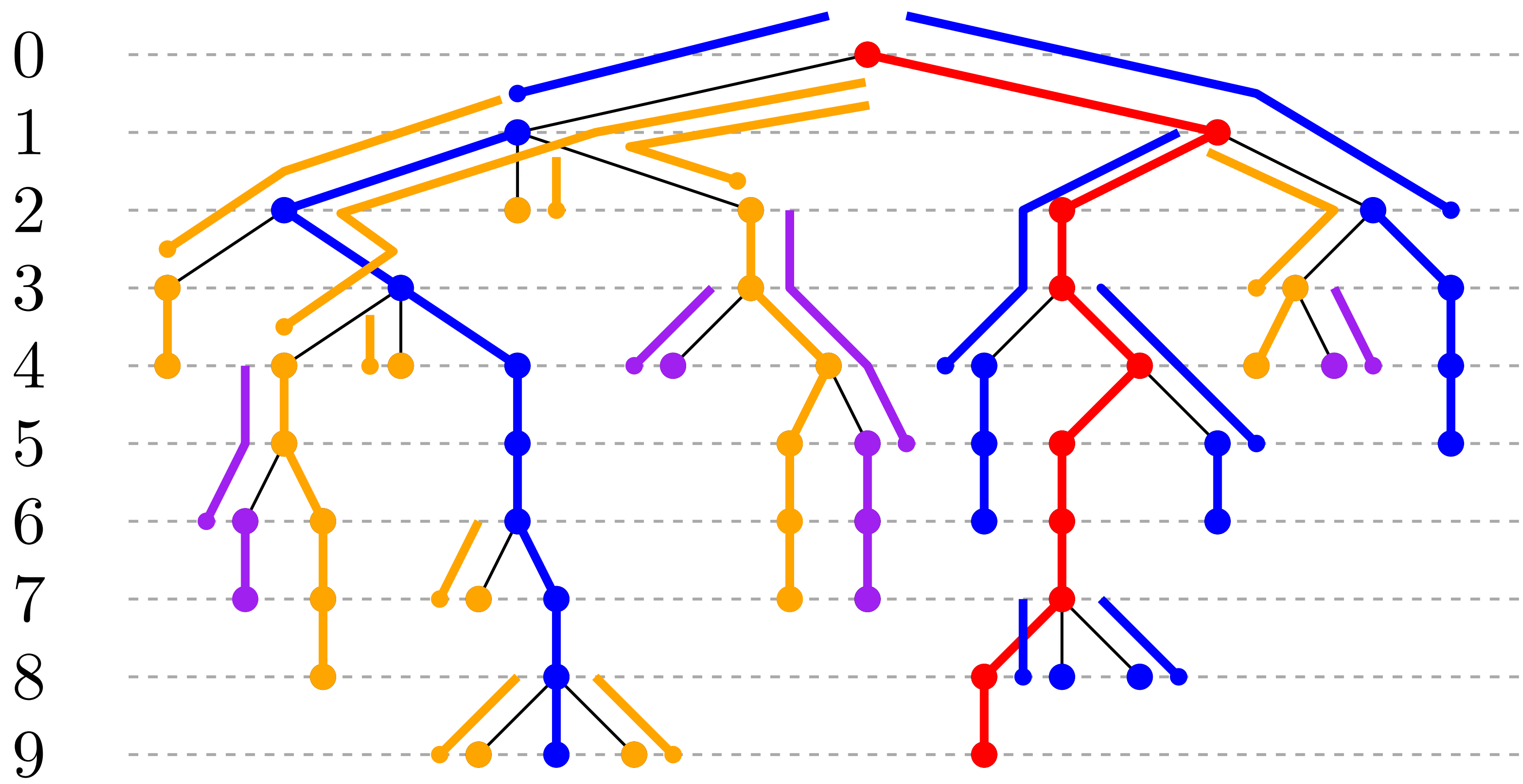
# Long Path Decomposition + Ladders




Let $\eta(P_v)$ be the number of vertices of the path $P_v \in \mathcal{D}$.

Extend each path $P_v \in \mathcal{D}$ into a *ladder* $L_v$ with $\eta(P_v)$ more vertices towards the root (if they exist).

# Long Path Decomposition + Ladders




Let $\eta(P_v)$ be the number of vertices of the path $P_v \in \mathcal{D}$.

Extend each path $P_v \in \mathcal{D}$ into a *ladder* $L_v$ with $\eta(P_v)$ more vertices towards the root (if they exist).

# Long Path Decomposition + Ladders

For each ladder $L_v = \langle v' = u_0, u_1, \ldots, v = u_j, \ldots, u_k \rangle$:

- Store, in $v$, an array $B_v$ of length $k + 1$ where $B_v[i]$ contains (a reference to) $u_i$
- Each $u_i$ with $i \geq j$ stores a reference $\tau(u_i)$ to $v$.

The length of $B_v$ is at most twice the length of $A_v \implies$ the total size is still $O(n)$.

# Long Path Decomposition + Ladders

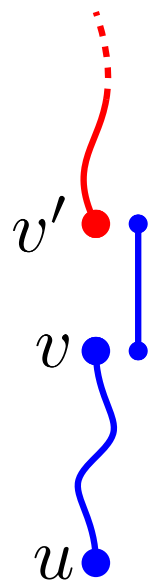
For each ladder $L_v = \langle v' = u_0, u_1, \ldots, v = u_j, \ldots, u_k \rangle$:

- Store, in $v$, an array $B_v$ of length $k+1$ where $B_v[i]$ contains (a reference to) $u_i$
- Each $u_i$ with $i \geq j$ stores a reference $\tau(u_i)$ to $v$.

The length of $B_v$ is at most twice the length of $A_v \implies$ the total size is still $O(n)$.

To report LA$(u, d)$:

- Let $v = \tau(u)$ and $v' = B_v[0]$.
- If $d \geq d_{v'}$: return $B_v[d - d_{v'}]$.
- If $d < d_{v'}$: return LA$(v', d)$. (recursively)

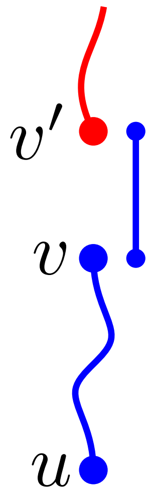



How many recursive calls?

# Long Path Decomposition + Ladders

How many recursive calls?

- If we recurse, $L_v$ cannot contain the root of $T$.




$$|L_v| = \eta(L_v) - 1 = 2\eta(P_v) - 1$$

- Since $u \in P_v$ we have:

$$h(v') \geq |L_v| = 2\eta(P_v) - 1 \geq 2(1 + h(u)) - 1 \geq 2h(u) + 1$$

- The height of the queried vertex doubles at every iteration $\implies O(\log n)$ iterations.

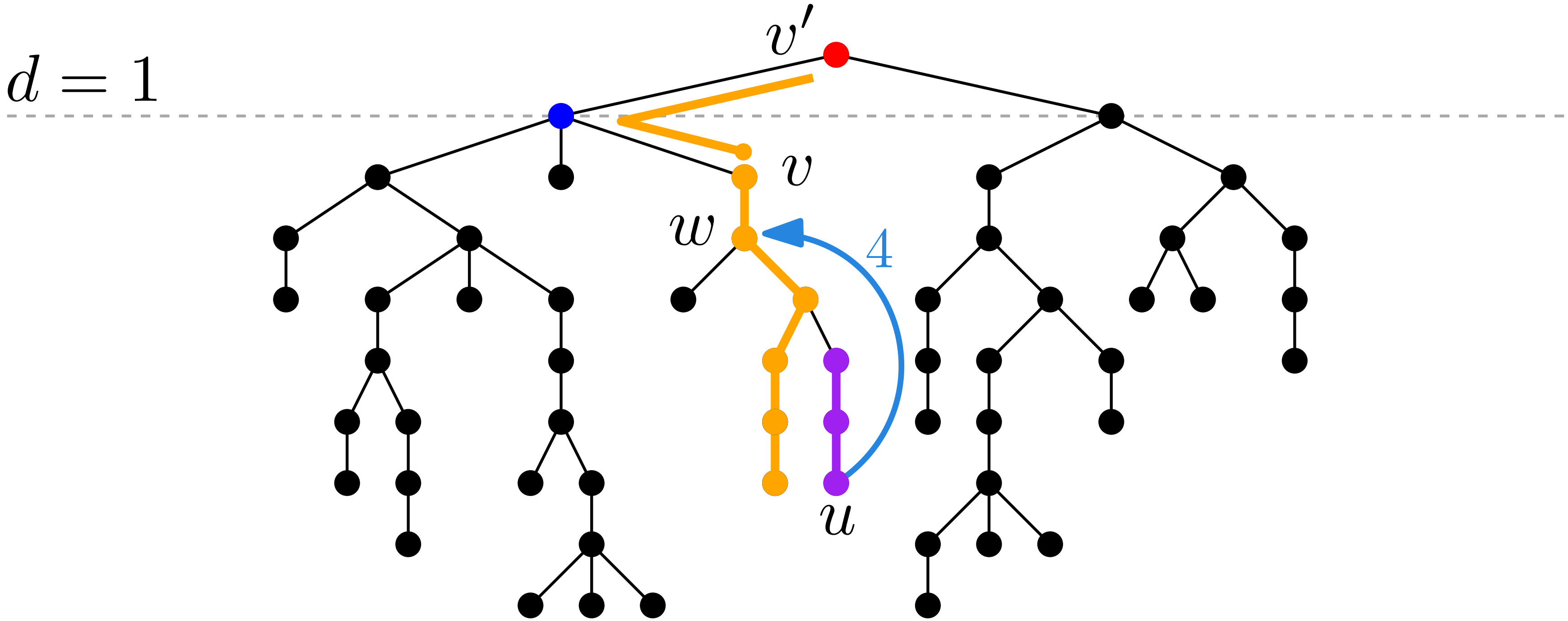
# Solutions so far

| Size | Preprocessing Time | Query Time | Notes |
|---|---|---|---|
| $O(n)$ | – | $O(n)$ | |
| $O(n^2)$ | $O(n^3)$ | $O(1)$ | |
| $O(n^2)$ | $O(n^2)$ | $O(1)$ | |
| $O(n \log n)$ | $O(n \log n)$ | $O(\log n)$ | Jump Pointers |
| $O(n)$ | $O(n)$ | $O(\sqrt{n})$ | Long Path Dec. |

# Solutions so far

| Size | Preprocessing Time | Query Time | Notes |
|---|---|---|---|
| $O(n)$ | – | $O(n)$ | |
| $O(n^2)$ | $O(n^3)$ | $O(1)$ | |
| $O(n^2)$ | $O(n^2)$ | $O(1)$ | |
| $O(n \log n)$ | $O(n \log n)$ | $O(\log n)$ | Jump Pointers |
| $O(n)$ | $O(n)$ | $O(\sqrt{n})$ | Long Path Dec. |
| $O(n)$ | $O(n)$ | $O(\log n)$ | $+$ Ladders |

# Long Path Dec. + Ladders + Jump Pointers




$$0 < d_u - d = 2^{\ell_k} + 2^{\ell_{k-1}} + \cdots + 2^{\ell_1}$$

To report LA$(u, d)$:

- Let $w = J(u, \ell_k)$, $v = \tau(w)$ and $v' = B_v[0]$.
  - Return $B_v[d_{v'} - d]$.

