

# Level Ancestor

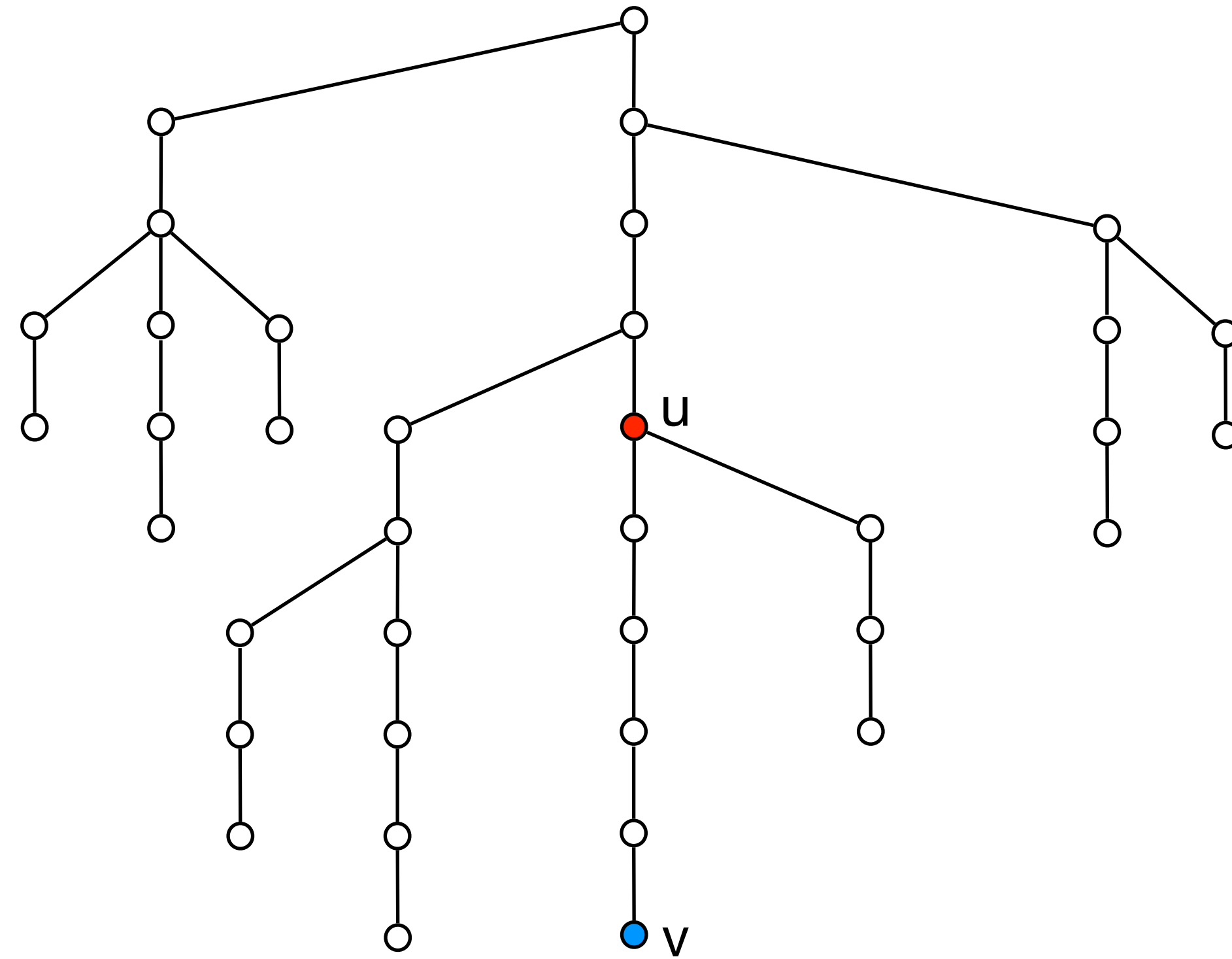
---

Philip Bille/Inge Li Gørtz

# Level Ancestor

---

- **Level ancestor problem.** Preprocess rooted tree  $T$  with  $n$  nodes to support
  - $LA(v,k)$ : return the  $k$ th ancestor of node  $v$ .



$$LA(v,5) = u$$

# Level Ancestor

---

- [Applications.](#)
  - Basic primitive for navigating trees (any hierarchical data).
  - Illustration of wealth of techniques for trees.
    - Path decompositions.
    - Tree decomposition.
    - Tree encoding and tabulation.

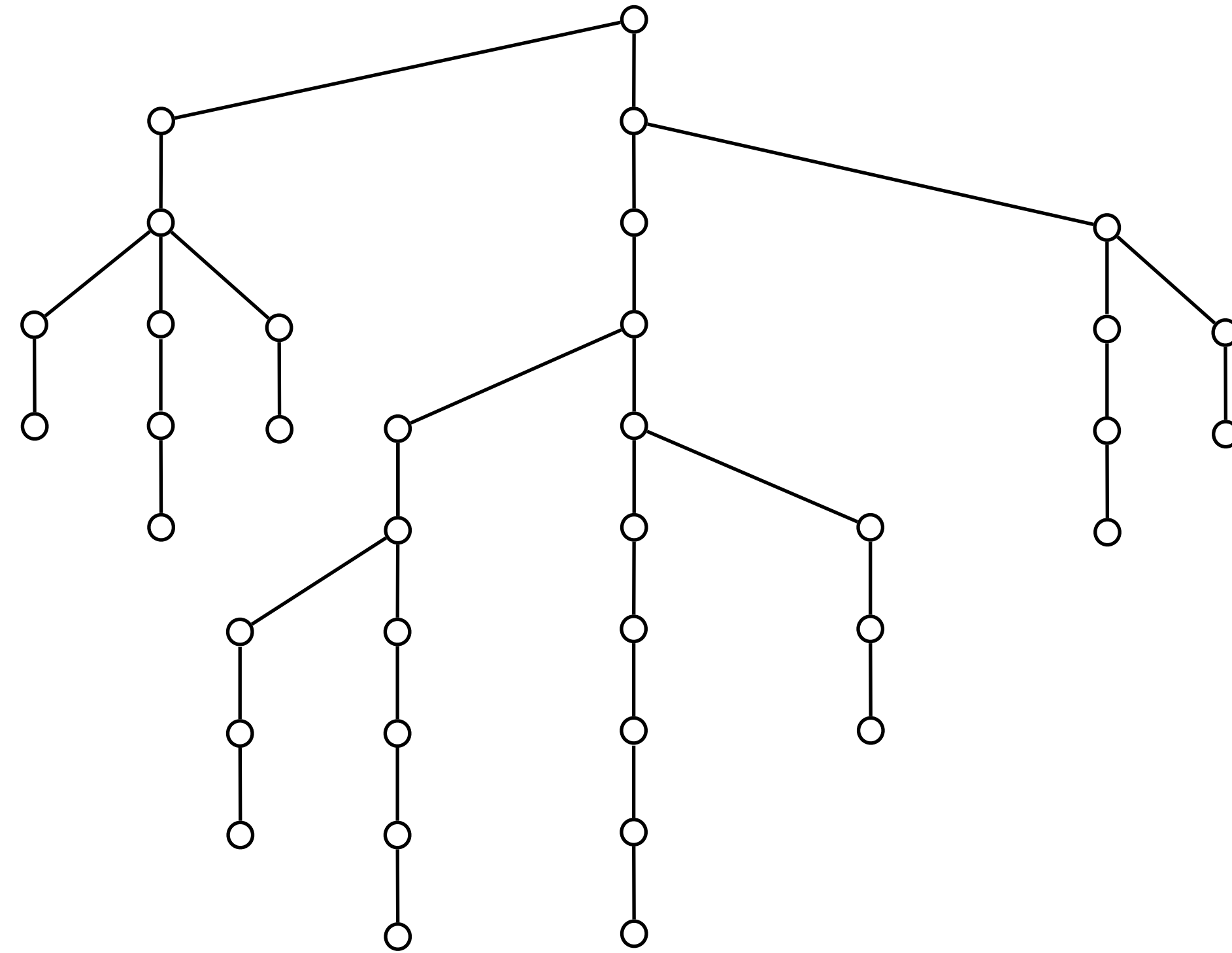
# Level Ancestor

---

- **Goal.** Linear space and constant time.
- **Solution in 7 steps (!).**
  - **No data structure.** Very slow, little space
  - **Direct shortcuts.** Very fast, lot of space.
  - ....
  - **Ladder decomposition + jump pointers + top-bottom decomposition.** Very fast, little space.

# Solution 1: No Data Structure

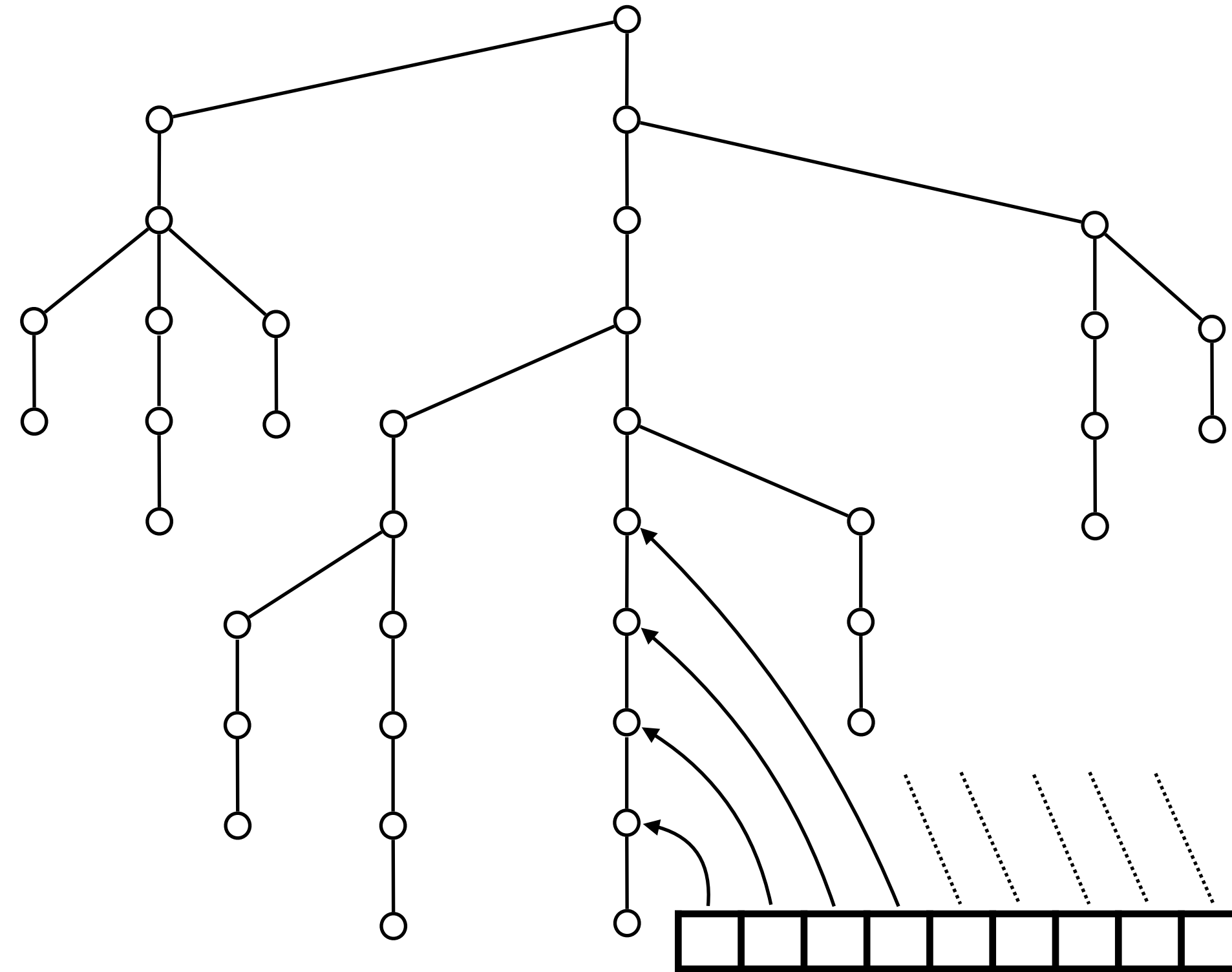
---



- **Data structure.** Store tree  $T$  (using pointers).
- $LA(v,k)$ : **Walk** up.
- **Time.**  $O(n)$
- **Space.**  $O(n)$

# Solution 2: Direct Shortcuts

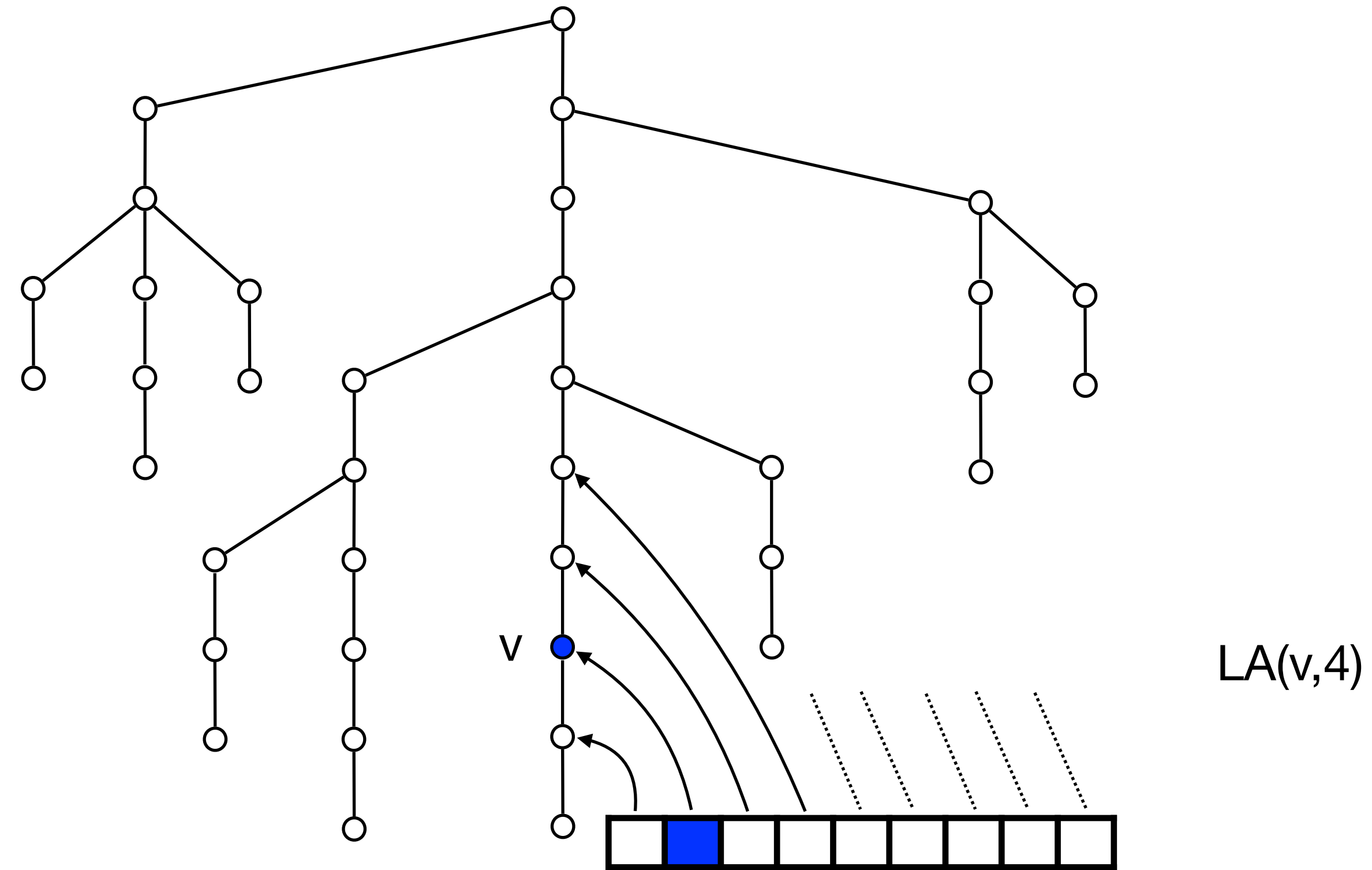
---



- **Data structure.** Store each root-to-leaf in array.
- **LA(v,k):** **Jump** up.
- **Time.**  $O(1)$
- **Space.**  $O(n^2)$

# Solution 2: Direct Shortcuts

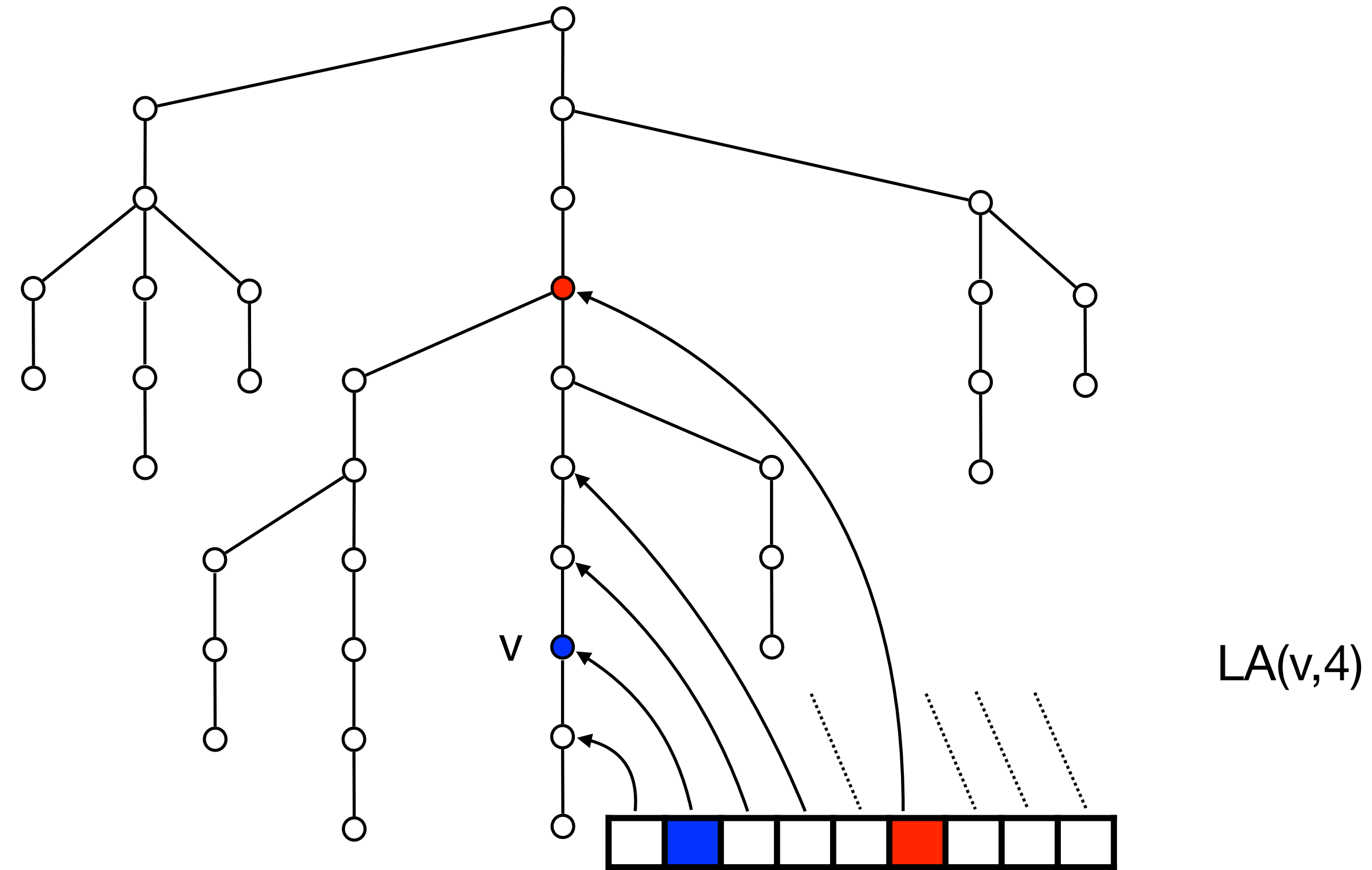
---



- **Data structure.** Store each root-to-leaf in array.
- **LA(v,k):** Jump up.
- **Time.**  $O(1)$
- **Space.**  $O(n^2)$

# Solution 2: Direct Shortcuts

---

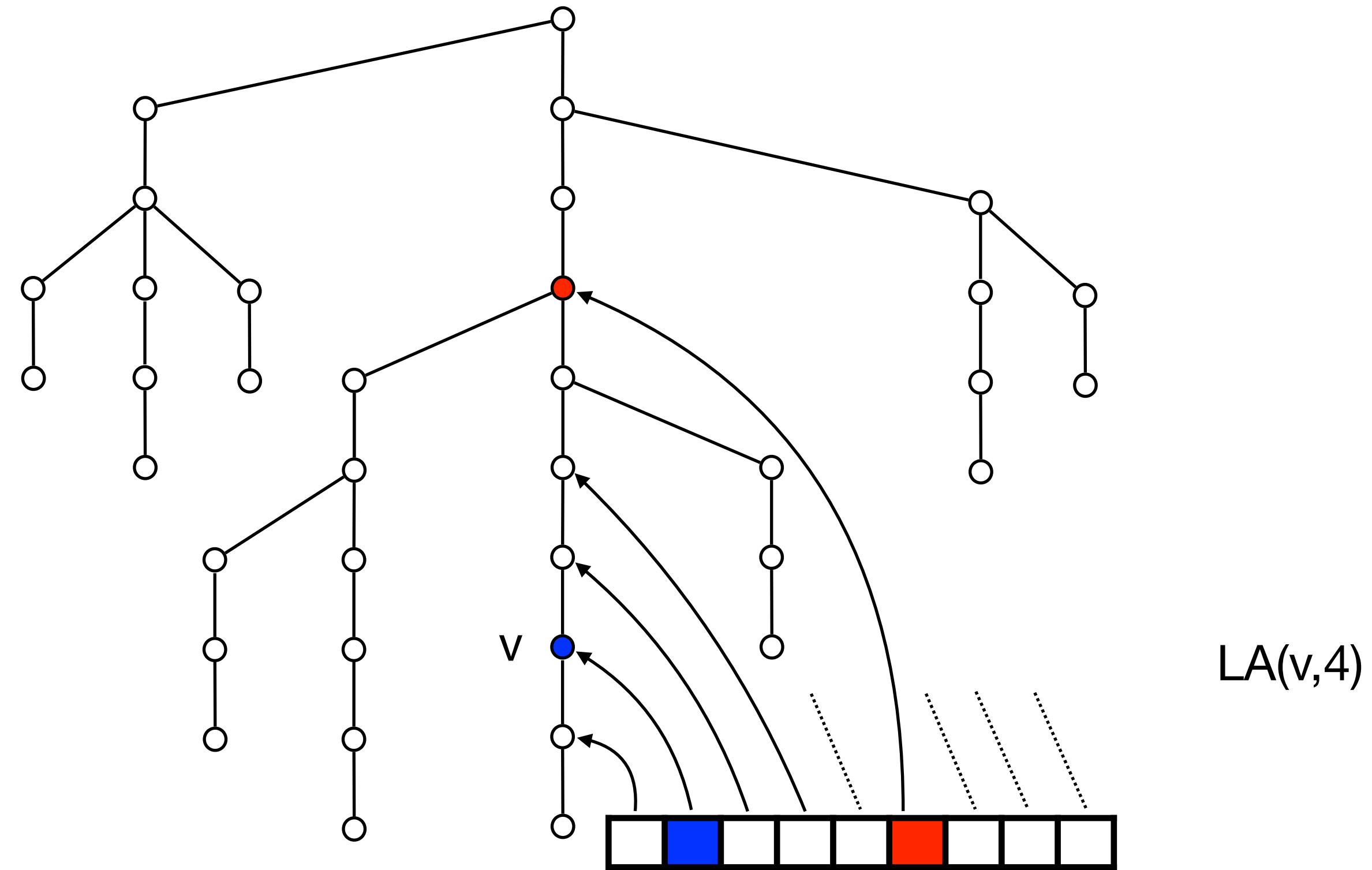


- **Data structure.** Store each root-to-leaf in array.
- **LA(v,k):** **Jump** up.
- **Time.**  $O(1)$
- **Space.**  $O(n^2)$



