Lecture 8

Red-Black Trees: Deletion (contd.), Augmenting Trees

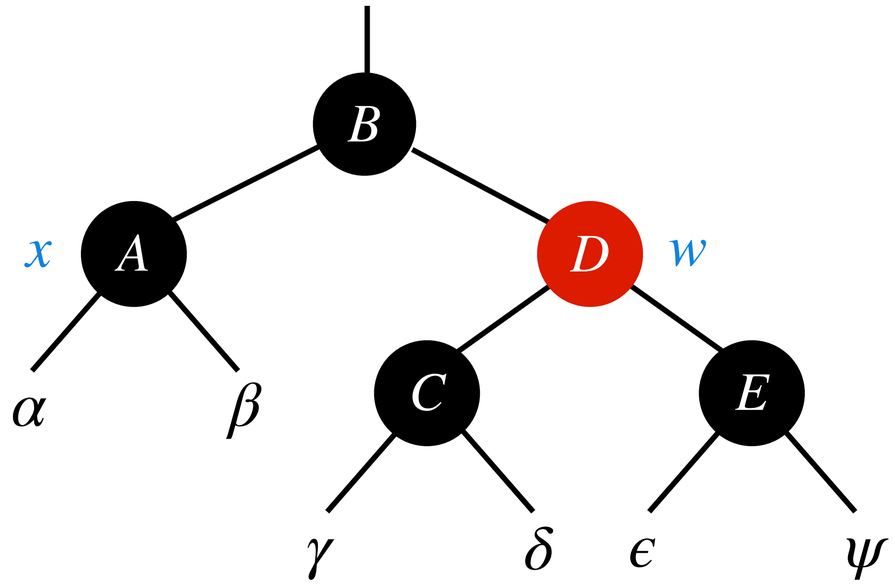
Fixing Violation of Only Property 5

Fixing Violation of Only Property 5

Case 1: x's sibling w is red.

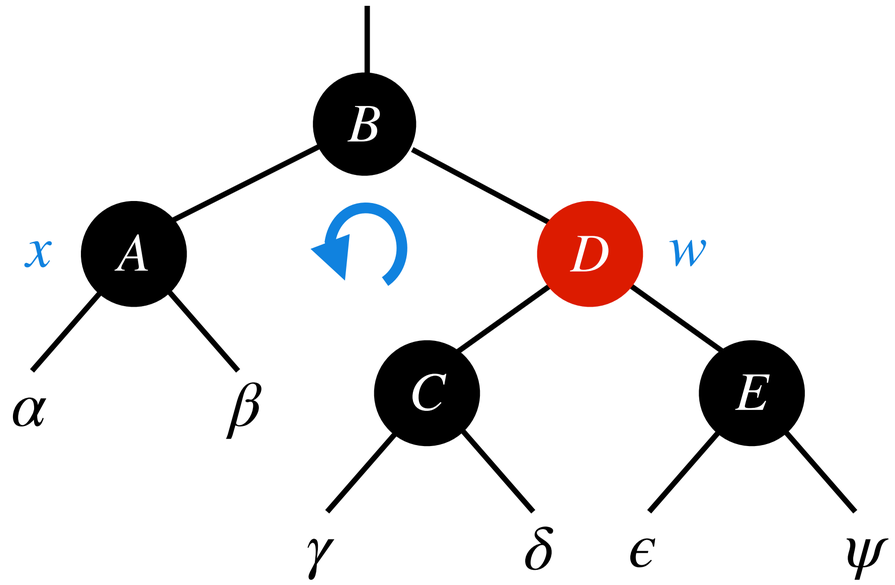
Fixing Violation of Only Property 5

Case 1: x's sibling w is red.



Fixing Violation of Only Property 5

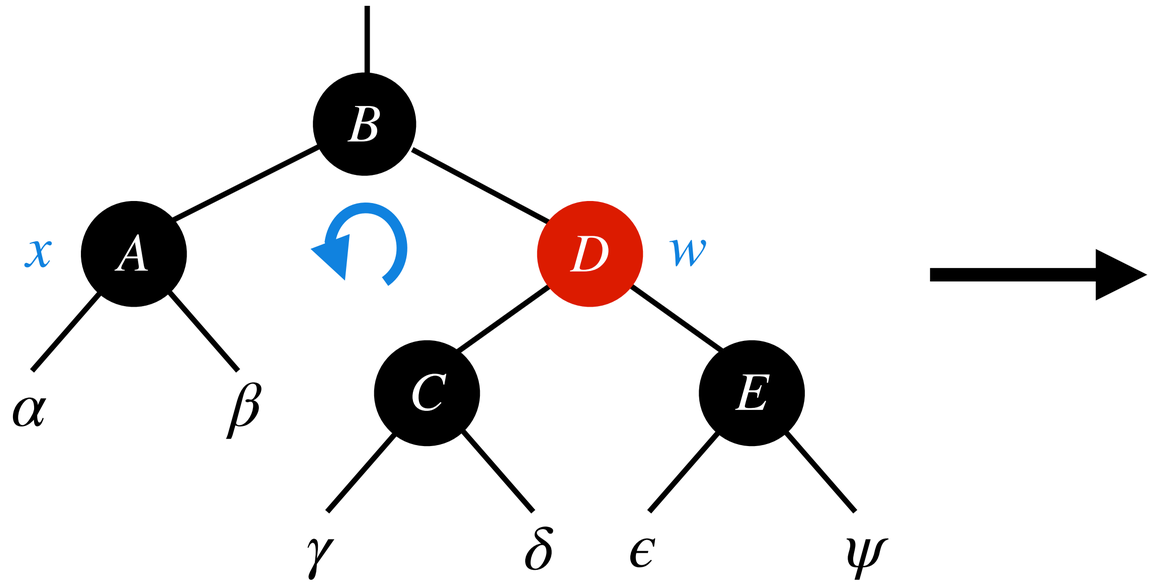
Case 1: x's sibling w is red.



Handling: Switch colours of w and $x\,.\,p$ and perform a left rotation at $x\,.\,p$.

Fixing Violation of Only Property 5

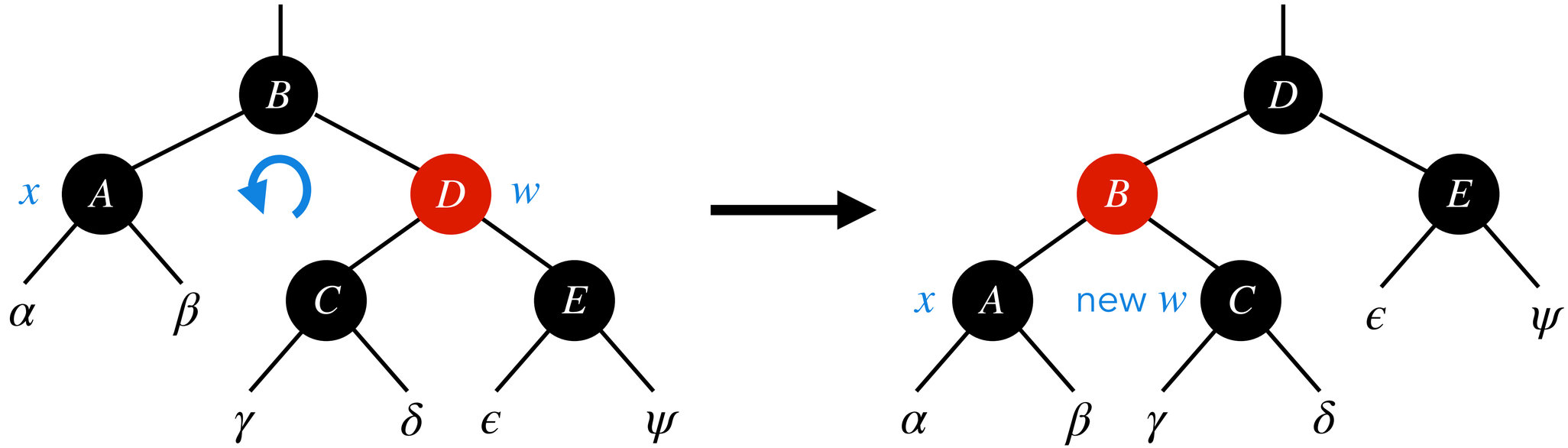
Case 1: x's sibling w is red.



Handling: Switch colours of w and $x\,.\,p$ and perform a left rotation at $x\,.\,p$.

Fixing Violation of Only Property 5

Case 1: x's sibling w is red.



Handling: Switch colours of w and $x\,.\,p$ and perform a left rotation at $x\,.\,p$.

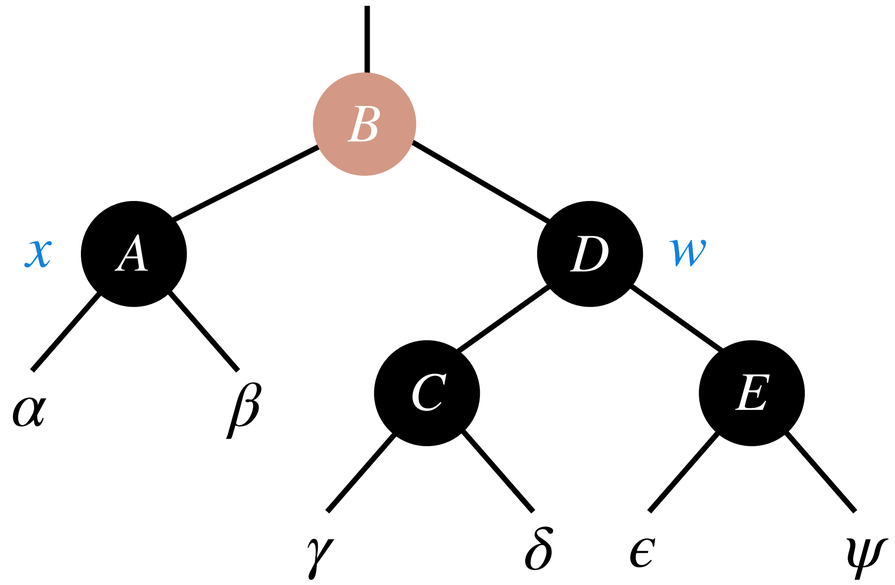
Fixing Violation of Only Property 5

Fixing Violation of Only Property 5

Case 2: x's sibling w is **black**, and both of w's children are **black**.