Query time: $O(1)$

# Long Path Dec. + Ladders + Jump Pointers

Space usage: $O(\underbrace{n +}_{\text{Ladders}} \overbrace{n \log n}^{\text{Jump Pointers}}) = O(n \log n)$

# Long Path Dec. + Ladders + Jump Pointers

Space usage: $O(\underbrace{n}_{\text{Ladders}} + \overbrace{n \log n}^{\text{Jump Pointers}}) = O(n \log n)$

**A trick to reduce space:**

- Only store jump pointers $J(v, \ell)$ in the leaves $v$ of $T$.
- For each node $u$ of $T$, store a reference to a leaf $\lambda_u$ in the subtree of $T$ rooted at $u$.
- $\mathsf{LA}(u, d) = \mathsf{LA}(\lambda_u, d)$

# Long Path Dec. + Ladders + Jump Pointers

Space usage: $O(\underbrace{n +}_{\text{Ladders}} \overbrace{n \log n}^{\text{Jump Pointers}}) = O(n \log n)$

**A trick to reduce space:**

- Only store jump pointers $J(v, \ell)$ in the leaves $v$ of $T$.
- For each node $u$ of $T$, store a reference to a leaf $\lambda_u$ in the subtree of $T$ rooted at $u$.
- $\mathsf{LA}(u, d) = \mathsf{LA}(\lambda_u, d)$

Space usage: $O(n + L \log n)$, where $L = \#$leaves of $T$.

# Solutions so far

| Size | Preprocessing Time | Query Time | Notes |
|---|---|---|---|
| $O(n)$ | – | $O(n)$ | |
| $O(n^2)$ | $O(n^3)$ | $O(1)$ | |
| $O(n^2)$ | $O(n^2)$ | $O(1)$ | |
| $O(n \log n)$ | $O(n \log n)$ | $O(\log n)$ | Jump Pointers |
| $O(n)$ | $O(n)$ | $O(\sqrt{n})$ | Long Path Dec. |
| $O(n)$ | $O(n)$ | $O(\log n)$ | $+$ Ladders |

# Solutions so far

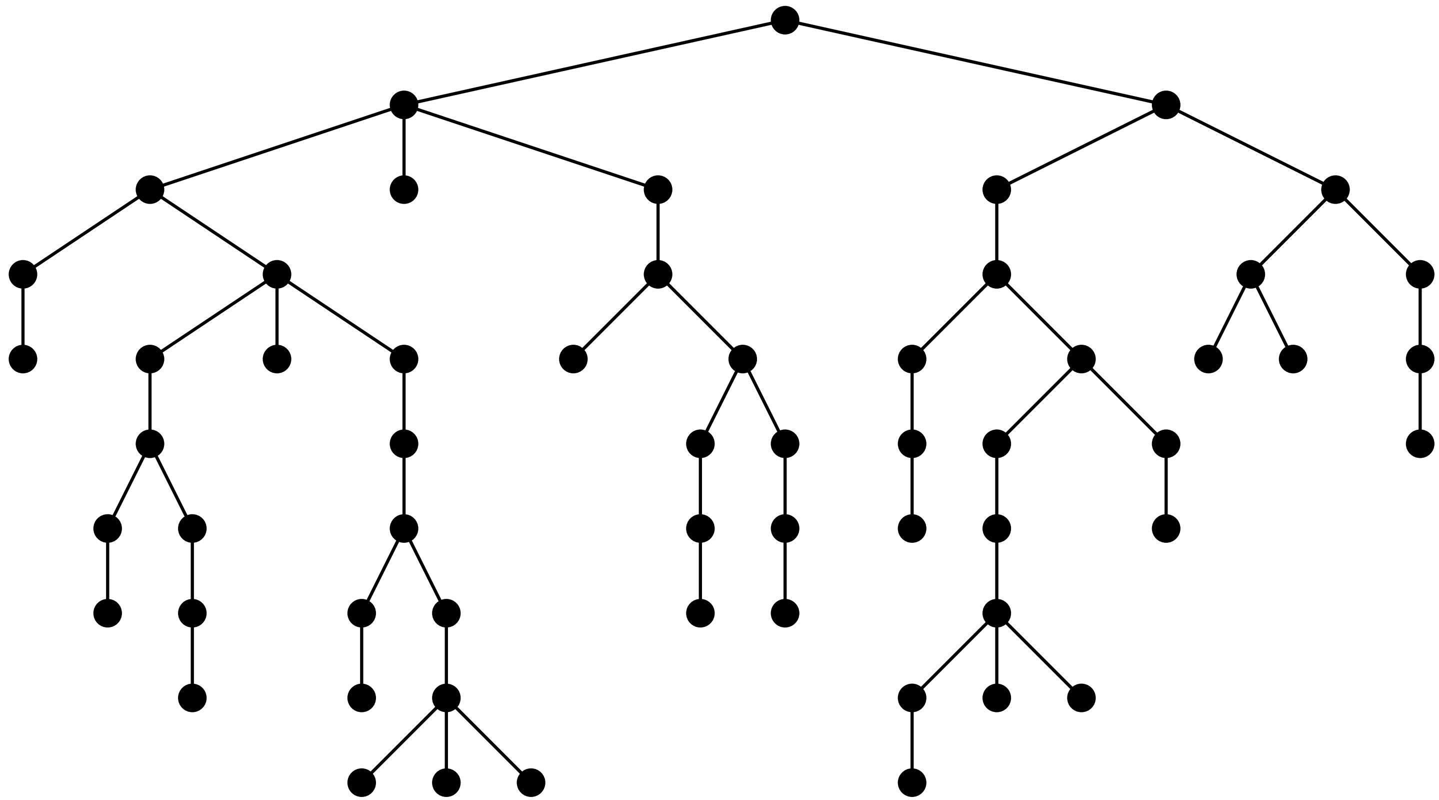
| Size | Preprocessing Time | Query Time | Notes |
|---|---|---|---|
| $O(n)$ | – | $O(n)$ | |
| $O(n^2)$ | $O(n^3)$ | $O(1)$ | |
| $O(n^2)$ | $O(n^2)$ | $O(1)$ | |
| $O(n \log n)$ | $O(n \log n)$ | $O(\log n)$ | Jump Pointers |
| $O(n)$ | $O(n)$ | $O(\sqrt{n})$ | Long Path Dec. |
| $O(n)$ | $O(n)$ | $O(\log n)$ | $+$ Ladders |
| $O(n + L \log n)$ | $O(n + L \log n)$ | $O(1)$ | $+$ Ladders, JP |

# Solutions so far

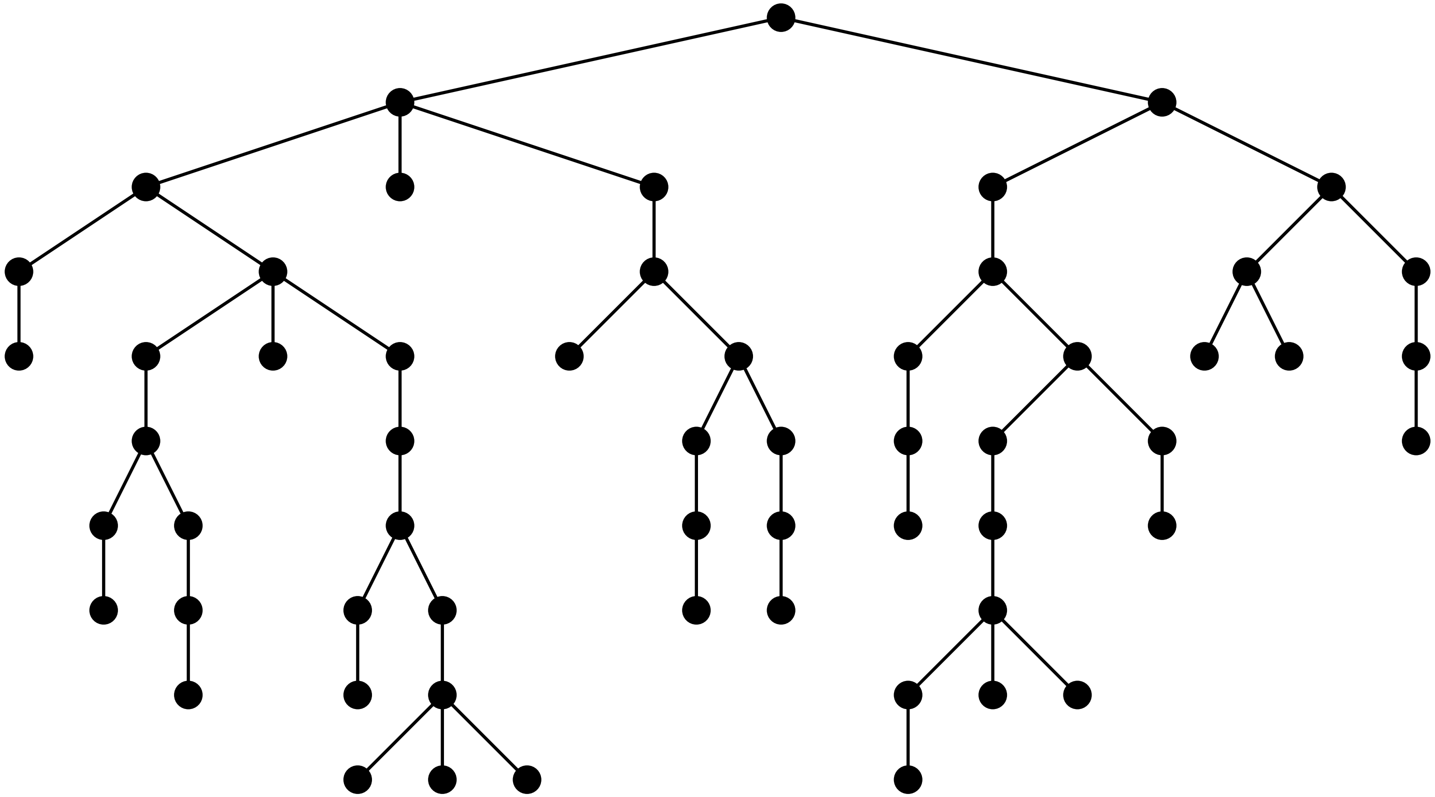
| Size | Preprocessing Time | Query Time | Notes |
|---|---|---|---|
| $O(n)$ | – | $O(n)$ | |
| $O(n^2)$ | $O(n^3)$ | $O(1)$ | |
| $O(n^2)$ | $O(n^2)$ | $O(1)$ | |
| $O(n \log n)$ | $O(n \log n)$ | $O(\log n)$ | Jump Pointers |
| $O(n)$ | $O(n)$ | $O(\sqrt{n})$ | Long Path Dec. |
| $O(n)$ | $O(n)$ | $O(\log n)$ | $+$ Ladders |
| $O(n + \underline{L \log n})$ | $O(n + \underline{L \log n})$ | $O(1)$ | $+$ Ladders, JP |

If only we had $O(\frac{n}{\log n})$ leaves...