# Solution 2: Direct Shortcuts

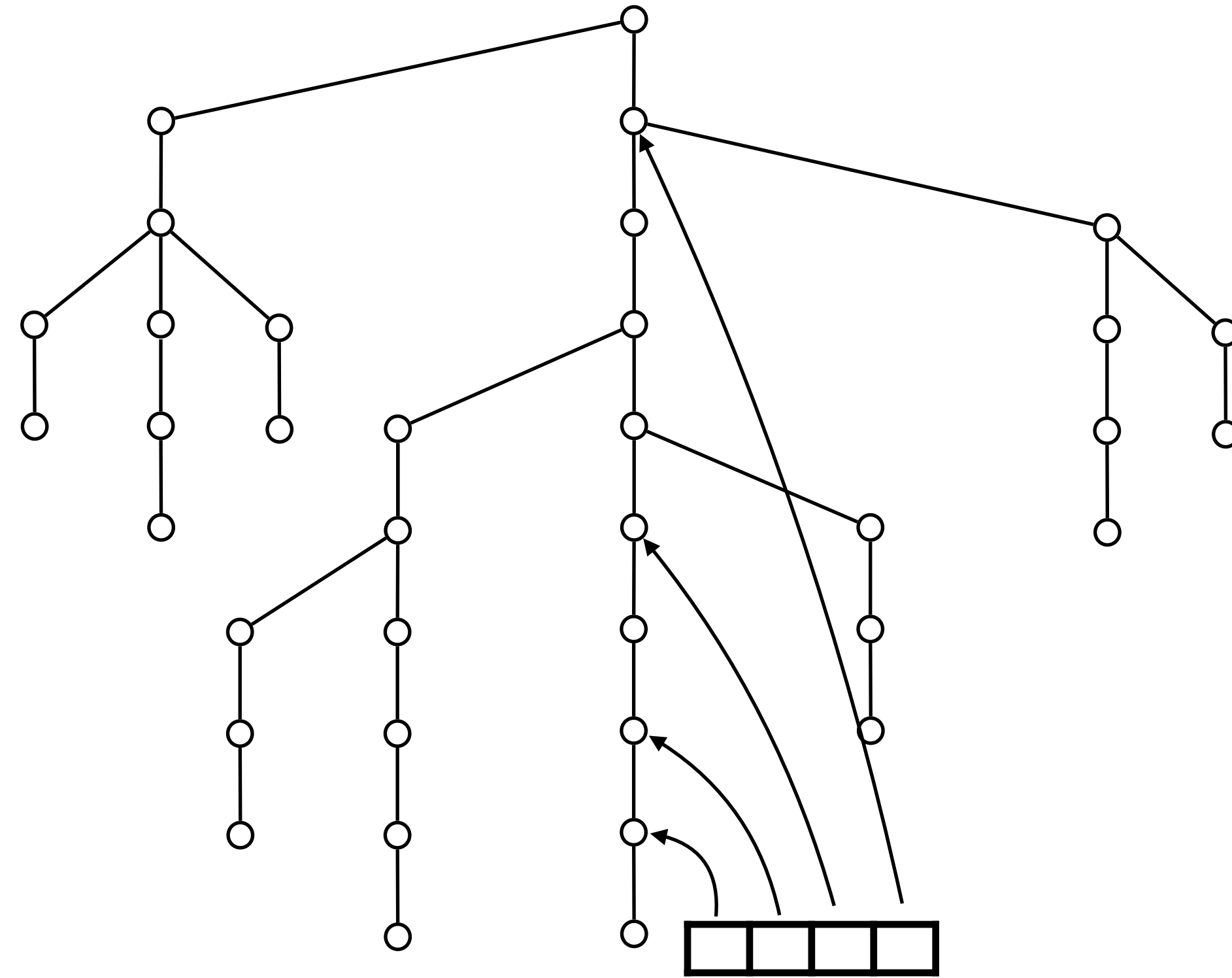
---



- **Data structure.** Store each root-to-leaf in array.
- **LA(v,k):** Jump up.
- **Time.**  $O(1)$
- **Space.**  $O(n^2)$

# Solution 3: Jump Pointers

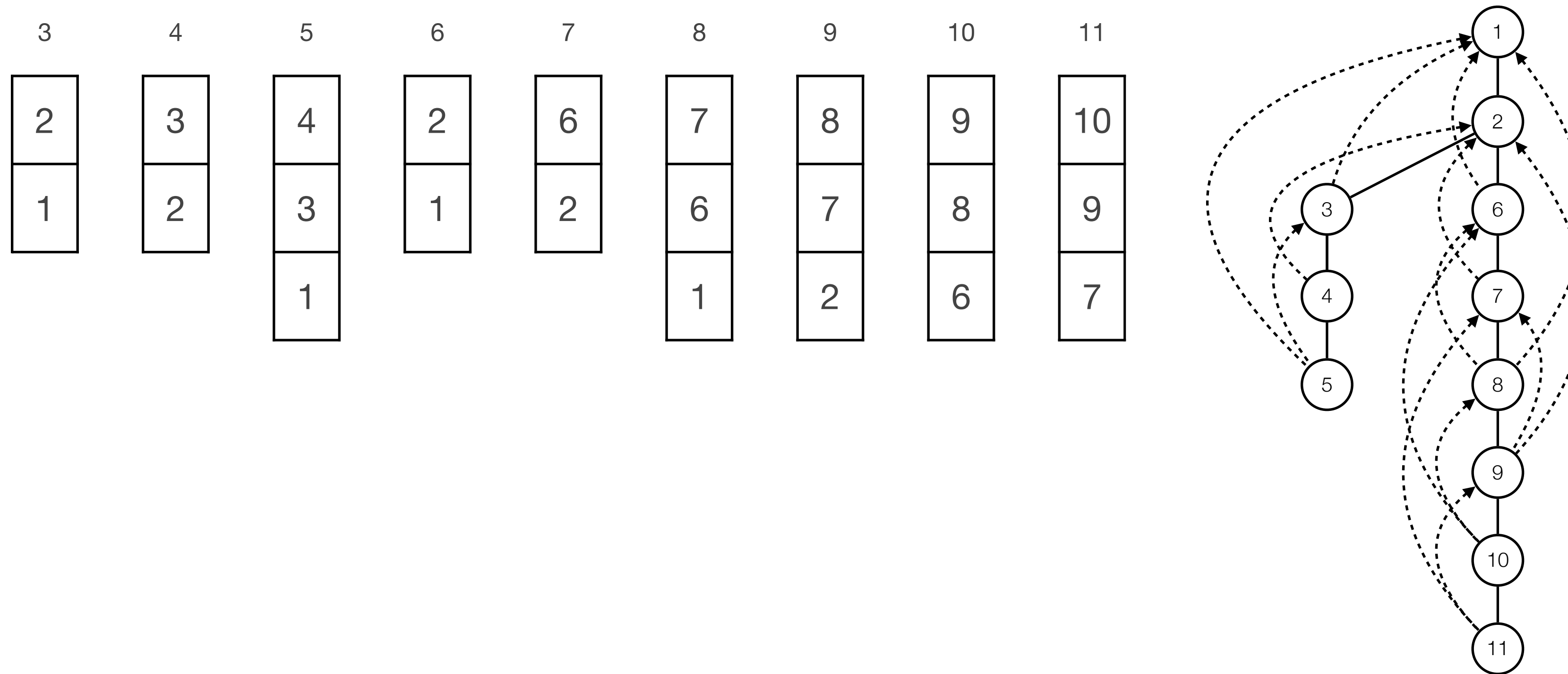
---



- **Data structure.** For each node  $v$ , store pointers to ancestors at distance 1,2,4, ..
- **LA( $v,k$ ):** Jump to most distant ancestor no further away than  $k$ . Repeat.
- **Time.**  $O(\log n)$
- **Space.**  $O(n \log n)$

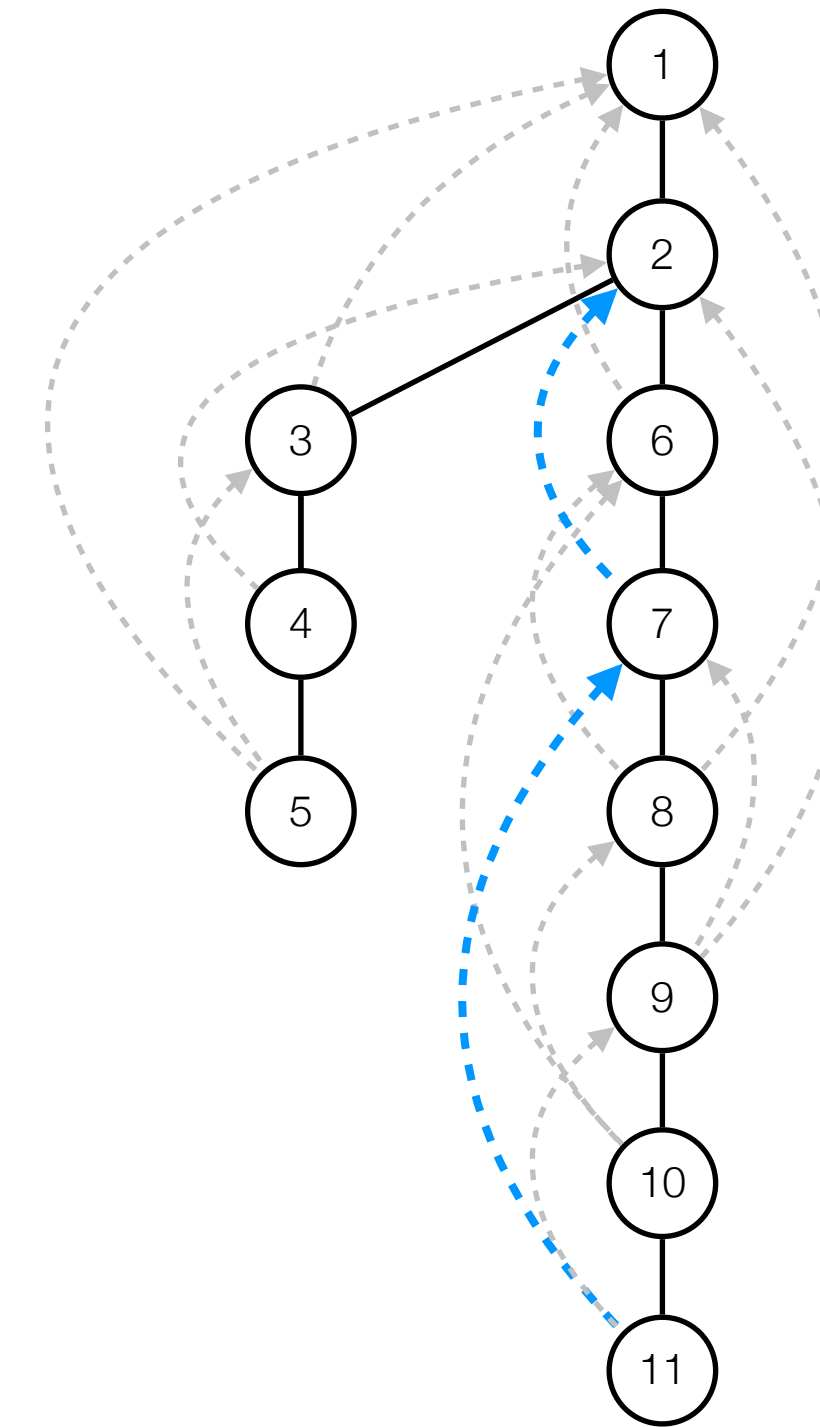
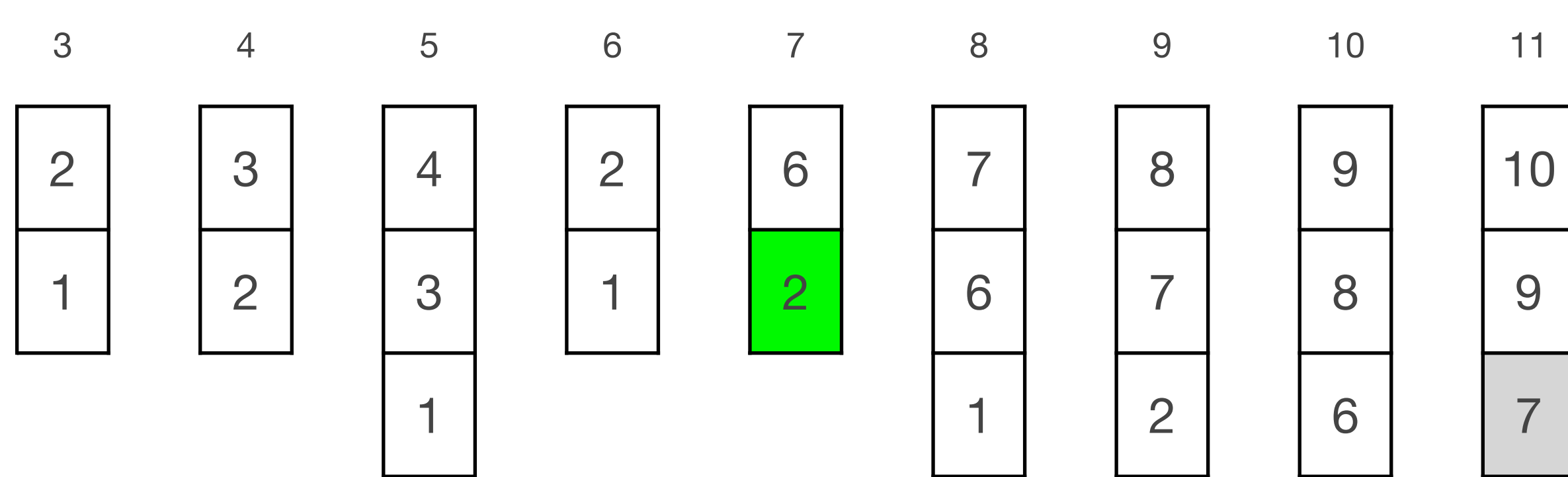
# Solution 3: Jump Pointers

---



- **Data structure.** For each node  $v$ , store pointers to ancestors at distance 1,2,4, ..
- **LA(v,k):** Jump to most distant ancestor no further away than  $k$ . Repeat.
- **Time.**  $O(\log n)$
- **Space.**  $O(n \log n)$

# Solution 3: Jump Pointers

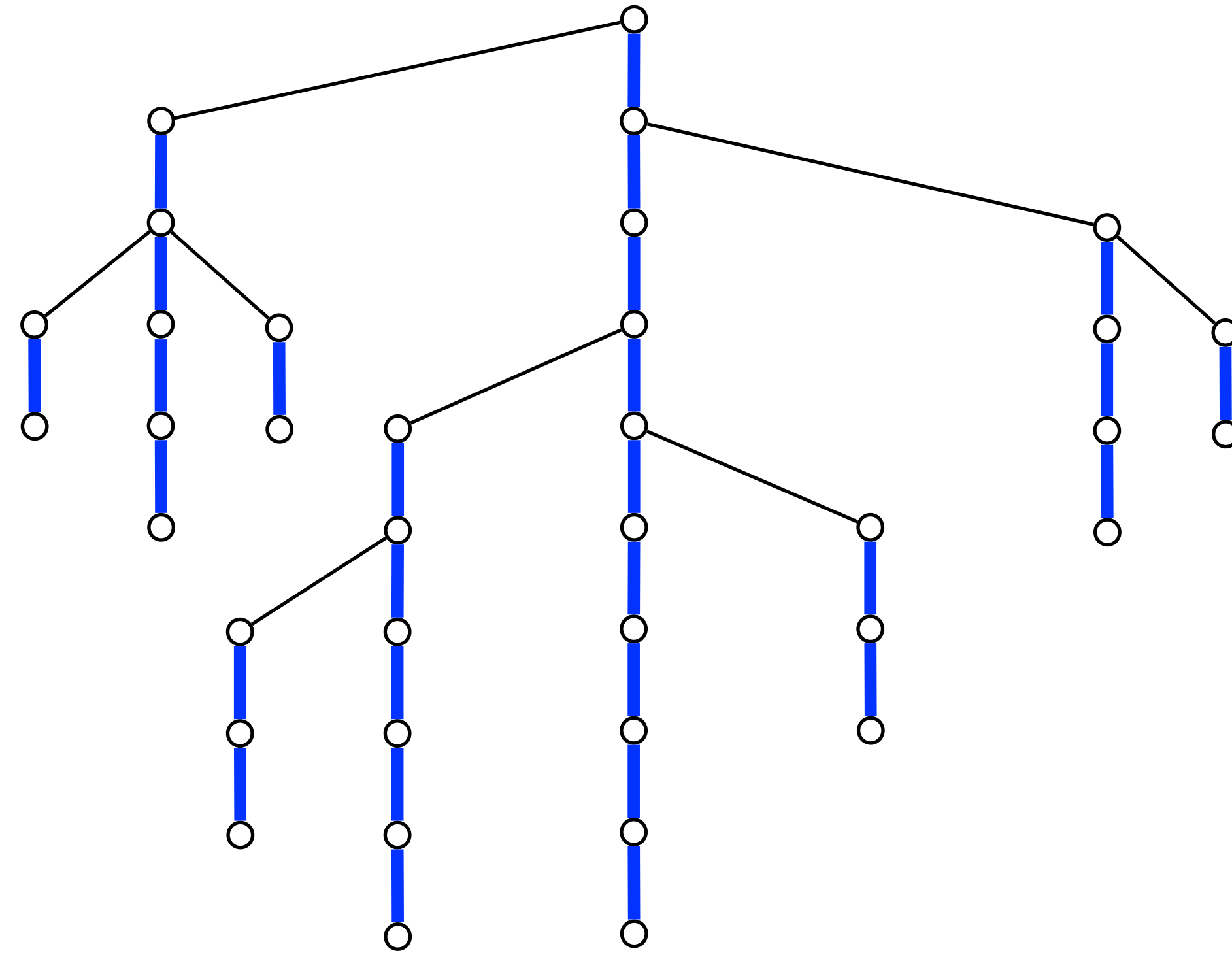


$$LA(11,6) = LA(7,2) = 2$$

- **Data structure.** For each node  $v$ , store pointers to ancestors at distance 1,2,4, ..
- **$LA(v,k)$ :** Jump to most distant ancestor no further away than  $k$ . Repeat.
- **Time.**  $O(\log n)$
- **Space.**  $O(n \log n)$

# Solution 4: Long Path Decomposition

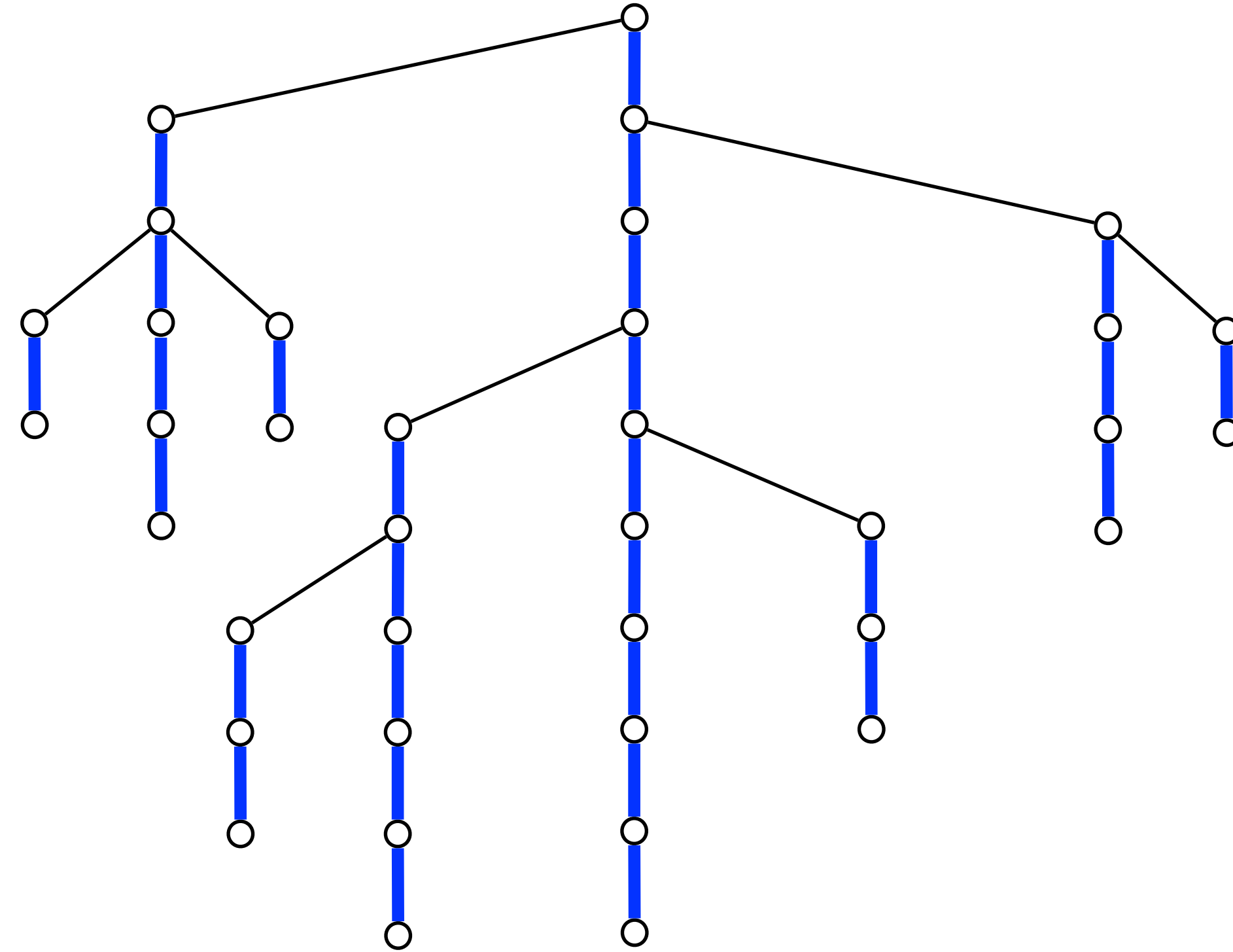
---



- Long path decomposition.
  - Find root-to-leaf path  $p$  of maximum length.
  - Recursively apply to subtrees hanging off  $p$ .
- **Lemma.** Any root-to-leaf path passes through at most  $O(n^{1/2})$  long paths.
- Longest paths partition  $T \Rightarrow$  total length (number of nodes) of all longest paths is  $= n$