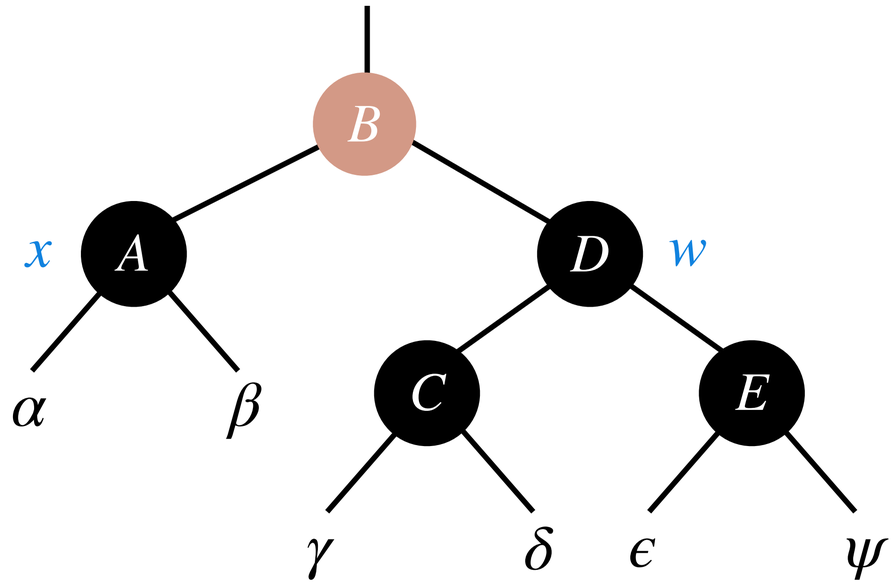
Fixing Violation of Only Property 5

Case 2: x's sibling w is **black**, and both of w's children are **black**.



Fixing Violation of Only Property 5

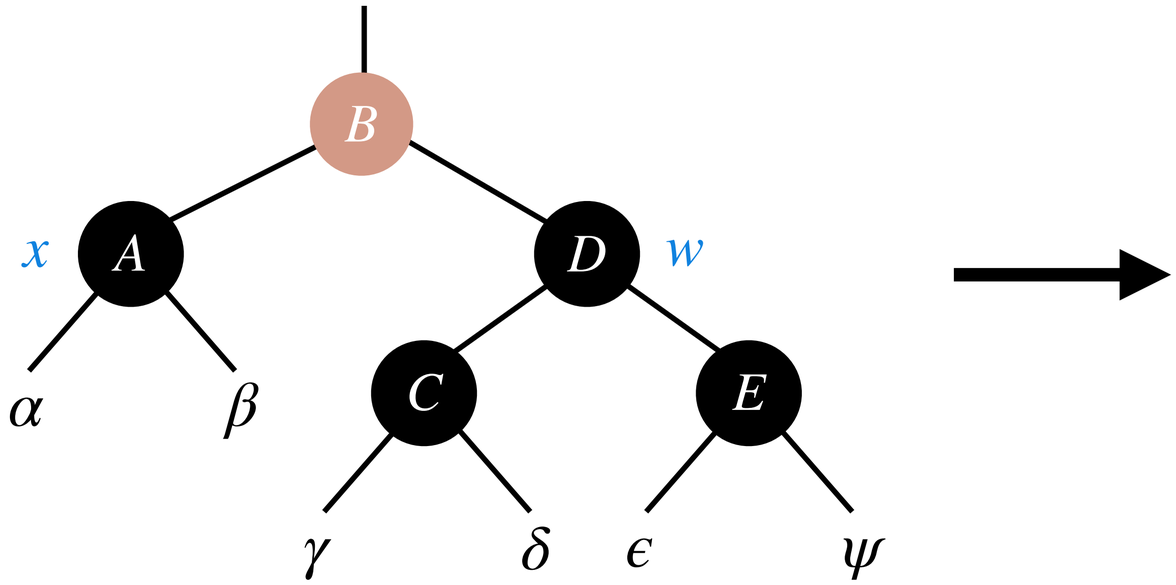
Case 2: x's sibling w is **black**, and both of w's children are **black**.



Handling: Move "extra blackness" of x and blackness of w to their **parent**.

Fixing Violation of Only Property 5

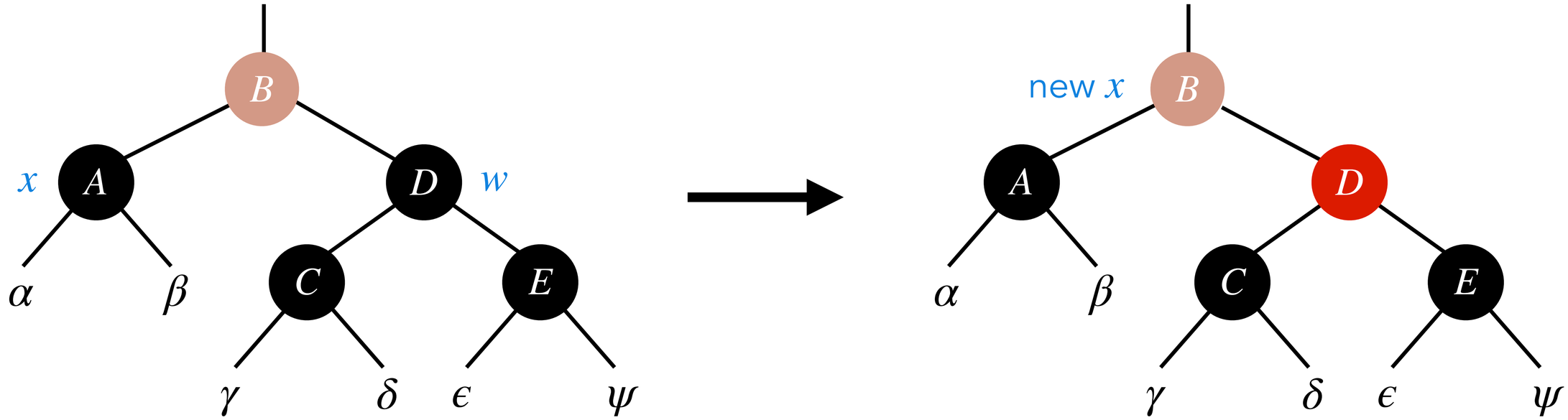
Case 2: x's sibling w is **black**, and both of w's children are **black**.



Handling: Move "extra blackness" of x and blackness of w to their **parent**.

Fixing Violation of Only Property 5

Case 2: x's sibling w is **black**, and both of w's children are **black**.

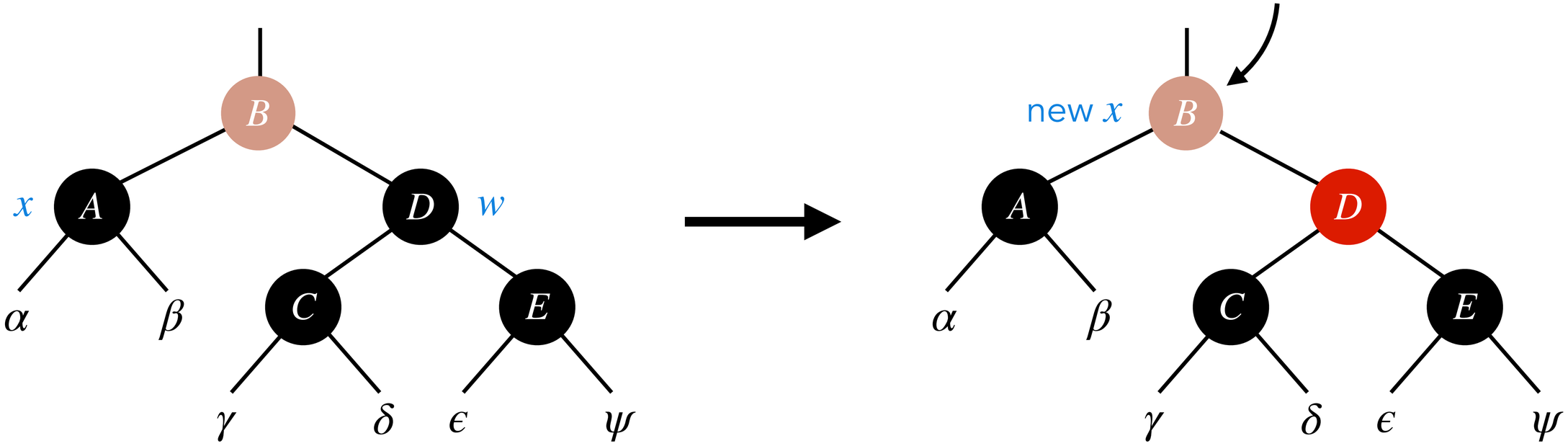


Handling: Move "extra blackness" of x and blackness of w to their **parent**.

Fixing Violation of Only Property 5

Case 2: x's sibling w is **black**, and both of w's children are **black**.

Process can terminate
if B is root or was Red earlier



Handling: Move "extra blackness" of x and blackness of w to their **parent**.

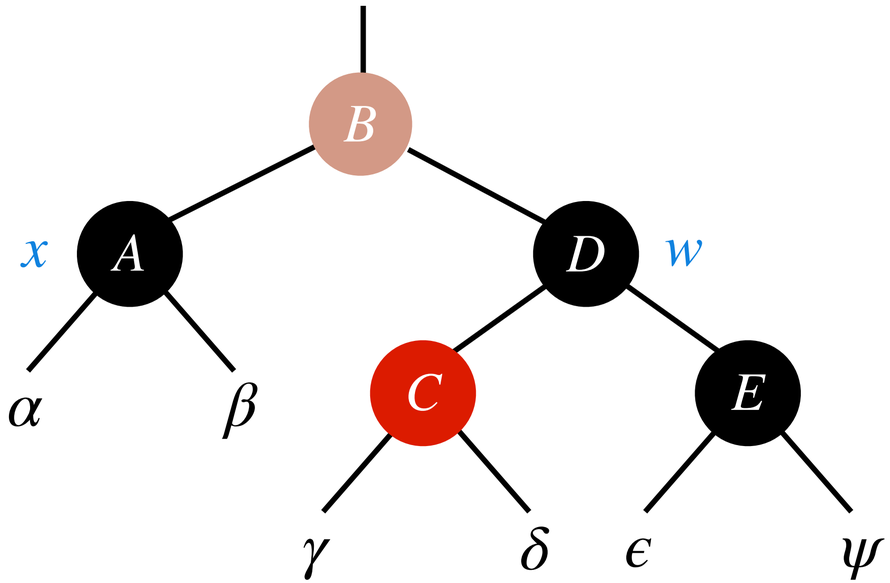
Fixing Violation of Only Property 5

Fixing Violation of Only Property 5

Case 3: x's sibling w is **black**, and w's left child is **red**, and w's right child is **black**.