# Macro-Micro trees



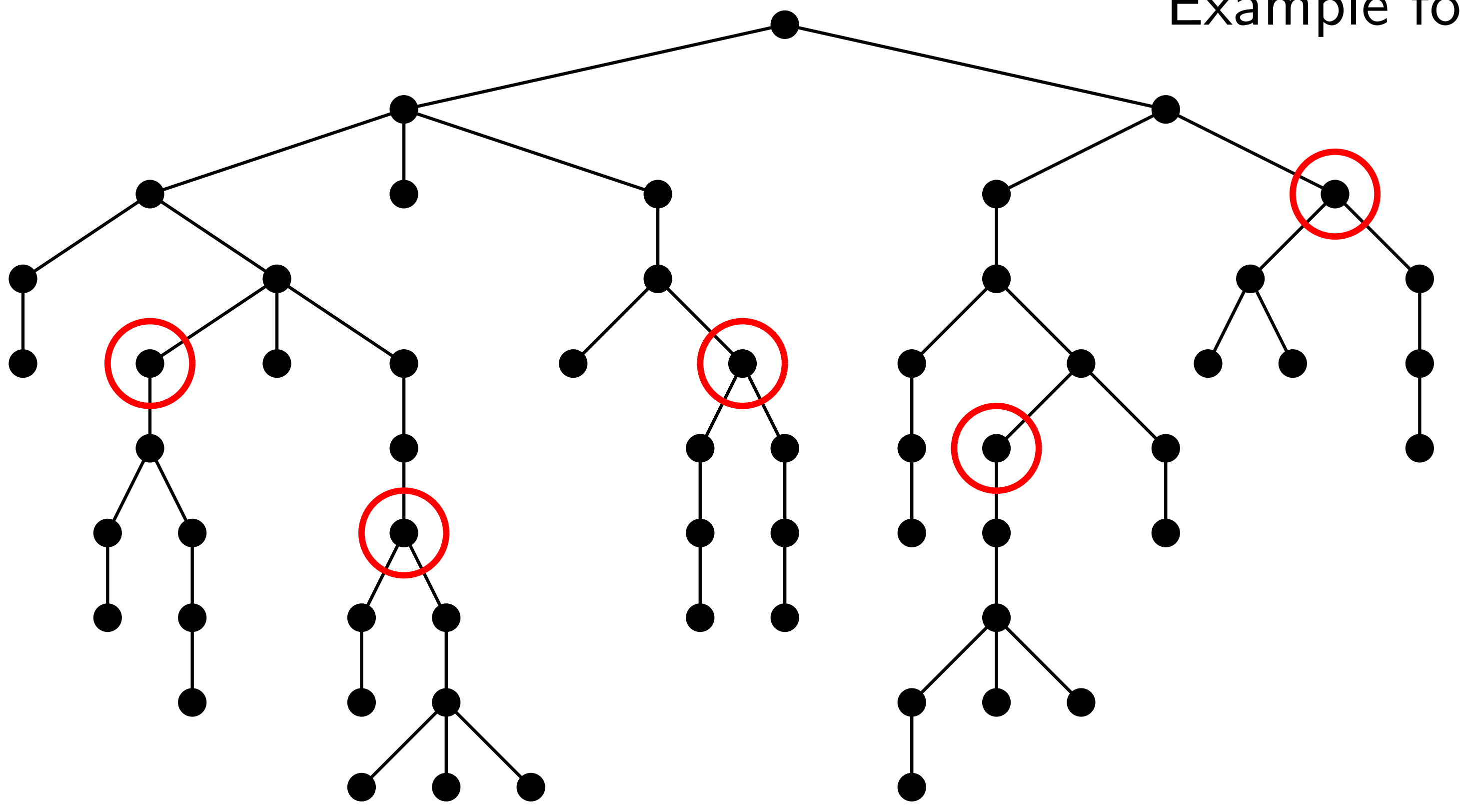
# Macro-Micro trees



Find the set $M$ of all maximally deep vertices with at least $x = \frac{1}{4}\log n$ descendants.

# Macro-Micro trees

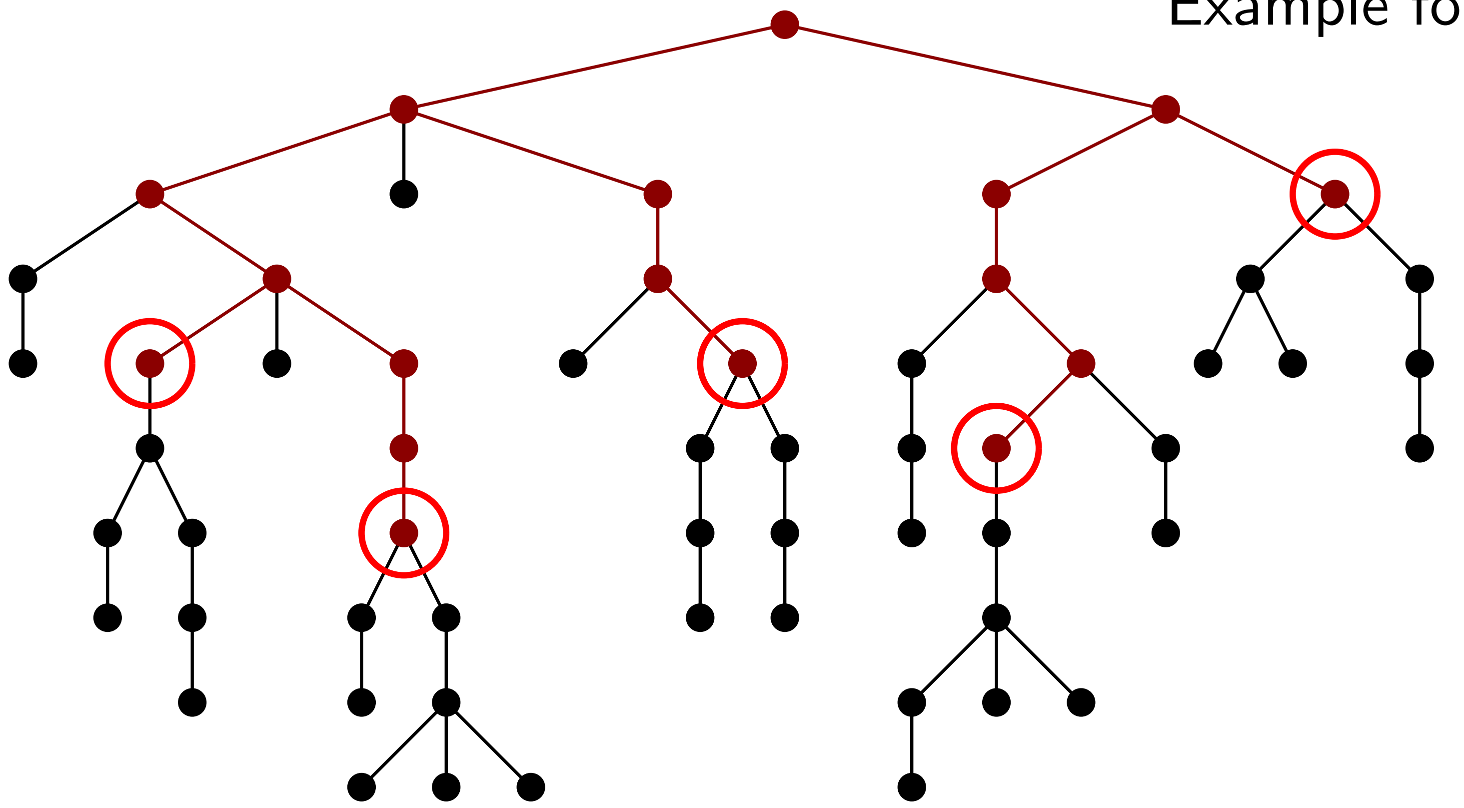
Example for $x = 7$



Find the set $M$ of all maximally deep vertices with at least $x = \frac{1}{4} \log n$ descendants.