# Solution 4: Long Path Decomposition

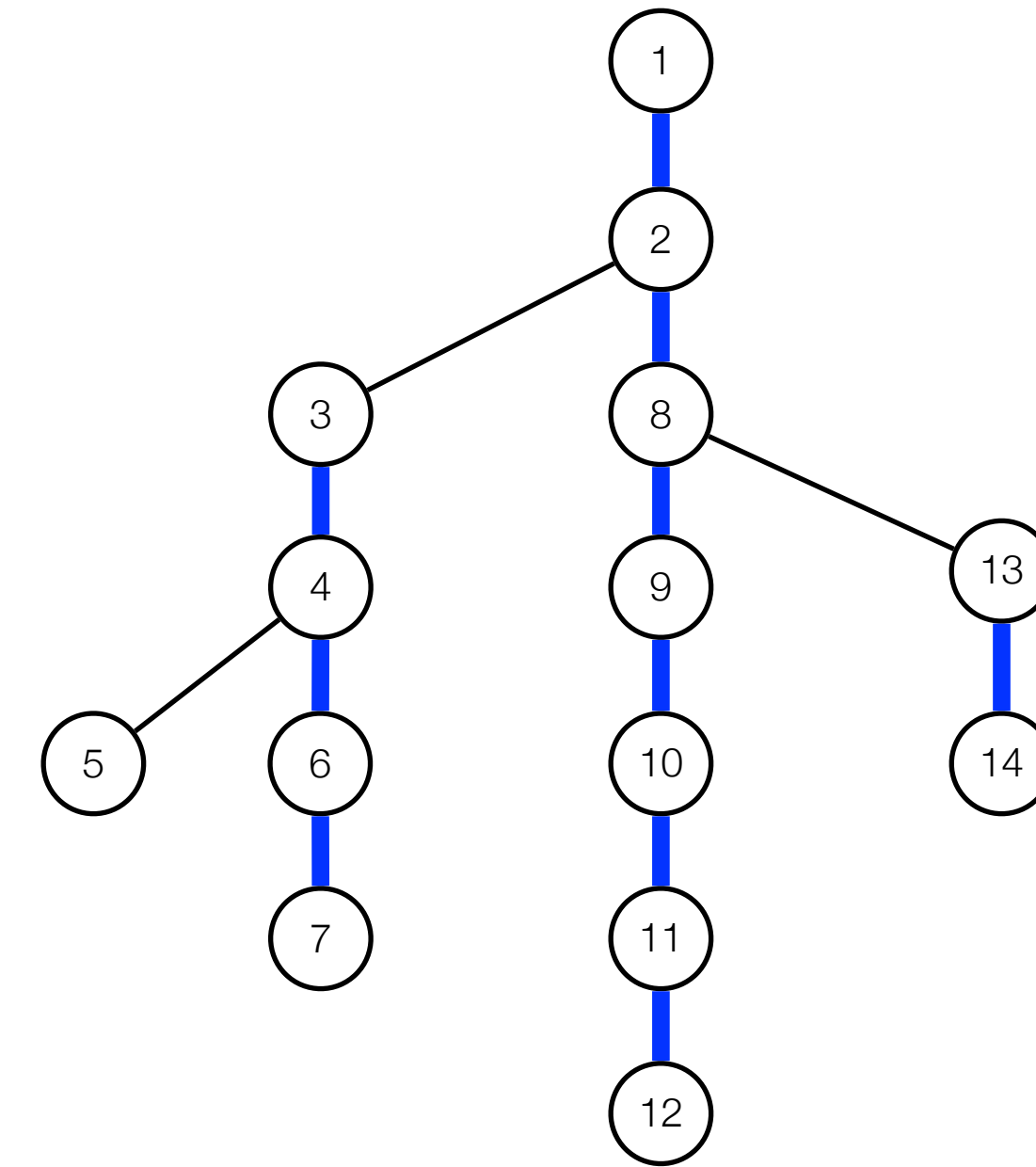
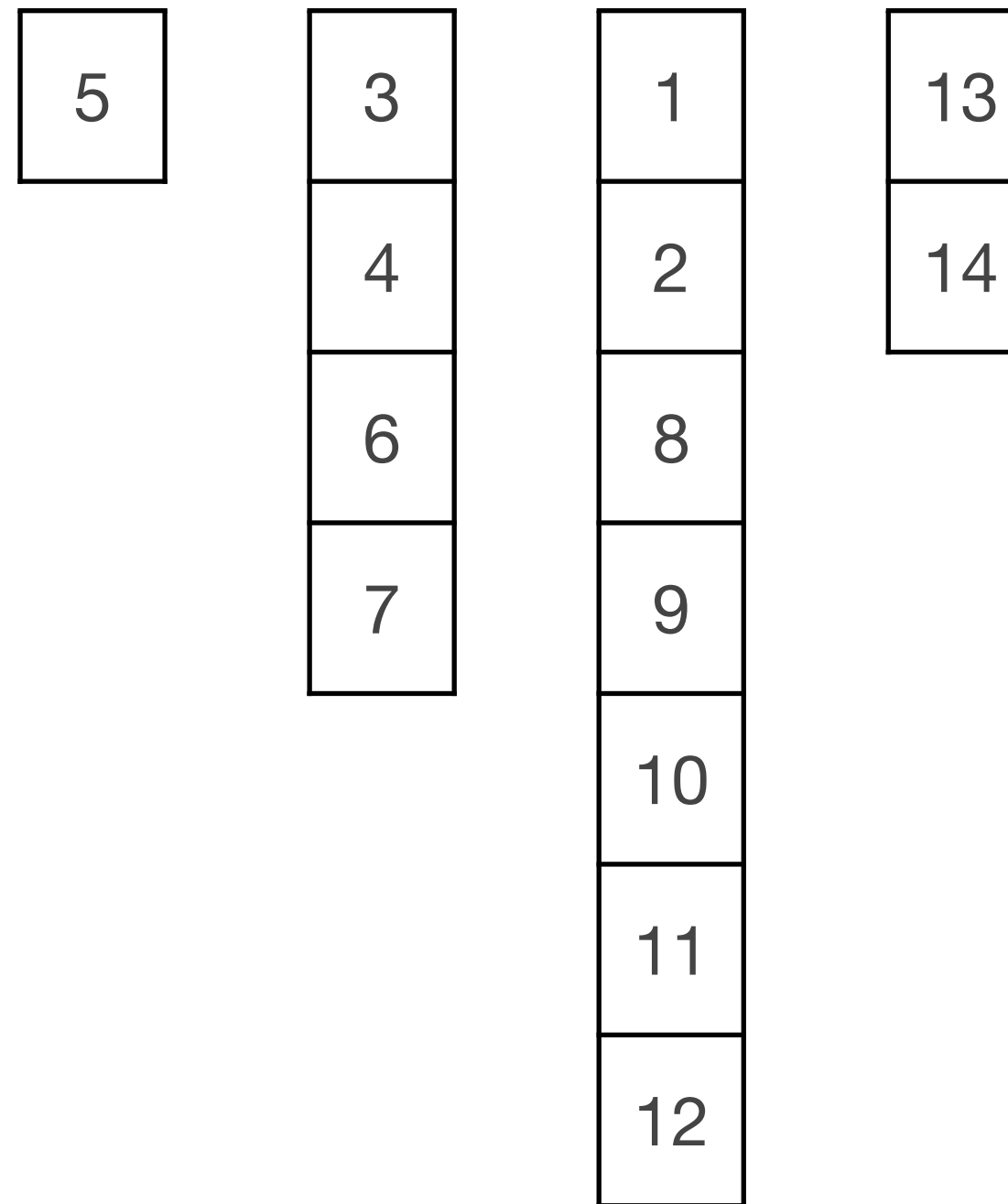
---



- **Data structure.** Store each long path in array.
- **LA(v,k):** Jump to kth ancestor or root of long path. Repeat.
- **Time.**  $O(n^{1/2})$
- **Space.**  $O(n)$

# Solution 4: Long Path Decomposition

---

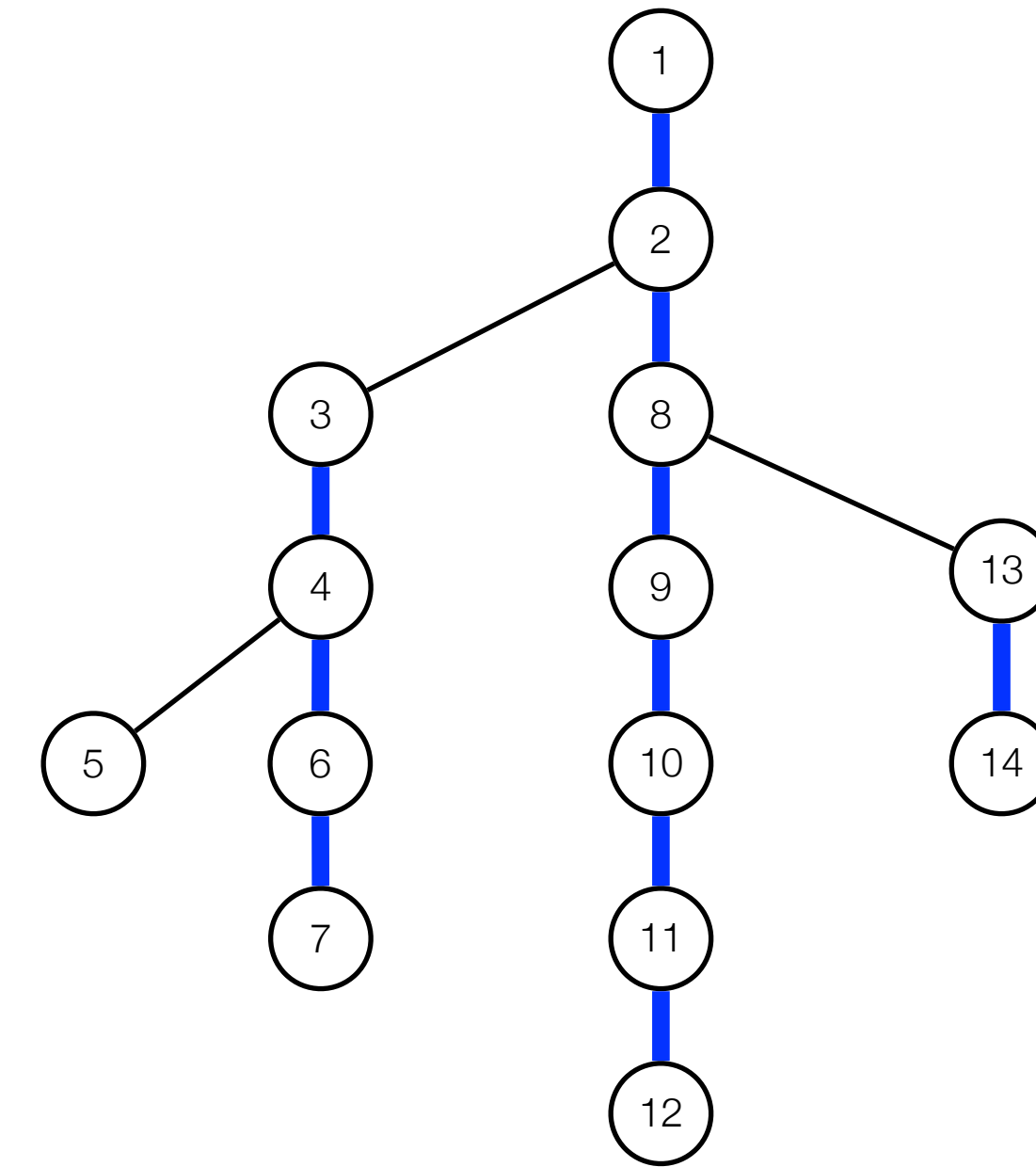
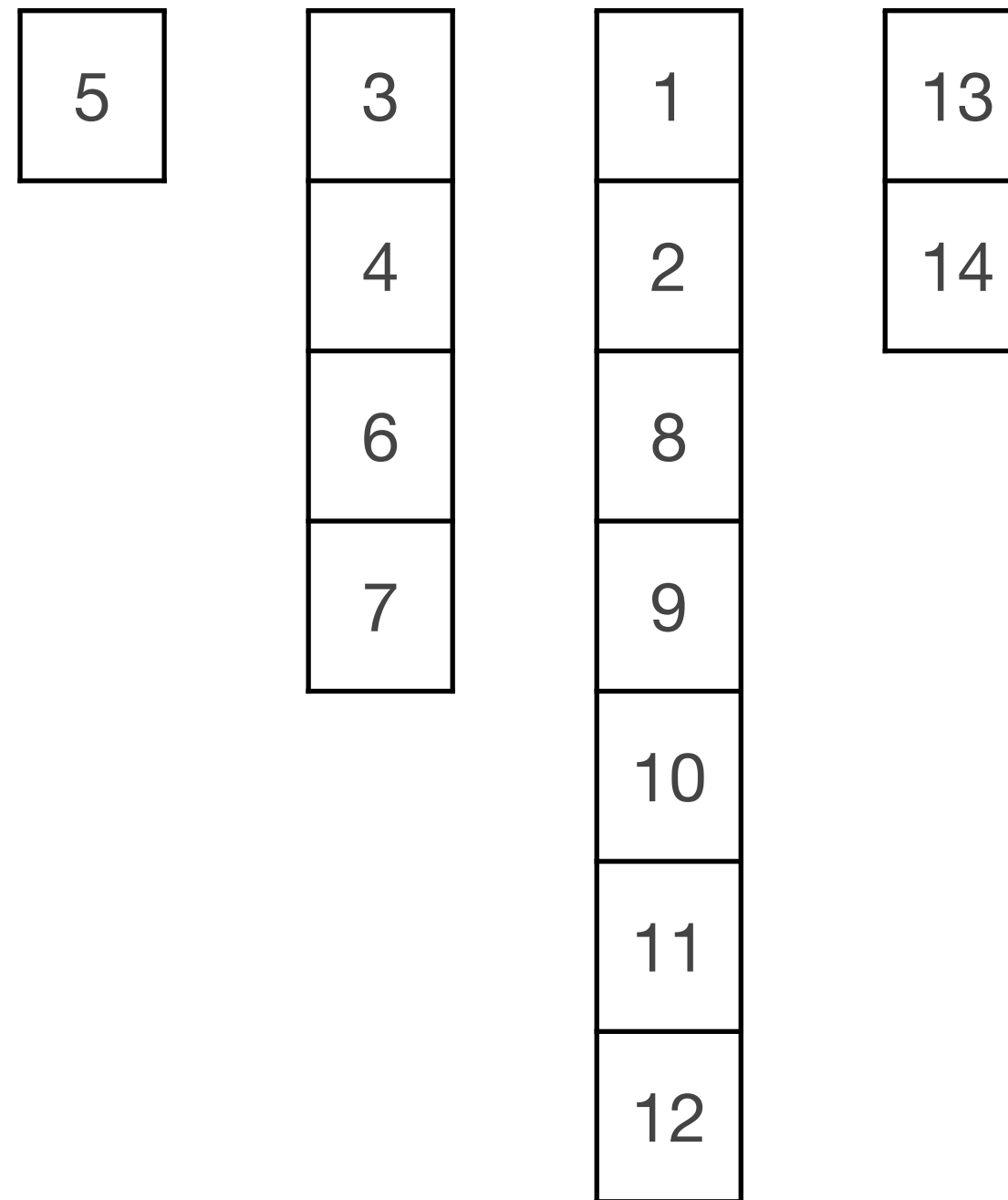


- **Data structure.** Store each long path in array.
- **LA(v,k):** Jump to kth ancestor or root of long path. Repeat.
- **Time.**  $O(n^{1/2})$
- **Space.**  $O(n)$

# Solution 4: Long Path Decomposition

---

LA(5,4)



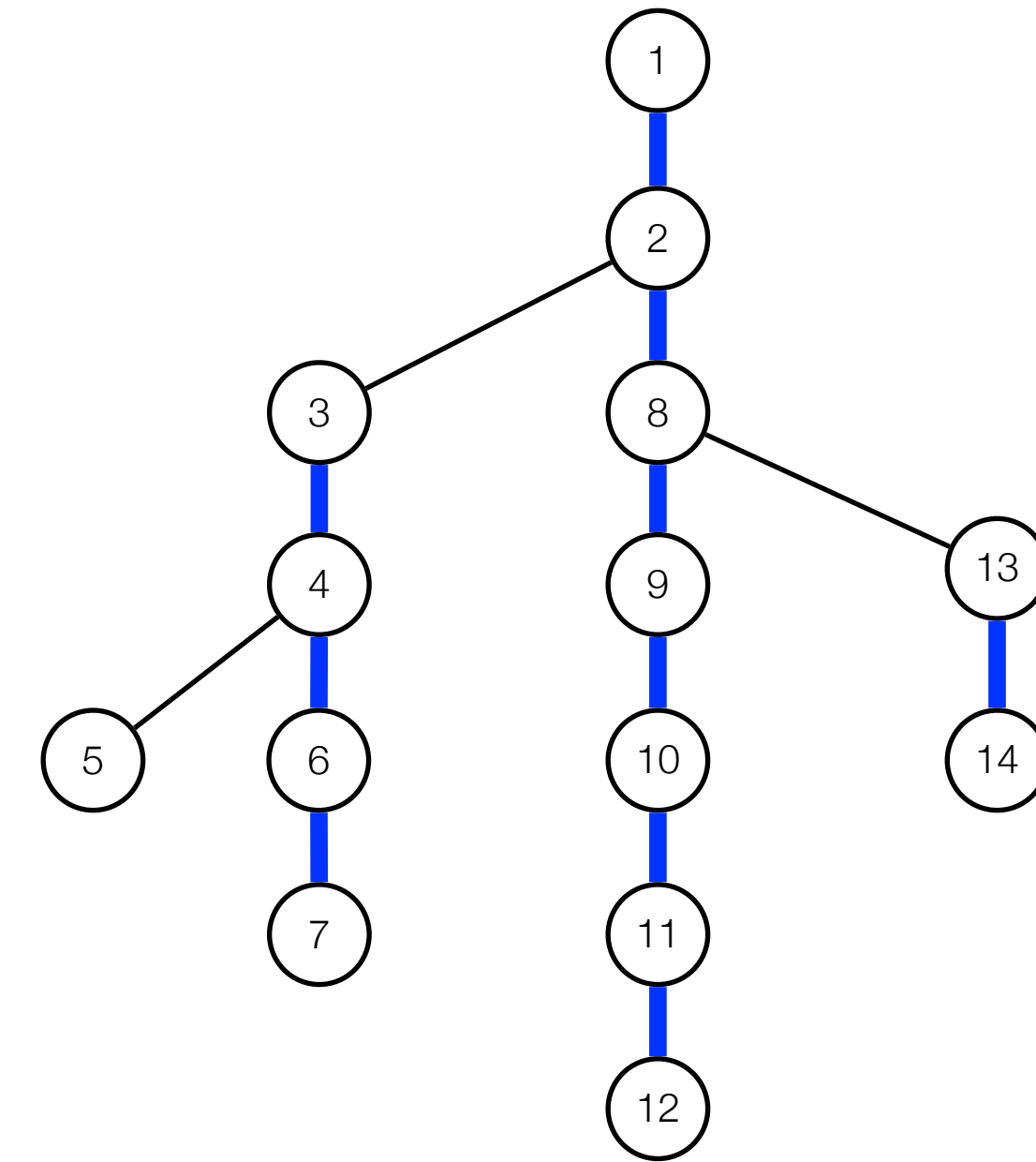
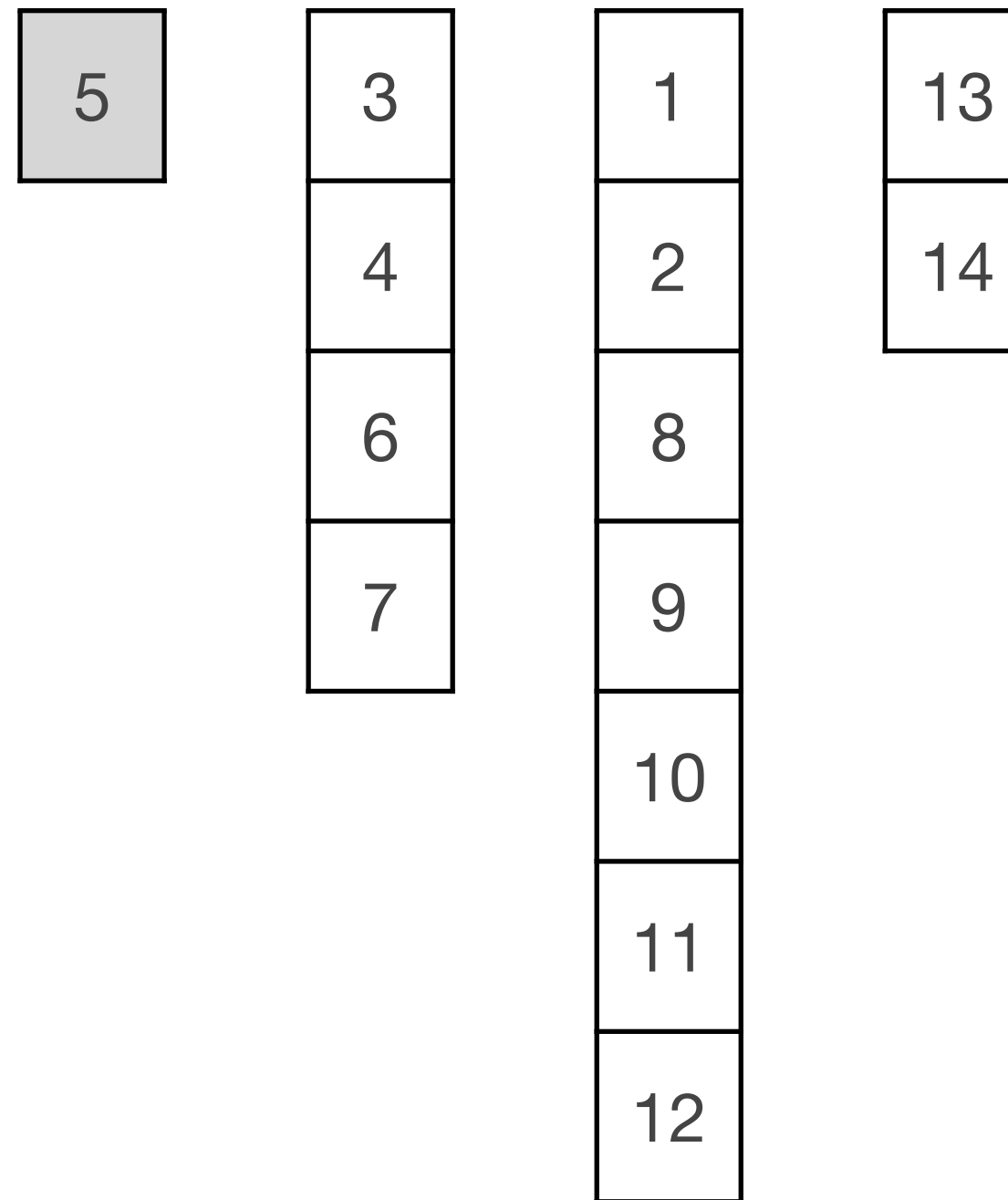
- **Data structure.** Store each long path in array.
- **LA(v,k):** Jump to kth ancestor or root of long path. Repeat.
- **Time.**  $O(n^{1/2})$
- **Space.**  $O(n)$



# Solution 4: Long Path Decomposition

---

LA(5,4)

