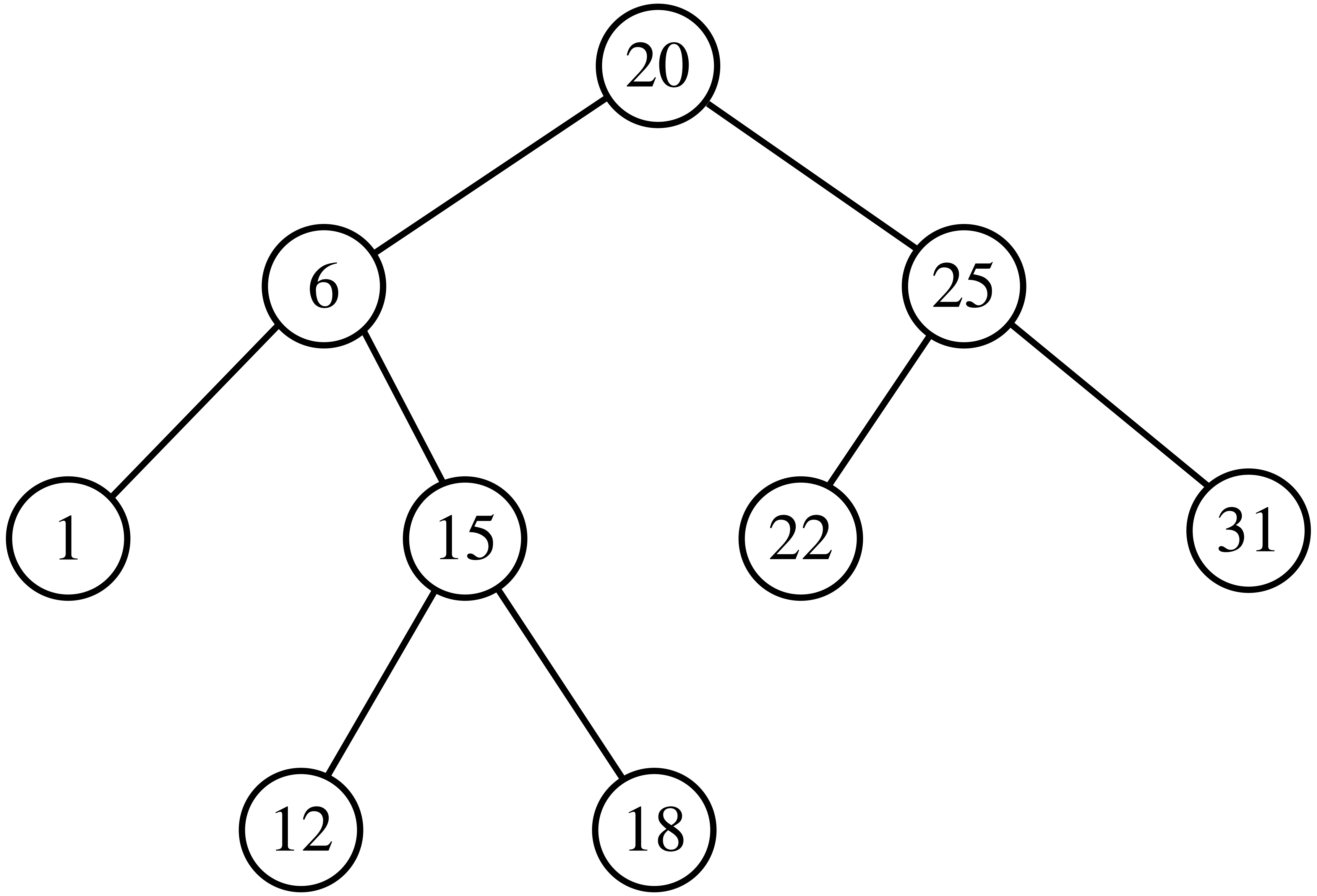
Lecture 7

BST (Insertion & Deletion), Red-Black Trees

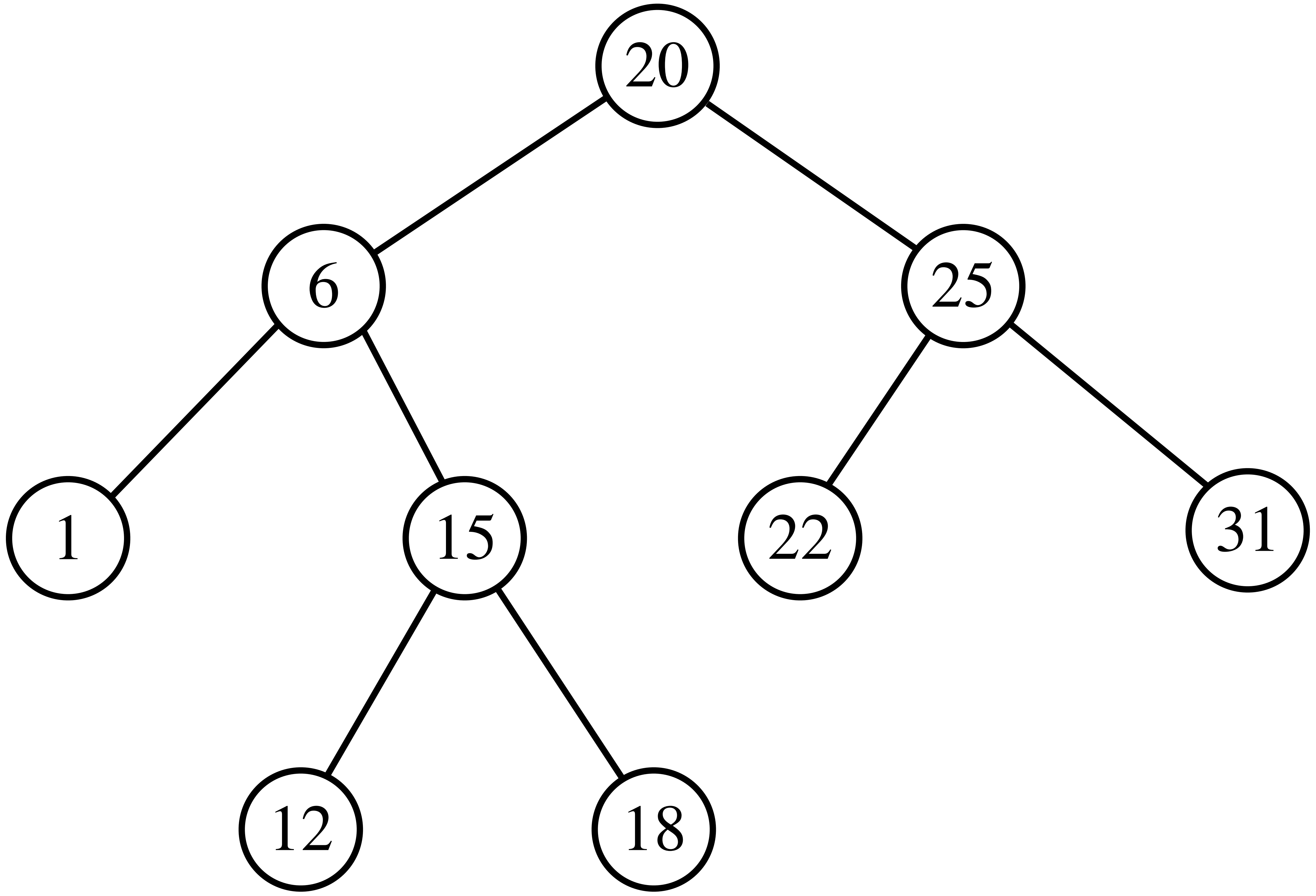
Modifying a BST: Insertion

Modifying a BST: Insertion



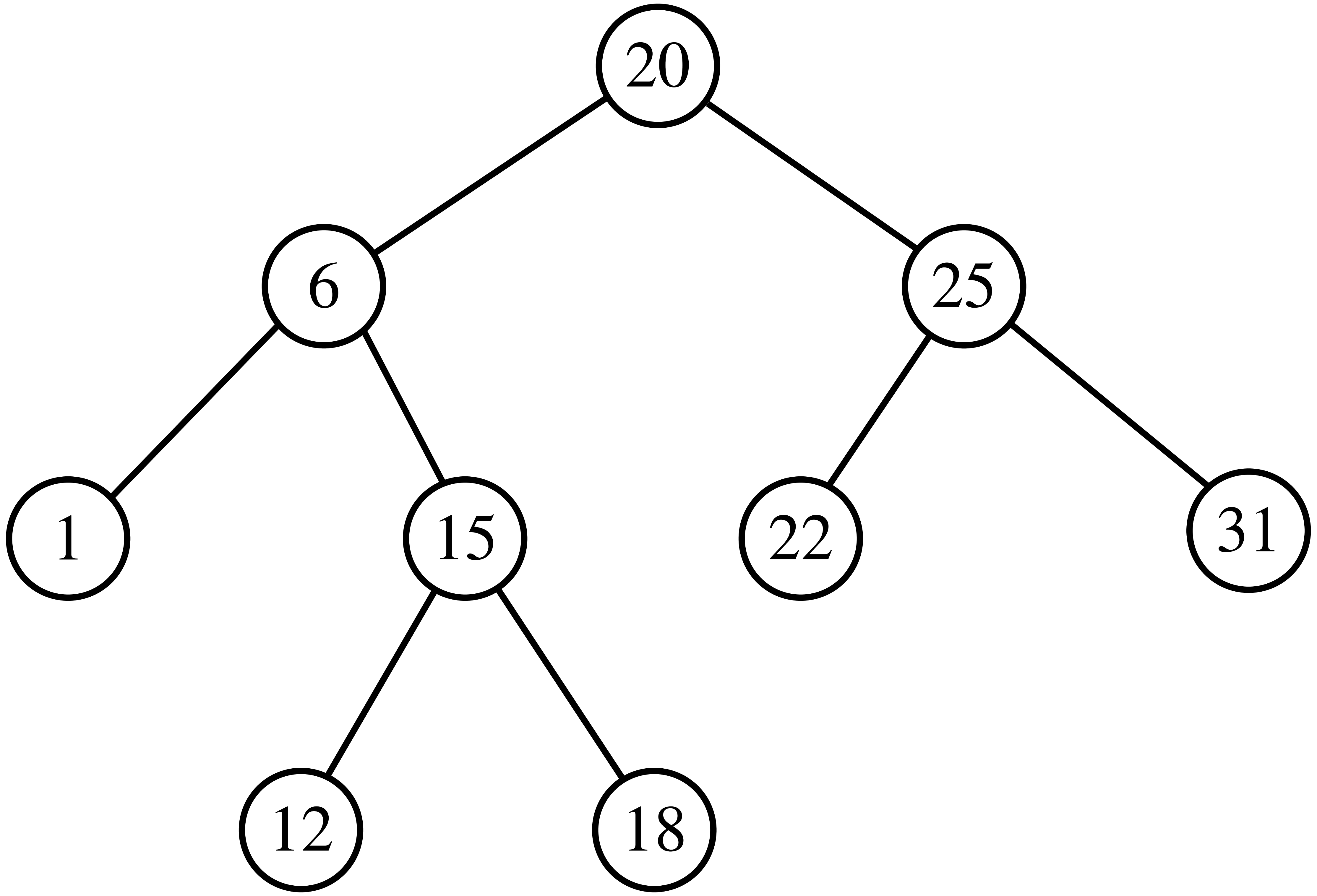
Modifying a BST: Insertion

Example:



Modifying a BST: Insertion