# Macro-Micro trees

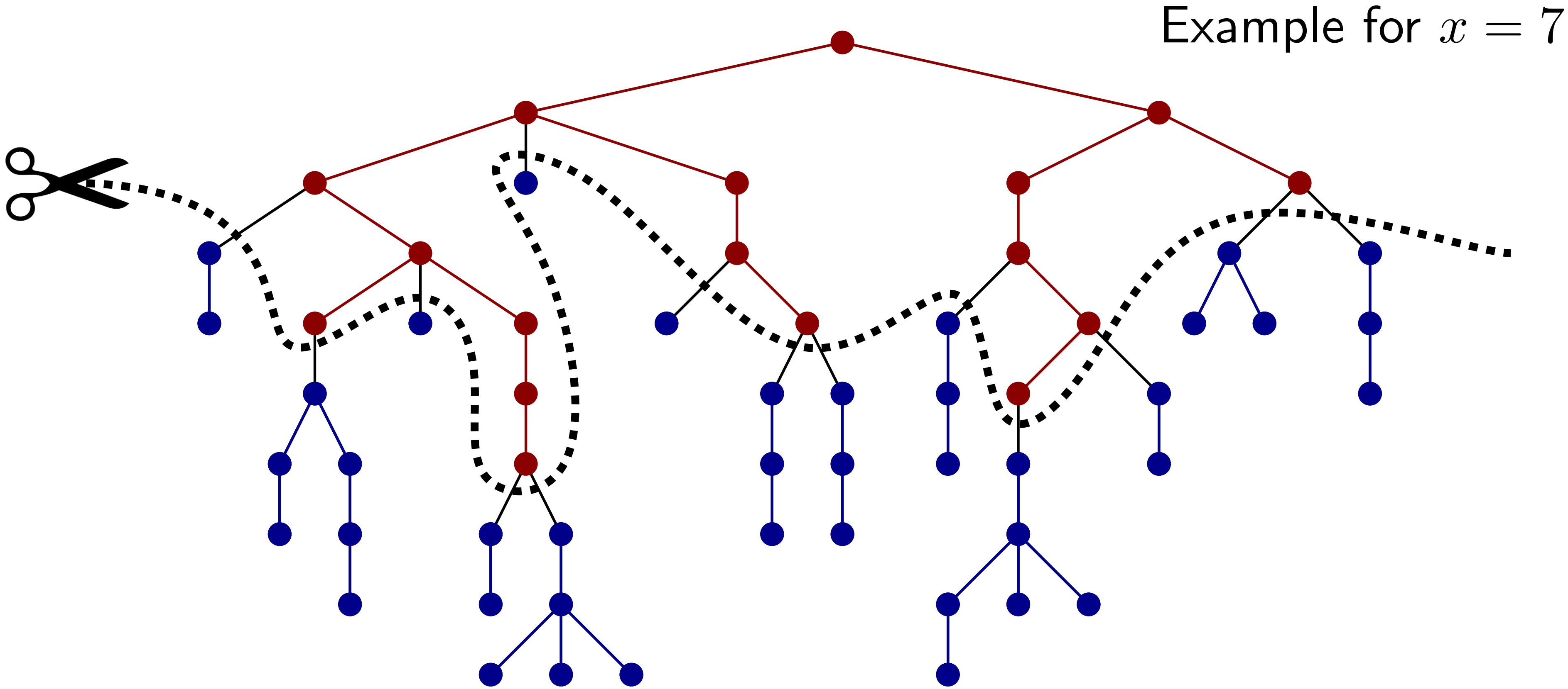
Example for $x = 7$



Find the set $M$ of all maximally deep vertices with at least $x = \frac{1}{4}\log n$ descendants.

Split $T$ into a macro-tree $T'$ containing all the ancestors of the vertices in $M$ and several micro-trees in $T \setminus T'$.

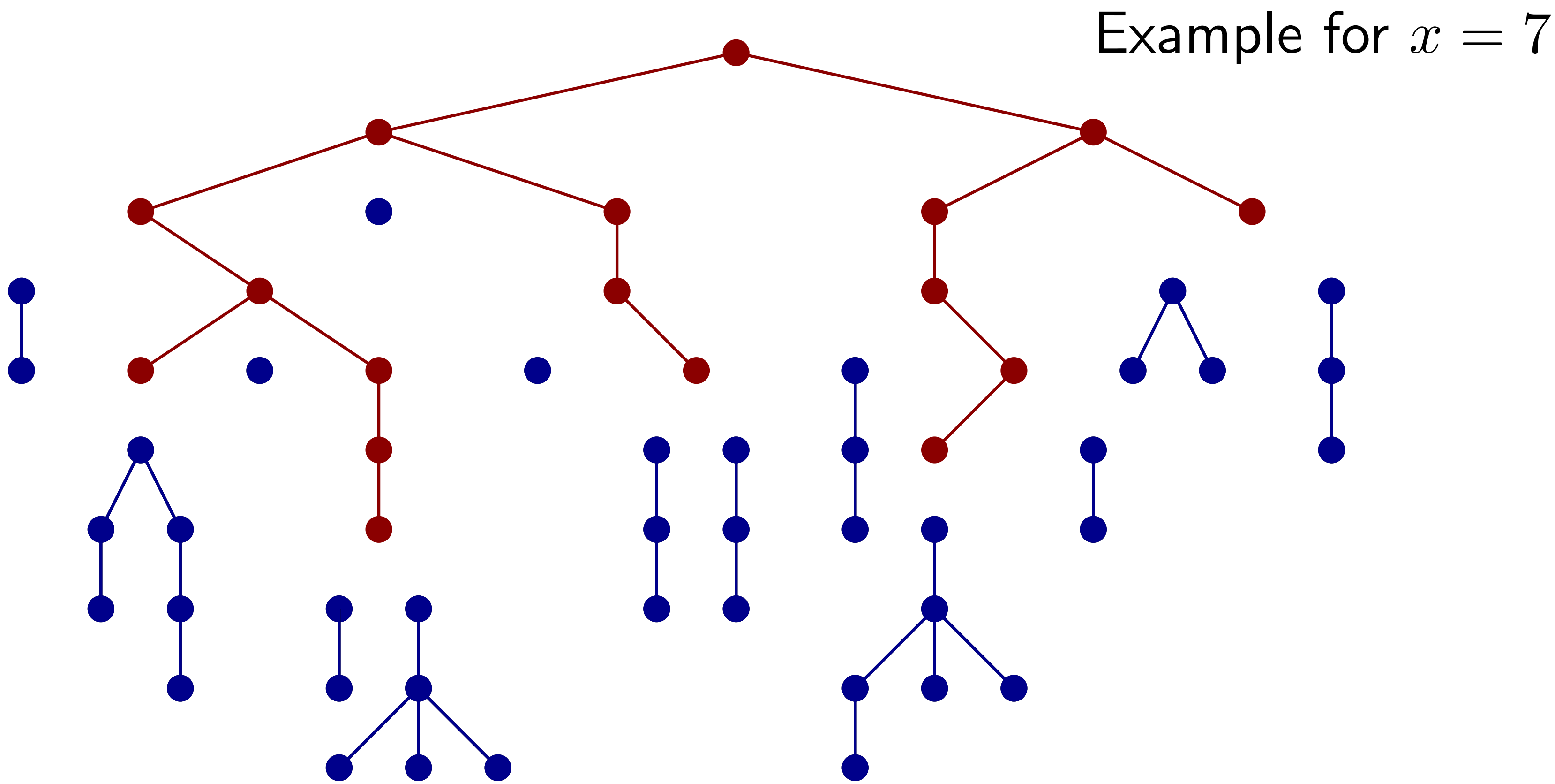
# Macro-Micro trees




Find the set $M$ of all maximally deep vertices with at least $x = \frac{1}{4}\log n$ descendants.

Split $T$ into a macro-tree $T'$ containing all the ancestors of the vertices in $M$ and several micro-trees in $T \setminus T'$.

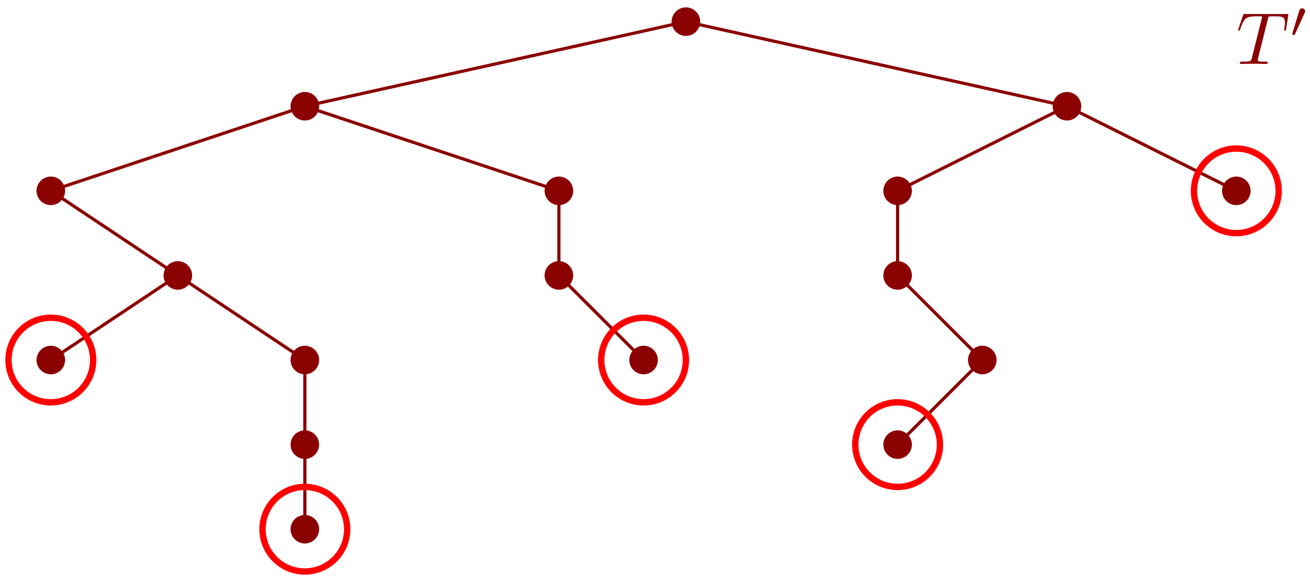
# Macro-Micro trees




Find the set $M$ of all maximally deep vertices with at least $x = \frac{1}{4} \log n$ descendants.

Split $T$ into a macro-tree $T'$ containing all the ancestors of the vertices in $M$ and several micro-trees in $T \setminus T'$.

# Handling the Macro-tree




How many leaves in $T'$? The leaves of $T'$ are the vertices in $M$.

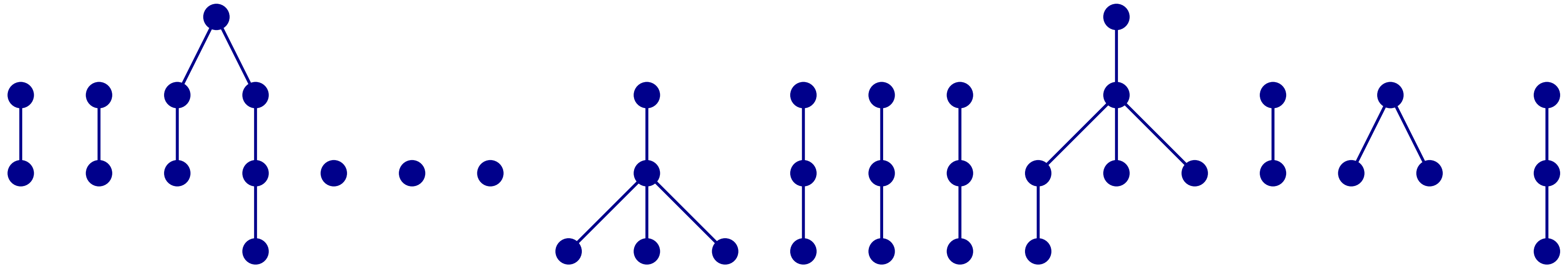
Each vertex in $M$ has at least $\frac{1}{4}\log n$ descendants in $T$.