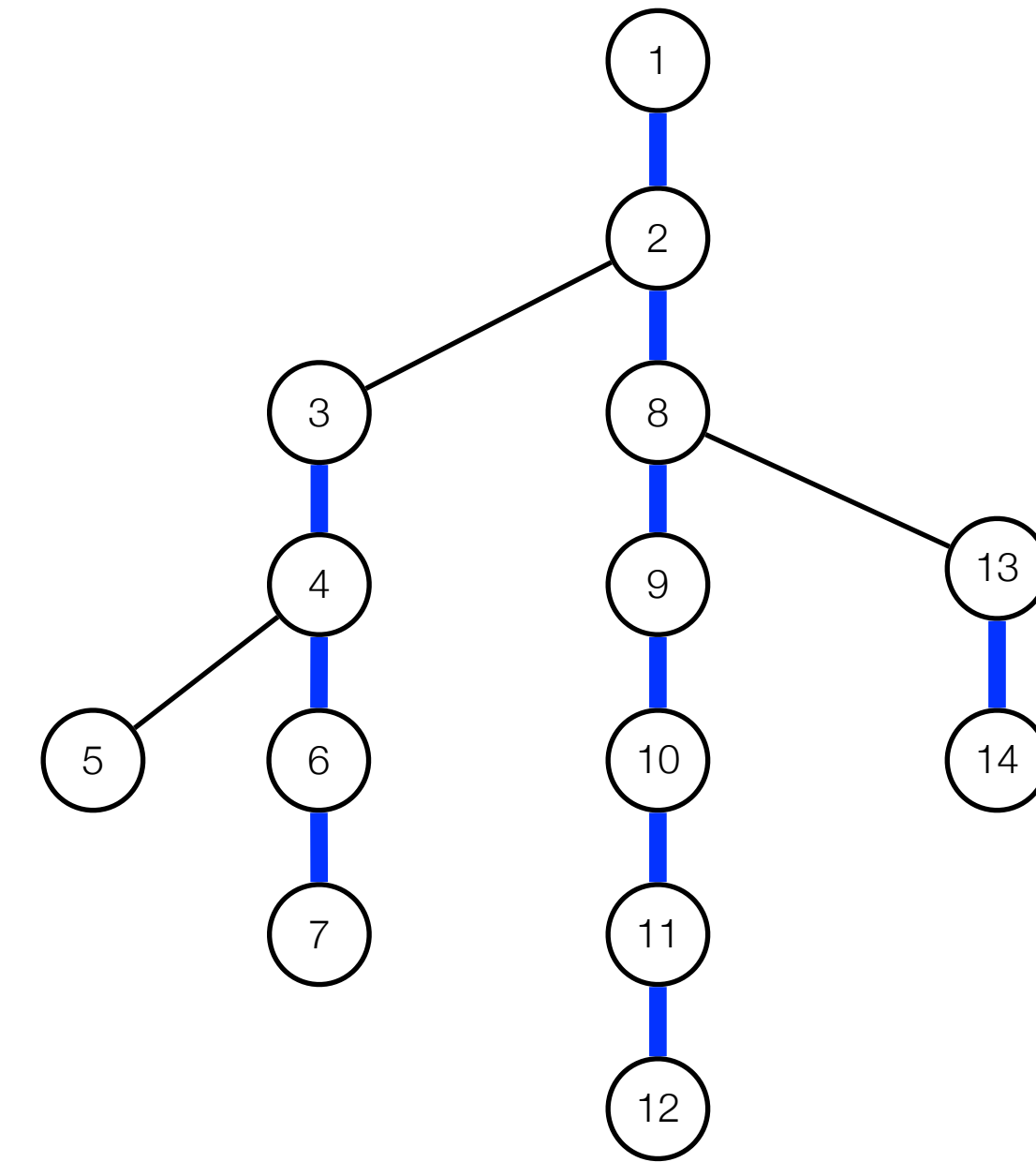
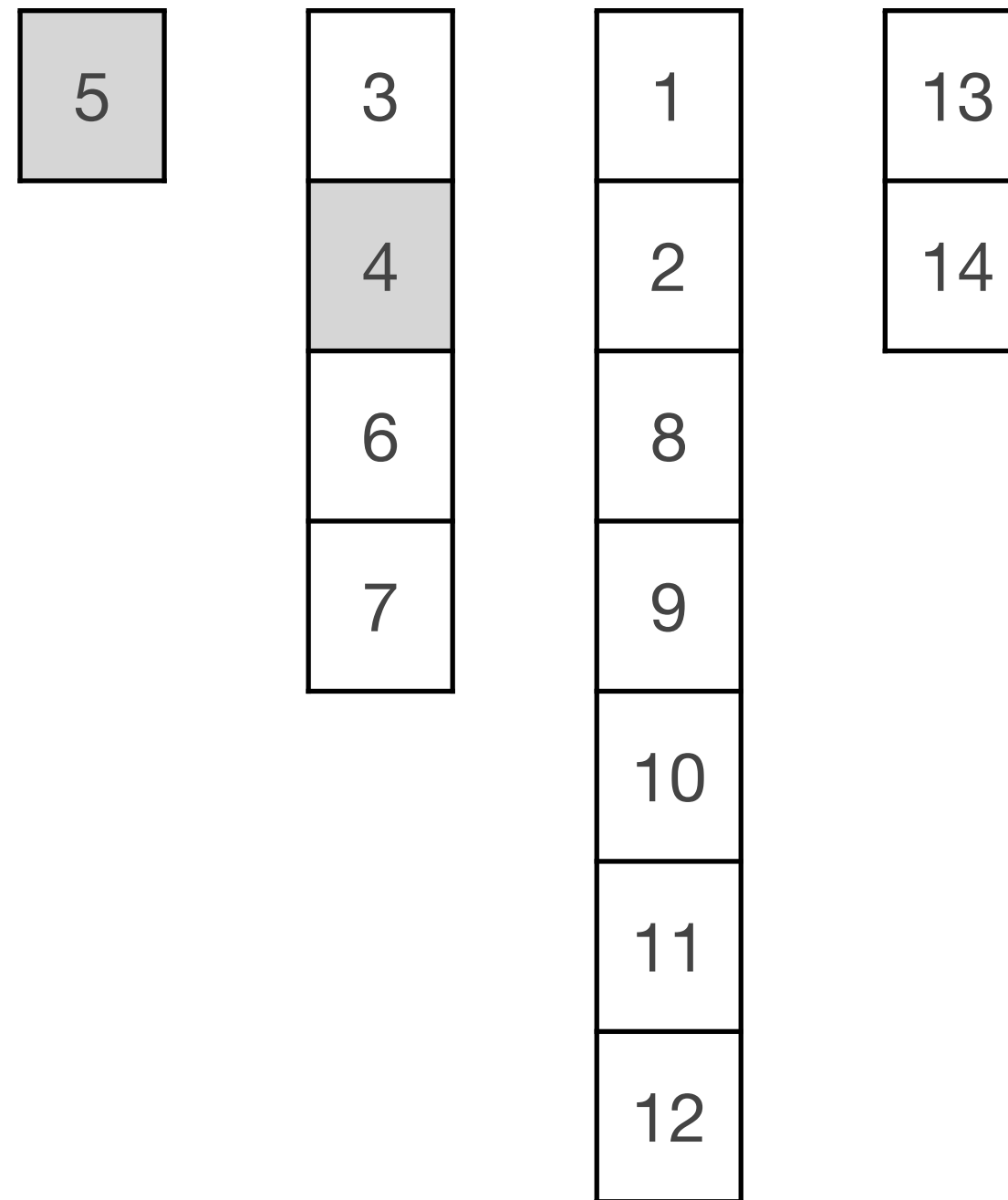


- **Data structure.** Store each long path in array.
- **LA(v,k):** Jump to kth ancestor or root of long path. Repeat.
- **Time.**  $O(n^{1/2})$
- **Space.**  $O(n)$

# Solution 4: Long Path Decomposition

---

$$LA(5,4) = LA(4,3)$$

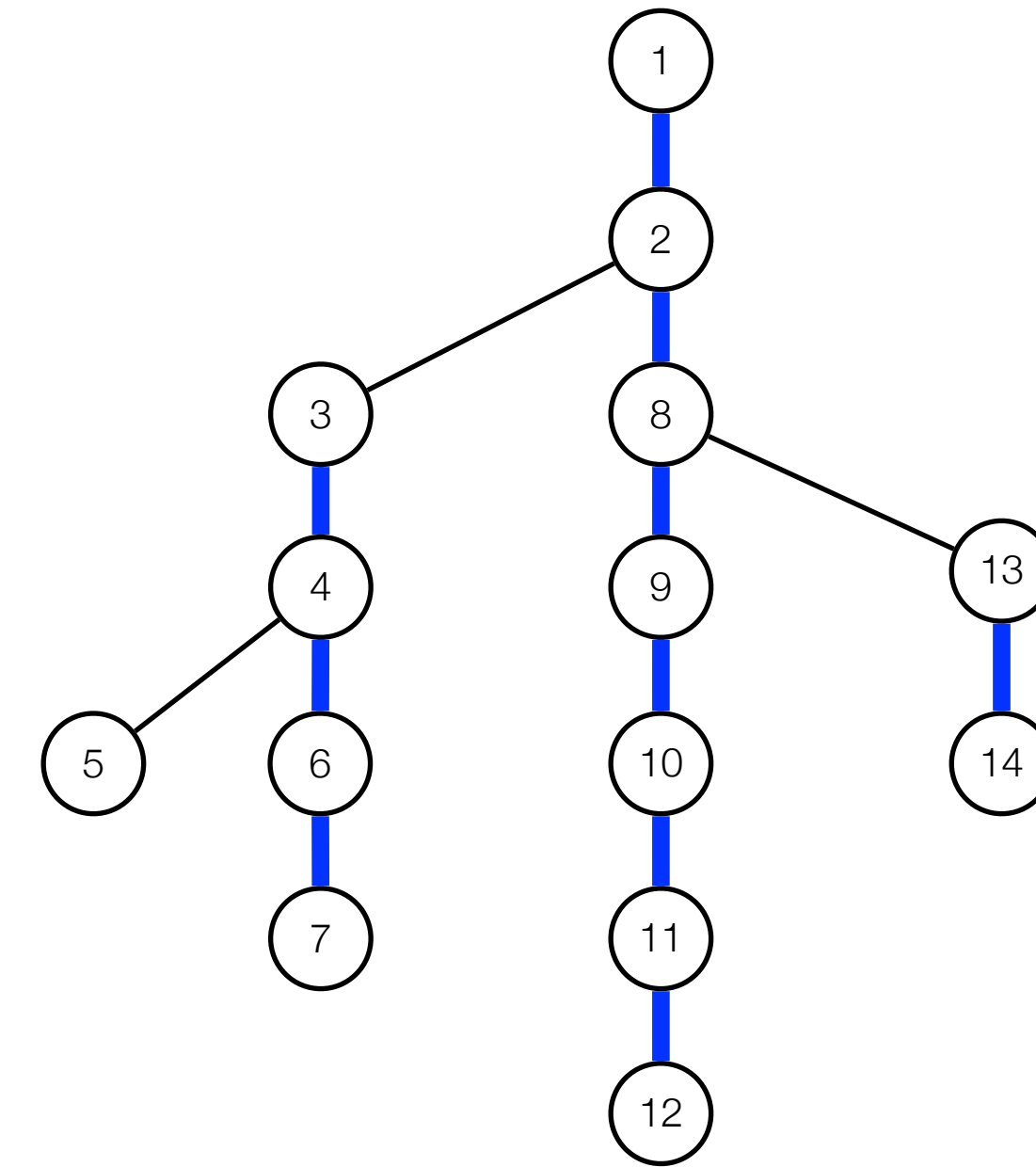
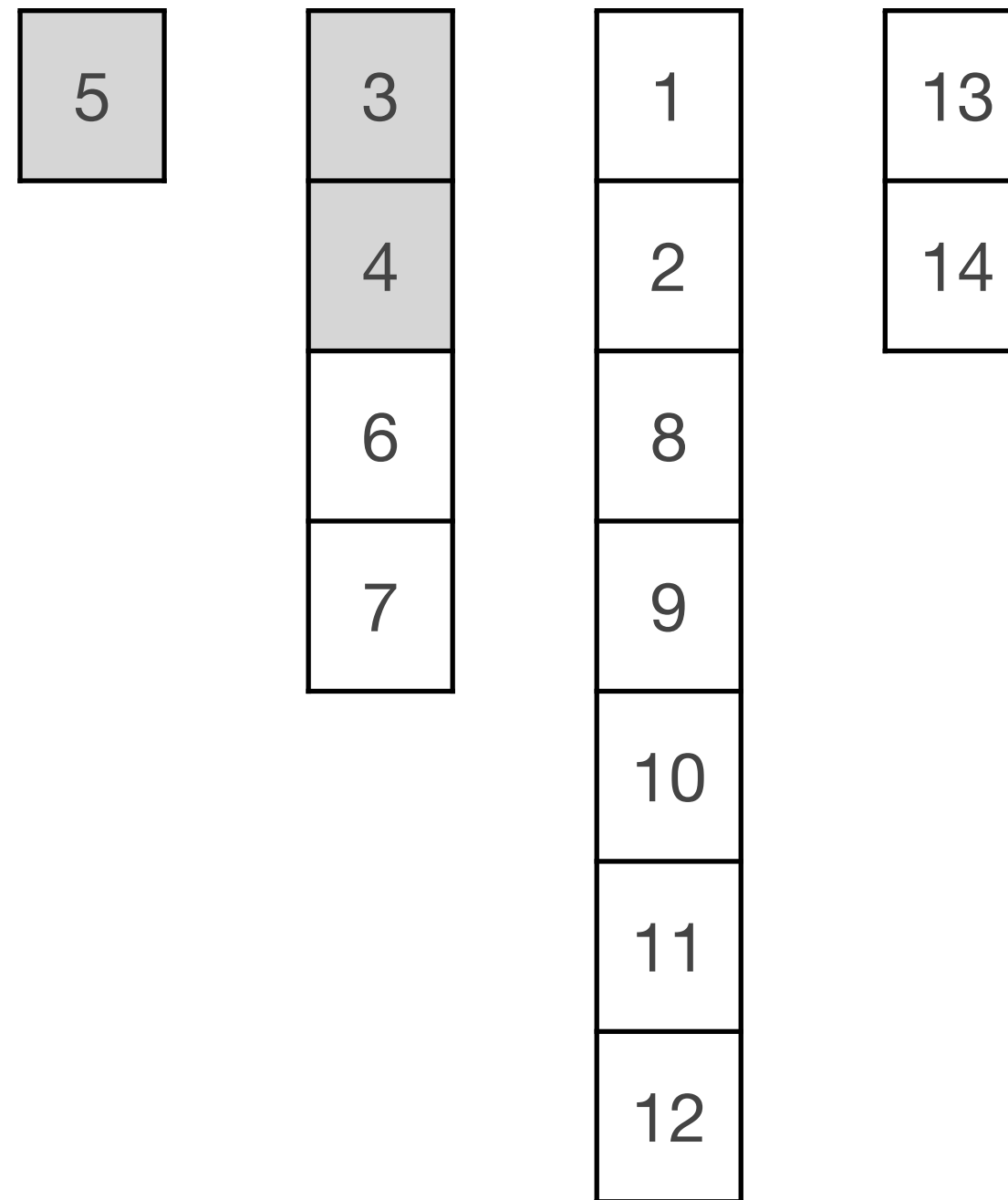


- **Data structure.** Store each long path in array.
- **LA(v,k):** Jump to kth ancestor or root of long path. Repeat.
- **Time.**  $O(n^{1/2})$
- **Space.**  $O(n)$

# Solution 4: Long Path Decomposition

---

$$LA(5,4) = LA(4,3) = LA(3,2)$$

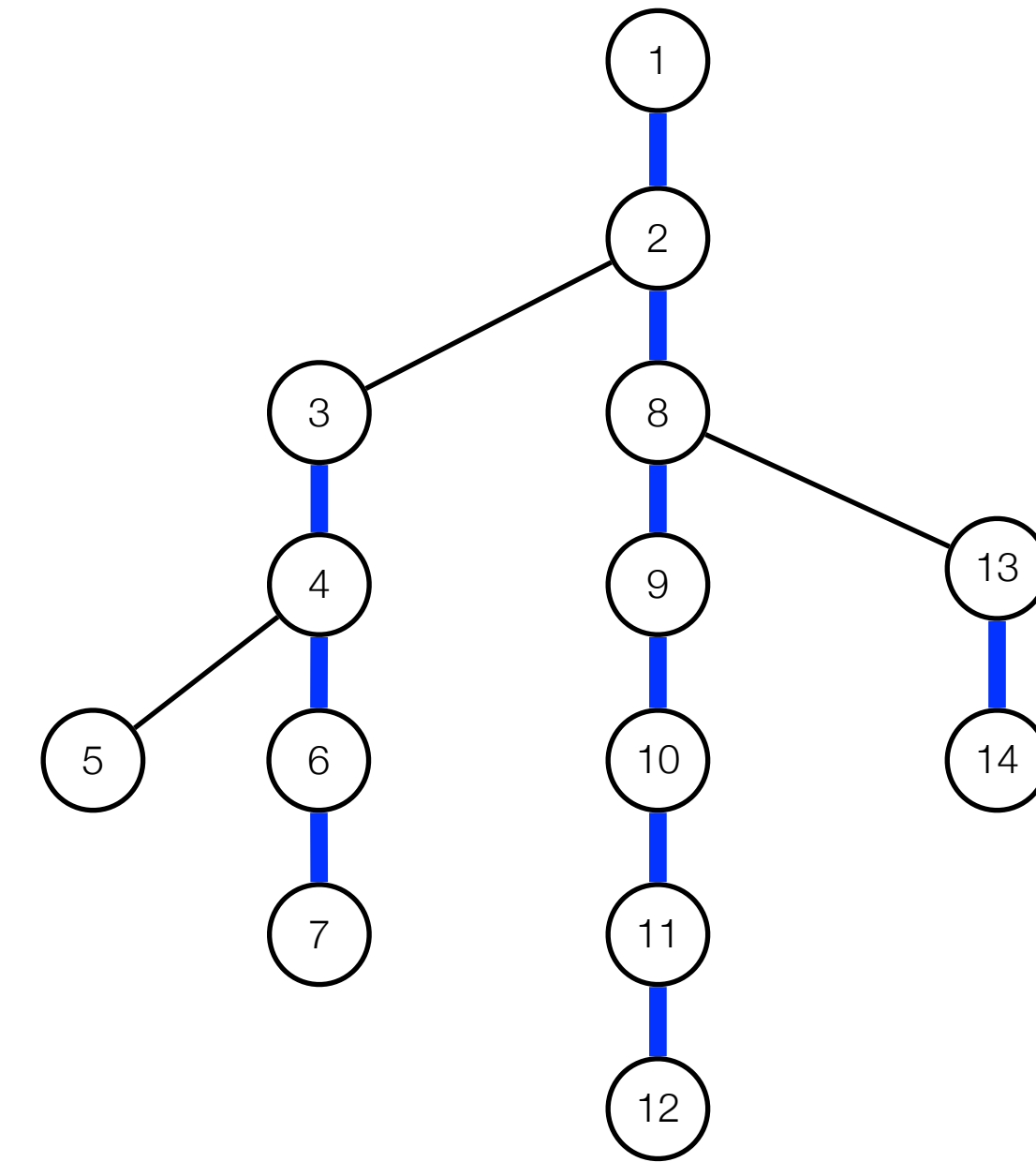
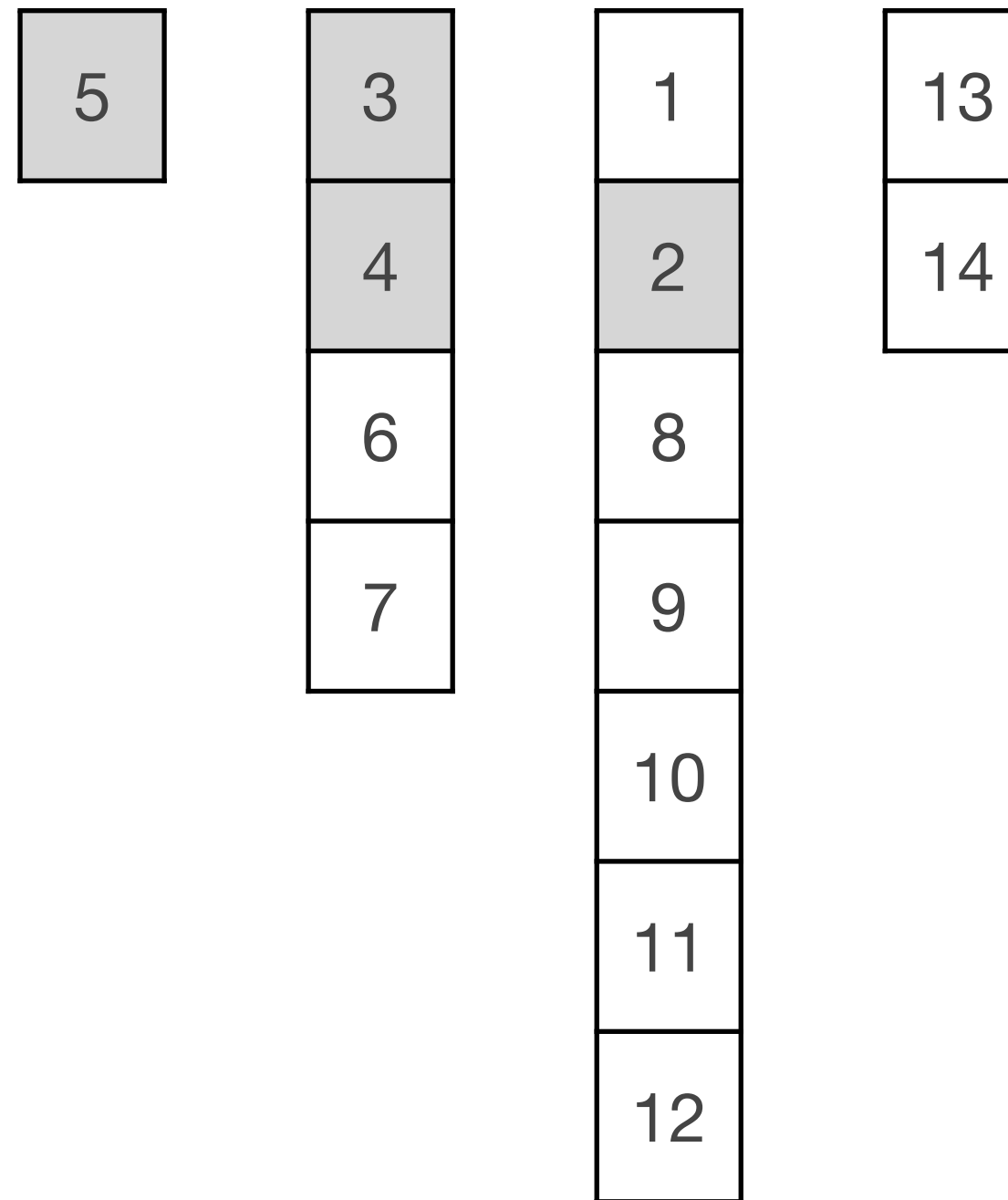


- **Data structure.** Store each long path in array.
- **LA(v,k):** Jump to kth ancestor or root of long path. Repeat.
- **Time.**  $O(n^{1/2})$
- **Space.**  $O(n)$