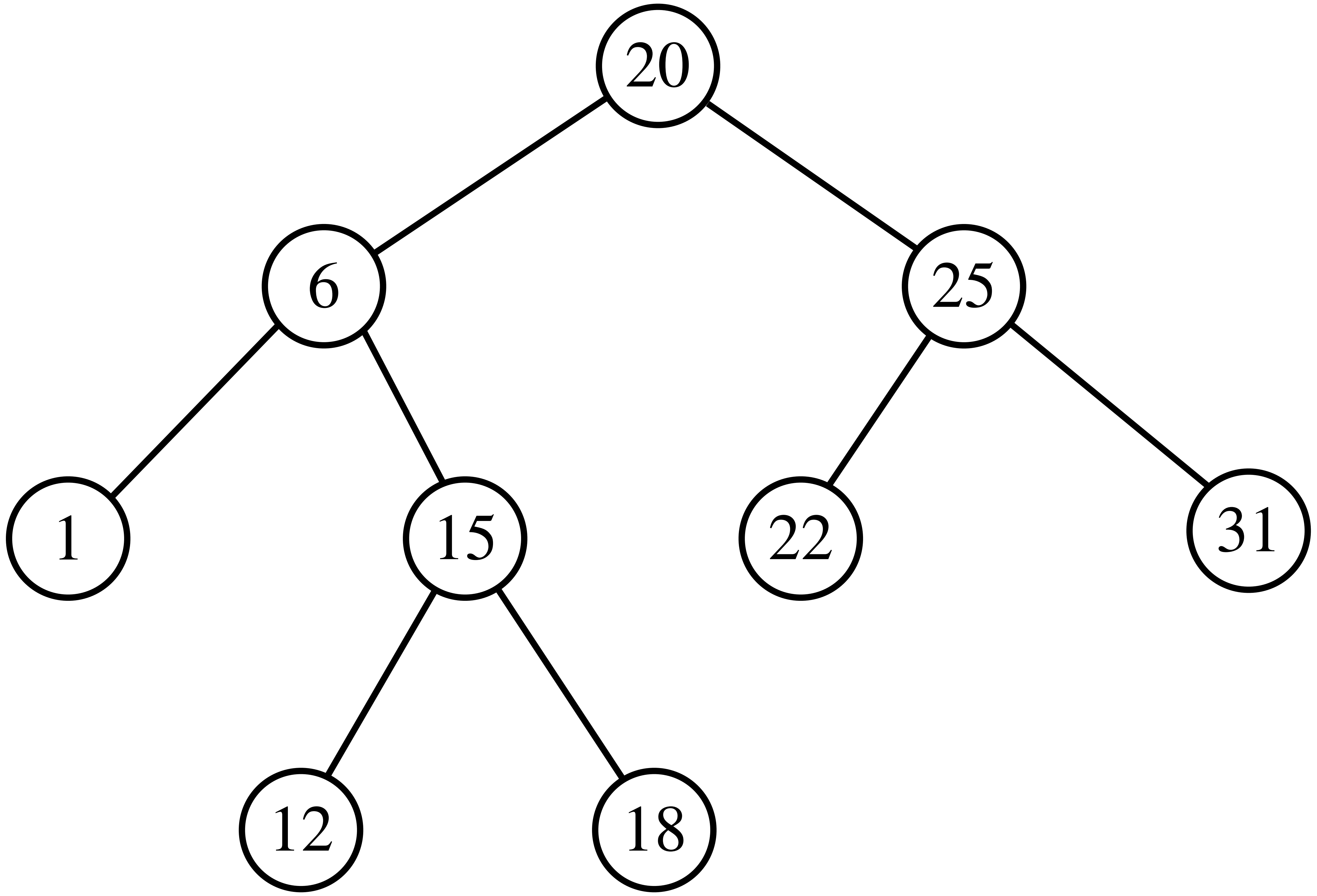
Example: Insert a node with 24 as key in the following BST.



Modifying a BST: Insertion

Example: Insert a node with 24 as key in the following BST.

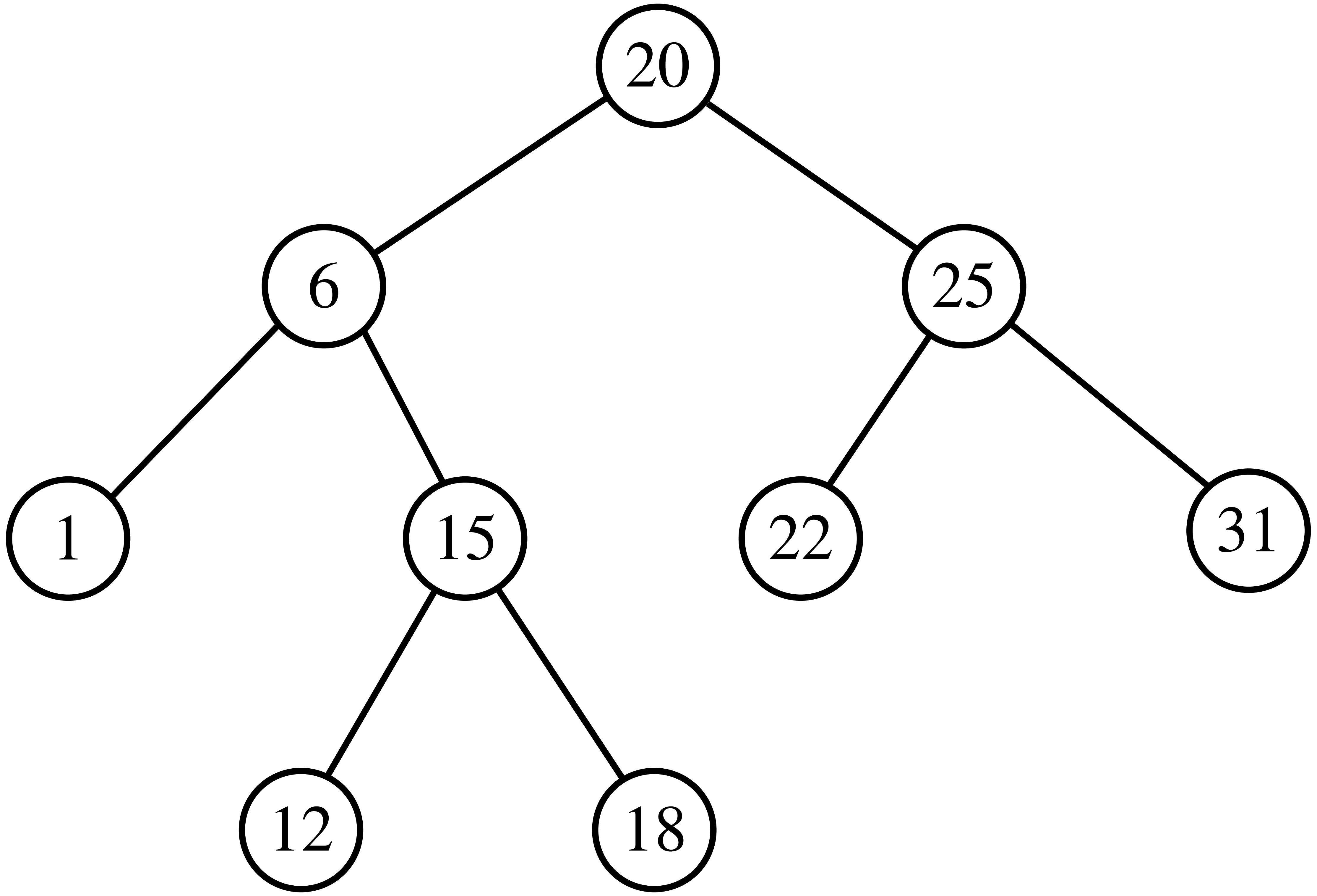
Idea:



Modifying a BST: Insertion

Example: Insert a node with 24 as key in the following BST.