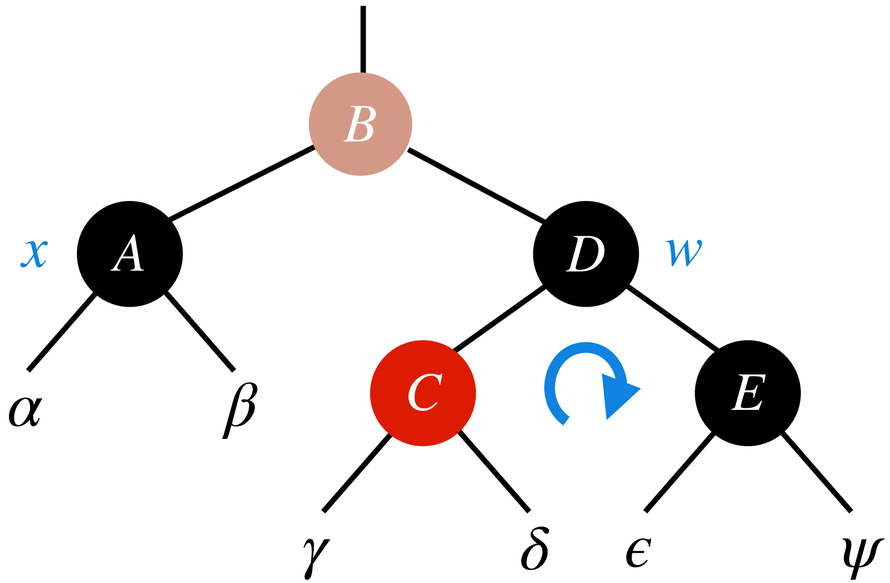
Fixing Violation of Only Property 5

Case 3: x's sibling w is **black**, and w's left child is red, and w's right child is **black**.



Fixing Violation of Only Property 5

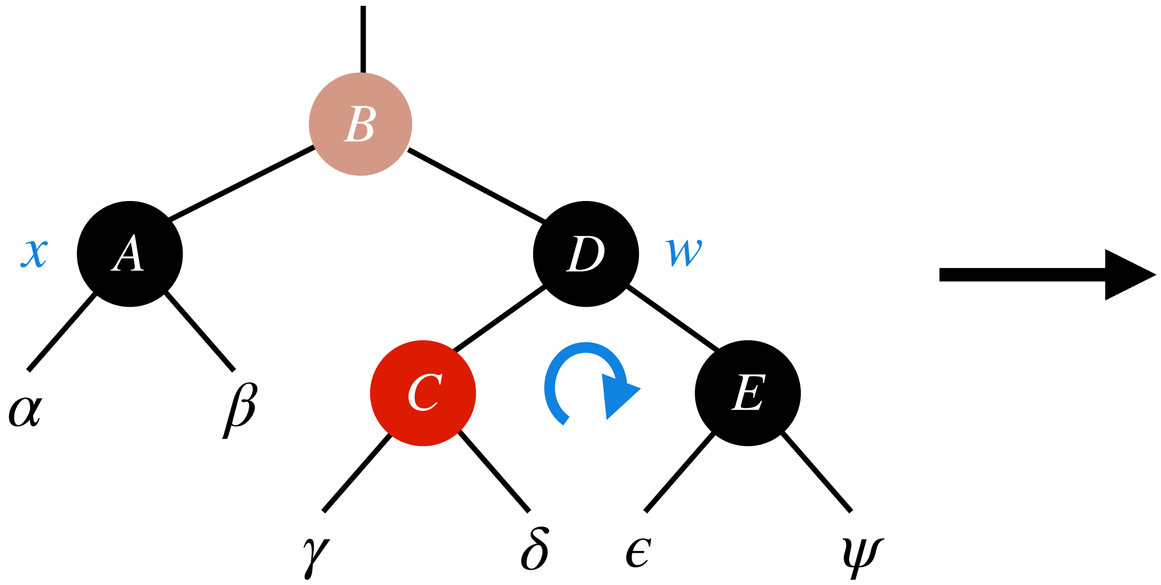
Case 3: x's sibling w is **black**, and w's left child is red, and w's right child is **black**.



Handling: Switch colour of w and its left child and then perform a right rotation on w.

Fixing Violation of Only Property 5

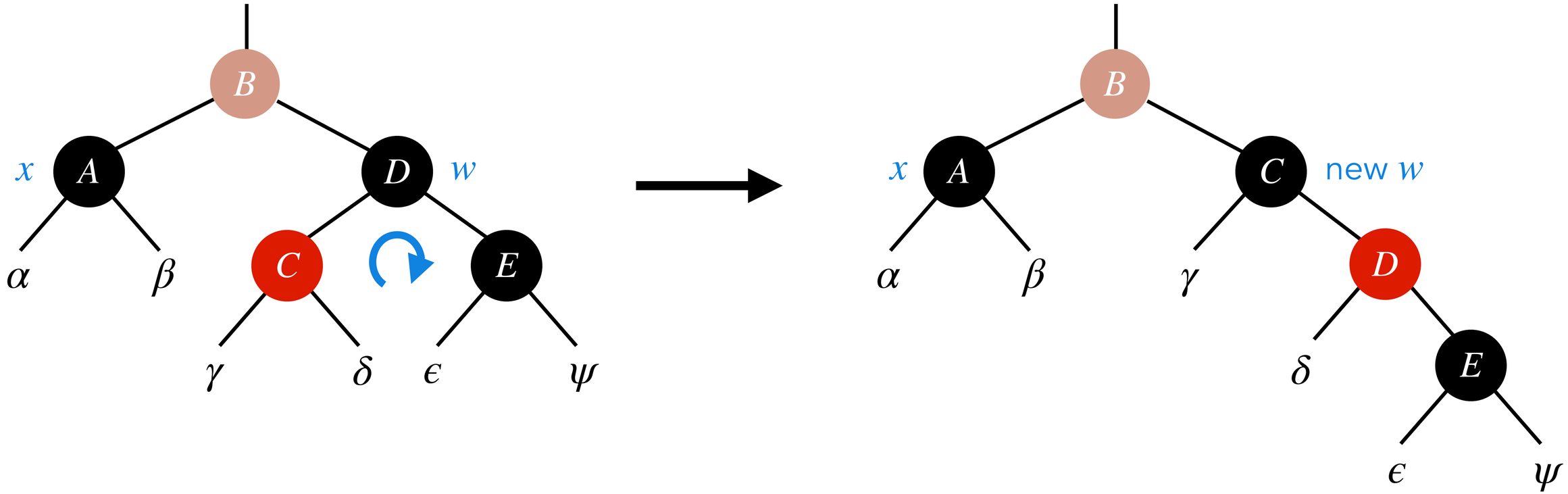
Case 3: x's sibling w is **black**, and w's left child is red, and w's right child is **black**.



Handling: Switch colour of w and its left child and then perform a right rotation on w.

Fixing Violation of Only Property 5

Case 3: x's sibling w is **black**, and w's left child is red, and w's right child is **black**.



Handling: Switch colour of w and its left child and then perform a right rotation on w.

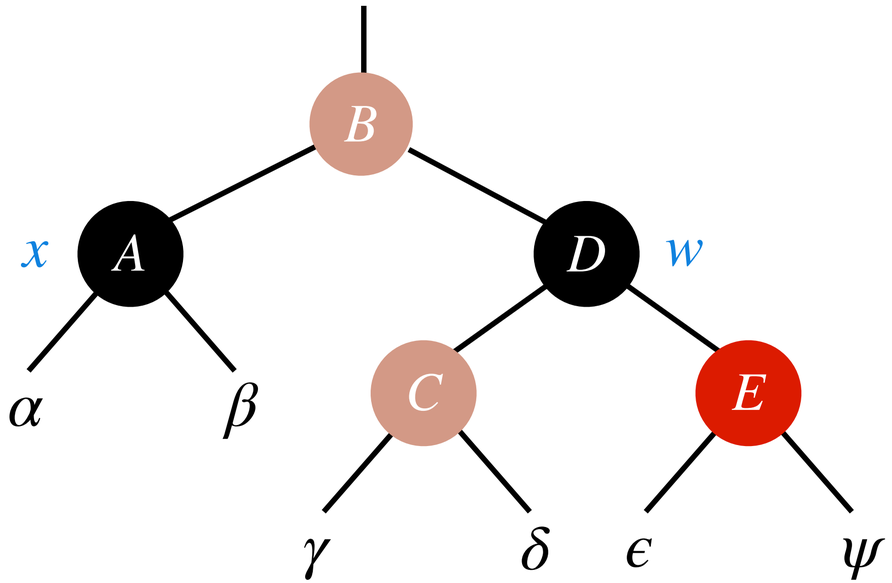
Fixing Violation of Only Property 5

Fixing Violation of Only Property 5

Case 4: x's sibling w is **black**, and w's right child is red.

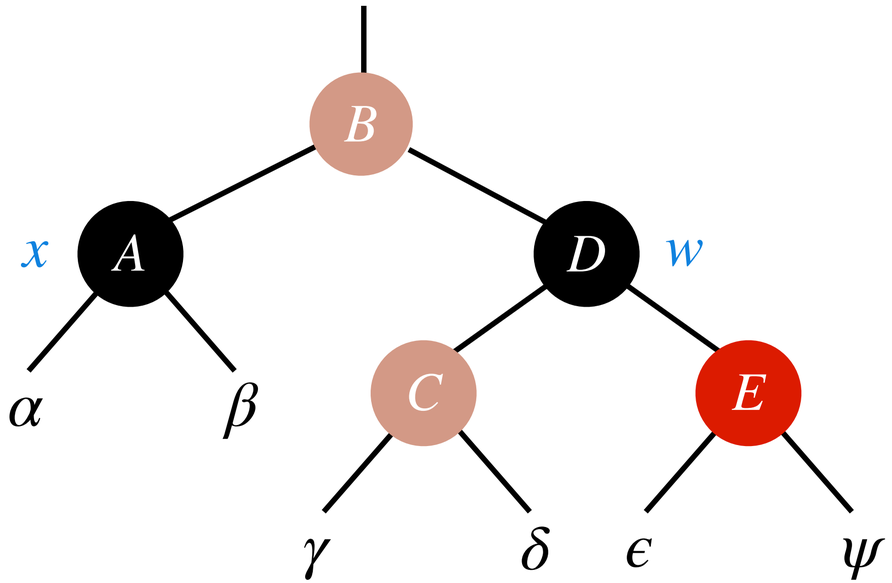
Fixing Violation of Only Property 5

Case 4: x's sibling w is **black**, and w's right child is red.