$|M| \cdot \frac{1}{4}\log n \leq n \implies |M| = O(\frac{n}{\log n})$.

**Build the previous LA oracle $\mathcal{O}'$ on $T'$.**

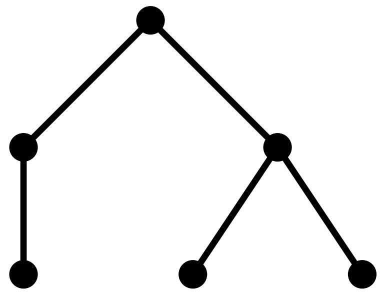
**Size/build time:** $O(n+|M|\log n)=O(n)$. **Query time:** $O(1)$.

# Handling the Micro-trees

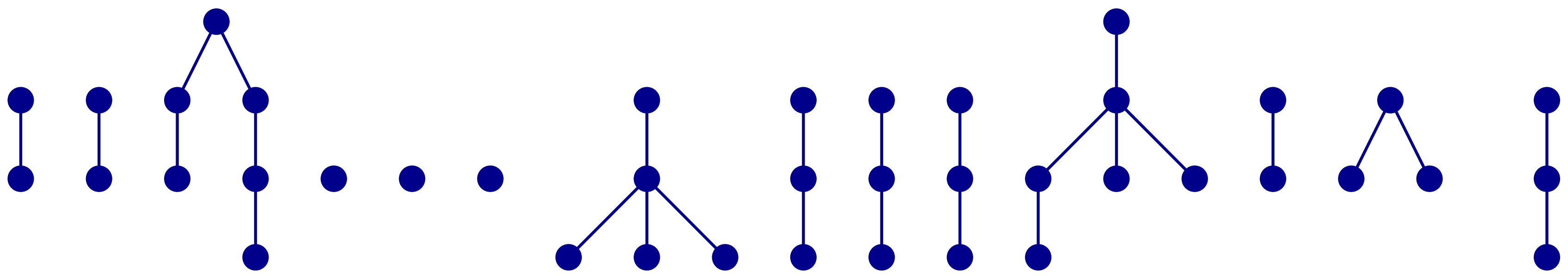


How many different *types* of micro-trees?

A rooted tree on $\leq m$ vertices can be uniquely represented by an array of $2(m-1)$ bits.



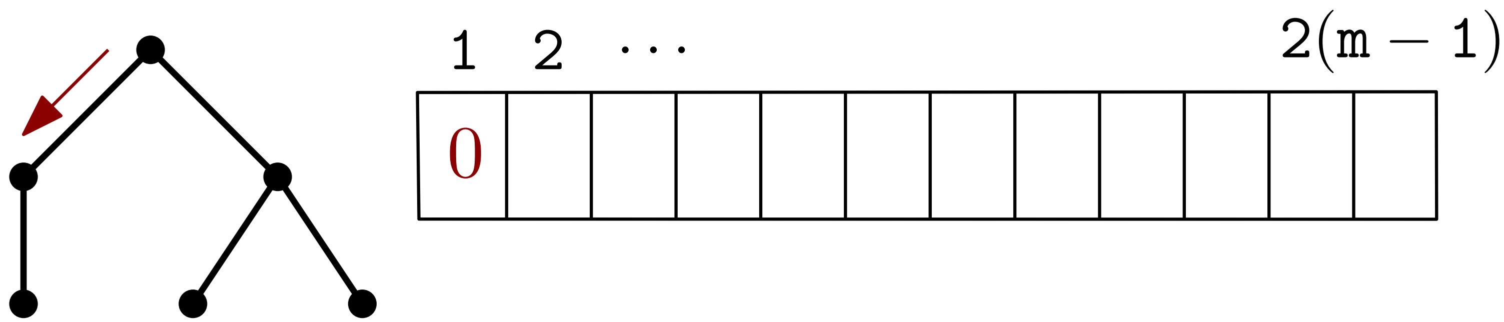
# Handling the Micro-trees



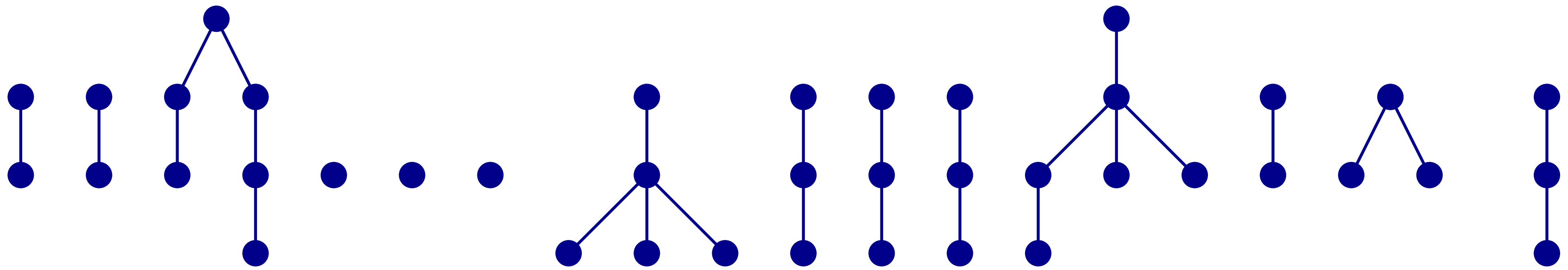
How many different *types* of micro-trees?

A rooted tree on $\leq m$ vertices can be uniquely represented by an array of $2(m-1)$ bits.

Perform a DFS traversal. Write $0$ when an edge is traversed towards the leaves, and $1$ when it is traversed towards the root.



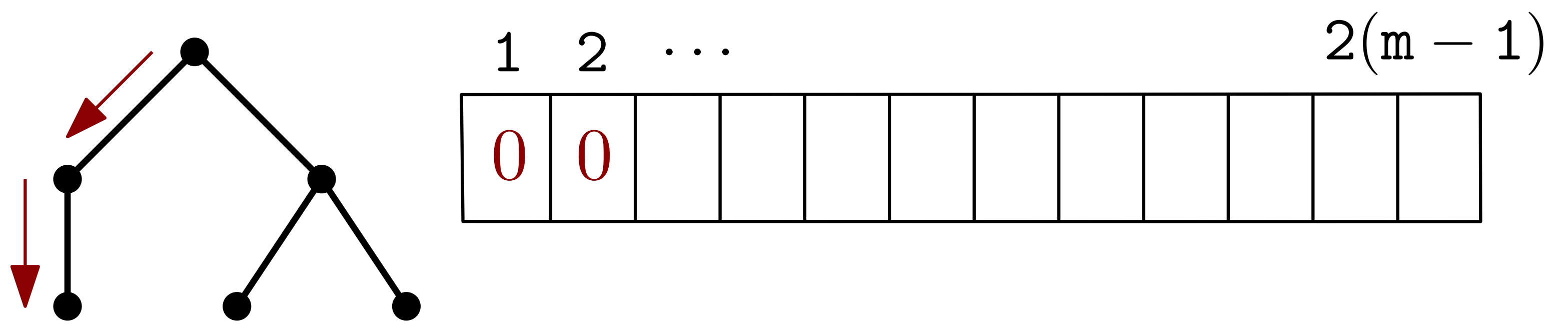
# Handling the Micro-trees



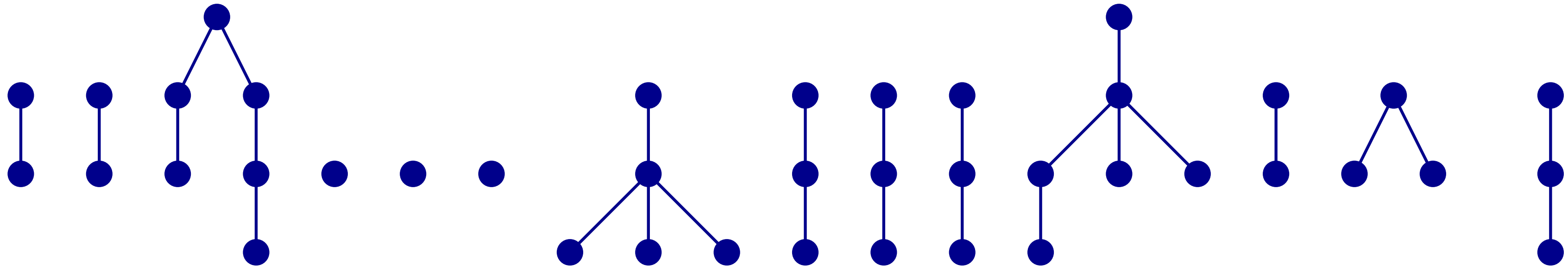
How many different *types* of micro-trees?

A rooted tree on $\leq m$ vertices can be uniquely represented by an array of $2(m-1)$ bits.

Perform a DFS traversal. Write $0$ when an edge is traversed towards the leaves, and $1$ when it is traversed towards the root.