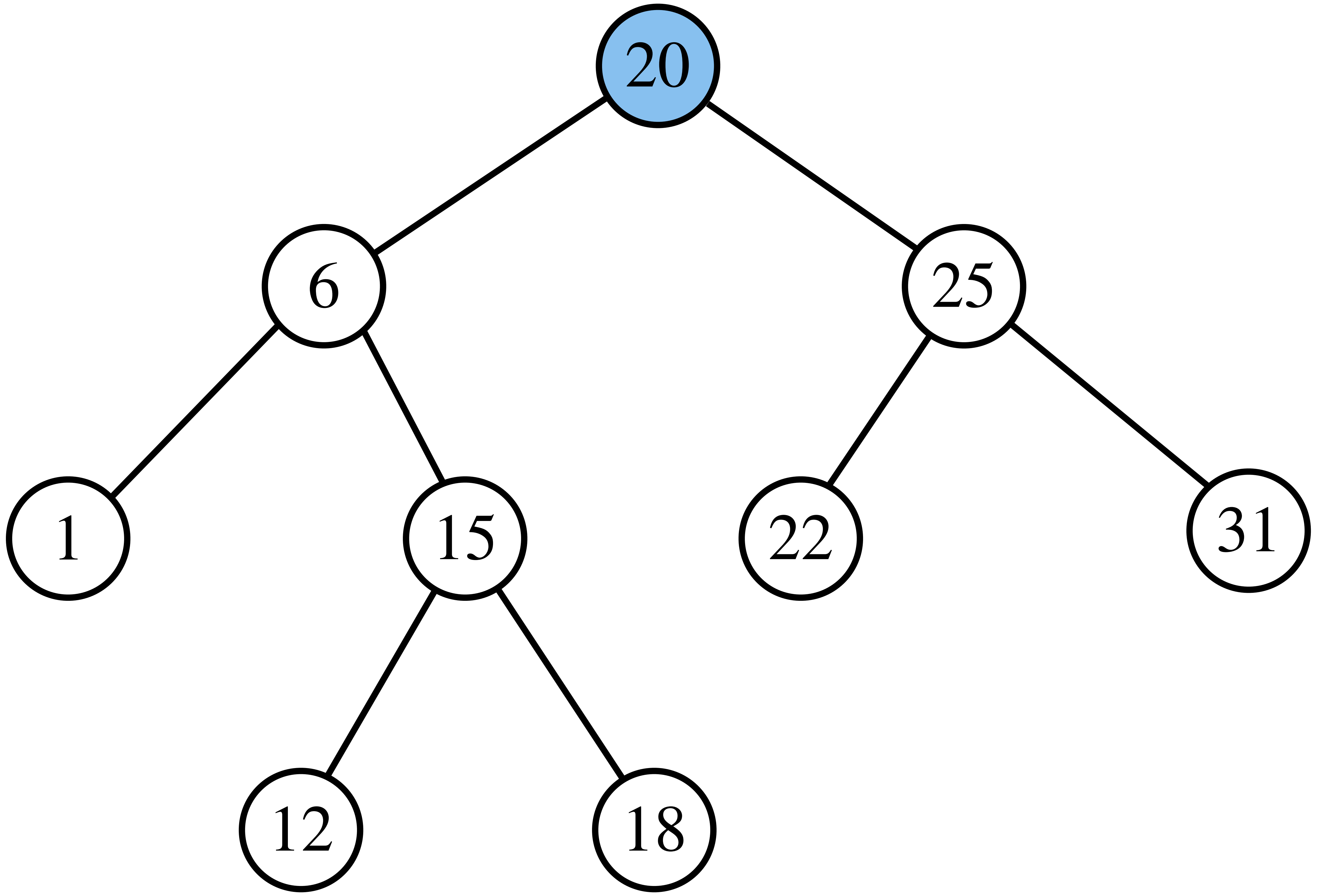
Idea: Find the correct leaf where it can be inserted.



Modifying a BST: Insertion

Example: Insert a node with 24 as key in the following BST.

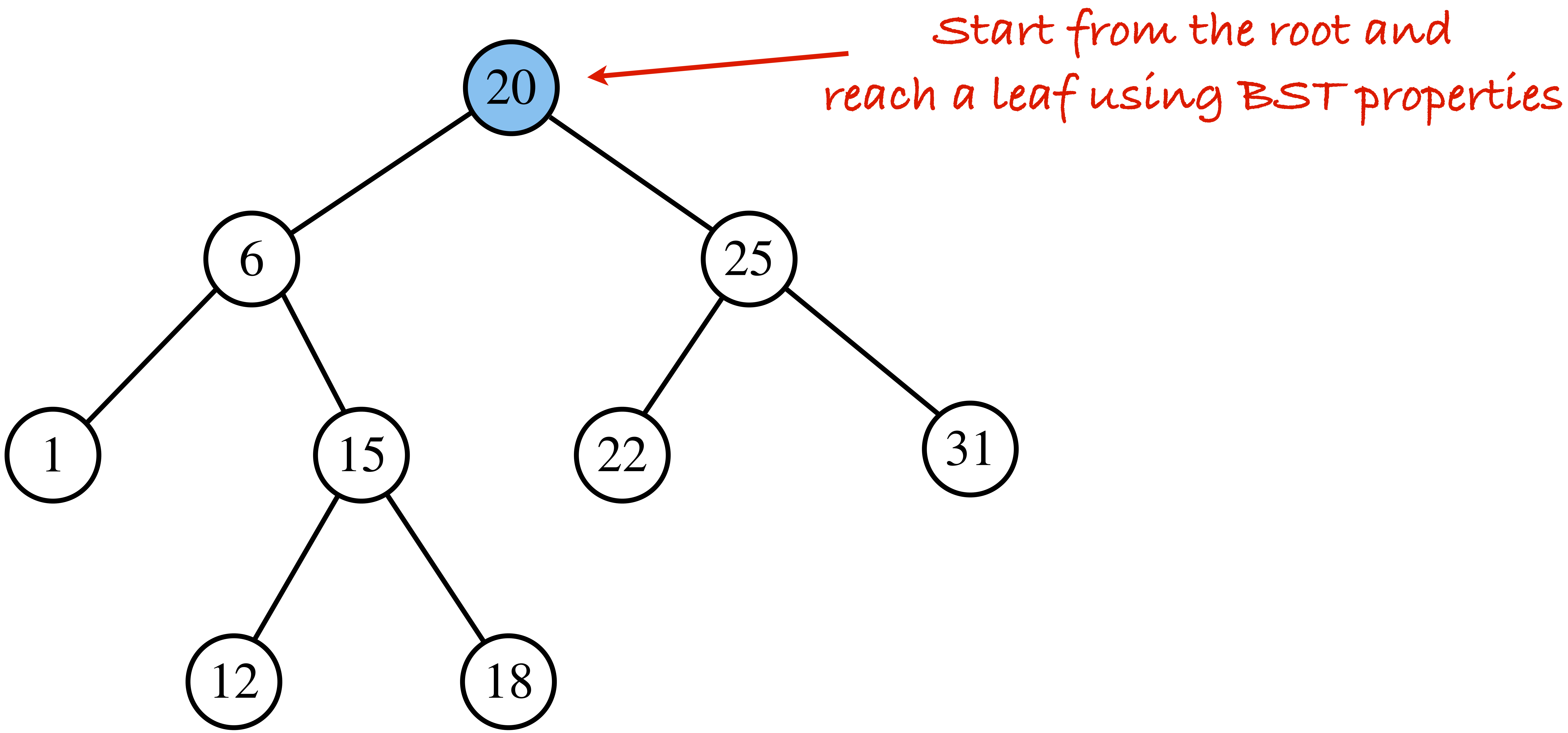
Idea: Find the correct leaf where it can be inserted.



Modifying a BST: Insertion

Example: Insert a node with 24 as key in the following BST.