Fixing Violation of Only Property 5

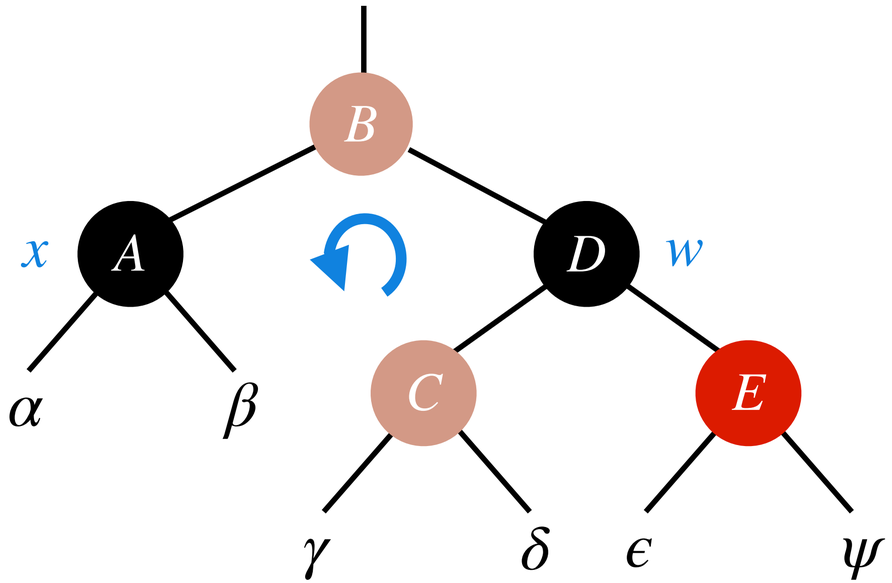
Case 4: x's sibling w is **black**, and w's right child is **red**.



Handling: Give w the colour of x's parent. Make x's parent and w's right child **black**.

Fixing Violation of Only Property 5

Case 4: x's sibling w is **black**, and w's right child is **red**.

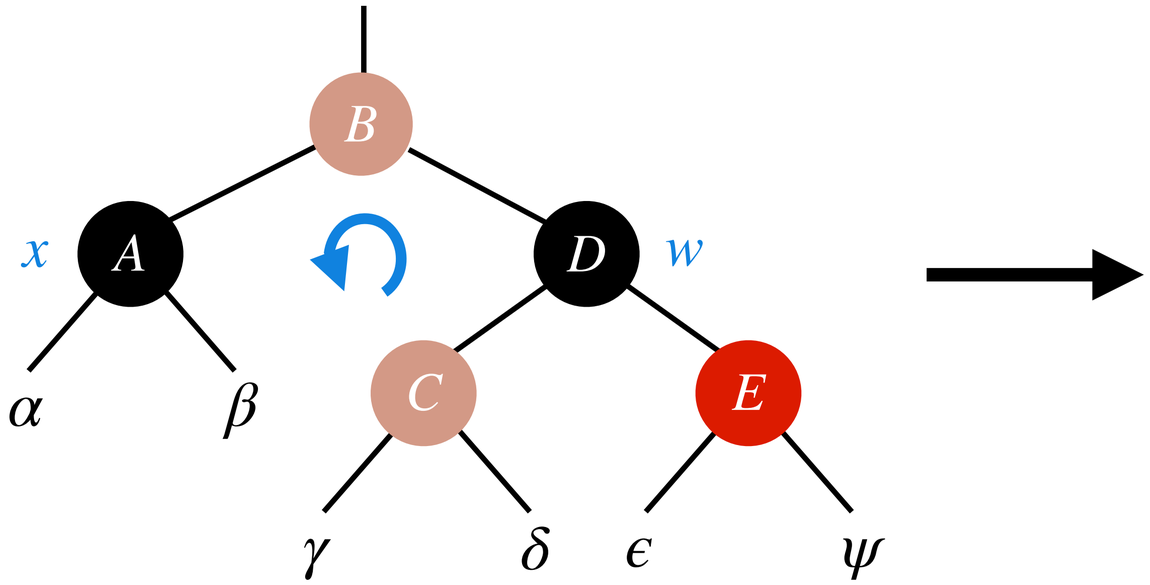


Handling: Give w the colour of x's parent. Make x's parent and w's right child **black**.

Perform left rotation at x's parent.

Fixing Violation of Only Property 5

Case 4: x's sibling w is **black**, and w's right child is **red**.

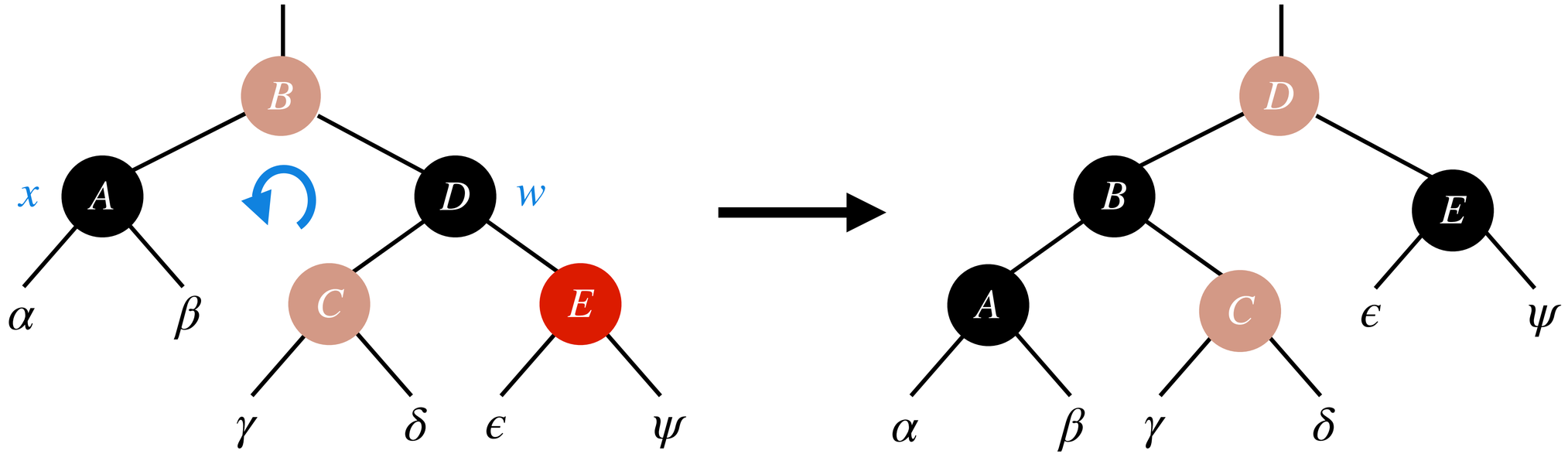


Handling: Give w the colour of x's parent. Make x's parent and w's right child **black**.

Perform left rotation at x's parent.

Fixing Violation of Only Property 5

Case 4: x's sibling w is **black**, and w's right child is **red**.



Handling: Give w the colour of x's parent. Make x's parent and w's right child **black**.

Perform left rotation at x's parent.

Facilitating New Operations

Facilitating New Operations

Defn: In a set, rank of an element is its position in the sorted order (w.r.t. keys) of the set.

Facilitating New Operations

Defn: In a set, rank of an element is its position in the sorted order (w.r.t. keys) of the set.

Example: Let $S = \{10,5,15,20,8,25,40,30\}$ be a set.

Facilitating New Operations

Defn: In a set, rank of an element is its position in the sorted order (w.r.t. keys) of the set.

Example: Let $S = \{10,5,15,20,8,25,40,30\}$ be a set.

Elements with ranks 1, 3, and 8 are 5, 10, and 40, respectively.

Facilitating New Operations

Defn: In a set, rank of an element is its position in the sorted order (w.r.t. keys) of the set.

Example: Let $S = \{10,5,15,20,8,25,40,30\}$ be a set.

Elements with ranks 1, 3, and 8 are 5, 10, and 40, respectively.

Ranks of elements 5, 15, and 25, are 1, 4, and 6, respectively.

Facilitating New Operations

Facilitating New Operations

Suppose we want to maintain a dynamic set with the following operations:

Facilitating New Operations

Suppose we want to maintain a dynamic set with the following operations:

- Find the element of the ith rank.

Facilitating New Operations

Suppose we want to maintain a dynamic set with the following operations:

- Find the element of the ith rank.
- Given an element find its rank in the set.

Facilitating New Operations

Suppose we want to maintain a dynamic set with the following operations:

- Find the element of the ith rank.
- Given an element find its rank in the set.

Should we learn or invent a new data structure to facilitate these operations?

Facilitating New Operations

Suppose we want to maintain a dynamic set with the following operations:

- Find the element of the ith rank.
- Given an element find its rank in the set.

Should we learn or invent a new data structure to facilitate these operations?

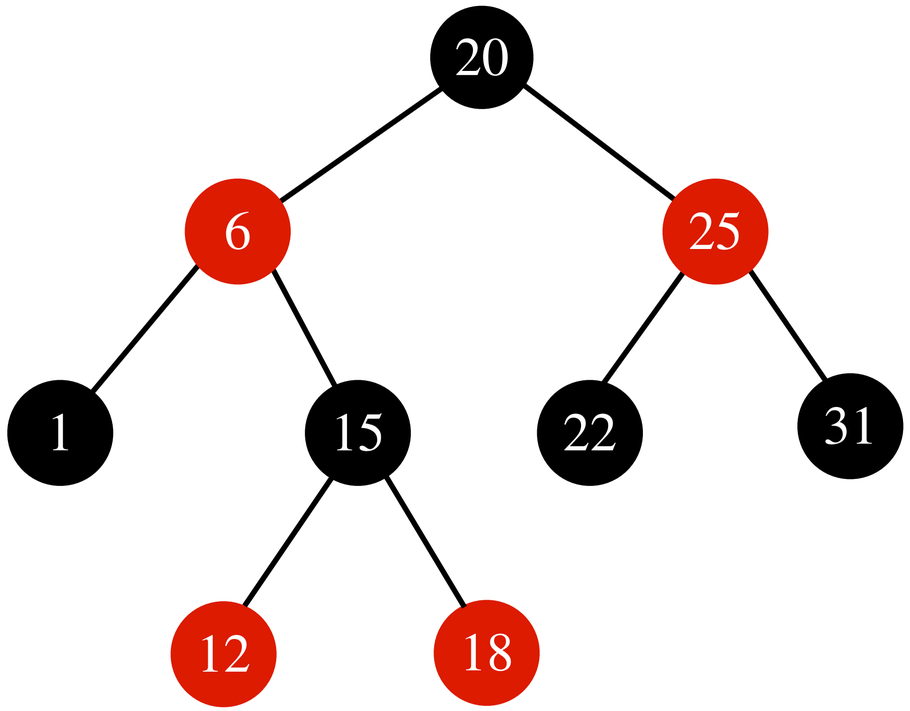
No. An easy modification to RB-trees is enough.