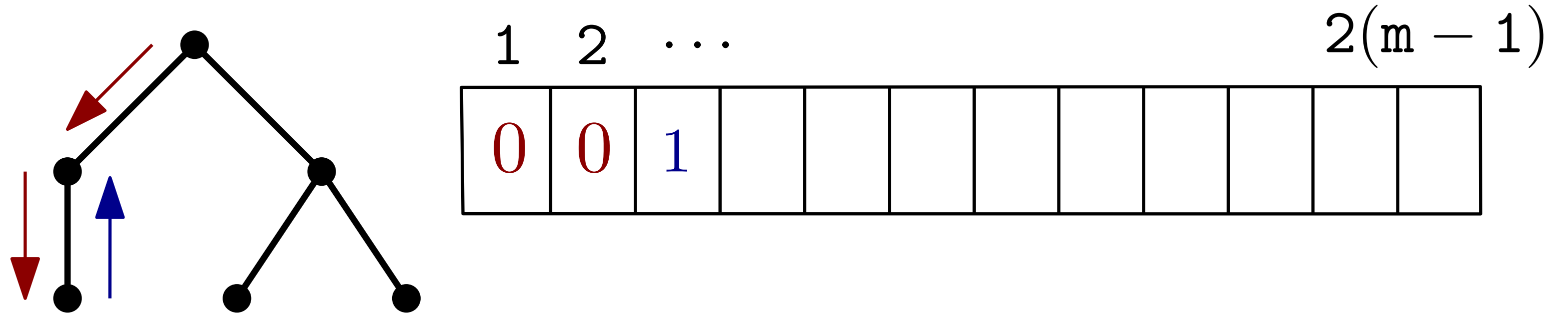
# Handling the Micro-trees



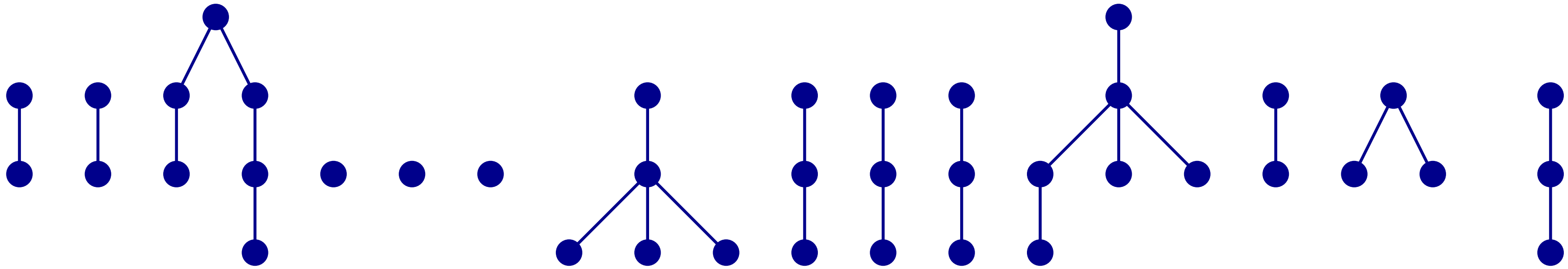
How many different *types* of micro-trees?

A rooted tree on $\leq m$ vertices can be uniquely represented by an array of $2(m-1)$ bits.

Perform a DFS traversal. Write $0$ when an edge is traversed towards the leaves, and $1$ when it is traversed towards the root.



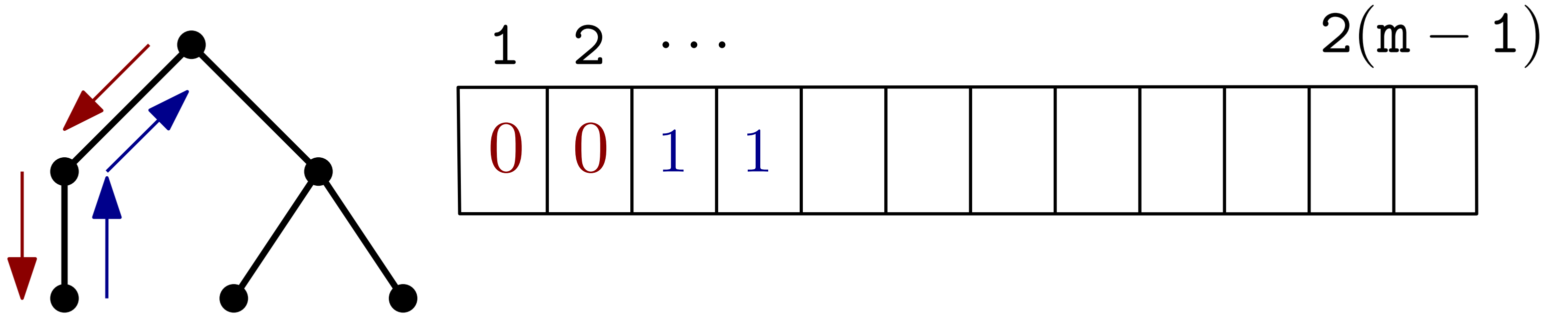
# Handling the Micro-trees



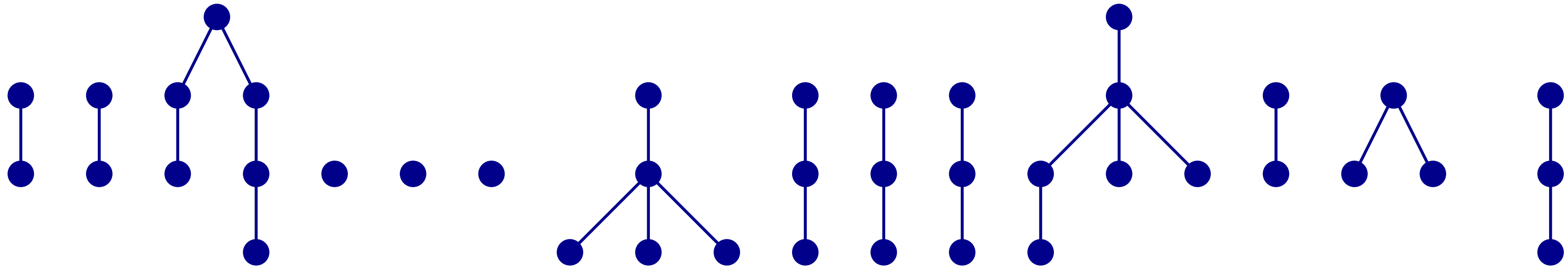
How many different *types* of micro-trees?

A rooted tree on $\leq m$ vertices can be uniquely represented by an array of $2(m-1)$ bits.

Perform a DFS traversal. Write $0$ when an edge is traversed towards the leaves, and $1$ when it is traversed towards the root.



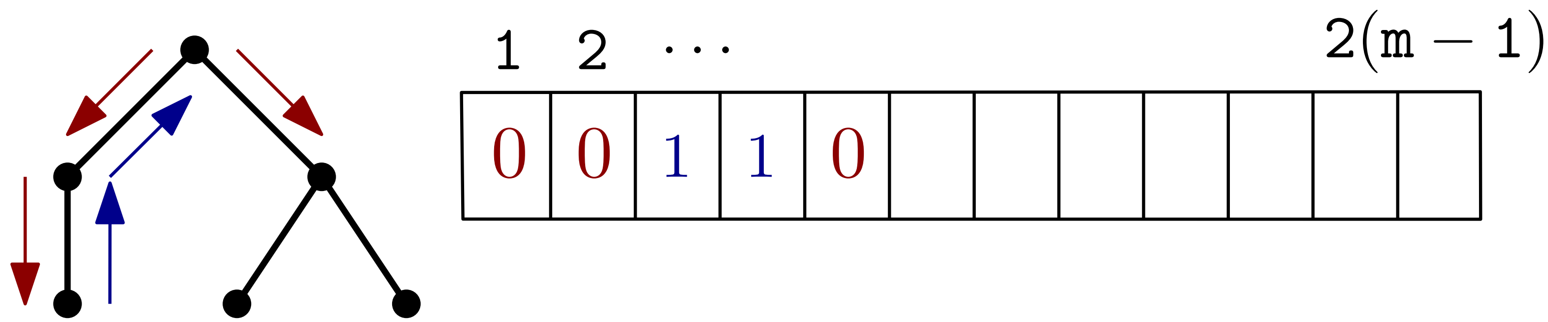
# Handling the Micro-trees



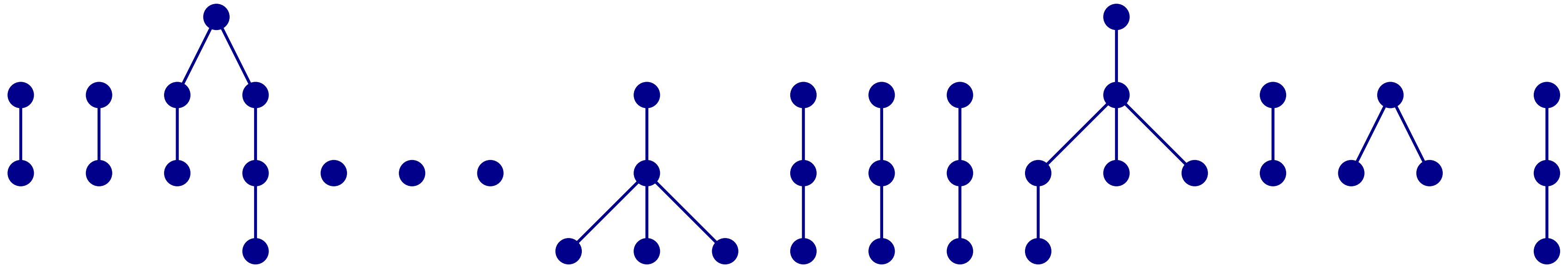
How many different *types* of micro-trees?

A rooted tree on $\leq m$ vertices can be uniquely represented by an array of $2(m-1)$ bits.

Perform a DFS traversal. Write $0$ when an edge is traversed towards the leaves, and $1$ when it is traversed towards the root.



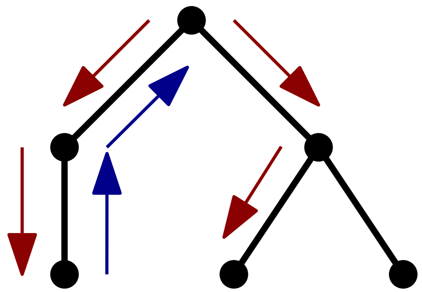
# Handling the Micro-trees



How many different *types* of micro-trees?

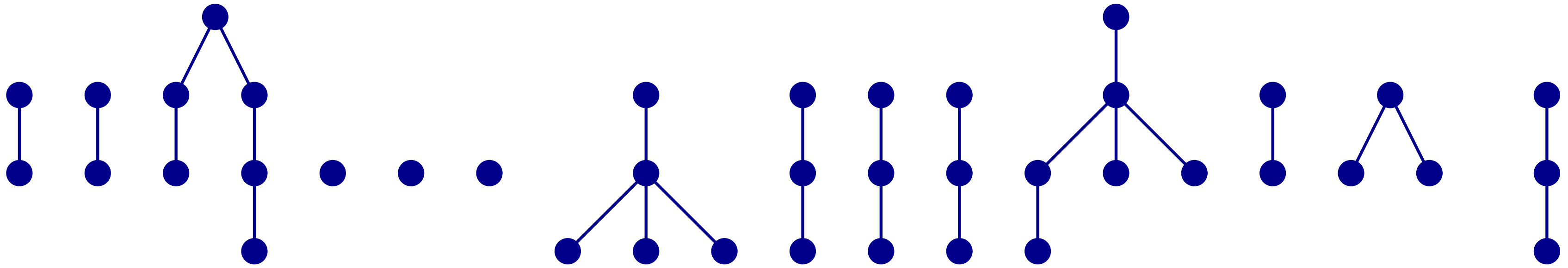
A rooted tree on $\leq m$ vertices can be uniquely represented by an array of $2(m-1)$ bits.