# Solution 4: Long Path Decomposition

---

$$LA(5,4) = LA(4,3) = LA(3,2) = LA(2,1)$$

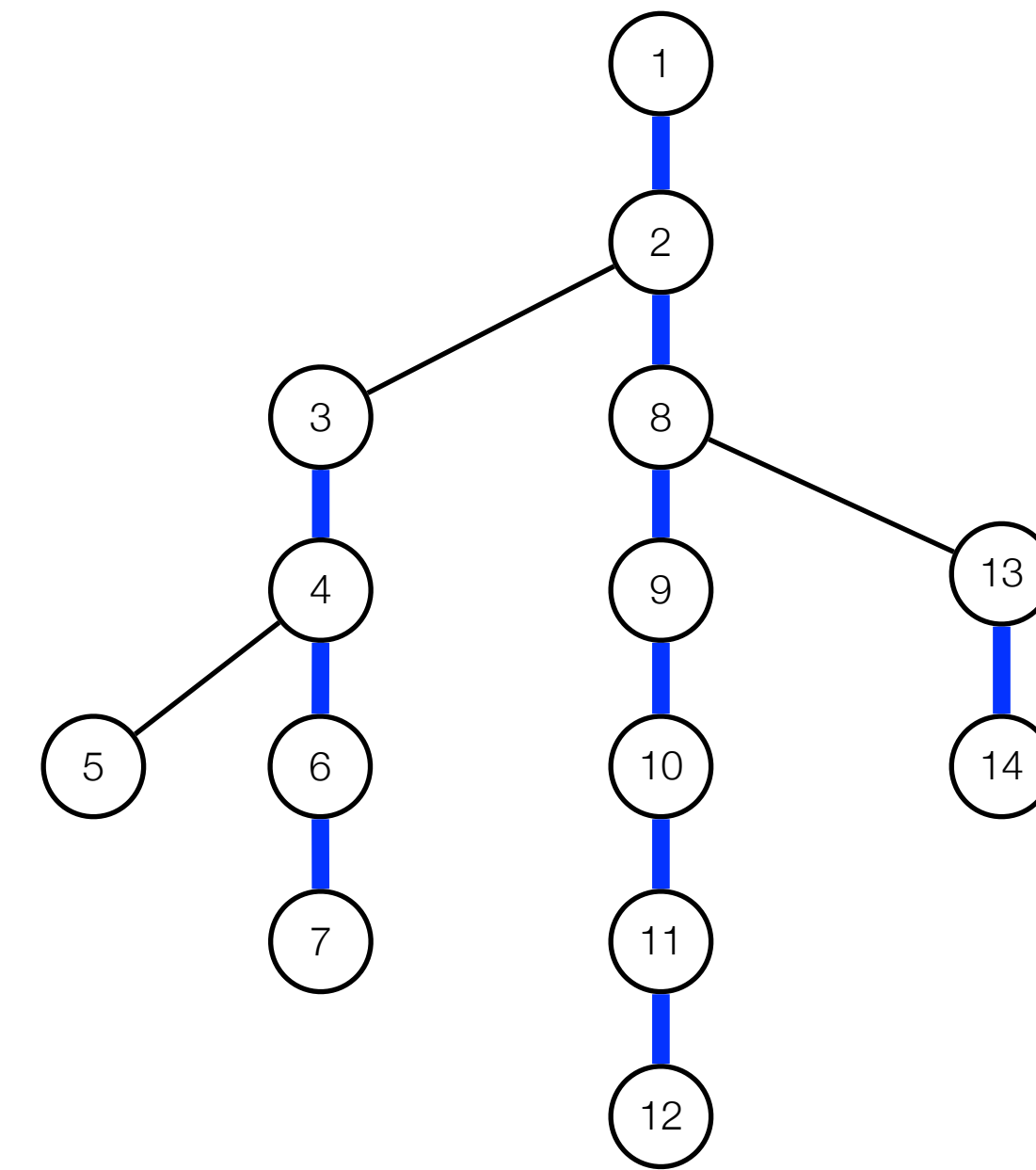
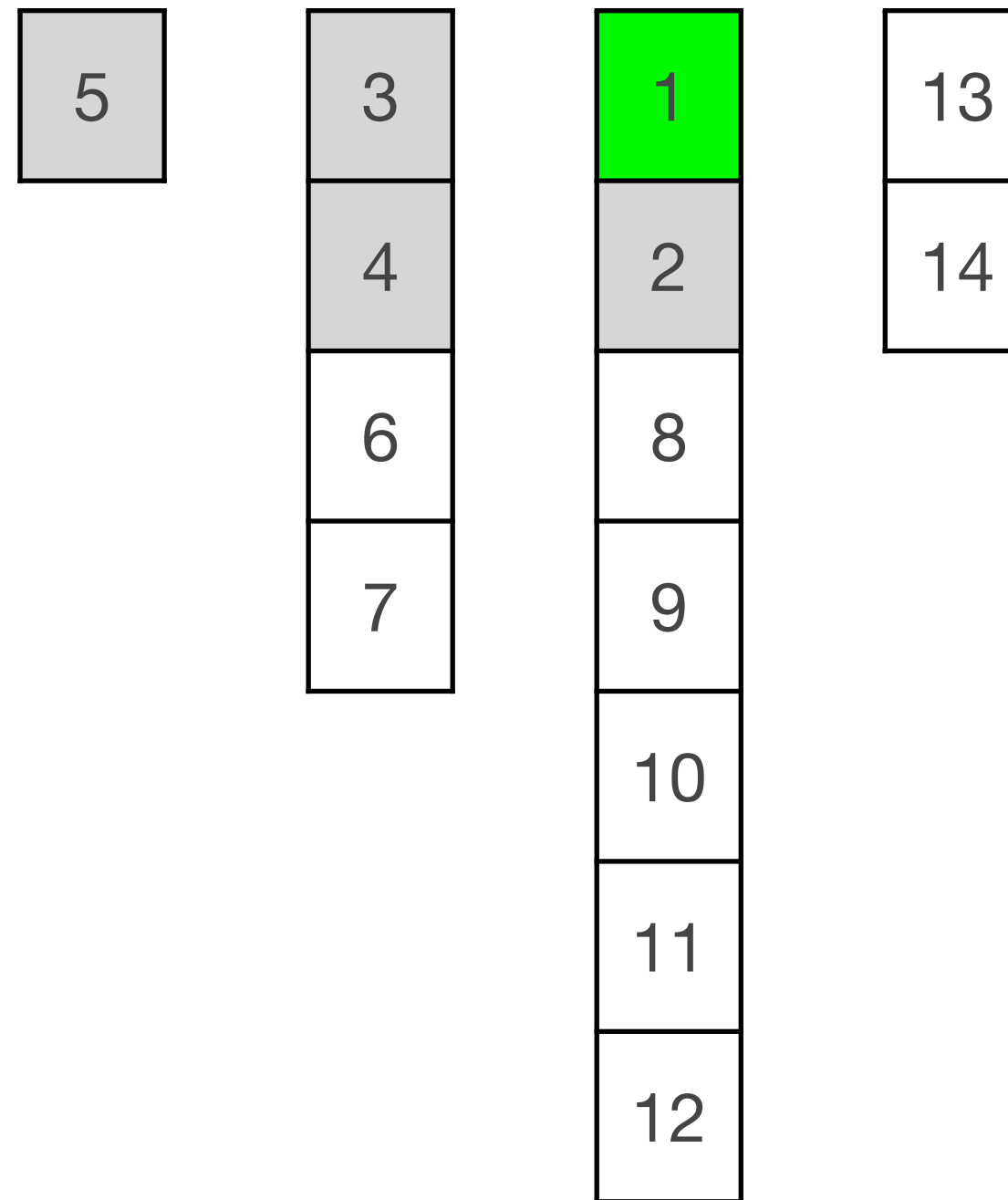


- **Data structure.** Store each long path in array.
- **$LA(v,k)$ :** Jump to  $k$ th ancestor or root of long path. Repeat.
- **Time.**  $O(n^{1/2})$
- **Space.**  $O(n)$

# Solution 4: Long Path Decomposition

---

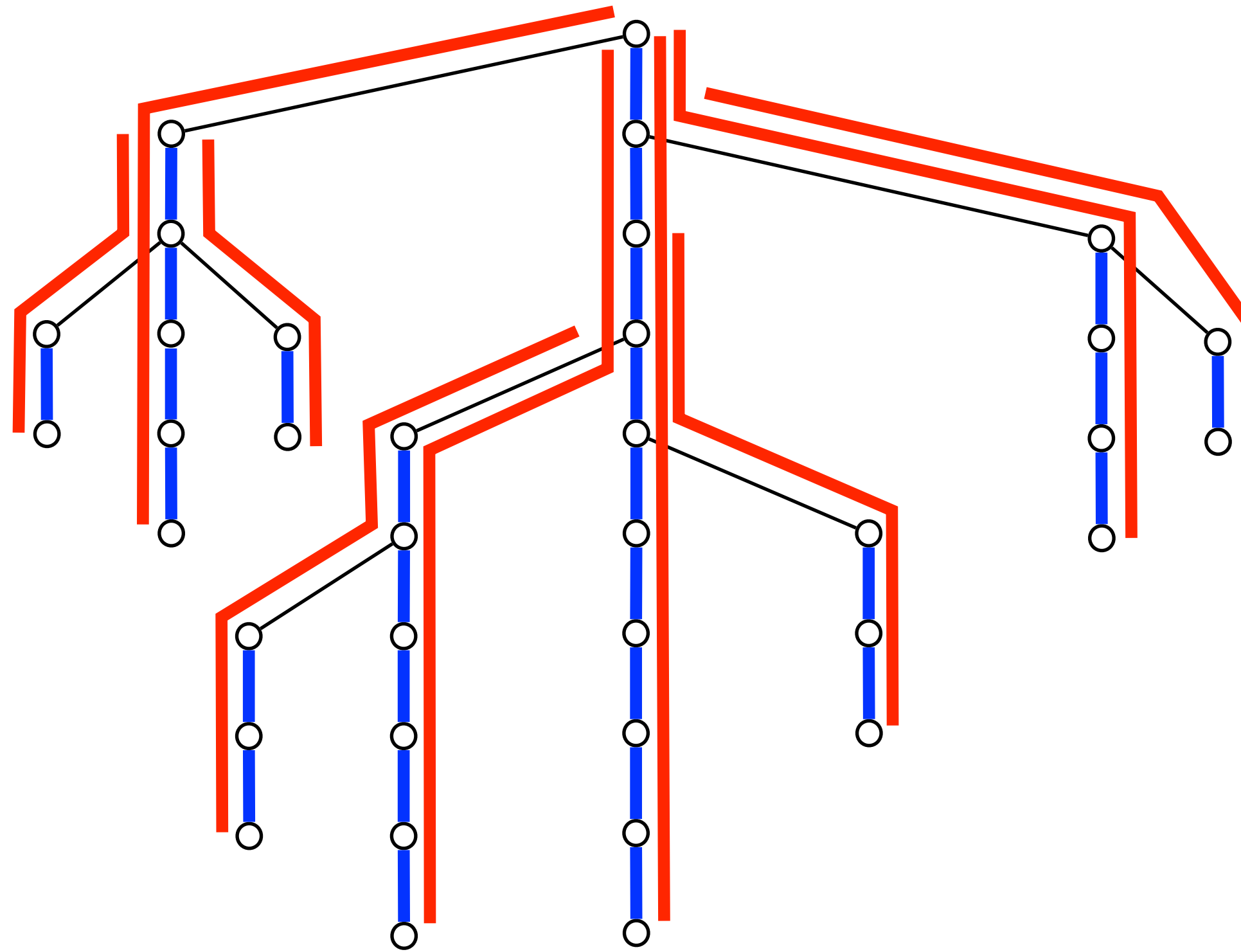
$$LA(5,4) = LA(4,4) = LA(3,3) = LA(2,1) = 1$$



- **Data structure.** Store each long path in array.
- **$LA(v,k)$ :** Jump to  $k$ th ancestor or root of long path. Repeat.
- **Time.**  $O(n^{1/2})$
- **Space.**  $O(n)$

# Solution 5: Ladder Decomposition

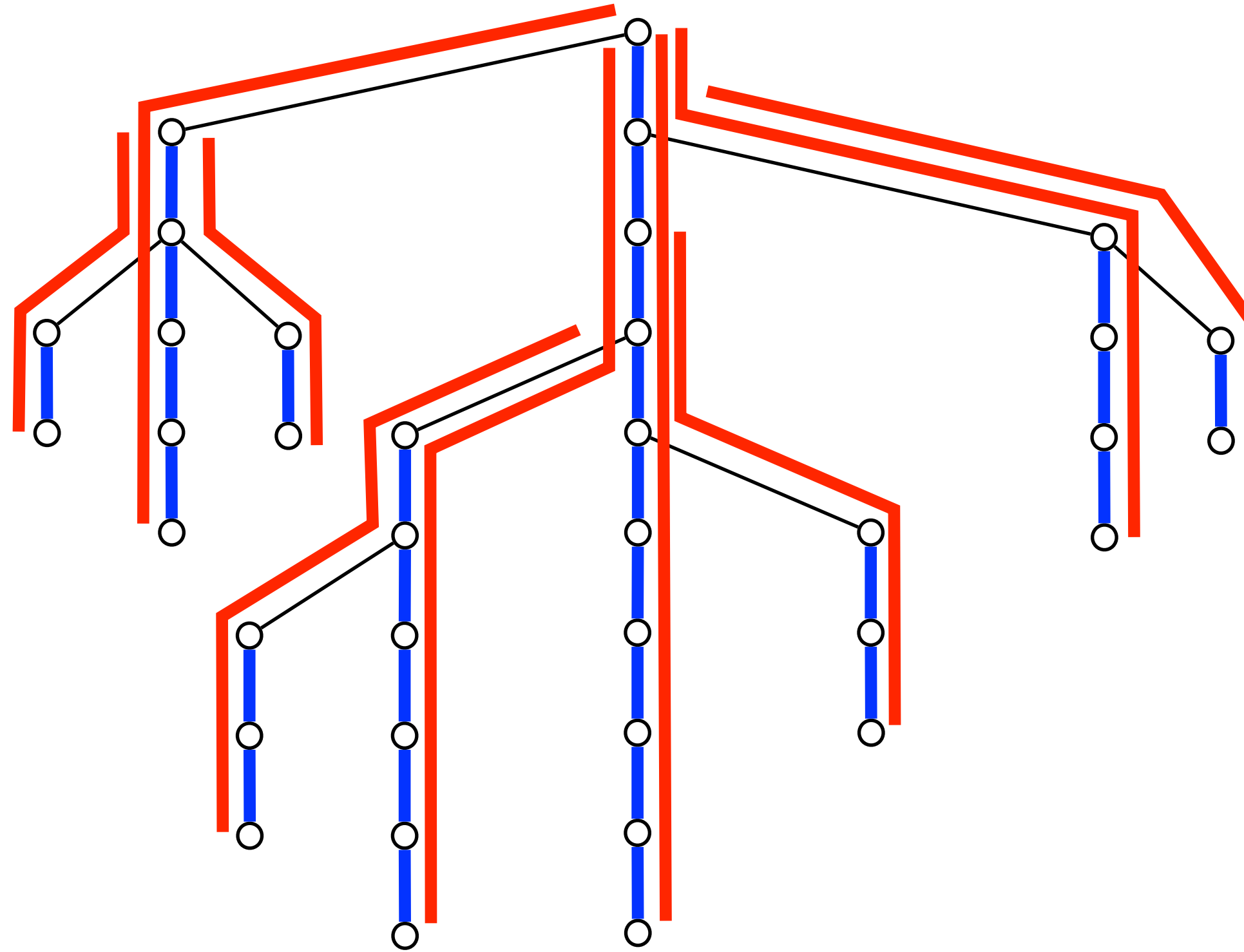
---



- Ladder decomposition.
  - Compute long path decomposition.
  - Double each long path.
- Lemma. Any root-to-leaf path passes through at most  $O(\log n)$  ladders.
- Total length of ladders is  $\leq 2n$ .

# Solution 5: Ladder Decomposition

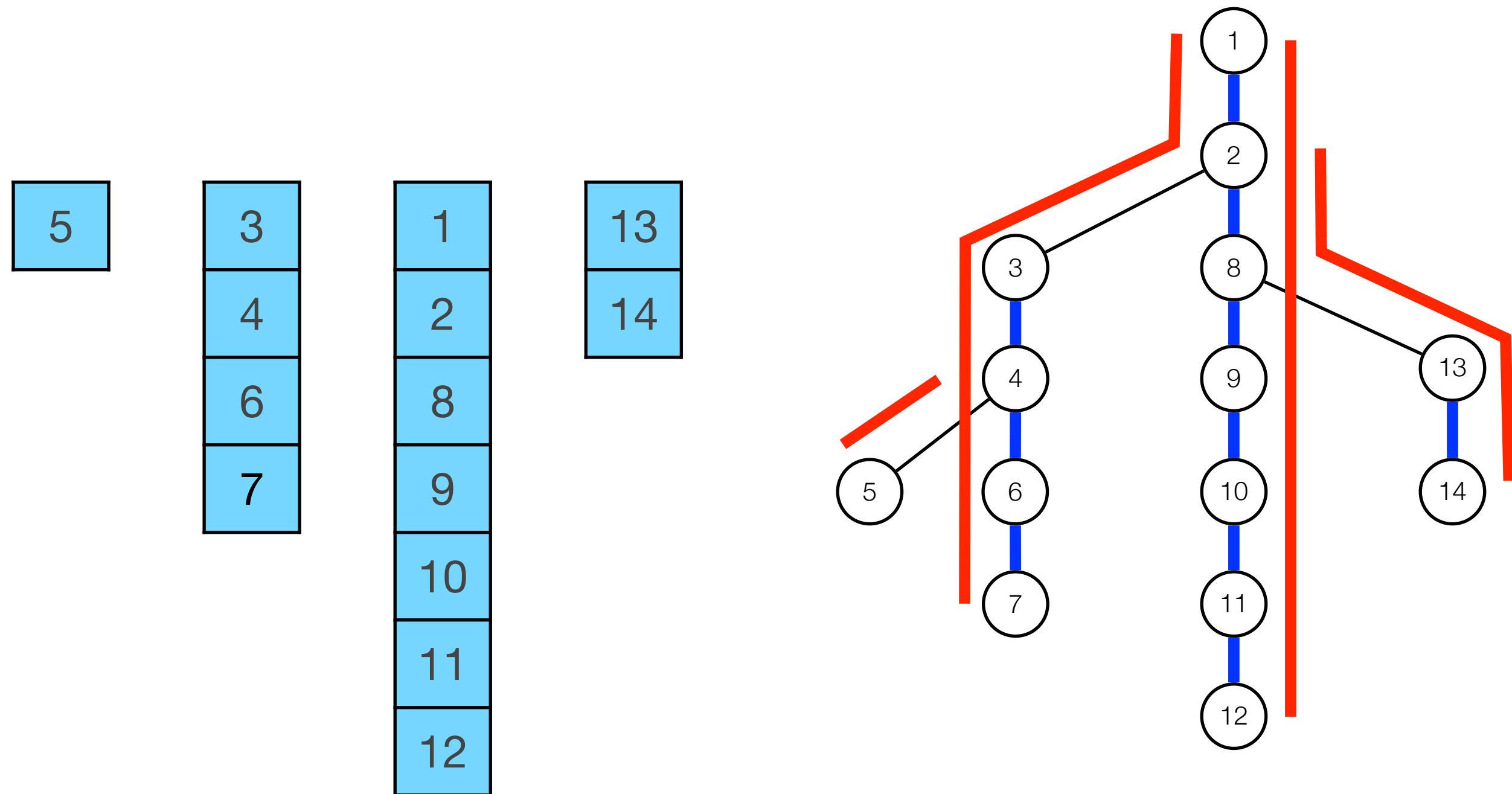
---



- **Data structure.**
  - Store each ladder in array.
  - Each node points to ladder corresponding to its longest path.
- **LA(v,k):** Jump to kth ancestor or root of ladder. Repeat.
- **Time.**  $O(\log n)$
- **Space.**  $O(n)$

# Solution 5: Ladder Decomposition

---

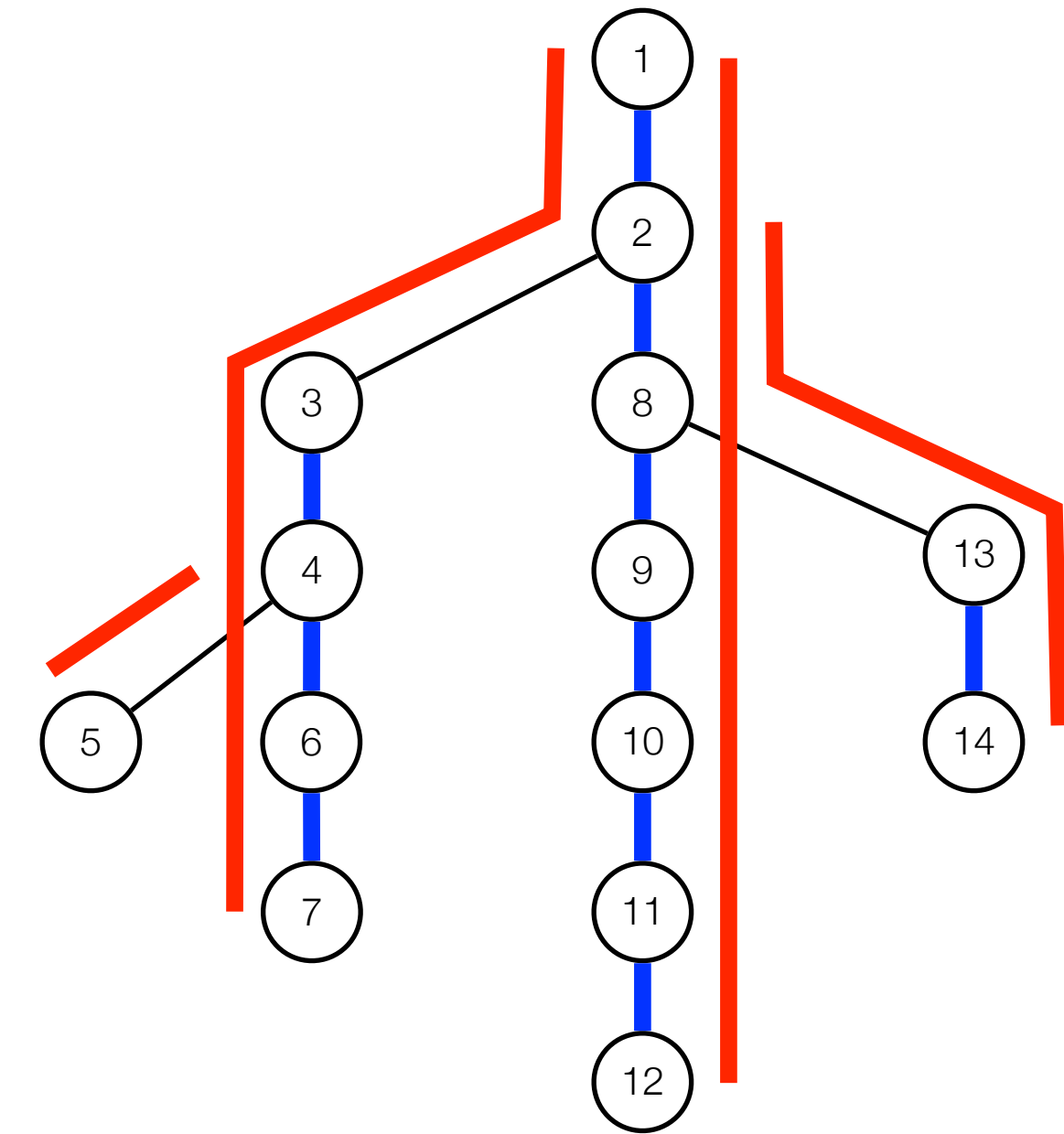
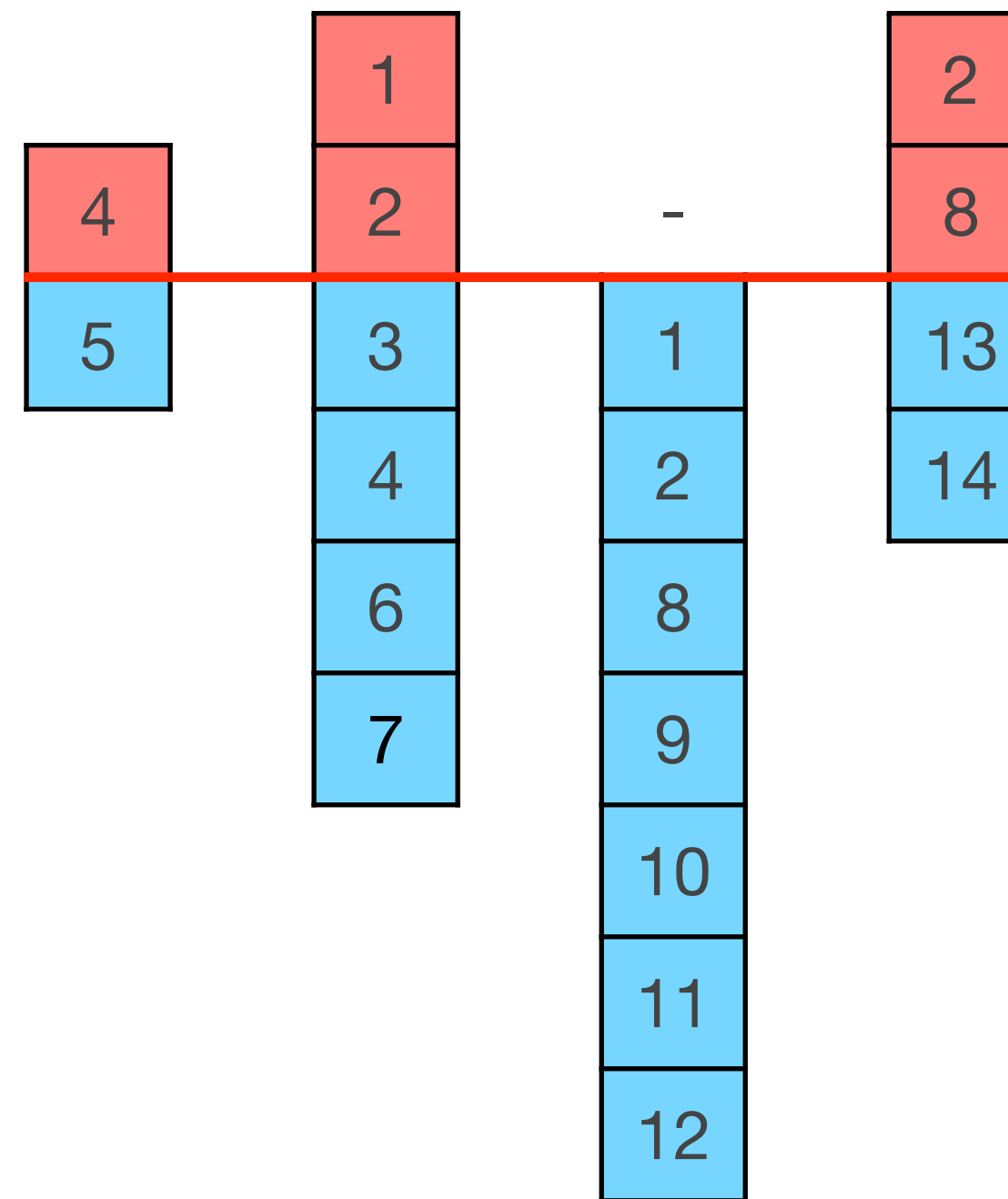