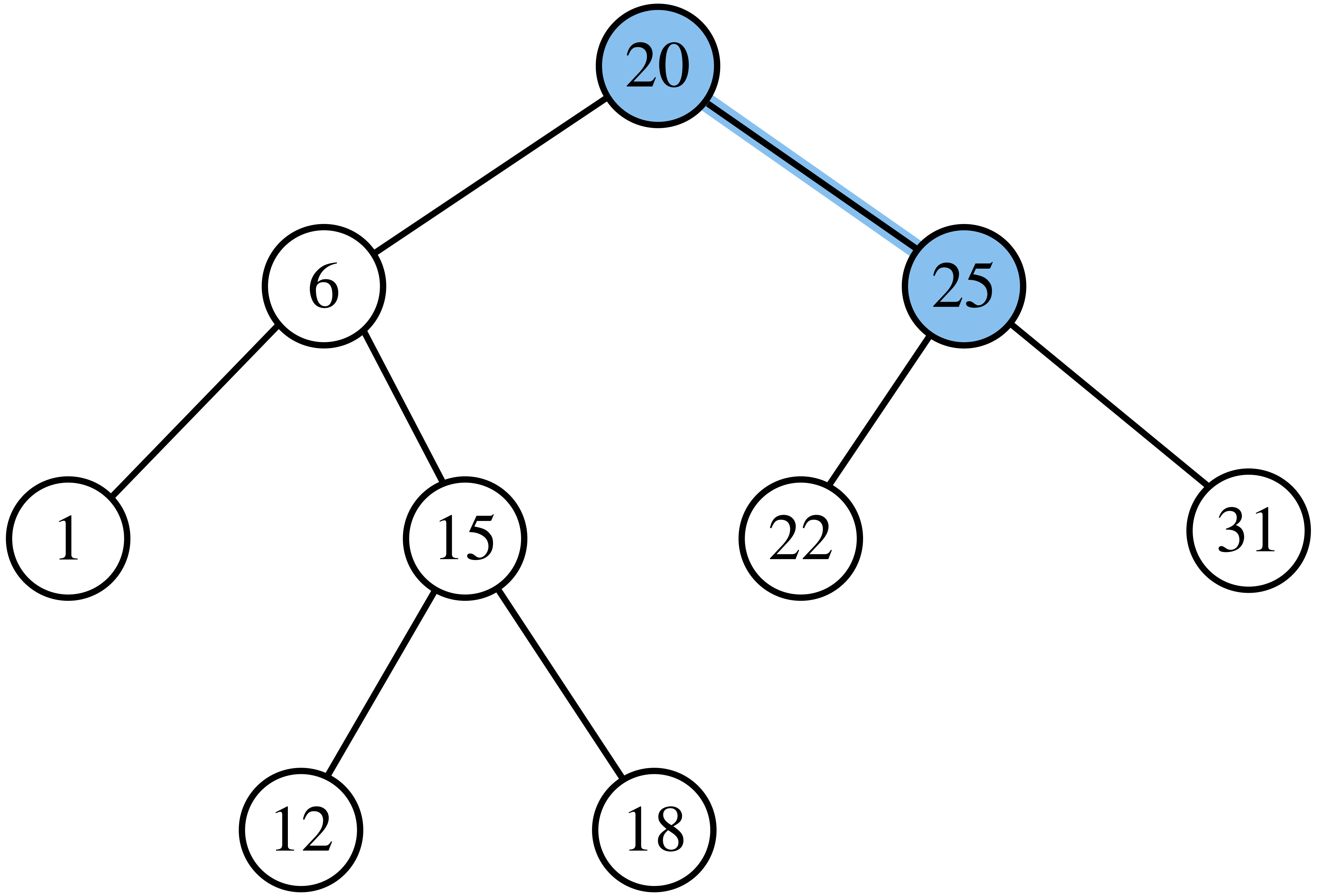
Idea: Find the correct leaf where it can be inserted.



Modifying a BST: Insertion

Example: Insert a node with 24 as key in the following BST.

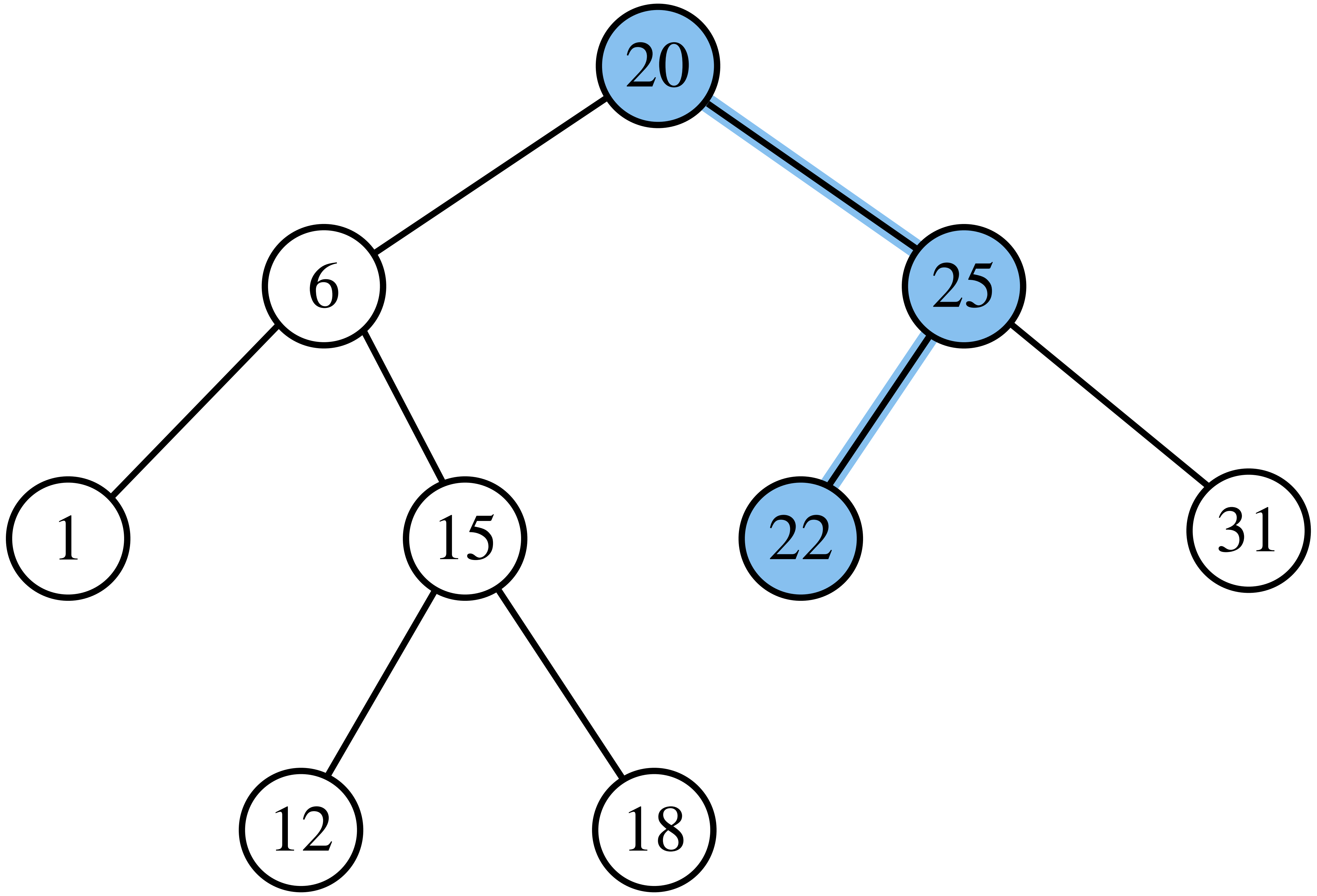
Idea: Find the correct leaf where it can be inserted.



Modifying a BST: Insertion

Example: Insert a node with 24 as key in the following BST.