A Minor Modification to RB-Tree

Every node also stores the number of internal nodes in the subtree rooted at that node.

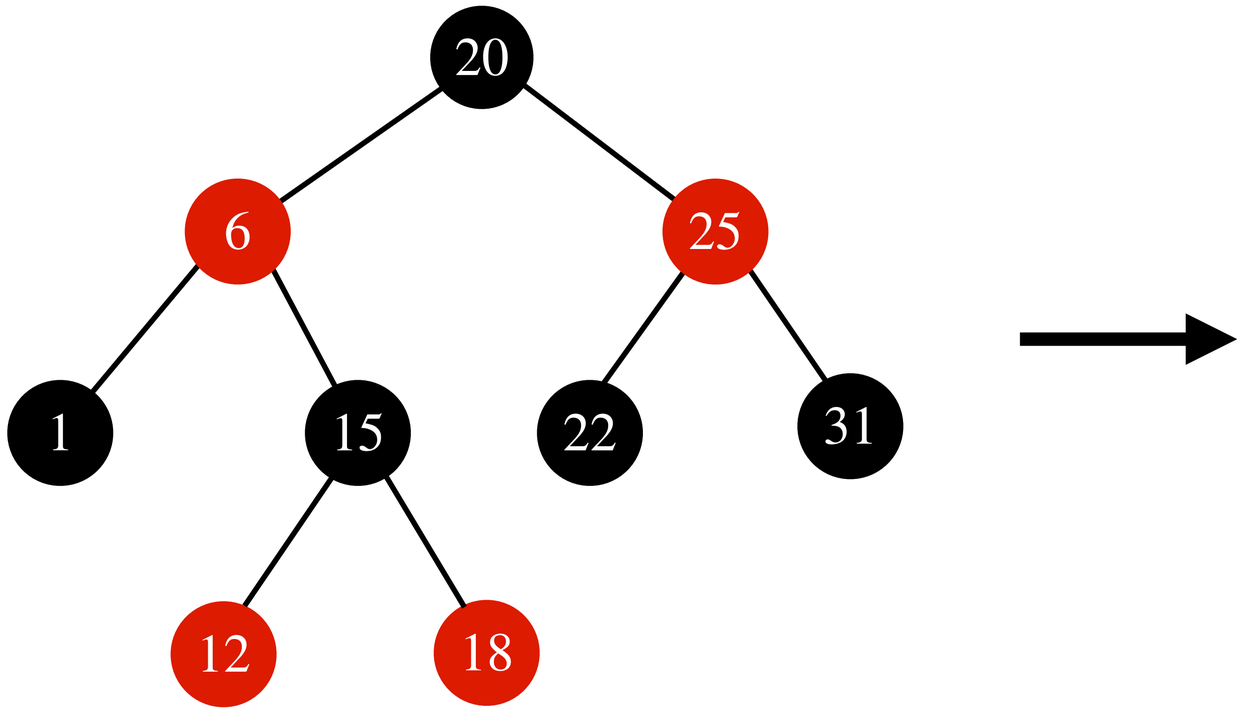
A Minor Modification to RB-Tree

Every node also stores the number of internal nodes in the subtree rooted at that node.



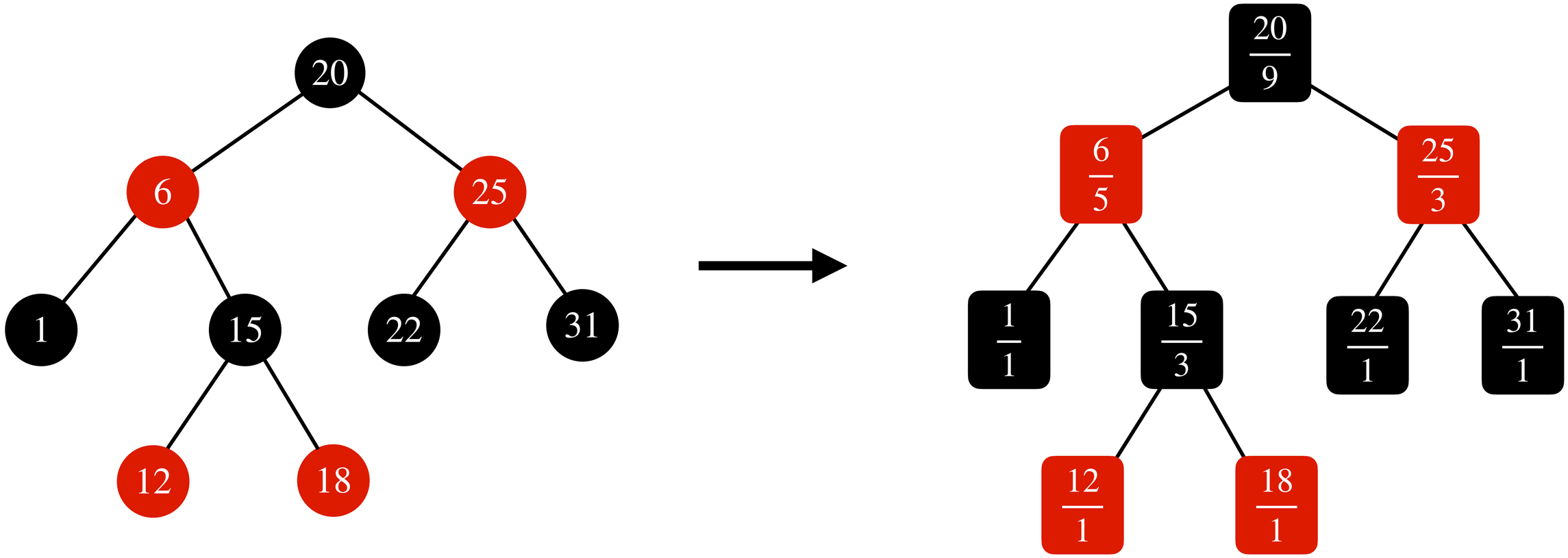
A Minor Modification to RB-Tree

Every node also stores the number of internal nodes in the subtree rooted at that node.



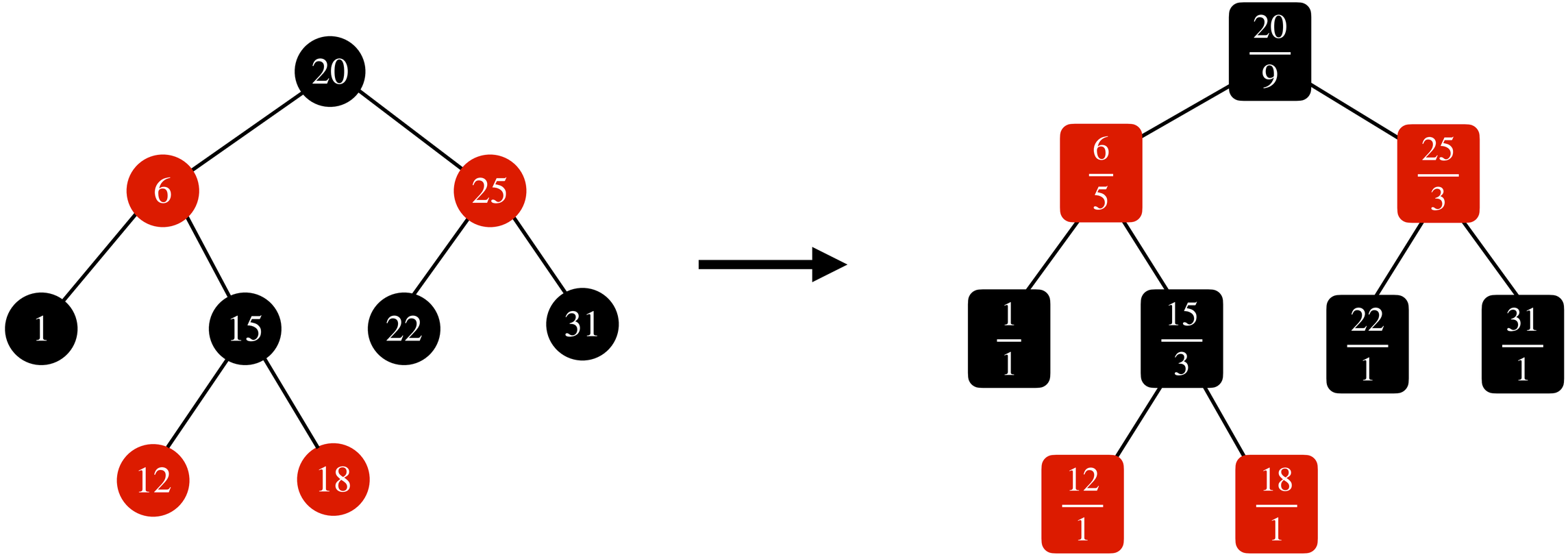
A Minor Modification to RB-Tree

Every node also stores the number of internal nodes in the subtree rooted at that node.