- **Data structure.**
  - Store each ladder in array.
  - Each node points to ladder corresponding to its longest path.
- **LA(v,k):** Jump to kth ancestor or root of ladder. Repeat.
- **Time.**  $O(\log n)$
- **Space.**  $O(n)$



# Solution 5: Ladder Decomposition

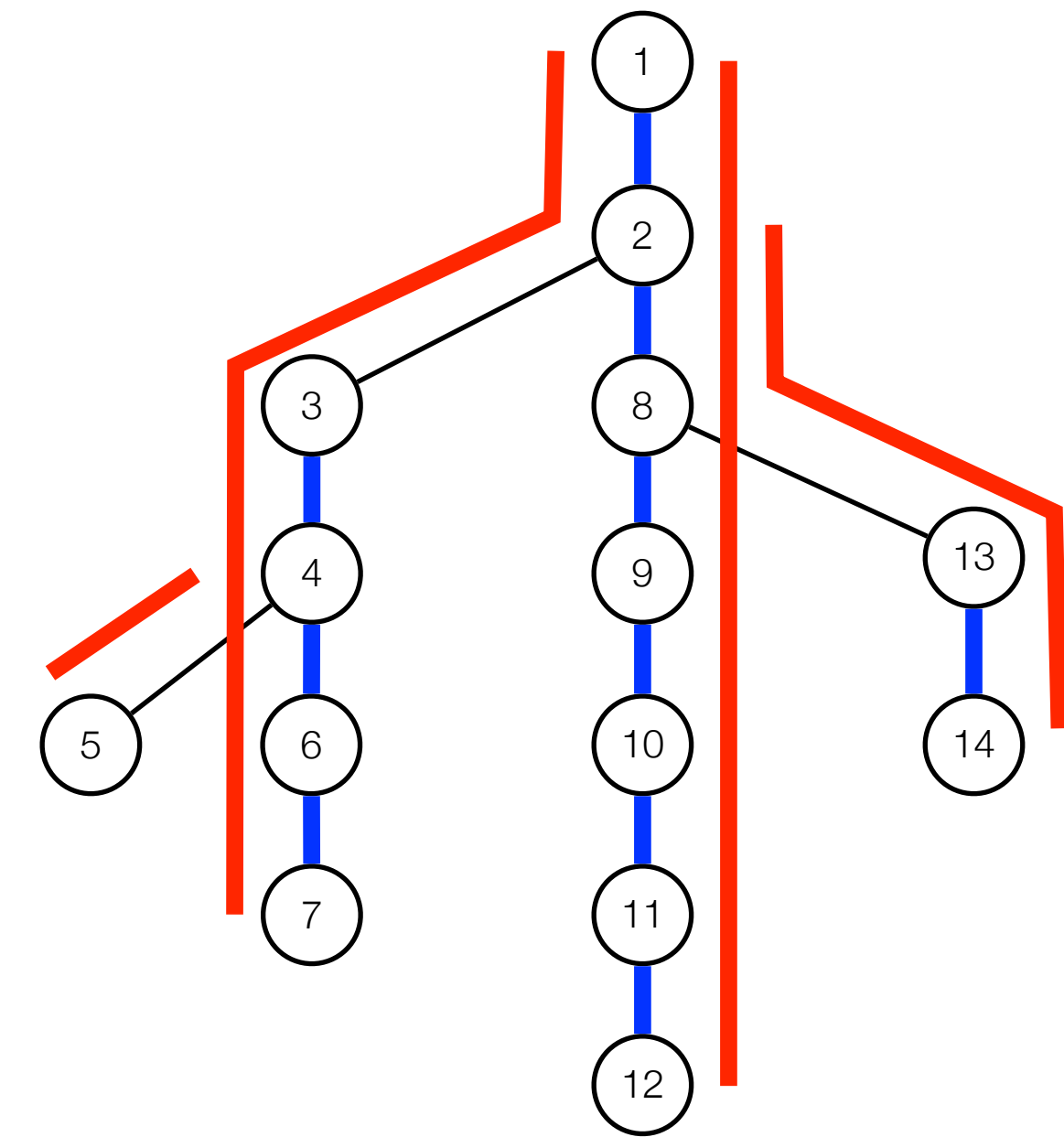
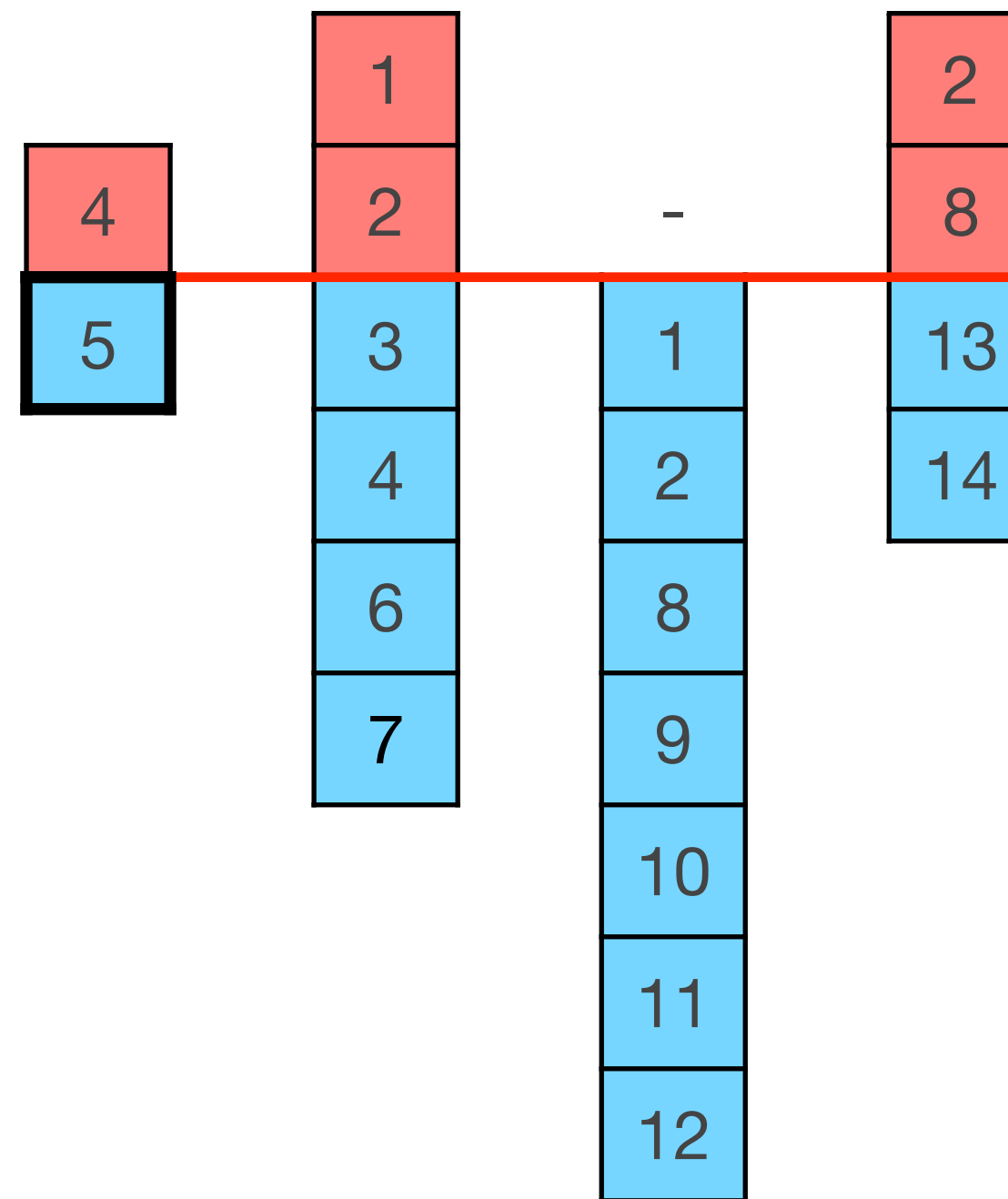
---



- **Data structure.**
  - Store each ladder in array.
  - Each node points to ladder corresponding to its longest path.
- **LA(v,k):** Jump to kth ancestor or root of ladder. Repeat.
- **Time.**  $O(\log n)$
- **Space.**  $O(n)$

# Solution 5: Ladder Decomposition

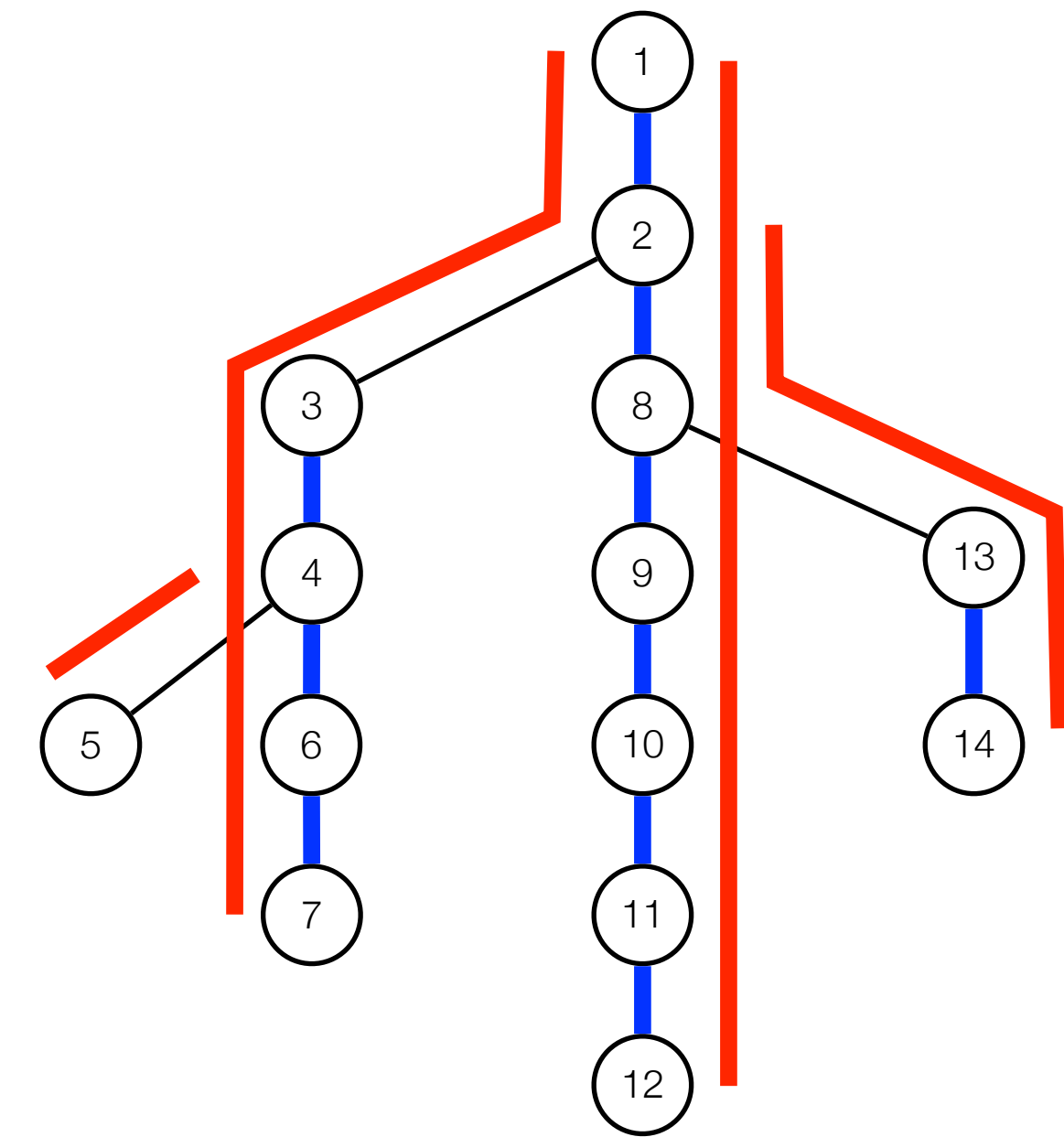
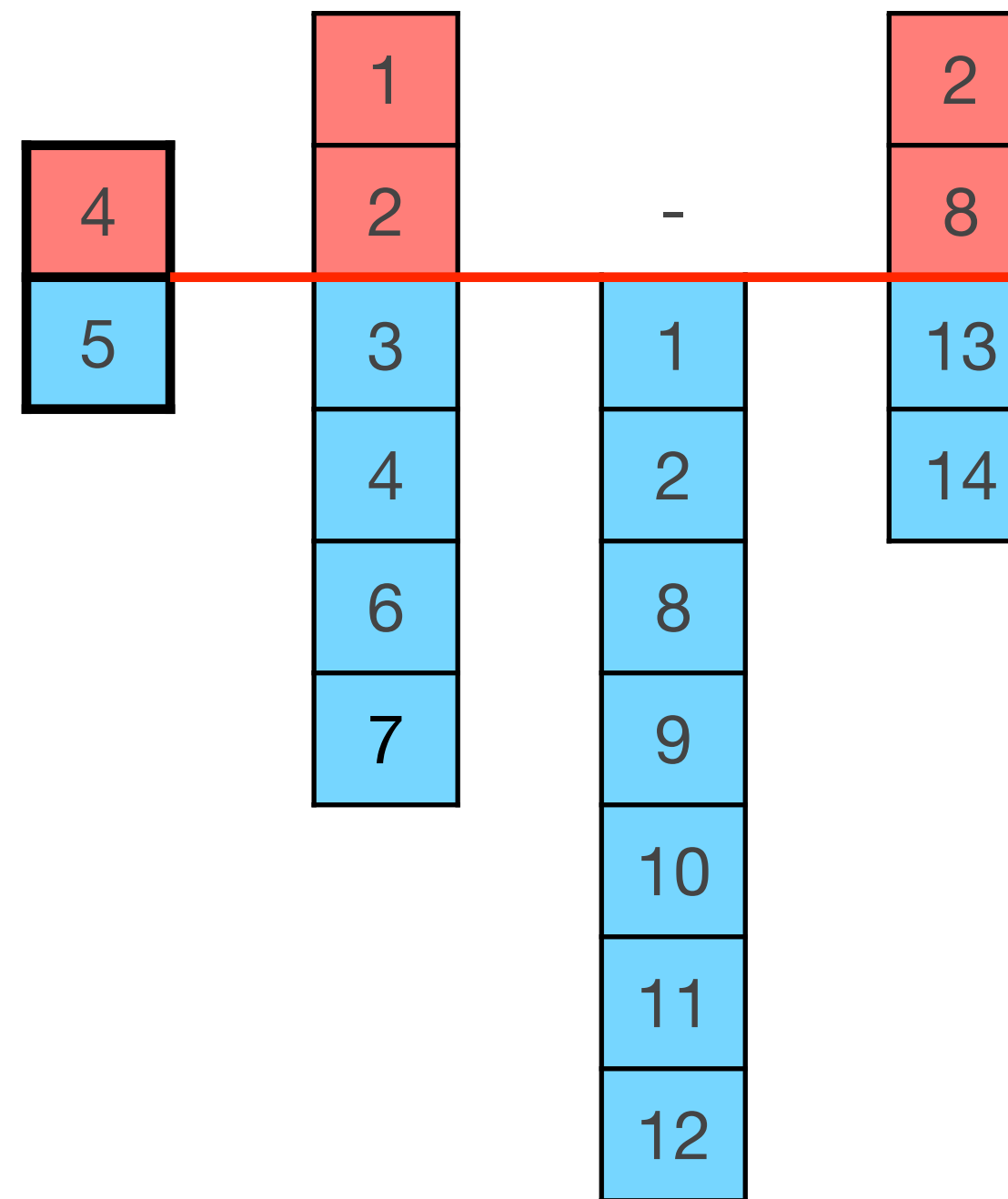
LA(5,4)



- **Data structure.**
  - Store each ladder in array.
  - Each node points to ladder corresponding to its longest path.
- **LA(v,k):** Jump to kth ancestor or root of ladder. Repeat.
- **Time.**  $O(\log n)$
- **Space.**  $O(n)$

# Solution 5: Ladder Decomposition

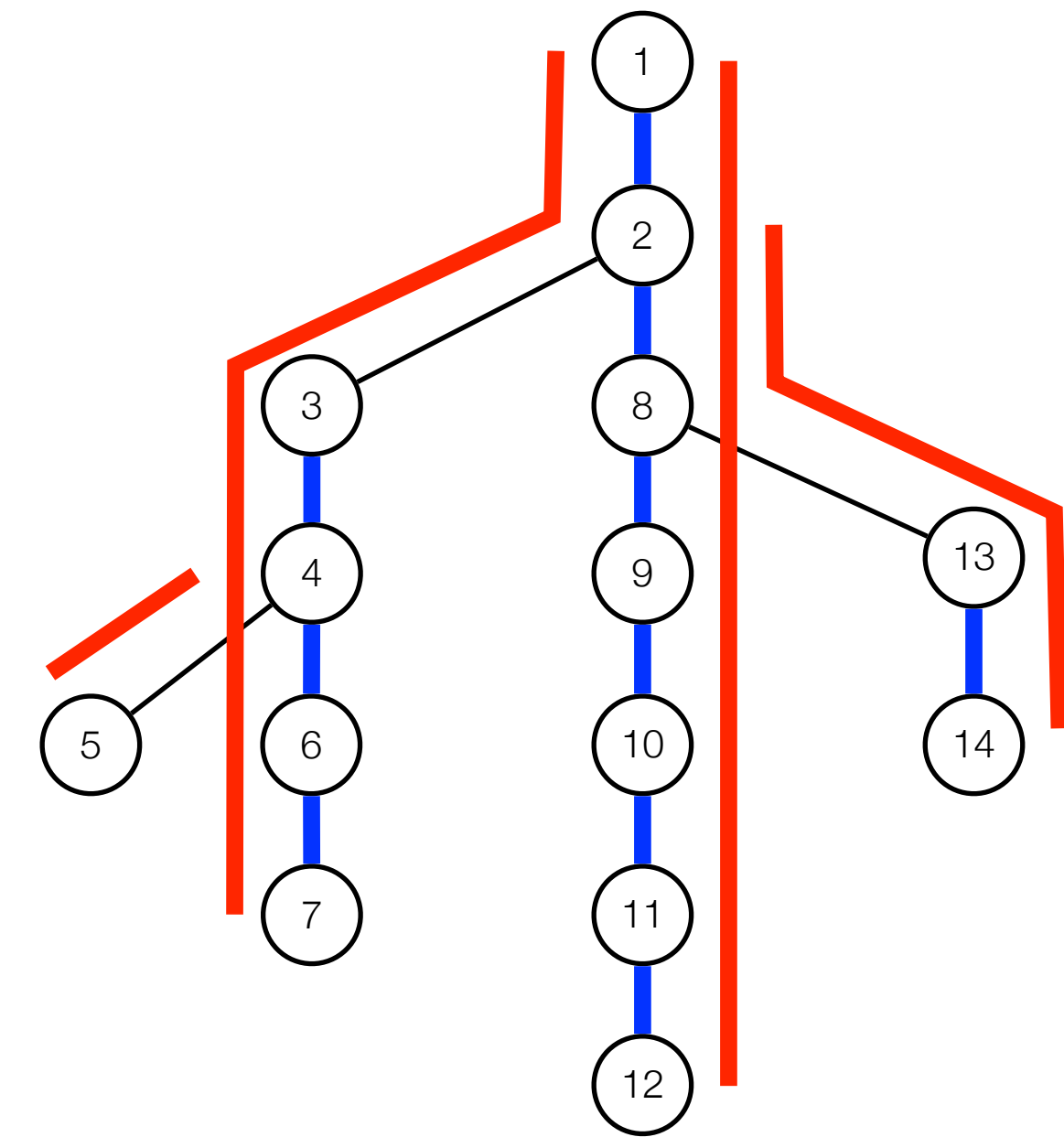
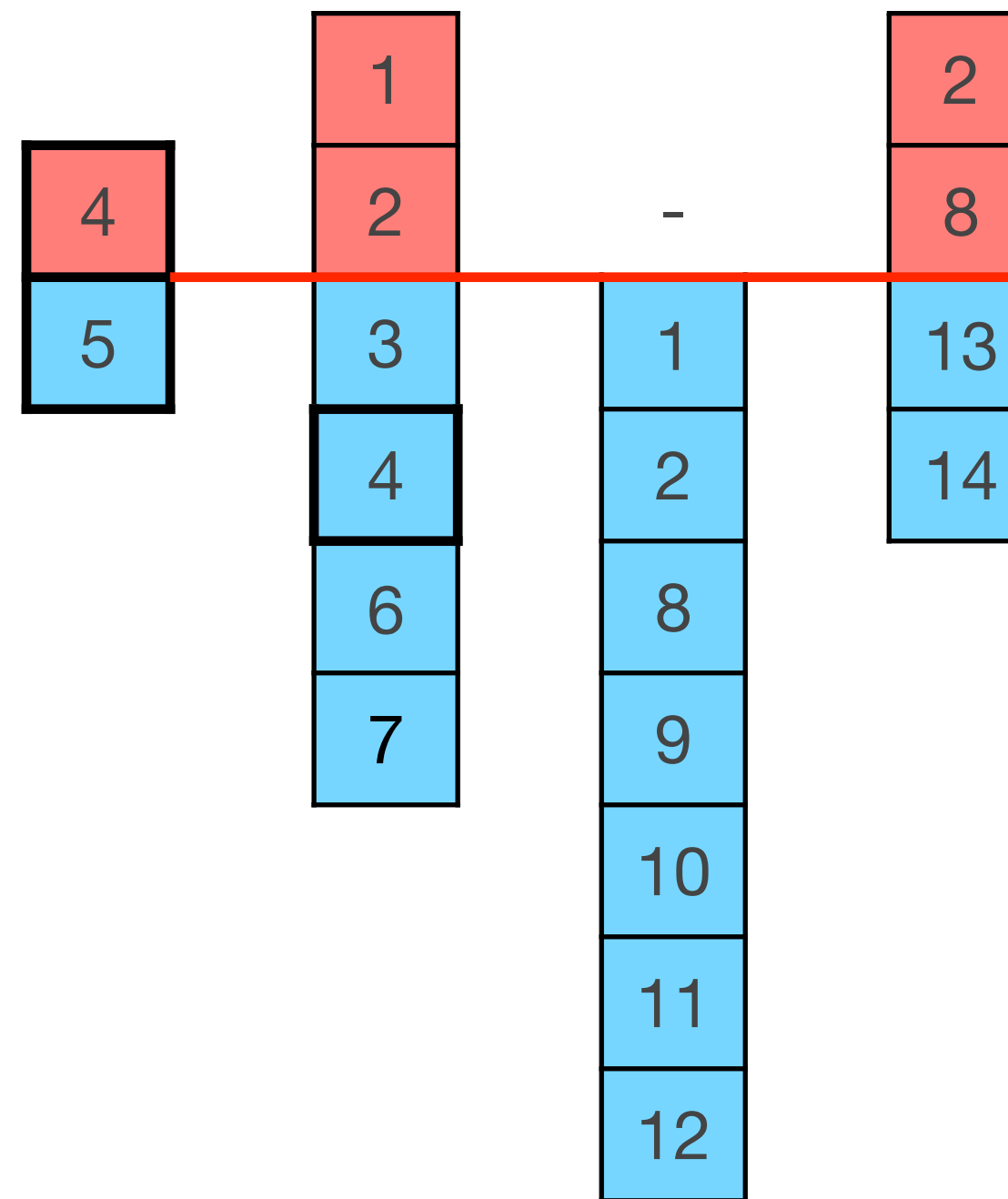
$$LA(5,4) = LA(4,3)$$