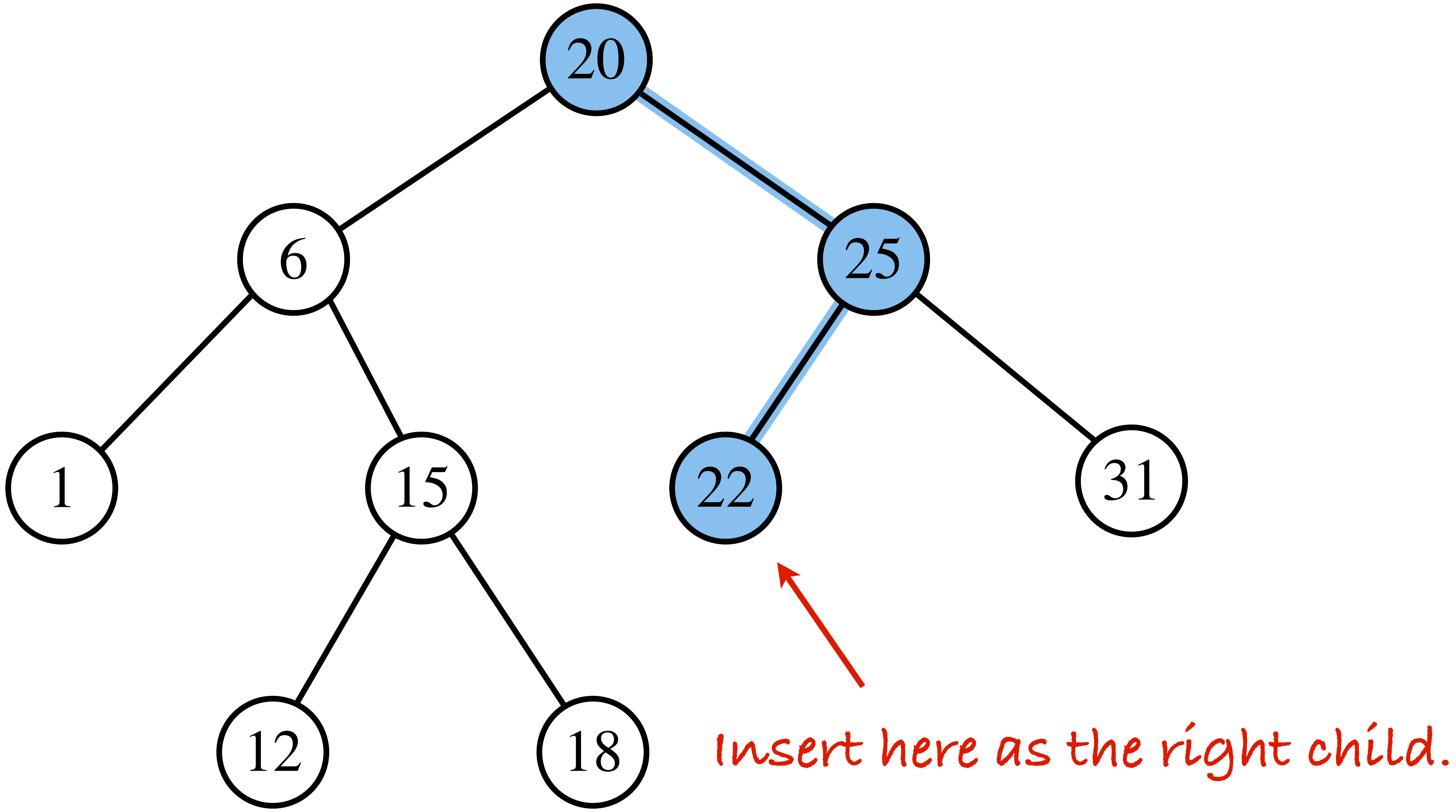
Idea: Find the correct leaf where it can be inserted.



Modifying a BST: Insertion

Example: Insert a node with 24 as key in the following BST.

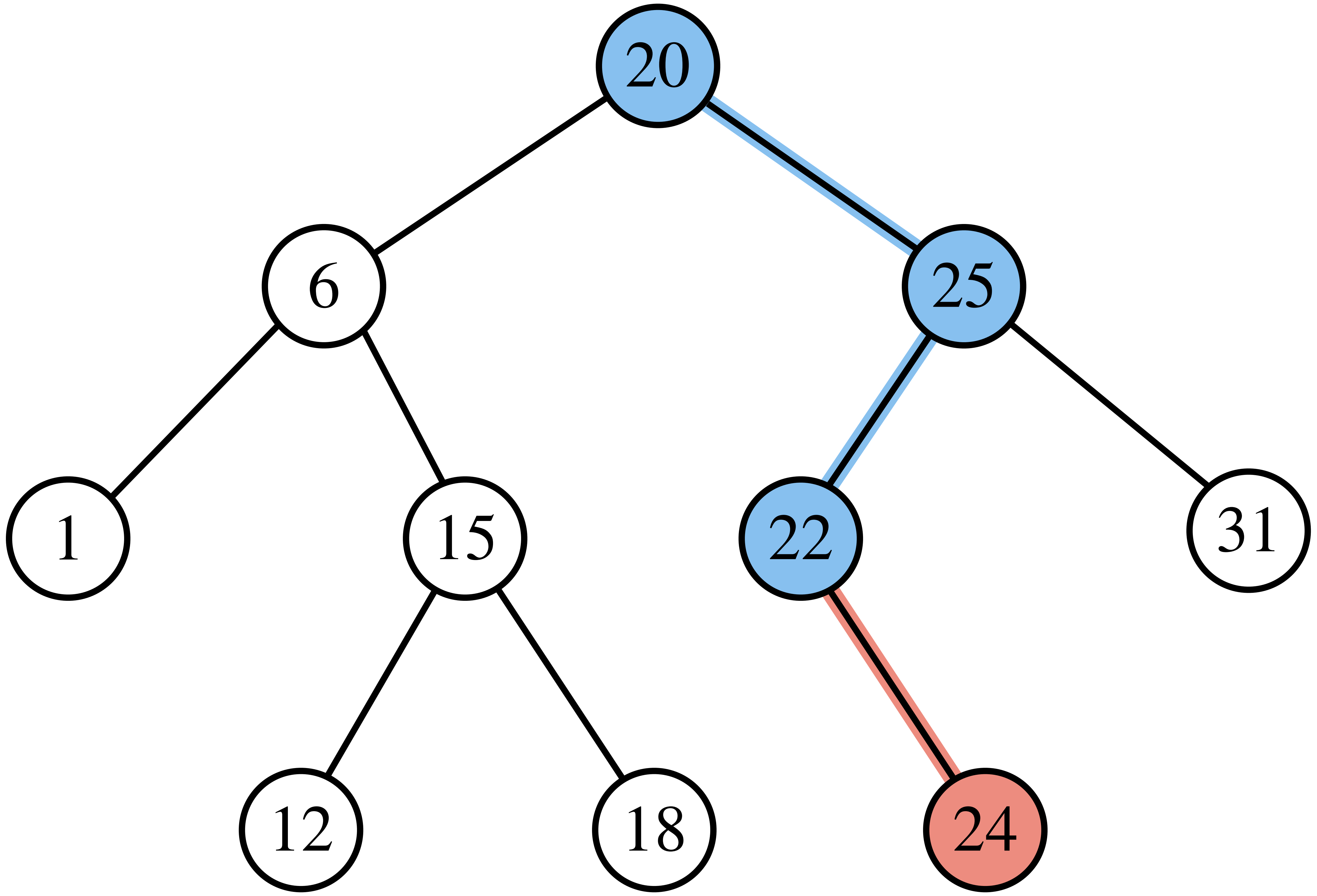
Idea: Find the correct leaf where it can be inserted.



Modifying a BST: Insertion

Example: Insert a node with 24 as key in the following BST.

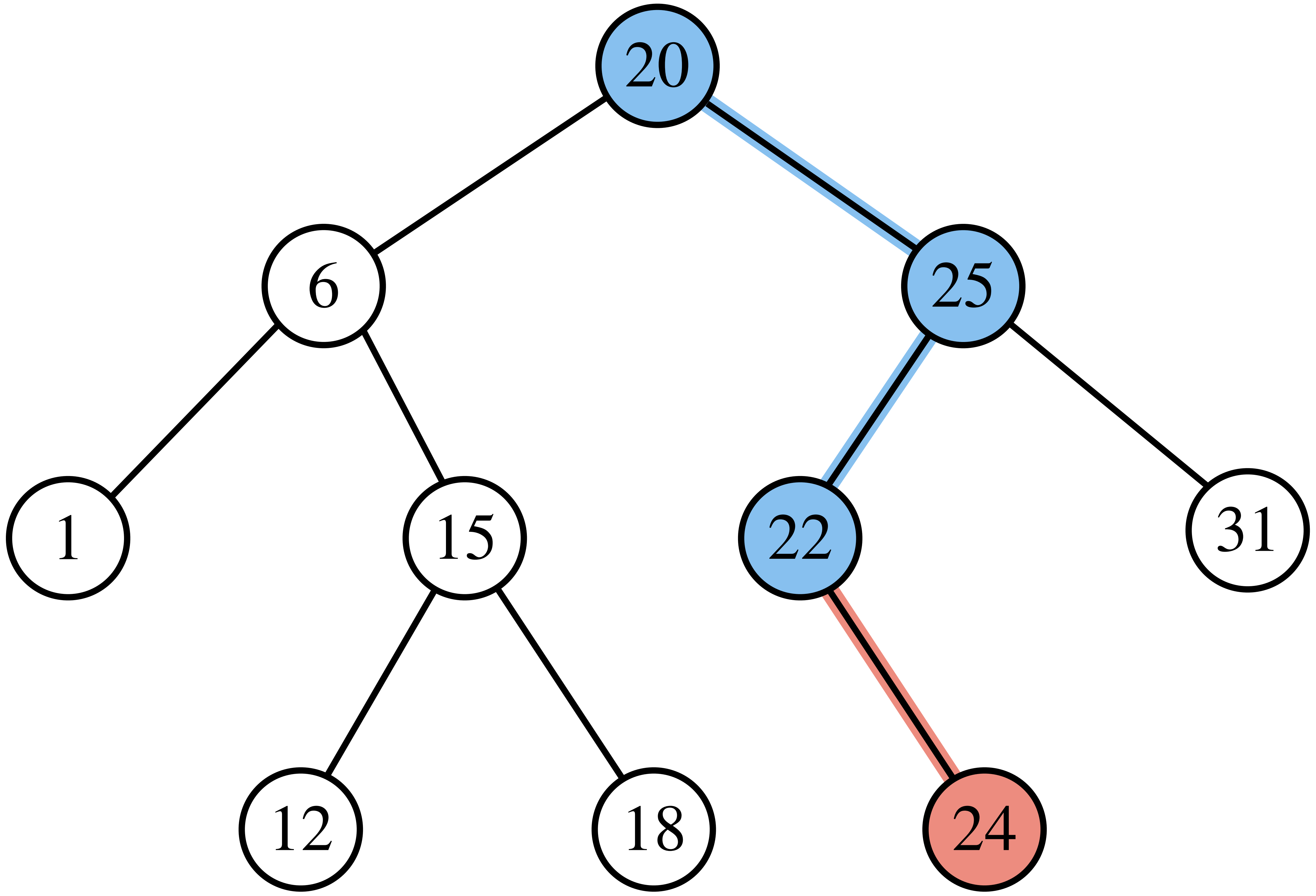
Idea: Find the correct leaf where it can be inserted.