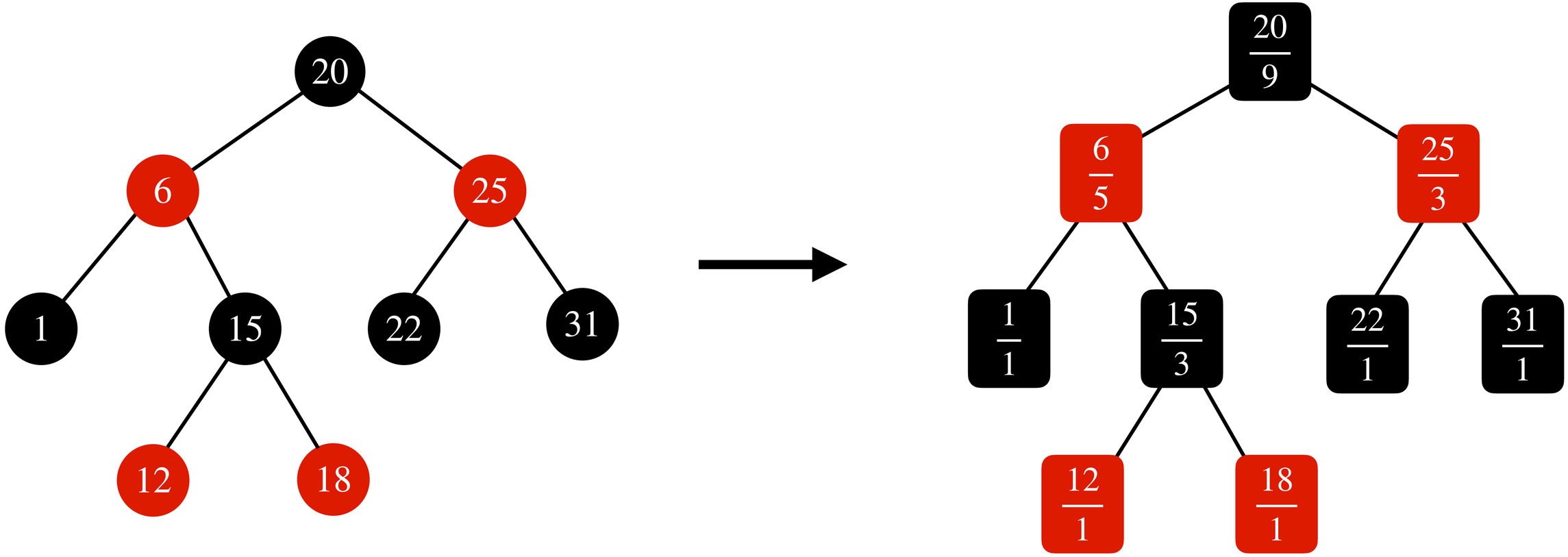
A Minor Modification to RB-Tree

Nodes are represented as $\frac{x}{y}$, where x is the key, and y is number of internal nodes in the subtree



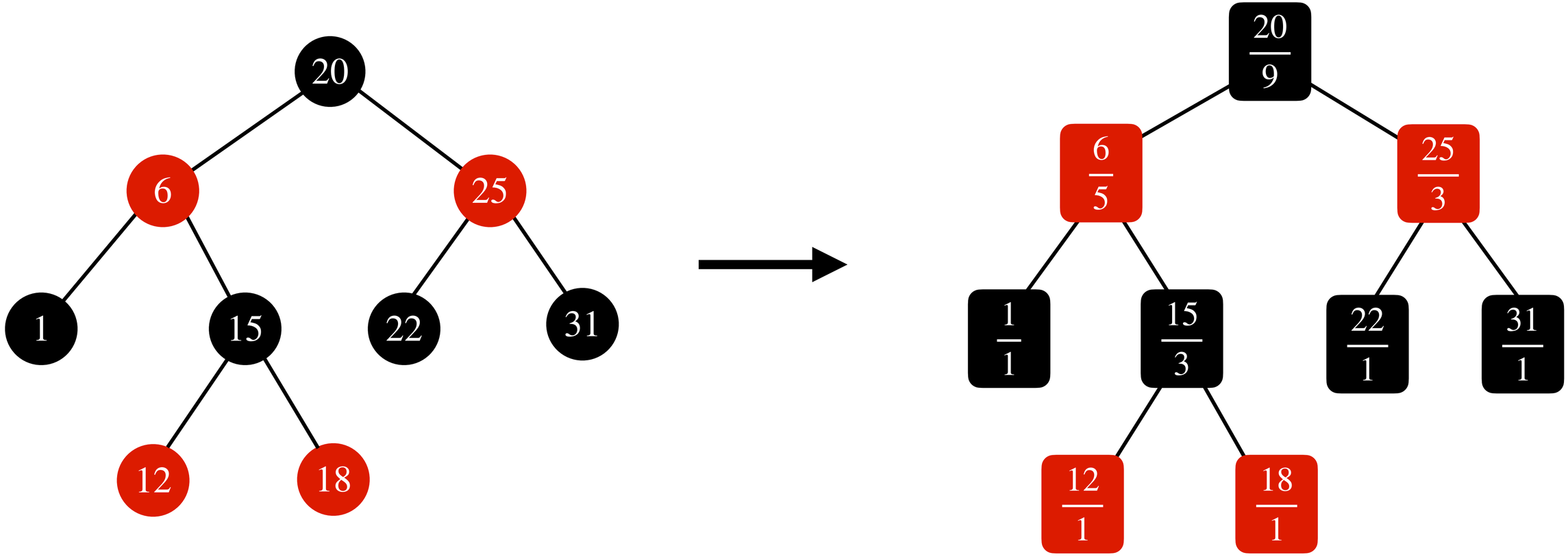
A Minor Modification to RB-Tree

Nodes are represented as $\frac{x}{y}$, where x is the key, and y is number of internal nodes in the subtree rooted at that node.



A Minor Modification to RB-Tree

Nodes are represented as $\frac{x}{y}$, where x is the key, and y is number of internal nodes in the subtree rooted at that node. For a node z, number of internal nodes in the subtree(z) = $z\,.\,size$.



Finding the Element with ith Rank

Recall that rank of an element is its position in the sorted order (w.r.t. keys) of the set.