- **Data structure.**
  - Store each ladder in array.
  - Each node points to ladder corresponding to its longest path.
- **LA(v,k):** Jump to kth ancestor or root of ladder. Repeat.
- **Time.**  $O(\log n)$
- **Space.**  $O(n)$

# Solution 5: Ladder Decomposition

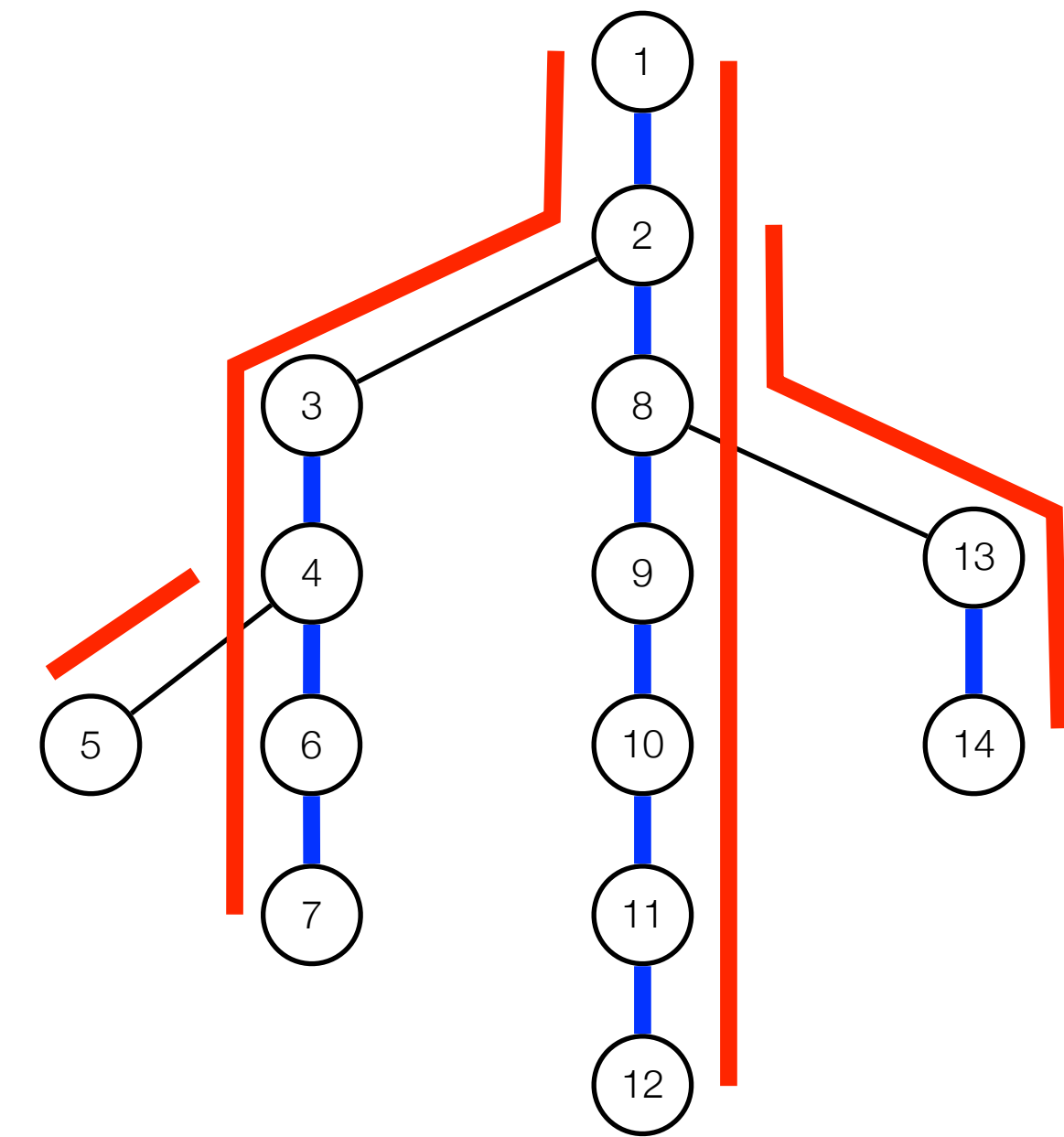
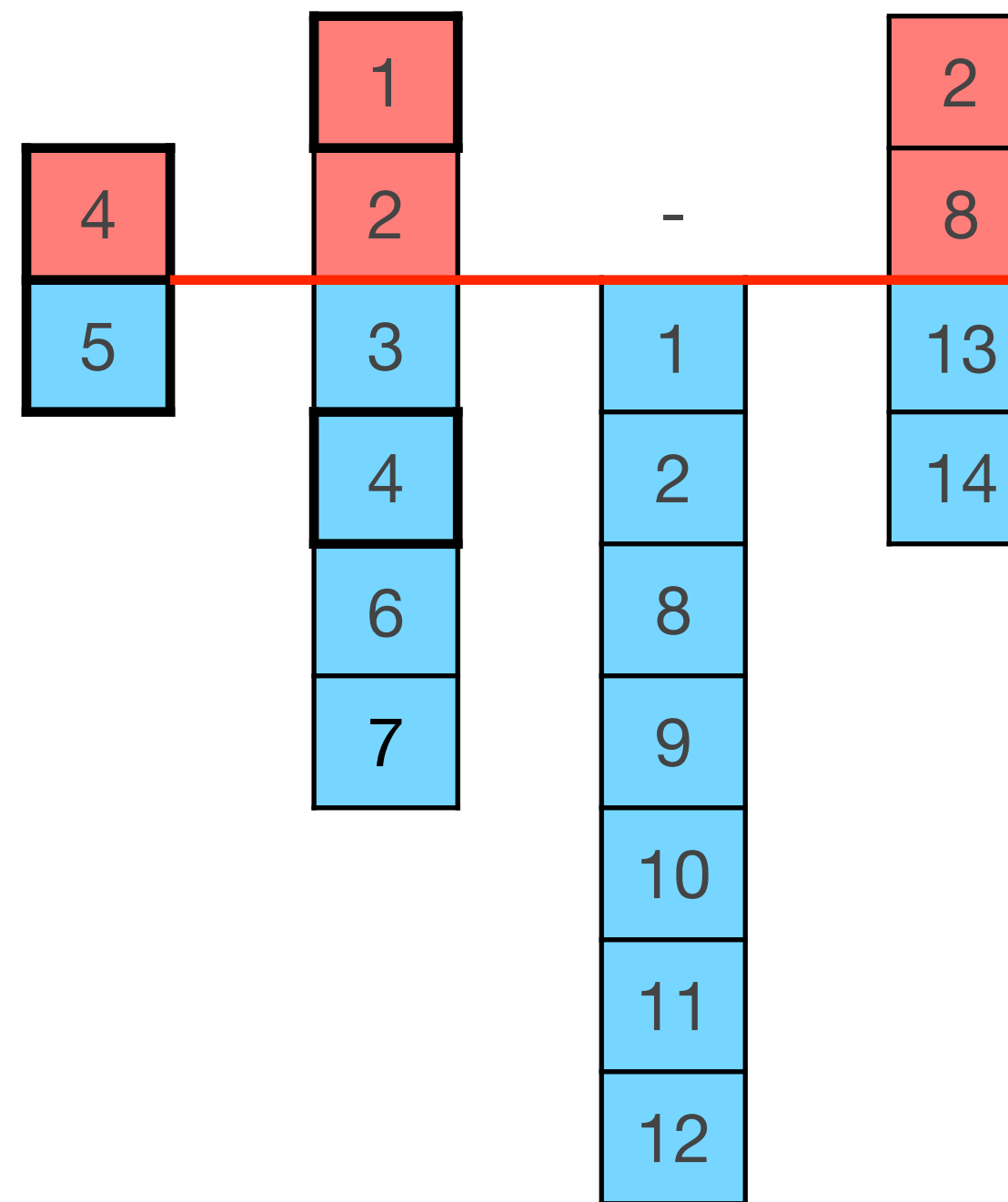
$$LA(5,4) = LA(4,3)$$



- **Data structure.**
  - Store each ladder in array.
  - Each node points to ladder corresponding to its longest path.
- **LA(v,k):** Jump to kth ancestor or root of ladder. Repeat.
- **Time.**  $O(\log n)$
- **Space.**  $O(n)$

# Solution 5: Ladder Decomposition

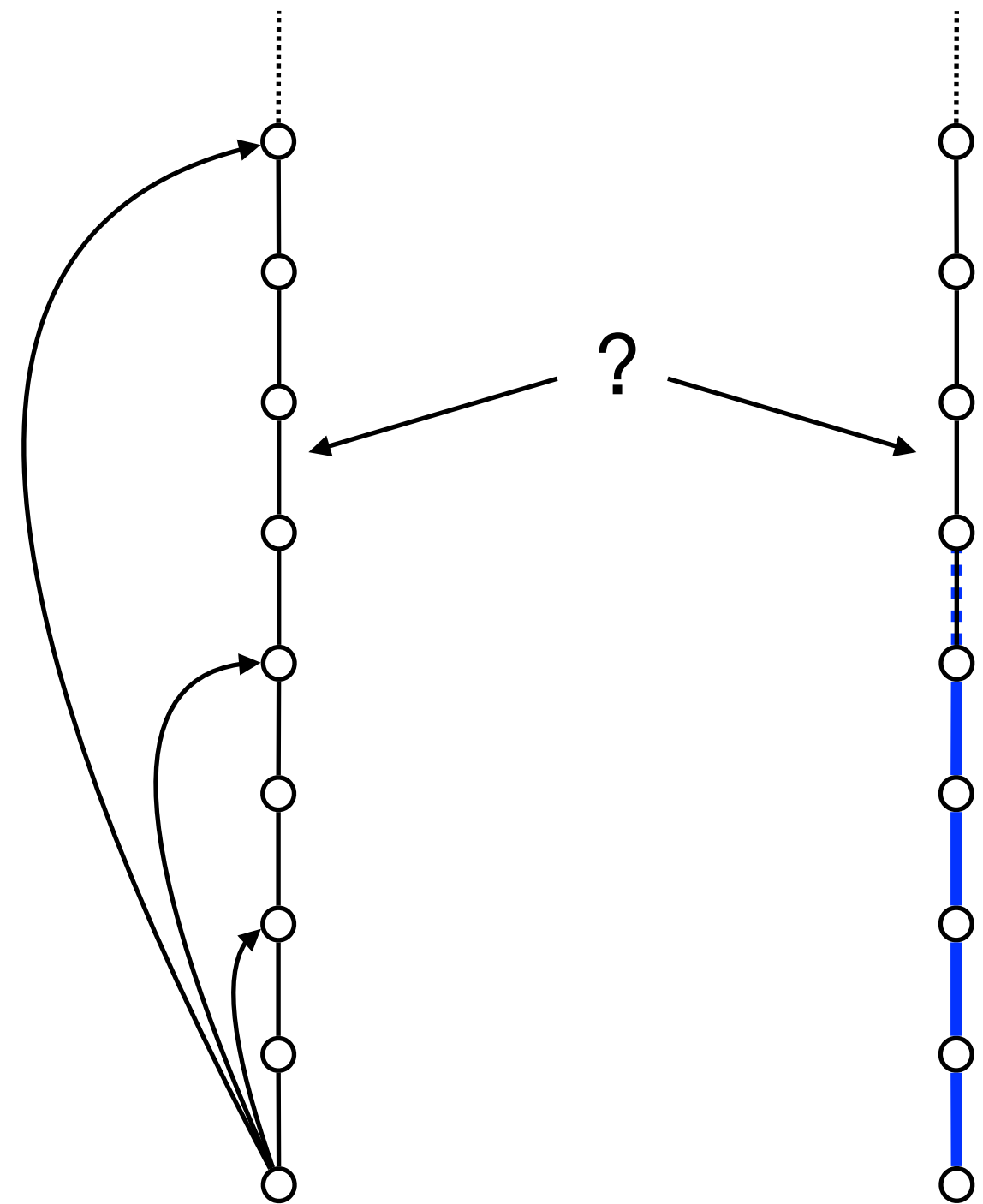
$$LA(5,4) = LA(4,3) = 1$$



- **Data structure.**
  - Store each ladder in array.
  - Each node points to ladder corresponding to its longest path.
- **LA(v,k):** Jump to kth ancestor or root of ladder. Repeat.
- **Time.**  $O(\log n)$
- **Space.**  $O(n)$

# Solution 6: Ladder Decomposition + Jump Pointers

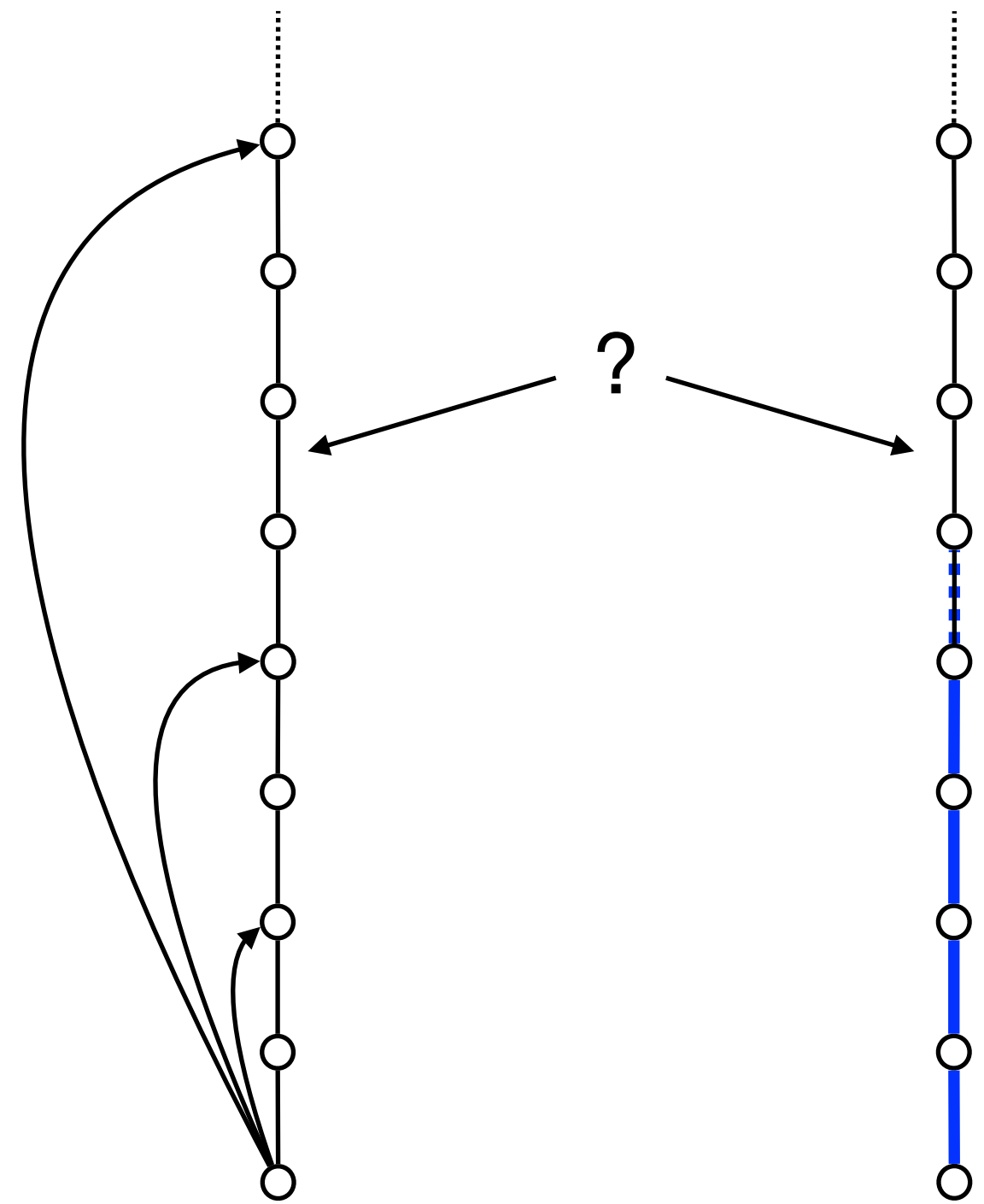
---



- **Data structure.** Ladder decomposition + Jump pointers.
- **LA(v,k):**
  - Jump to most distant ancestor not further away than k using jump pointer.
  - Jump to kth ancestor using ladder.
- **Time.**  $O(1)$
- **Space.**  $O(n) + O(n \log n) = O(n \log n)$

# Solution 6: Ladder Decomposition + Jump Pointers

---

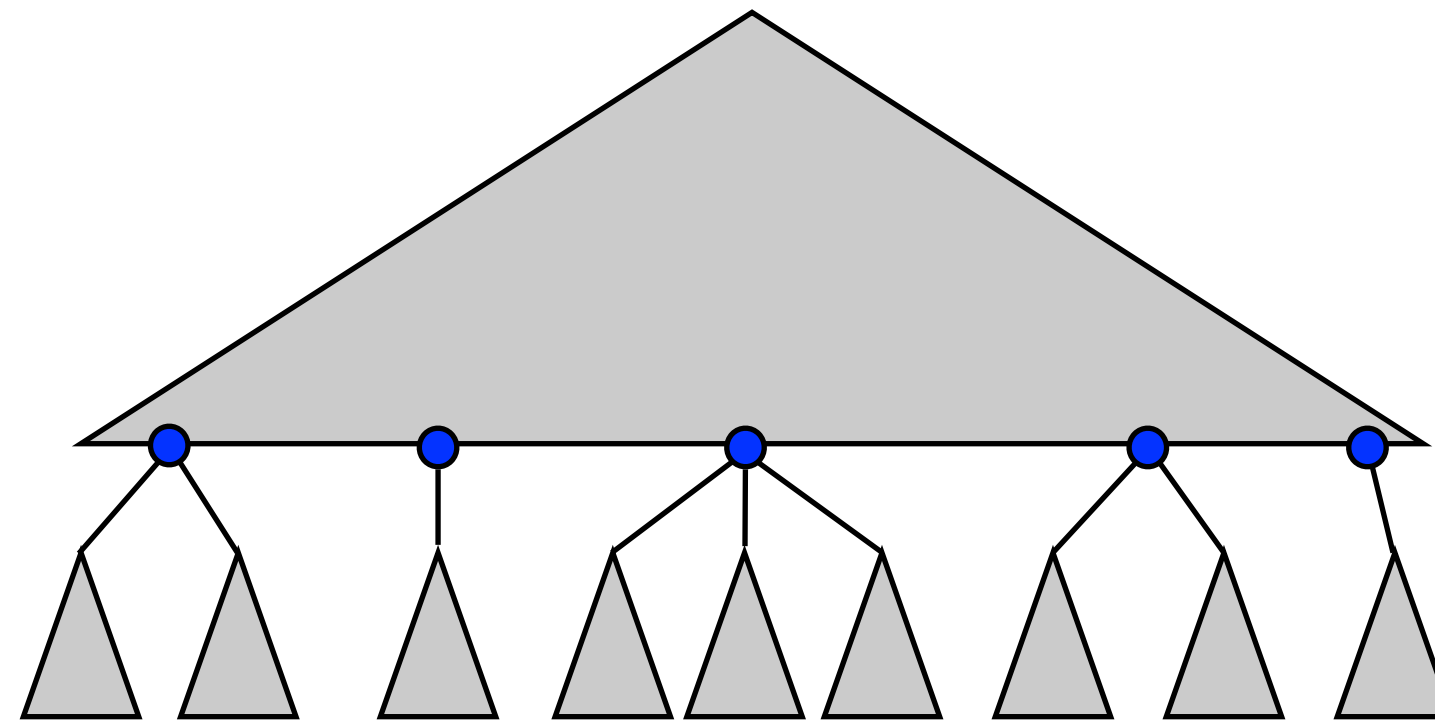


- **Correctness.**

- A node at height  $x$  is on a ladder of height at least  $2x$ .
- After jump we are at a node of height at least  $k/2$ .
- $\Rightarrow$  after jump we are at a ladder that contains our goal.

# Solution 7: Top-Bottom Decomposition

---

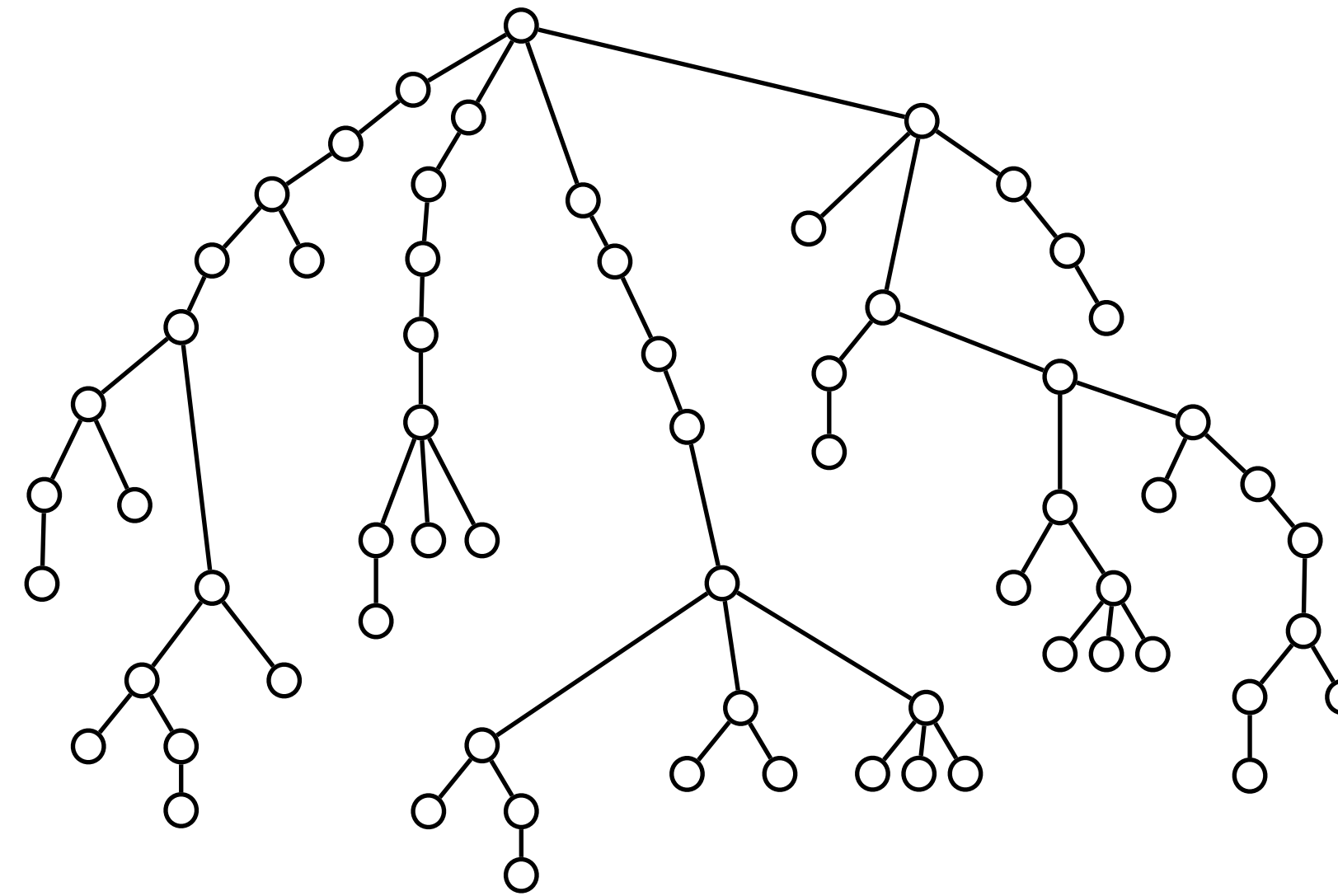


- **Jump nodes.** Maximal **deep** nodes with  $\geq 1/4 \log n$  descendants.
- **Top tree.** Jump nodes + ancestors.
- **Bottom trees.** Below top tree.