Modifying a BST: Insertion

Example: Insert a node with 24 as key in the following BST.

Idea: Find the correct leaf where it can be inserted.



Searching for the correct leaf and insertion takes $O(h)$ time.

Modifying a BST: Deletion

Modifying a BST: Deletion

Deletion can be more tricky than Insertion.

Modifying a BST: Deletion

Deletion can be more tricky than Insertion.

Let z be the node we want to delete.

Modifying a BST: Deletion

Deletion can be more tricky than Insertion.

Let z be the node we want to delete. Then, the following cases are possible:

Modifying a BST: Deletion

Deletion can be more tricky than Insertion.

Let z be the node we want to delete. Then, the following cases are possible:

- **Case** 1: z has no children.

Modifying a BST: Deletion

Deletion can be more tricky than Insertion.

Let z be the node we want to delete. Then, the following cases are possible:

- **Case** 1: z has no children.
- **Case** 2: z has only single child.

Modifying a BST: Deletion

Deletion can be more tricky than Insertion.

Let z be the node we want to delete. Then, the following cases are possible:

- **Case** 1: z has no children.
- **Case** 2: z has only single child.
- **Case** 3: z has two children.

Modifying a BST: Deletion

Deletion can be more tricky than Insertion.

Let z be the node we want to delete. Then, the following cases are possible:

- **Case** 1: z has no children.
- **Case** 2: z has only single child.
- **Case** 3: z has two children.

Easy

Modifying a BST: Deletion

Deletion can be more tricky than Insertion.

Let z be the node we want to delete. Then, the following cases are possible:

- **Case** 1: z has no children.
- **Case** 2: z has only single child.
- **Case** 3: z has two children.

Easy

Tricky

Modifying a BST: Deletion

Deletion can be more tricky than Insertion.

Let z be the node we want to delete. Then, the following cases are possible:

- **Case** 1: z has no children.
- **Case** 2: z has only single child.

Easy

- **Case** 3: z has two children.

Tricky

Note: Node z is provided as input.

Modifying a BST: Deletion

Modifying a BST: Deletion

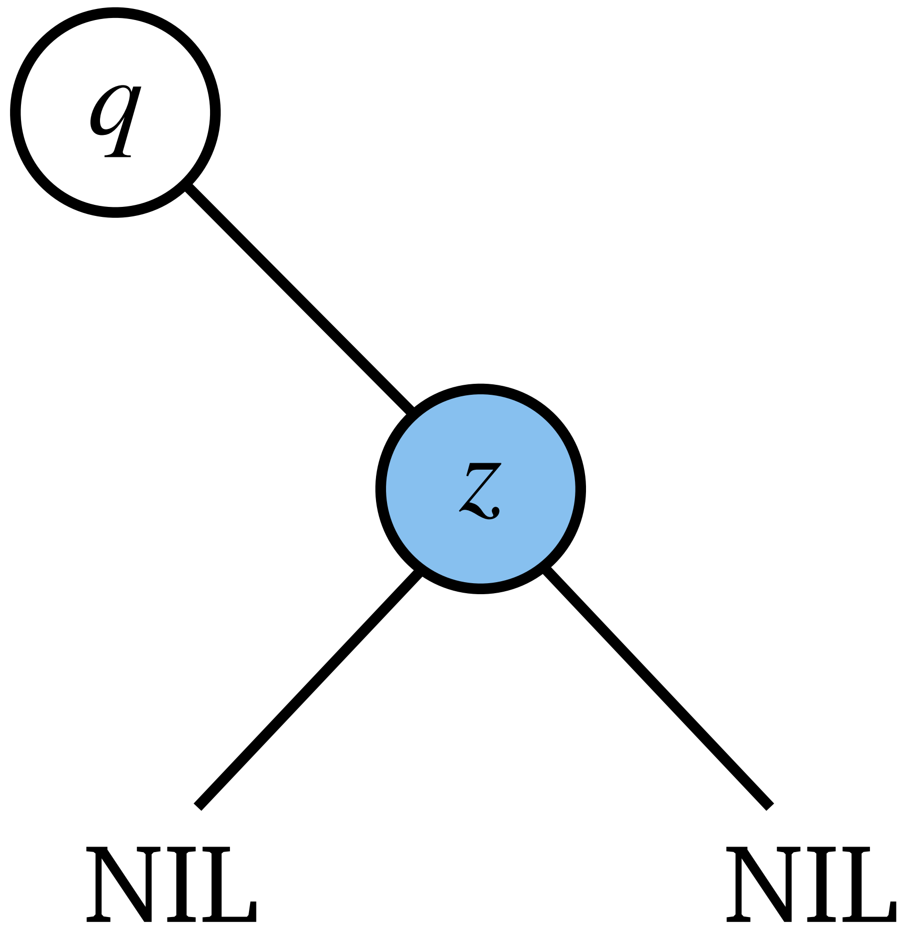
Case 1: z has no children.

Modifying a BST: Deletion

Case 1: z has no children. (WLOG assume z is a right child.)

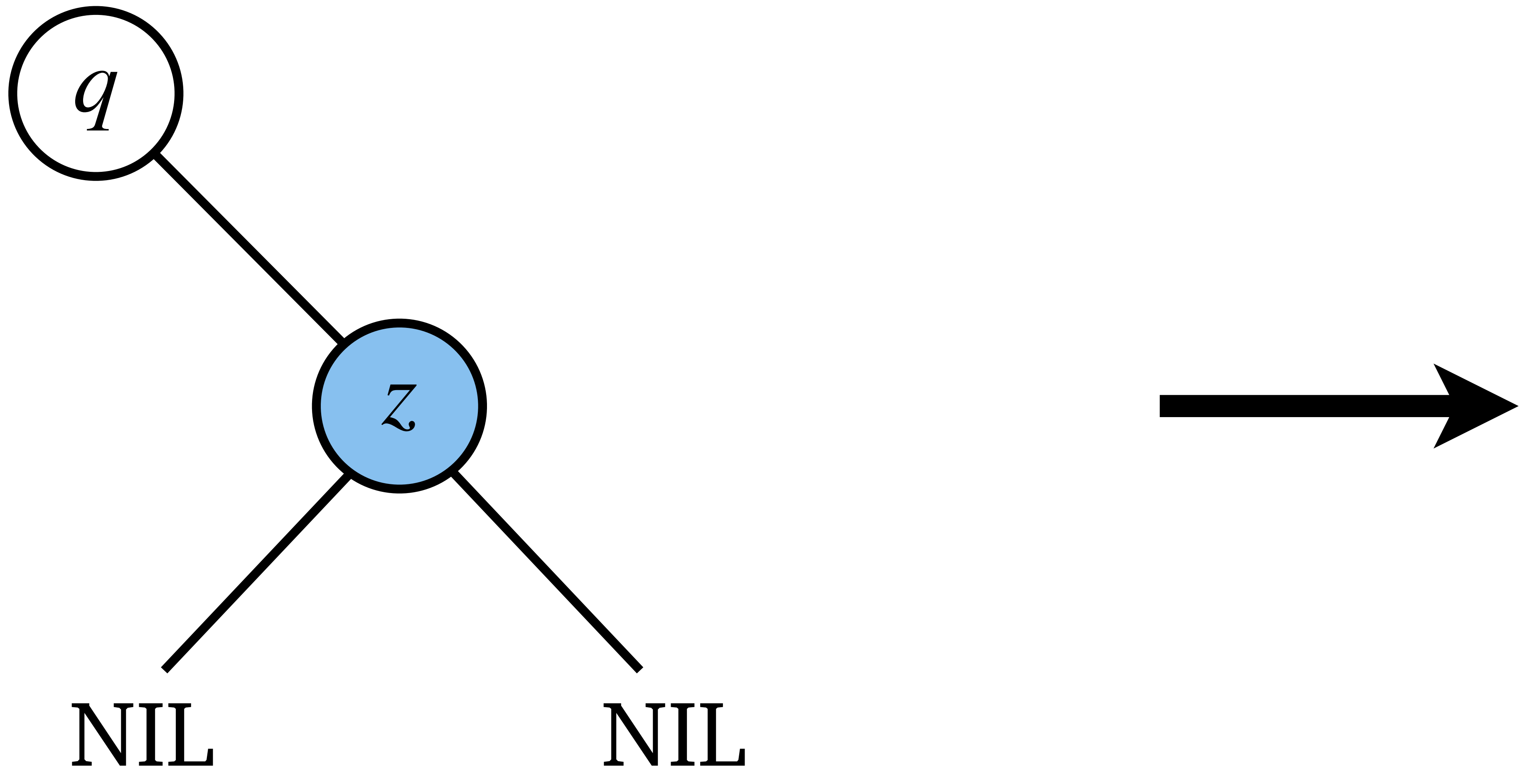
Modifying a BST: Deletion

Case 1: z has no children. (WLOG assume z is a right child.)