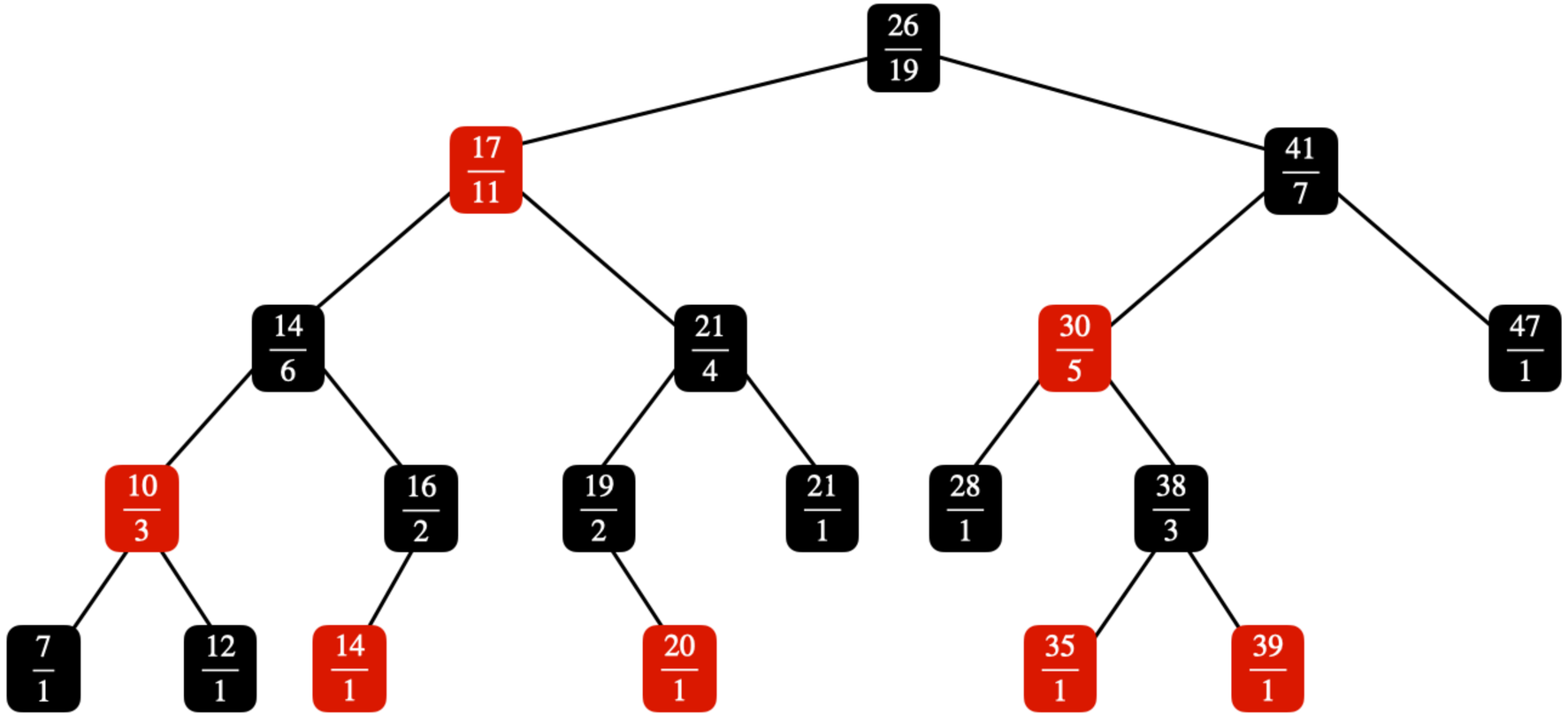
Finding the Element with ith Rank

Finding the Element with ith Rank

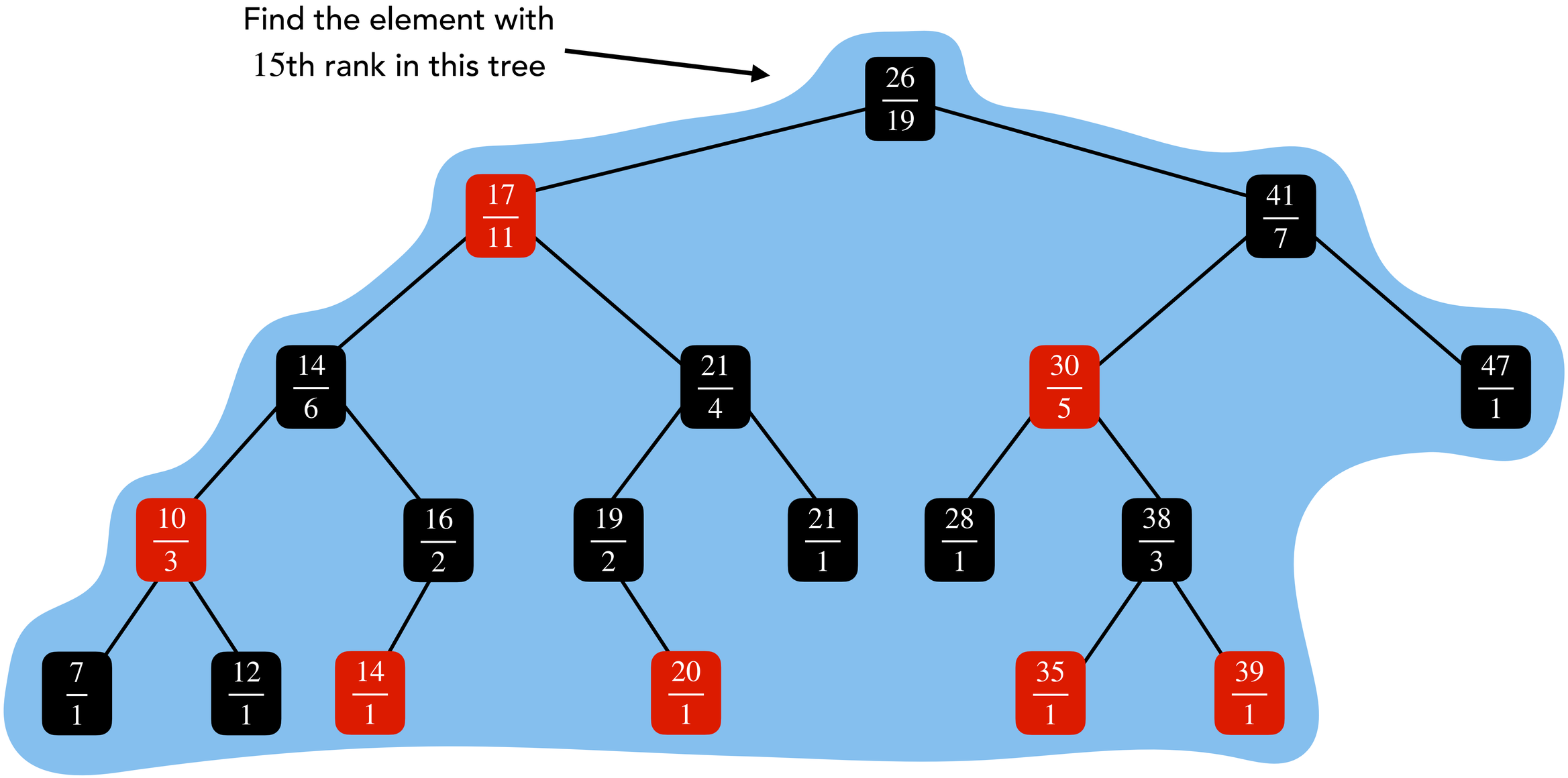
Example: Find an element with 15th rank in the below **set** or RB-tree.

Finding the Element with ith Rank

Example: Find an element with 15th rank in the below **set** or RB-tree.