# Solution 7: Top-Bottom Decomposition

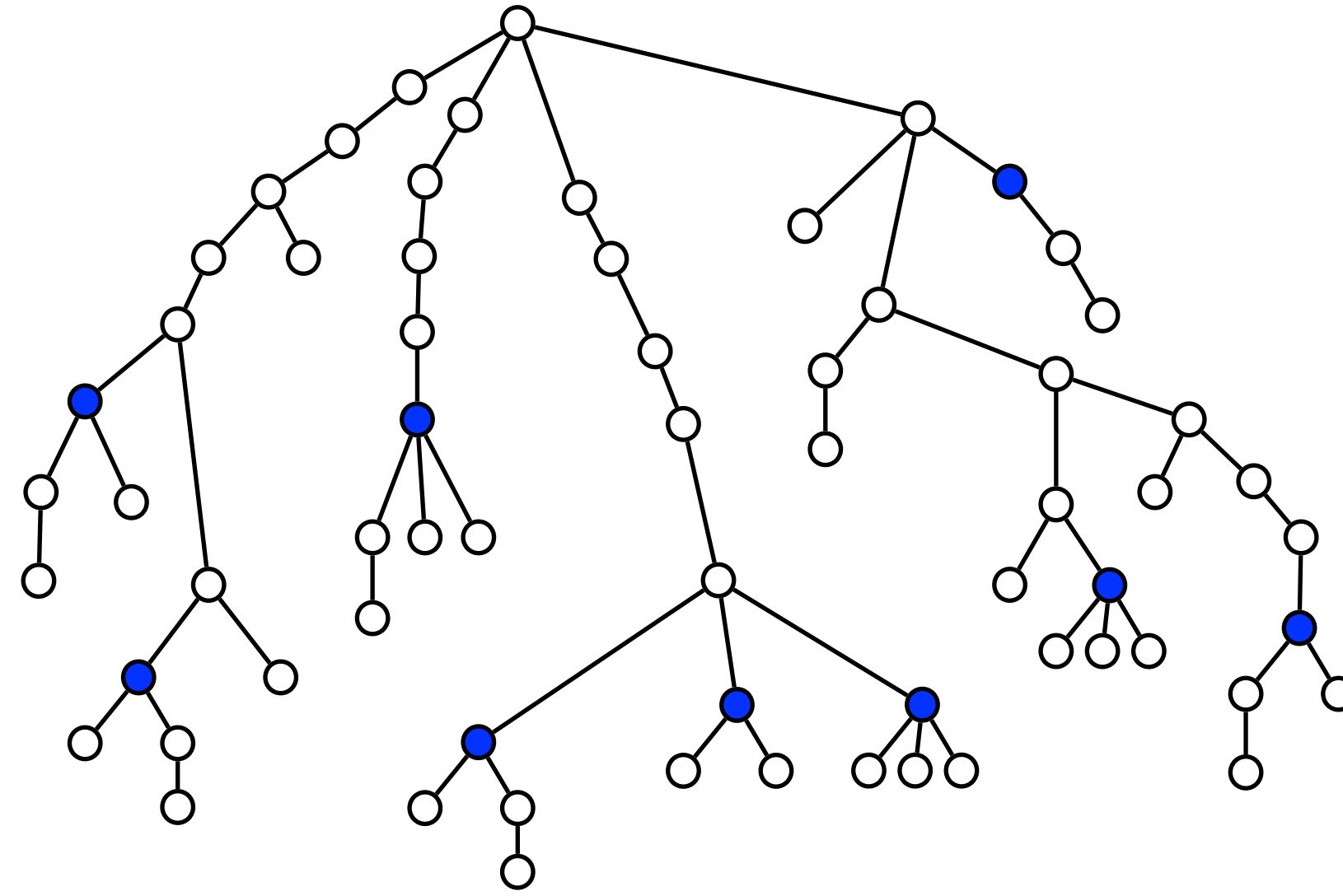
---



- **Jump nodes.** Maximal **deep** nodes with  $\geq 1/4 \log n$  descendants.
- **Top tree.** Jump nodes + ancestors.
- **Bottom trees.** Below top tree.

# Solution 7: Top-Bottom Decomposition

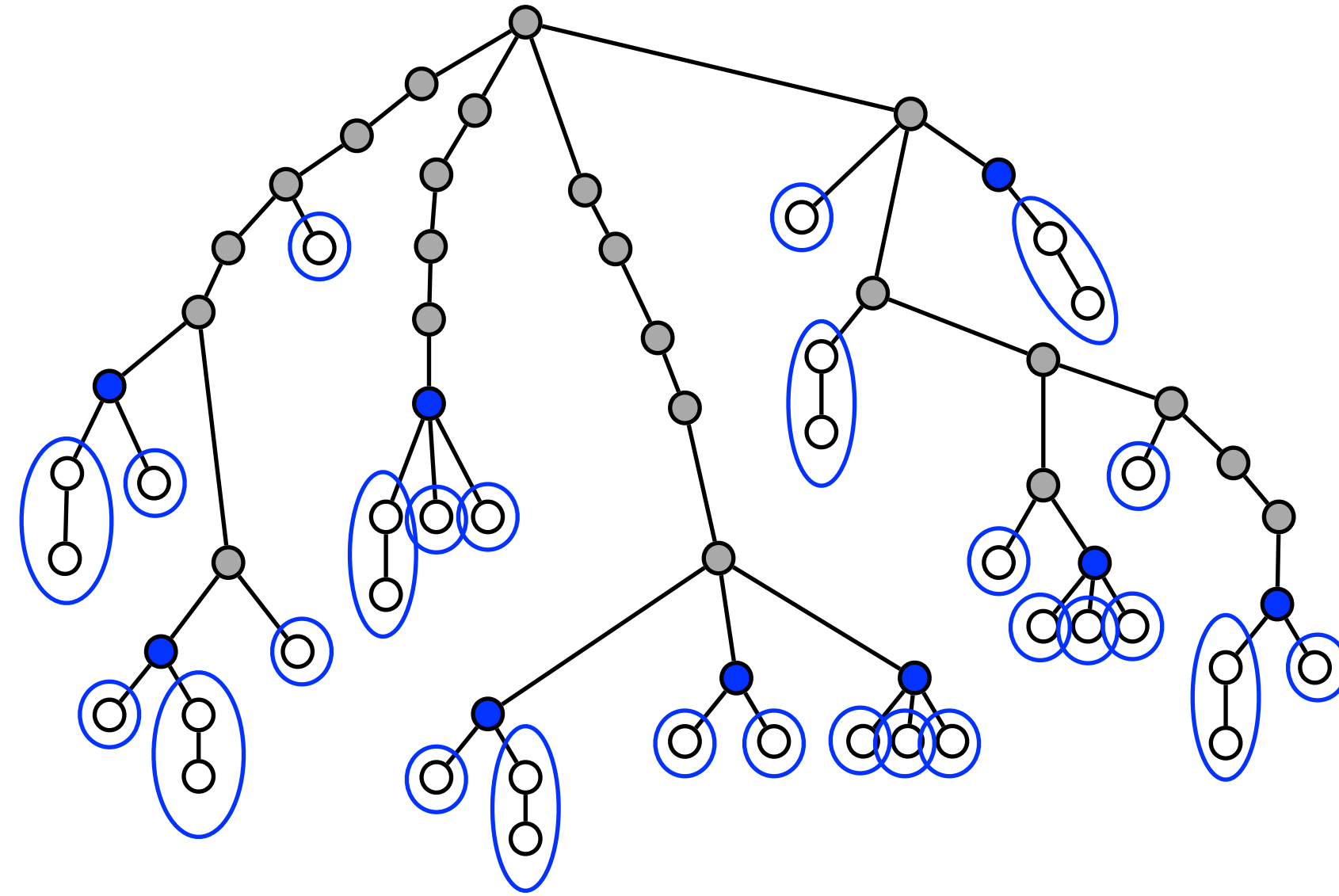
---



- **Jump nodes.** Maximal **deep** nodes with  $\geq 1/4 \log n$  descendants.
- **Top tree.** Jump nodes + ancestors.
- **Bottom trees.** Below top tree.

# Solution 7: Top-Bottom Decomposition

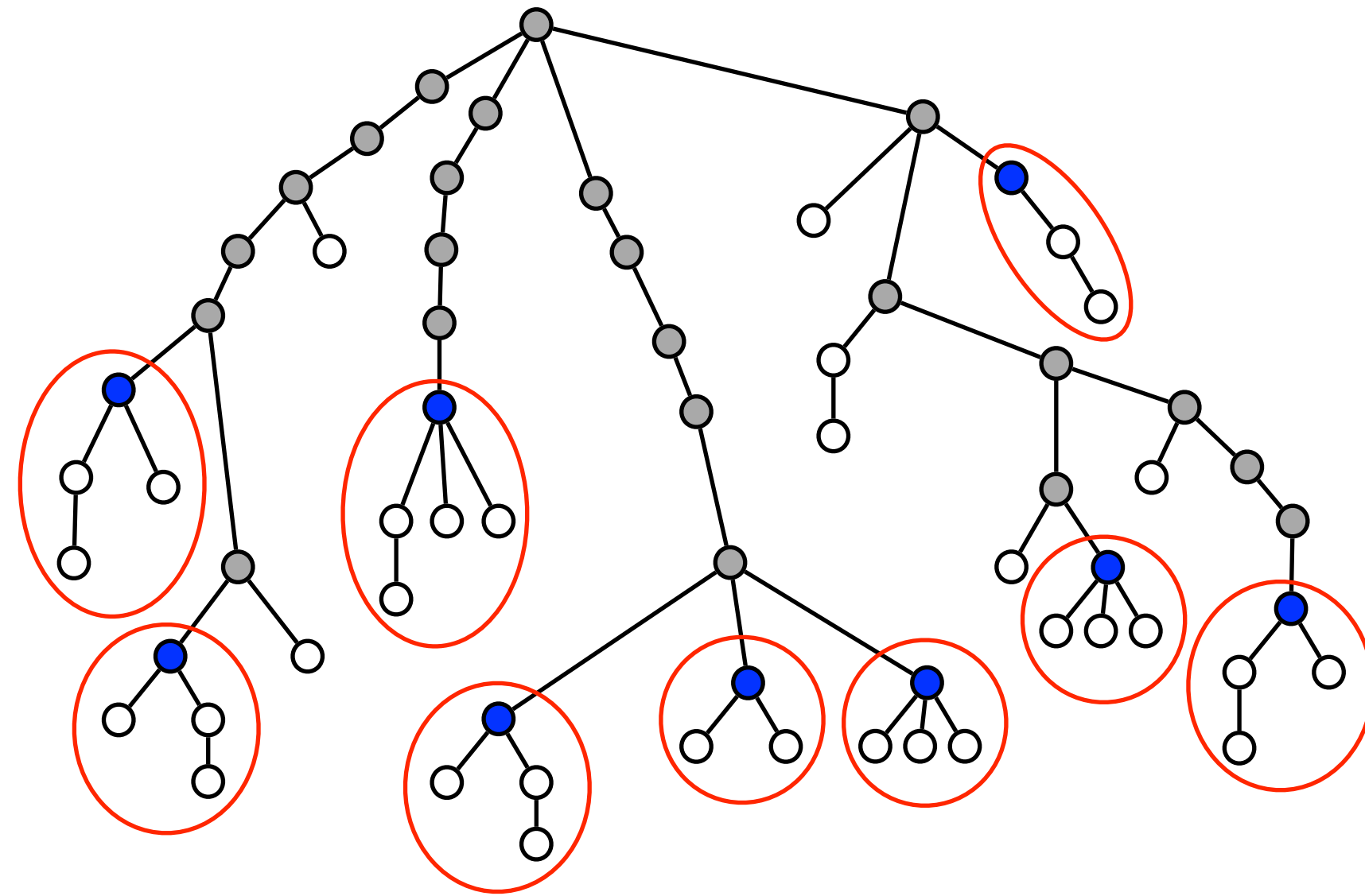
---



- **Jump nodes.** Maximal **deep** nodes with  $\geq 1/4 \log n$  descendants.
- **Top tree.** Jump nodes + ancestors.
- **Bottom trees.** Below top tree.
  
- Size of each bottom tree  $< 1/4 \log n$ .

# Solution 7: Top-Bottom Decomposition

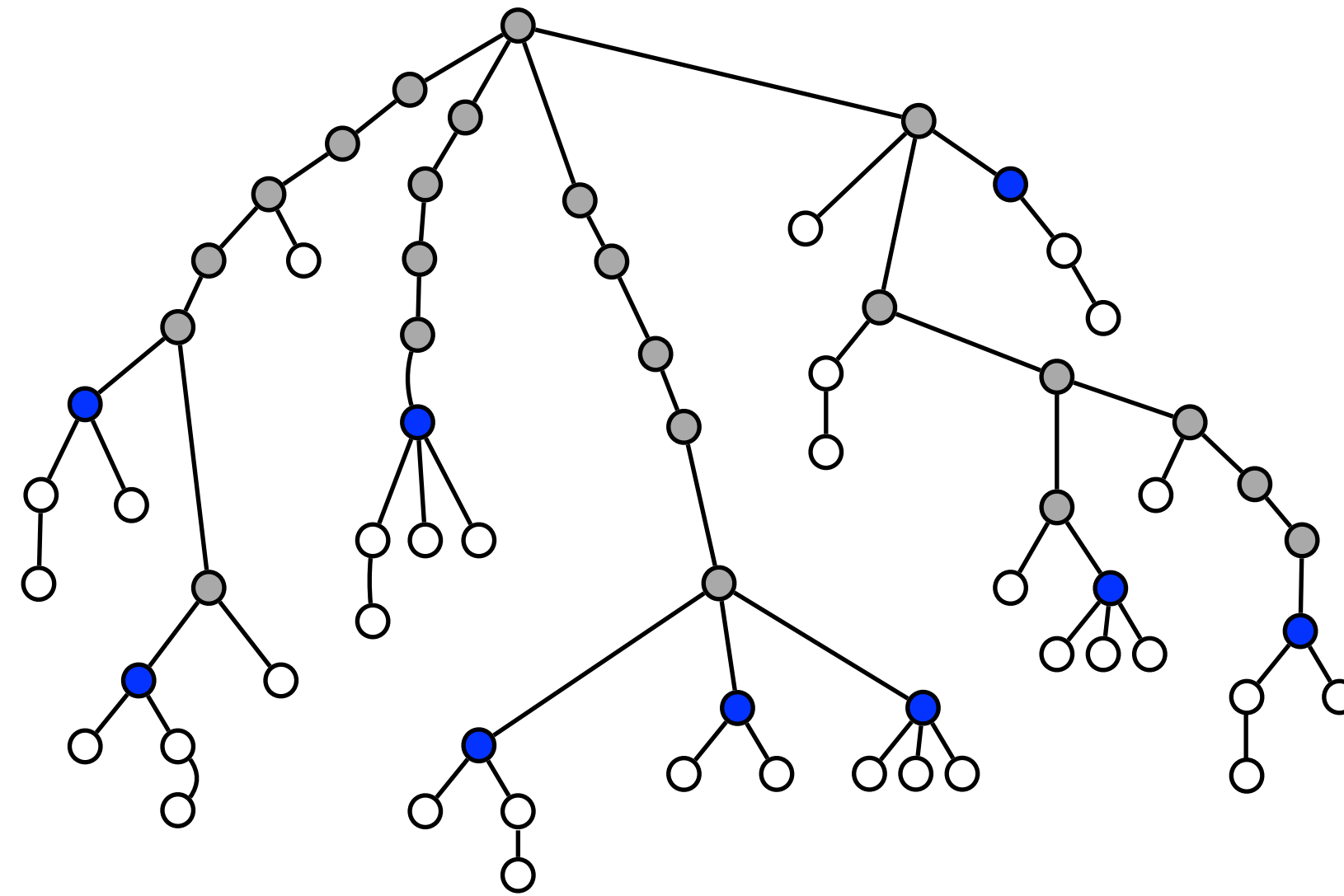
---



- **Jump nodes.** Maximal **deep** nodes with  $\geq 1/4 \log n$  descendants.
- **Top tree.** Jump nodes + ancestors.
- **Bottom trees.** Below top tree.
  
- Size of each bottom tree  $< 1/4 \log n$ .
- Number of jump nodes is at most  $O(n/\log n)$ .

# Solution 7: Top-Bottom Decomposition

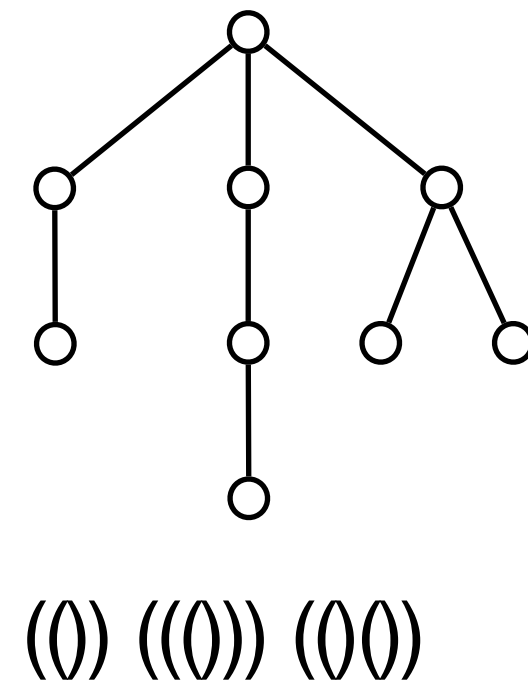
---



- **Data structure for top.**
  - Ladder decomposition + Jump pointers for jump nodes.
  - For each internal node pointer to some jump node below.
- **LA(v,k) in top:**
  - Follow pointer to jump node below v.
  - Jump pointer + ladder solution.
- **Time.**  $O(1)$
- **Space.**  $O(n) + (n/\log n \cdot \log n) = O(n)$

# Solution 7: Top-Bottom Decomposition

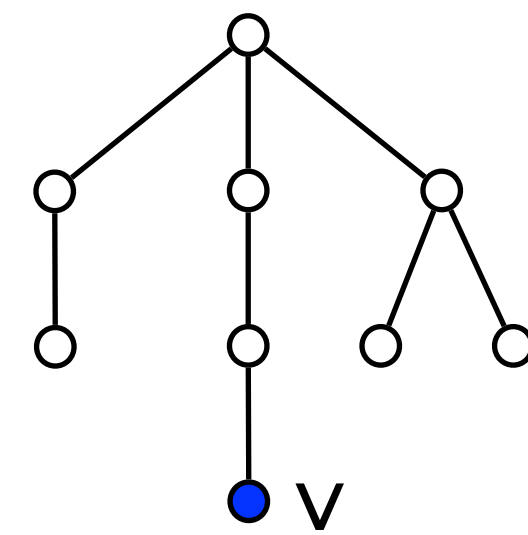
---



- **Tree encoding.** Encode each bottom tree  $B$  using balanced parentheses representation.
  - $< 2 \cdot 1/4 \log n = 1/2 \log n$  bits.

# Solution 7: Top-Bottom Decomposition

---



v has preorder number 5 =  $0101_2$

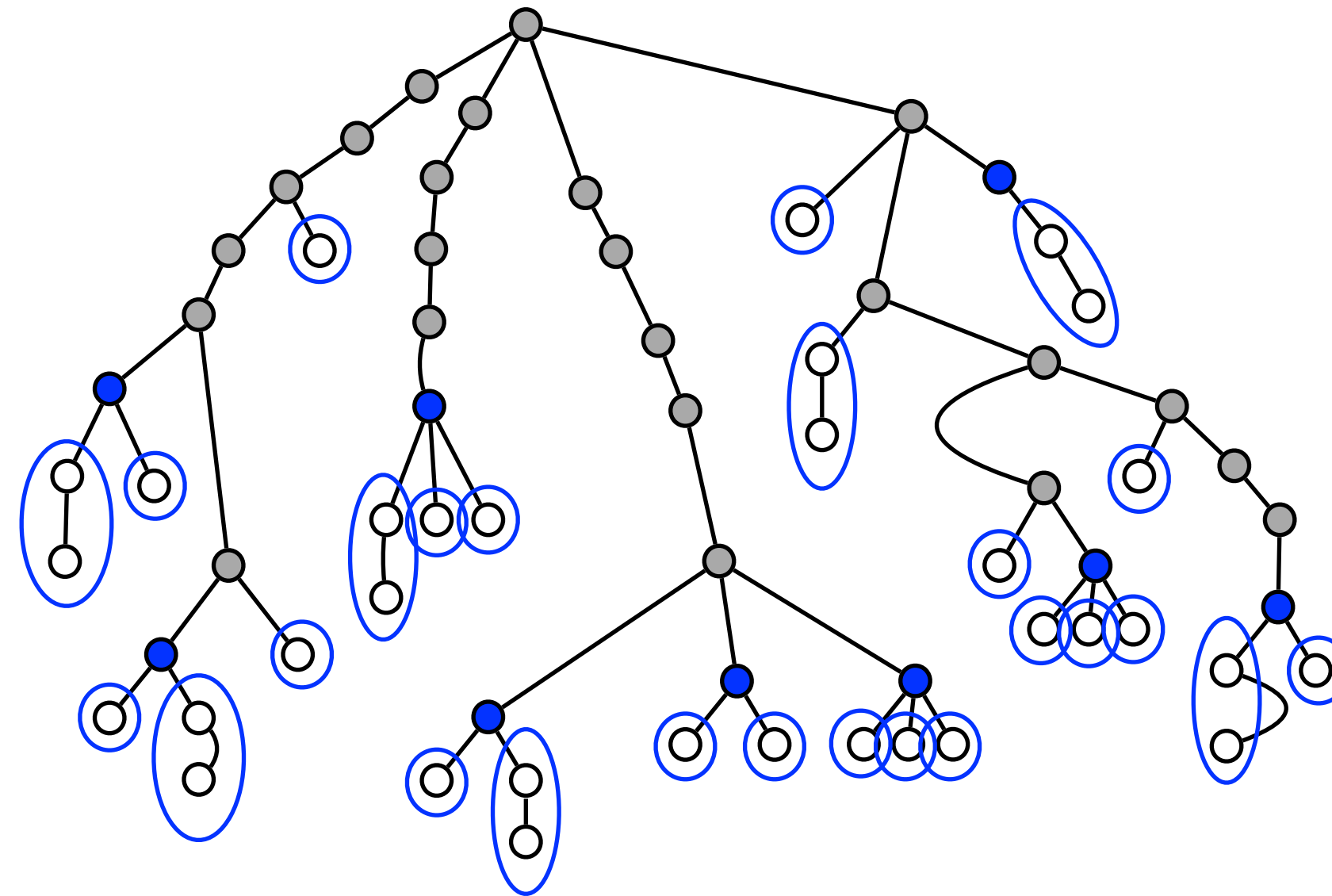
(()) ((())) (())

$$\text{Code}(B, v, 2) = 0011000111001011 \mathbf{0101} \mathbf{0010}$$

- **Tree encoding.** Encode each bottom tree B using balanced parentheses representation.
  - $< 2 \cdot 1/4 \log n = 1/2 \log n$  bits.
- **Integer encoding.** Encode inputs v and k to LA
  - $< 2 \cdot \log(1/4 \log n) < 2 \log \log n$  bits.
- **LA encoding.** Concatenate into  $\text{code}(B, v, k)$ 
  - $\Rightarrow |\text{code}(B, v, k)| < 1/2 \log n + 2 \log \log n$  bits.

# Solution 7: Top-Bottom Decomposition

---



- **Data structure for bottom.**
  - Build table A s.t.  $A[\text{code}(B, v, k)] = \text{LA}(v, k)$  in bottom tree B.
- **LA(v,k) in bottom:** Lookup in A.
- **Time.**  $O(1)$
- **Space.**  $2^{|\text{code}|} < 2^{1/2 \log n + 2 \log \log n} = n^{1/2} \log^2 n = o(n)$ .
- Combine bottom and top data structures  $\Rightarrow O(n)$  space and  $O(1)$  query time.



# Solution 7: Top-Bottom Decomposition

---

- **Theorem.** We can solve the level ancestor problem in linear space and constant query time.