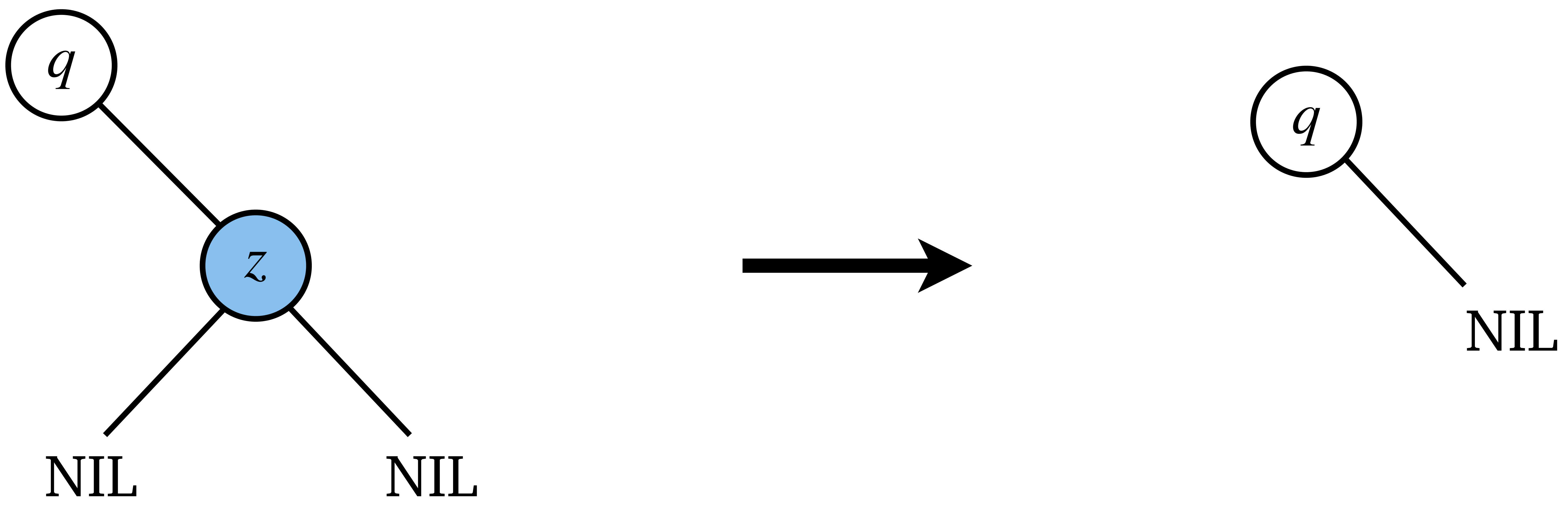
Modifying a BST: Deletion

Case 1: z has no children. (WLOG assume z is a right child.)



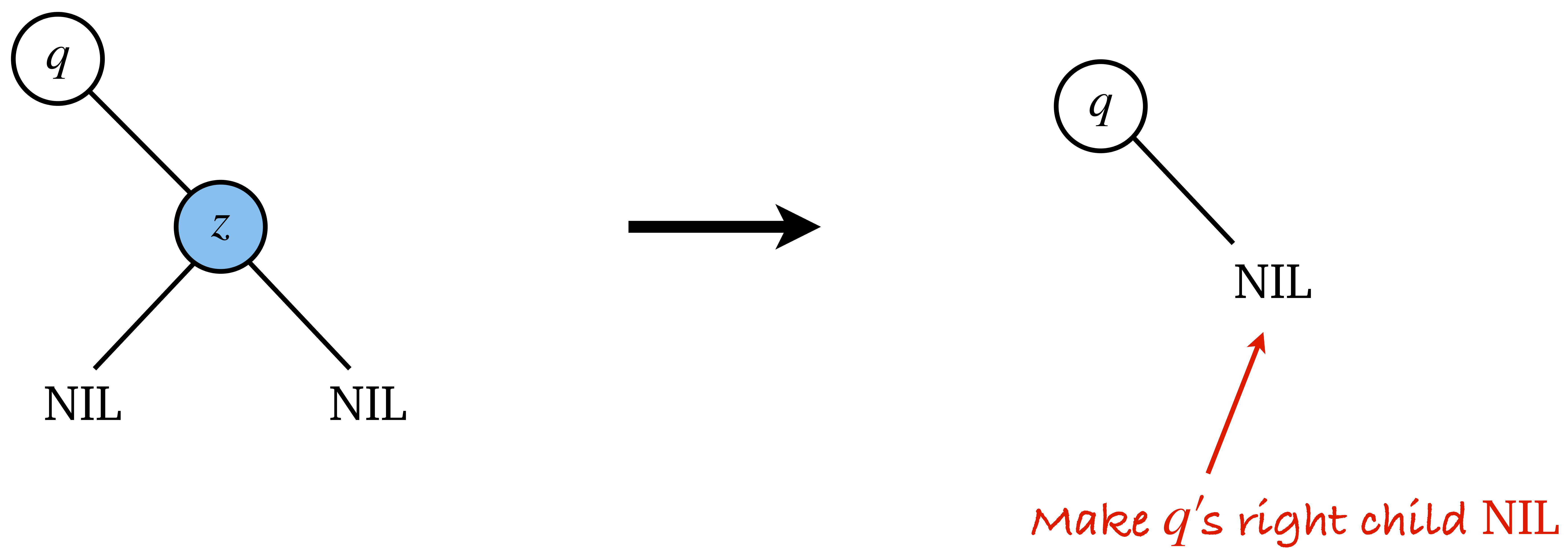
Modifying a BST: Deletion

Case 1: z has no children. (WLOG assume z is a right child.)



Modifying a BST: Deletion

Case 1: z has no children. (WLOG assume z is a right child.)