Perform a DFS traversal. Write $0$ when an edge is traversed towards the leaves, and $1$ when it is traversed towards the root.



| 1 | 2 | $\cdots$ | | | | | | | | | $2(\mathtt{m}-1)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | | | | | | |

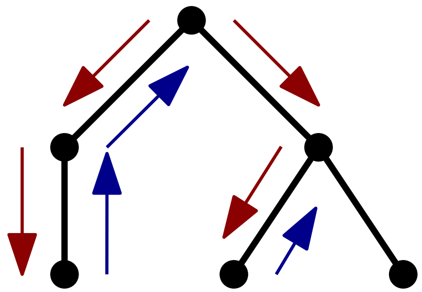
# Handling the Micro-trees



How many different *types* of micro-trees?

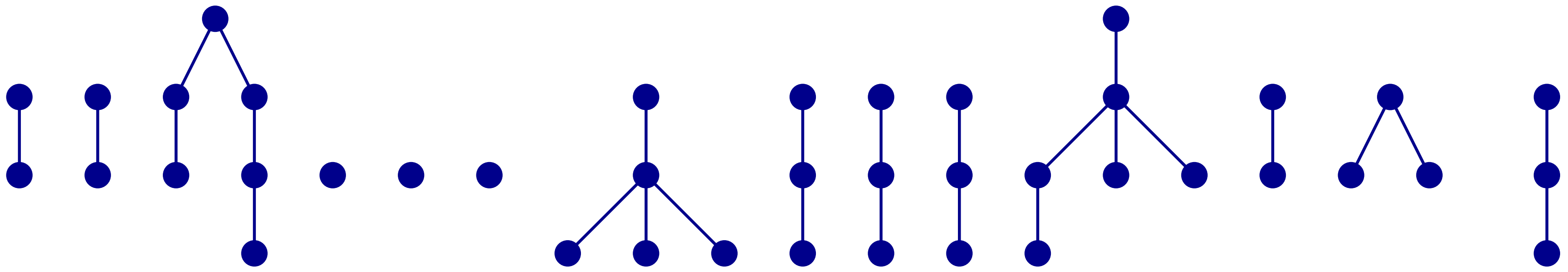
A rooted tree on $\leq m$ vertices can be uniquely represented by an array of $2(m-1)$ bits.

Perform a DFS traversal. Write $0$ when an edge is traversed towards the leaves, and $1$ when it is traversed towards the root.



| 1 | 2 | $\cdots$ | | | | | | | | | $2(\mathtt{m}-1)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | | | | | |

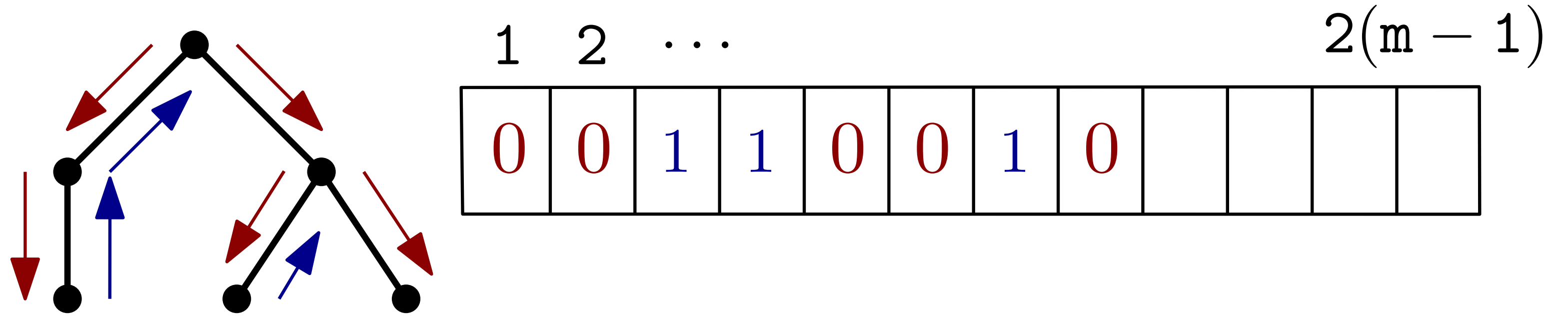
# Handling the Micro-trees



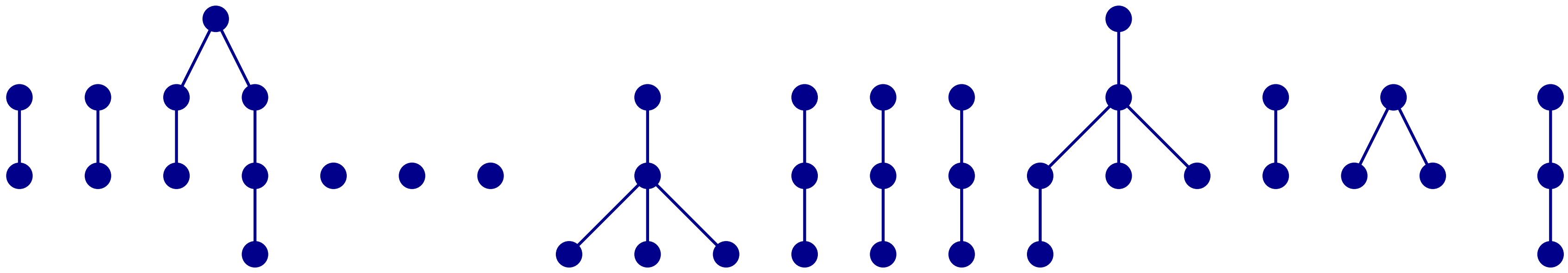
How many different *types* of micro-trees?

A rooted tree on $\leq m$ vertices can be uniquely represented by an array of $2(m-1)$ bits.

Perform a DFS traversal. Write $0$ when an edge is traversed towards the leaves, and $1$ when it is traversed towards the root.



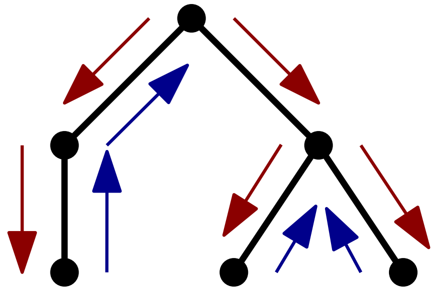
# Handling the Micro-trees



How many different *types* of micro-trees?

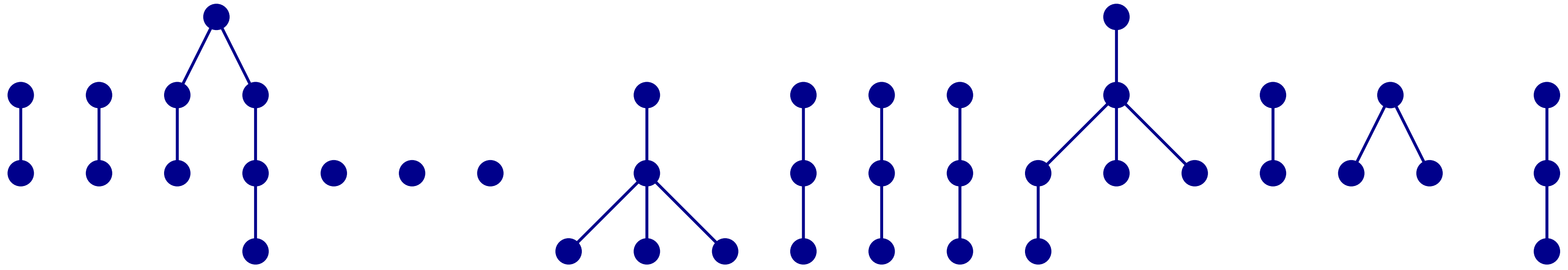
A rooted tree on $\leq m$ vertices can be uniquely represented by an array of $2(m-1)$ bits.

Perform a DFS traversal. Write $0$ when an edge is traversed towards the leaves, and $1$ when it is traversed towards the root.



| 1 | 2 | $\cdots$ | | | | | | | | | $2(\mathtt{m}-1)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | | | |

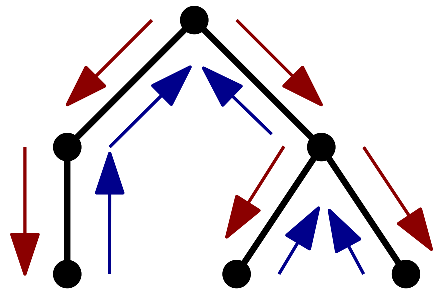
# Handling the Micro-trees



How many different *types* of micro-trees?

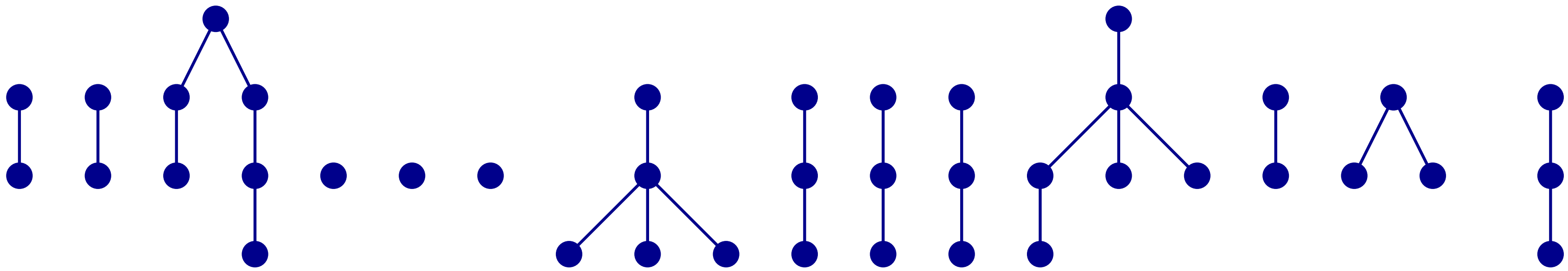
A rooted tree on $\leq m$ vertices can be uniquely represented by an array of $2(m-1)$ bits.