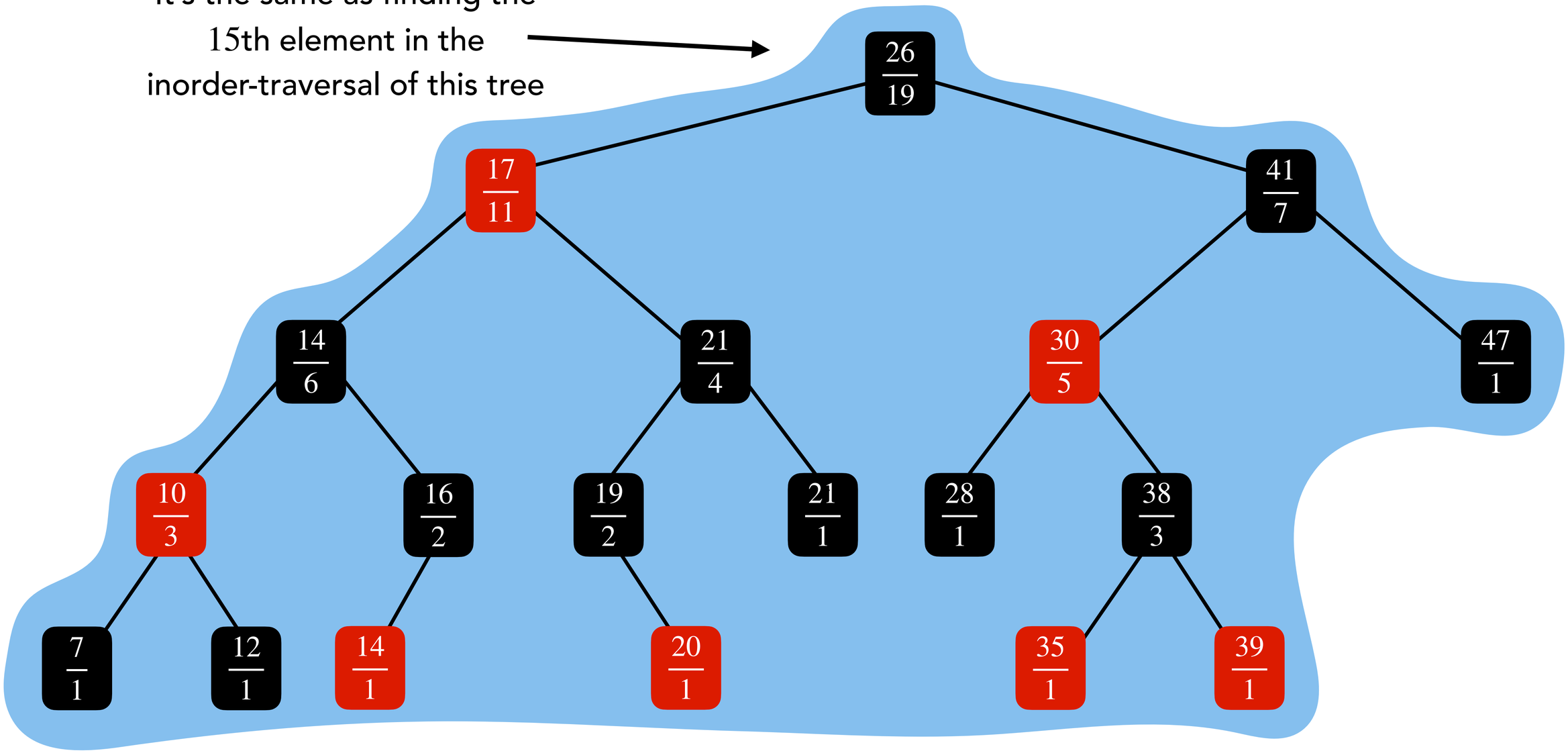


Finding the Element with ith Rank

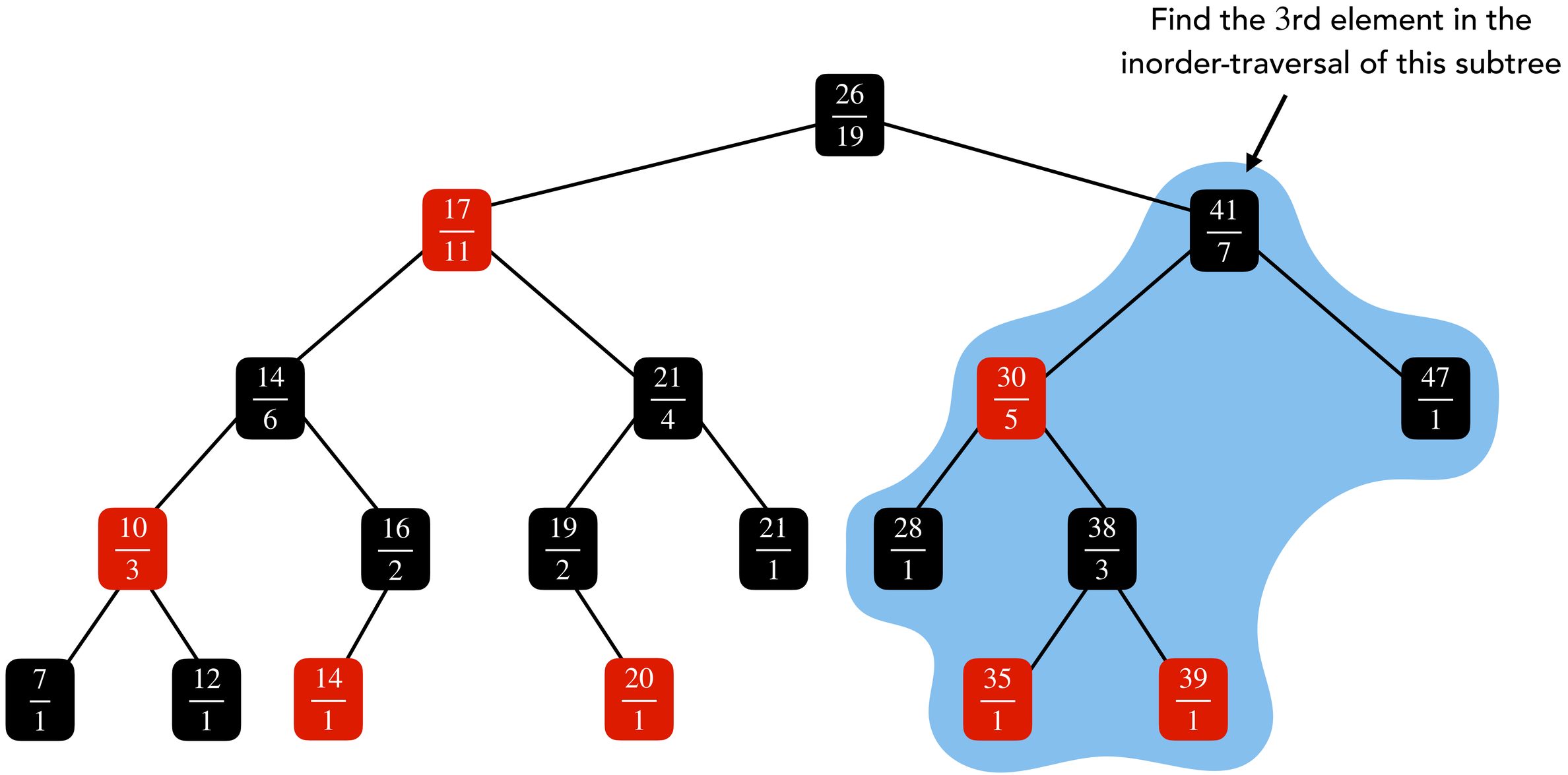


Finding the Element with ith Rank

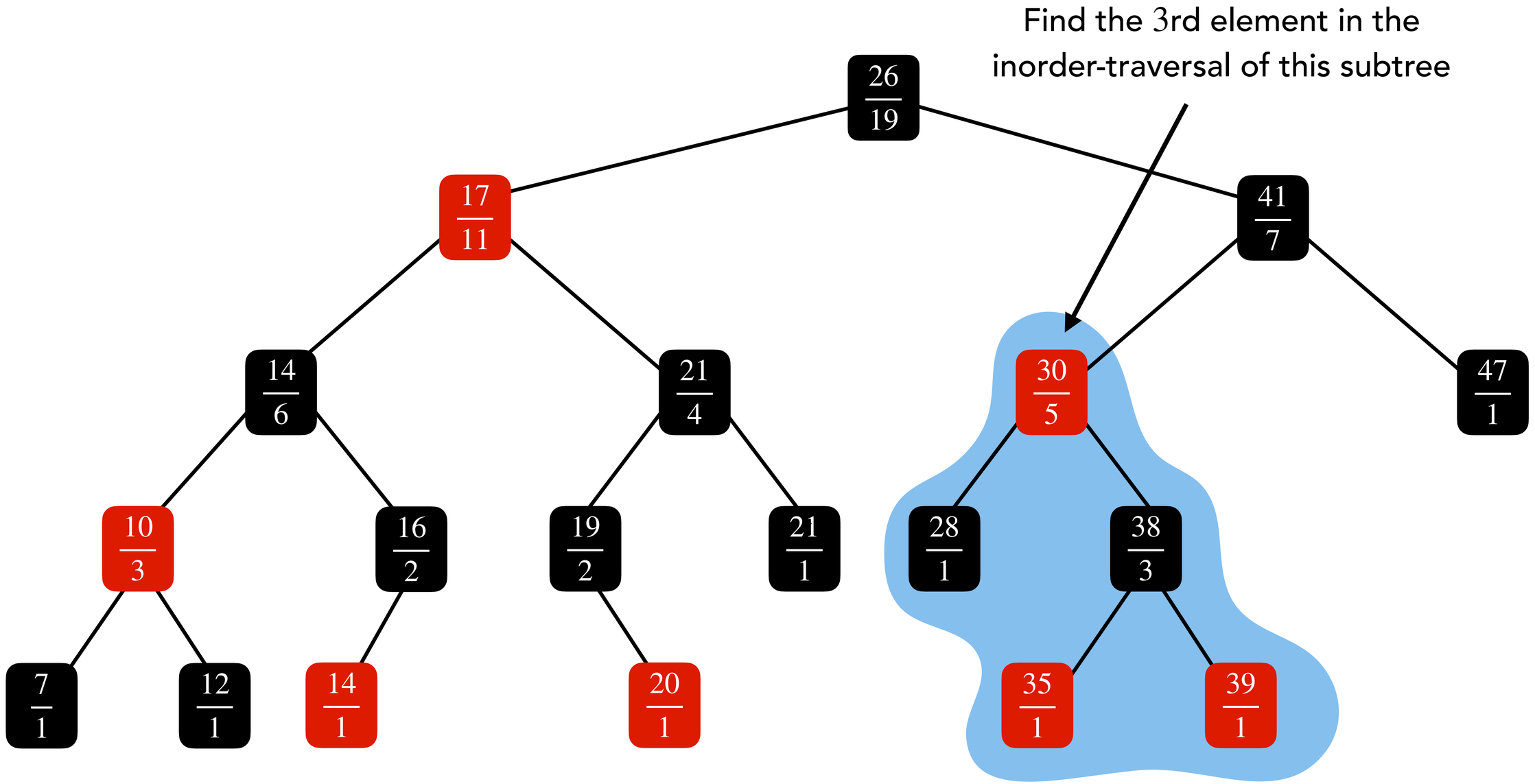
It's the same as finding the 15th element in the inorder-traversal of this tree



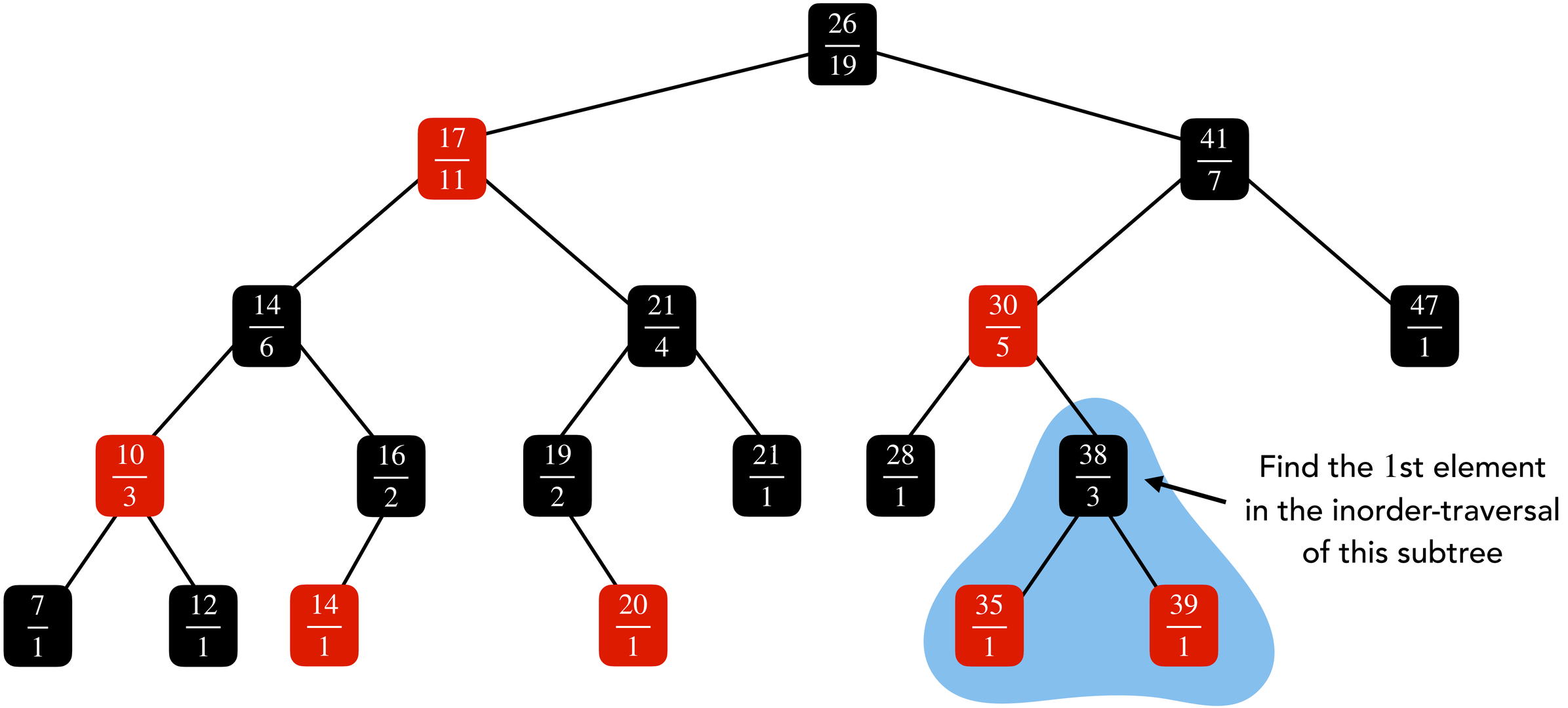
Finding the Element with ith Rank



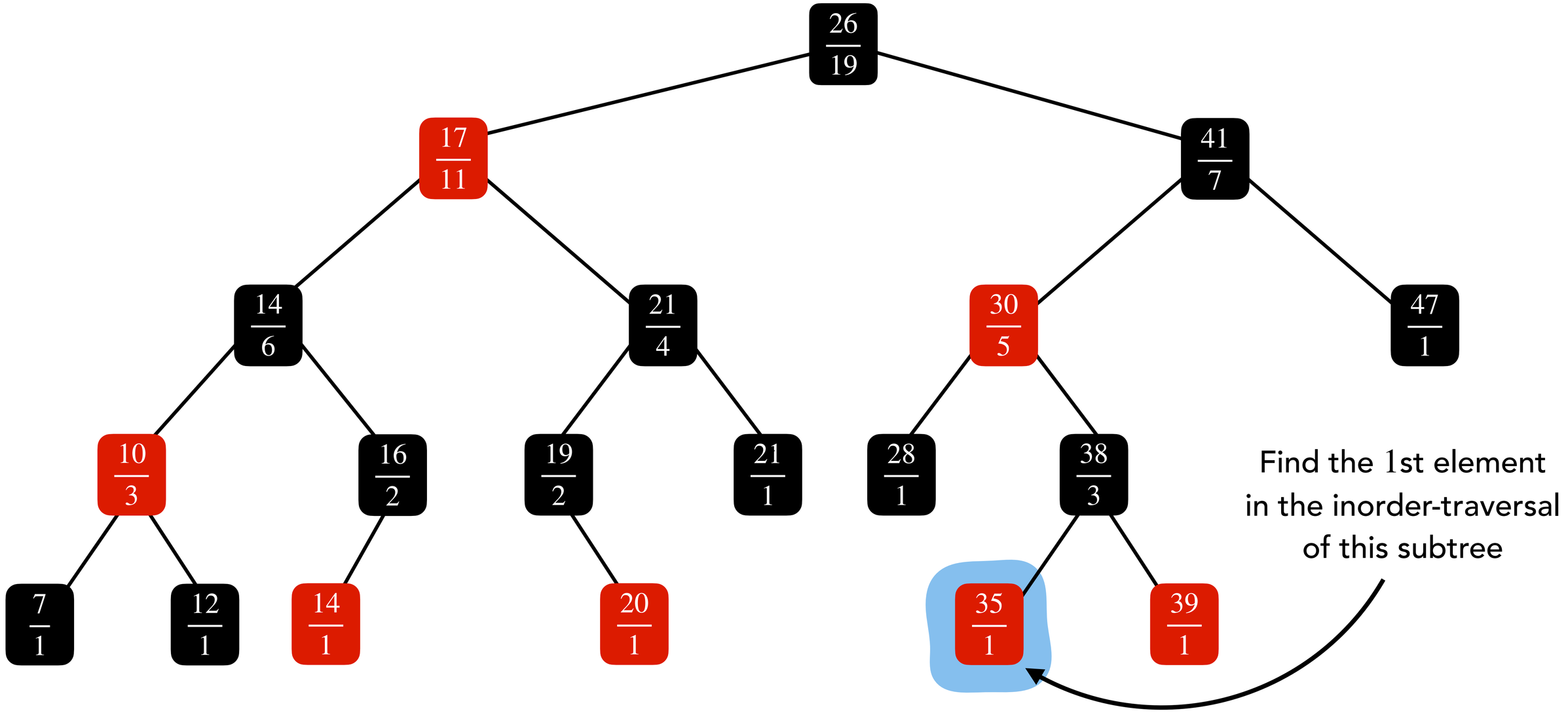
Finding the Element with ith Rank



Finding the Element with ith Rank



Finding the Element with ith Rank



Finding the Element with ith Rank