Perform a DFS traversal. Write $0$ when an edge is traversed towards the leaves, and $1$ when it is traversed towards the root.



| 1 | 2 | $\cdots$ | | | | | | | | | $2(m-1)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | | |

# Handling the Micro-trees

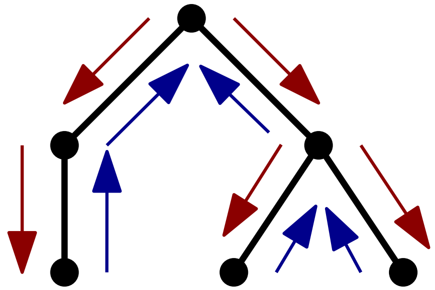


How many different *types* of micro-trees?

A rooted tree on $\leq m$ vertices can be uniquely represented by an array of $2(m-1)$ bits.

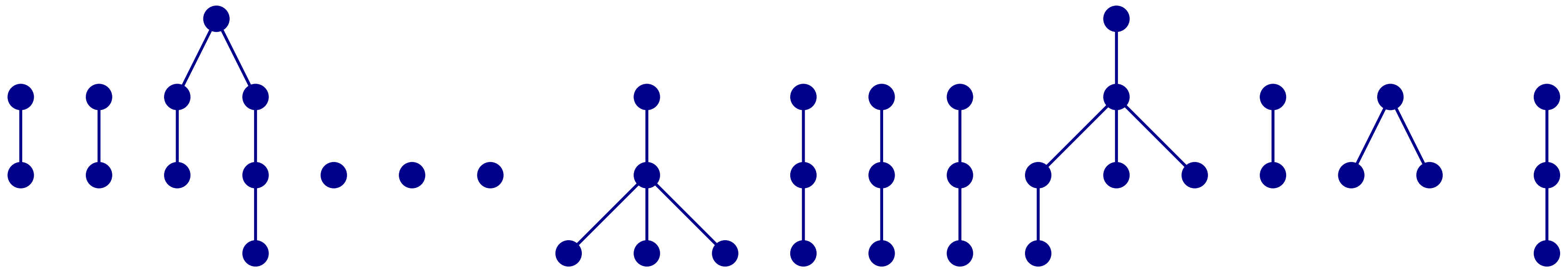
Perform a DFS traversal. Write $0$ when an edge is traversed towards the leaves, and $1$ when it is traversed towards the root.

Pad with $1$s.



| 1 | 2 | $\cdots$ | | | | | | | | | $2(\mathtt{m}-1)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |

# Handling the Micro-trees

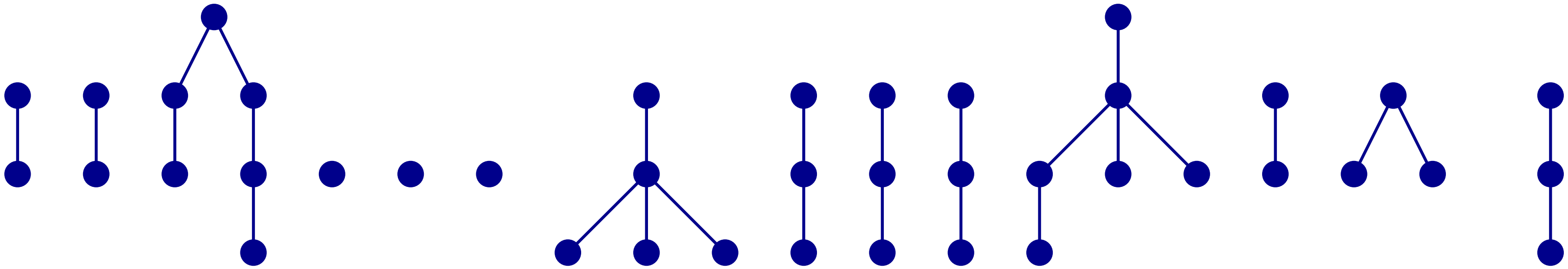


How many different *types* of micro-trees?

A rooted tree on $\leq m$ vertices can be uniquely represented by an array of $2(m-1)$ bits.

At most $2^{2(m-1)} < 2^{2m}$ trees with up to $m$ vertices
$\implies O(2^{2\frac{1}{4}\log n}) = O(\sqrt{n})$ micro-tree types.

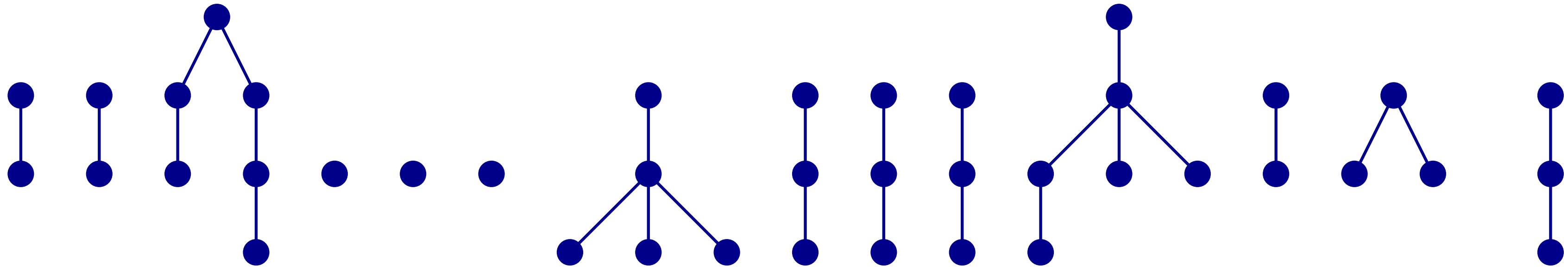
# Handling the Micro-trees



For each of the $O(\sqrt{n})$ distinct micro-trees types $T_i$

- Build the trival oracle $\mathcal{O}_i$ with size/preprocessing time $O(|T_i|^2)$ and query time $O(1)$.

# Handling the Micro-trees



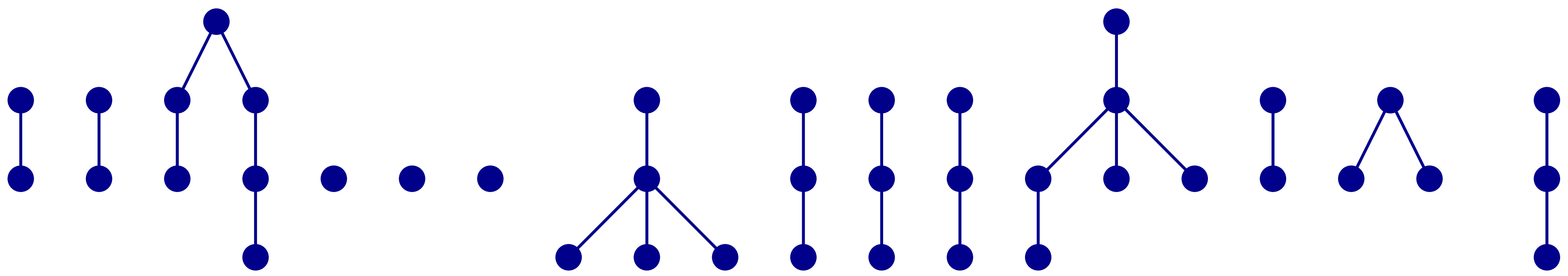
For each of the $O(\sqrt{n})$ distinct micro-trees types $T_i$

- Build the trival oracle $\mathcal{O}_i$ with size/preprocessing time $O(|T_i|^2)$ and query time $O(1)$.

For each vertex $u$ of $T$ that belongs to a micro tree:

- Store, in $u$, the index $i$ of the *type* of its micro-tree.
- Store, in $u$, the vertex $\mu(u)$ in $T_i$ corresponding to $u$.
- Store, in $u$, the root $\rho(u)$ of its micro-tree.

# Handling the Micro-trees



For each of the $O(\sqrt{n})$ distinct micro-trees types $T_i$

- Build the trival oracle $\mathcal{O}_i$ with size/preprocessing time $O(|T_i|^2)$ and query time $O(1)$.

For each vertex $u$ of $T$ that belongs to a micro tree:

- Store, in $u$, the index $i$ of the *type* of its micro-tree.
- Store, in $u$, the vertex $\mu(u)$ in $T_i$ corresponding to $u$.
- Store, in $u$, the root $\rho(u)$ of its micro-tree.

Total size/time: $O(\sqrt{n}) \cdot O(\log^2 n) + O(n) = O(n)$.

# Answering a Query

To answer LA$(u, d)$:

- If $u$ is in the macro tree $T'$: query $\mathcal{O}'$ for LA$(u, d)$.
- If $u$ is in a micro-tree $T''$:
  - If $d < d_{\rho(u)}$: query $\mathcal{O}'$ for LA$(\text{parent}(\rho(u)), d)$.
  - Otherwise:
    - Let $i$ be the type of the micro-tree containing $u$.
    - Query $O_i$ for LA$(\mu(u), d - d_{\rho(u)})$.

      (and map it back to a vertex in $T''$)

**Query time:** $O(1)$.

# Solutions so far

| Size | Preprocessing Time | Query Time | Notes |
|---|---|---|---|
| $O(n)$ | – | $O(n)$ | |
| $O(n^2)$ | $O(n^3)$ | $O(1)$ | |
| $O(n^2)$ | $O(n^2)$ | $O(1)$ | |
| $O(n \log n)$ | $O(n \log n)$ | $O(\log n)$ | Jump Pointers |
| $O(n)$ | $O(n)$ | $O(\sqrt{n})$ | Long Path Dec. |
| $O(n)$ | $O(n)$ | $O(\log n)$ | $+$ Ladders |
| $O(n + L \log n)$ | $O(n + L \log n)$ | $O(1)$ | $+$ Ladders, JP |

# Solutions so far

| Size | Preprocessing Time | Query Time | Notes |
|---|---|---|---|
| $O(n)$ | – | $O(n)$ | |
| $O(n^2)$ | $O(n^3)$ | $O(1)$ | |
| $O(n^2)$ | $O(n^2)$ | $O(1)$ | |
| $O(n \log n)$ | $O(n \log n)$ | $O(\log n)$ | Jump Pointers |
| $O(n)$ | $O(n)$ | $O(\sqrt{n})$ | Long Path Dec. |
| $O(n)$ | $O(n)$ | $O(\log n)$ | $+$ Ladders |
| $O(n + L \log n)$ | $O(n + L \log n)$ | $O(1)$ | $+$ Ladders, JP |
| $O(n)$ | $O(n)$ | $O(1)$ | $+$ Macro-Micro trees |