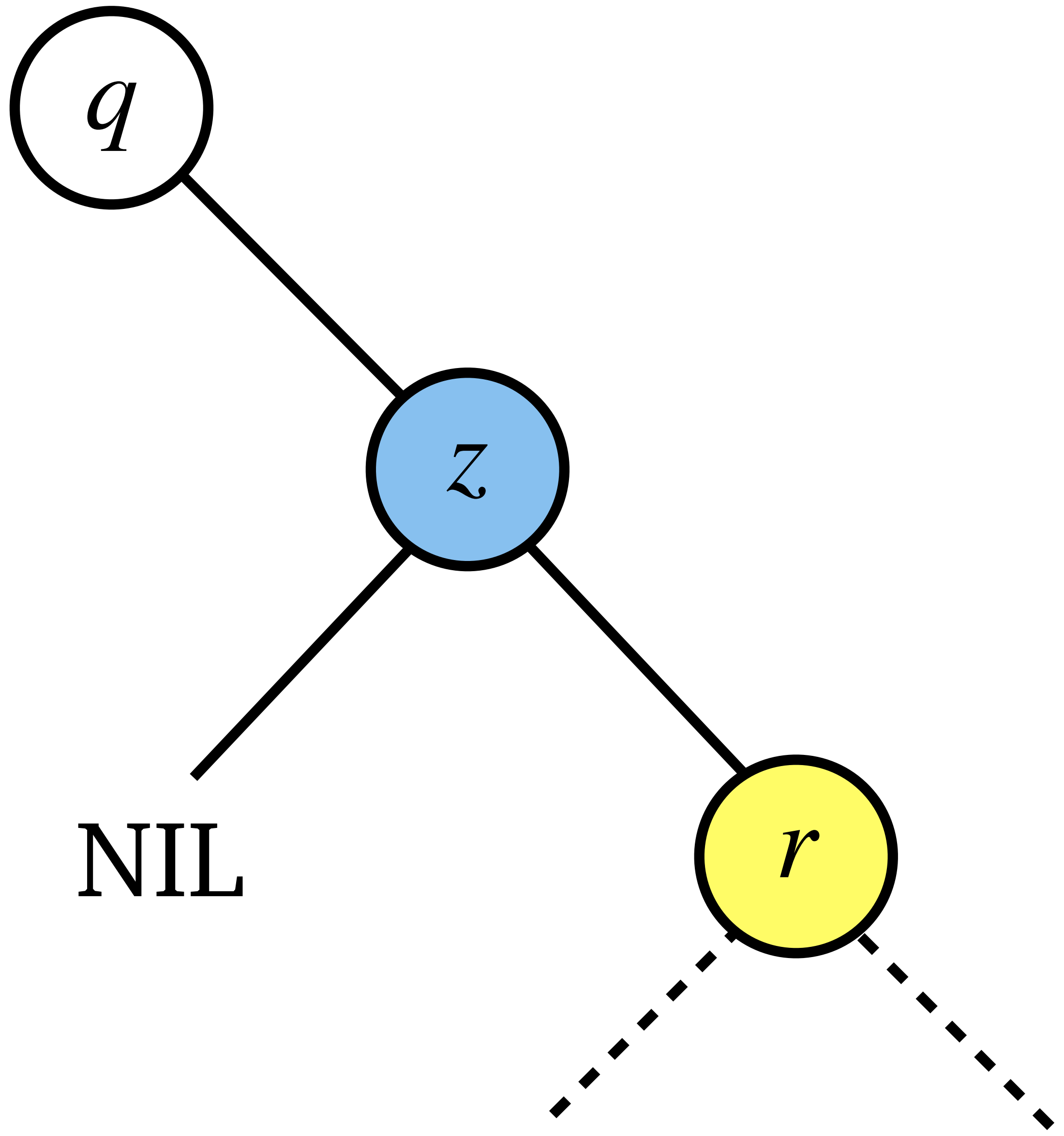
Modifying a BST: Deletion

Modifying a BST: Deletion

Case 2: z has one child. (WLOG assume z is a right child.)

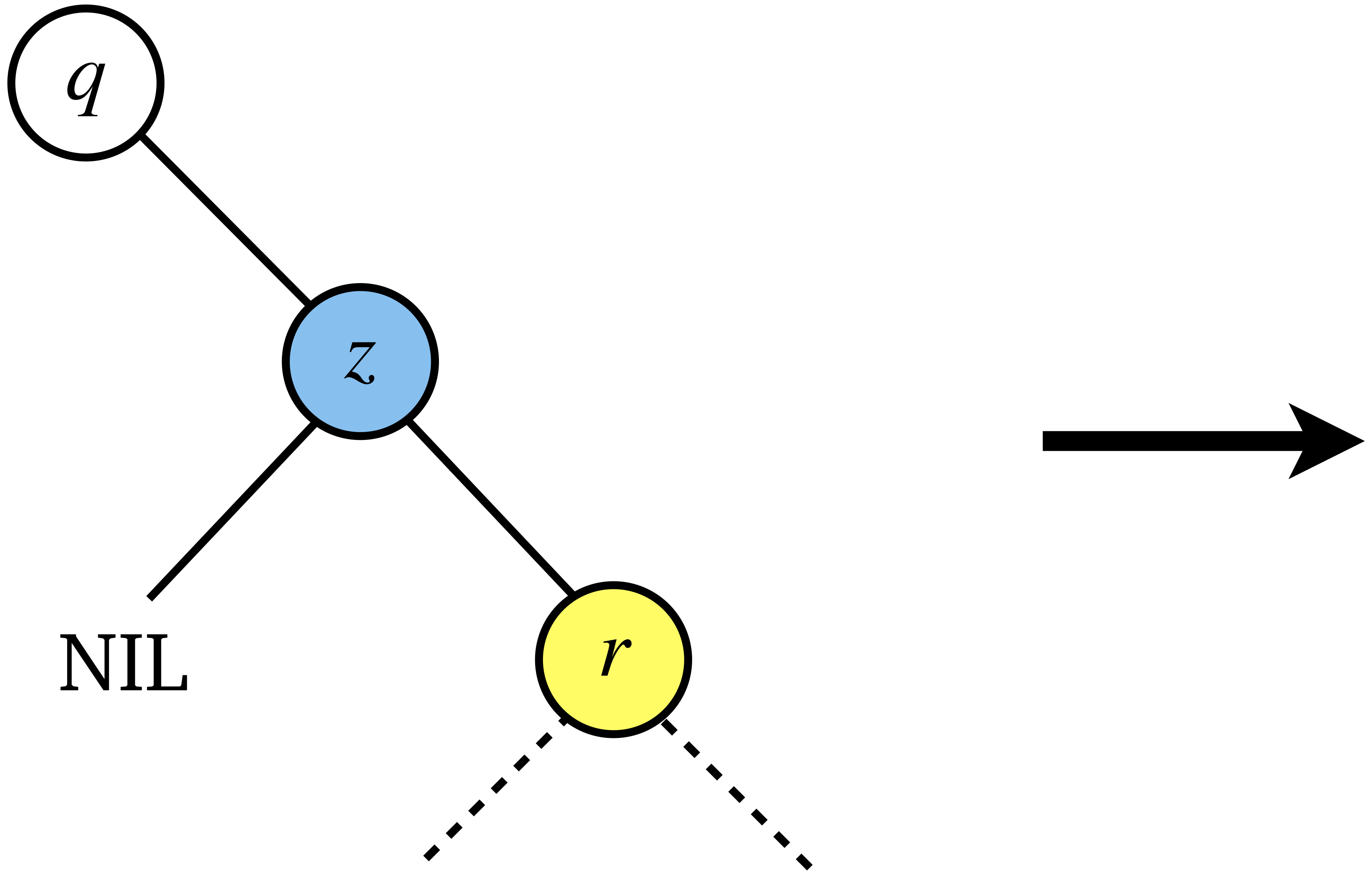
Modifying a BST: Deletion

Case 2: z has one child. (WLOG assume z is a right child.)



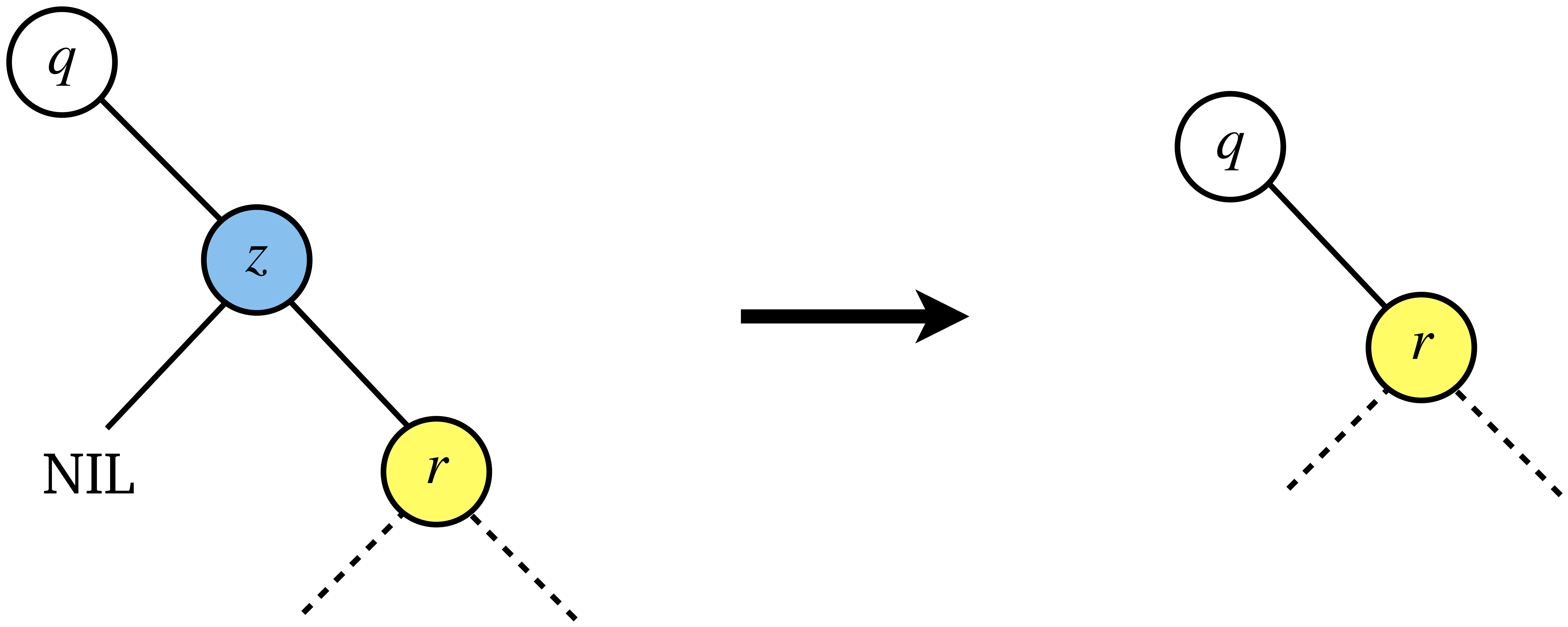
Modifying a BST: Deletion

Case 2: z has one child. (WLOG assume z is a right child.)



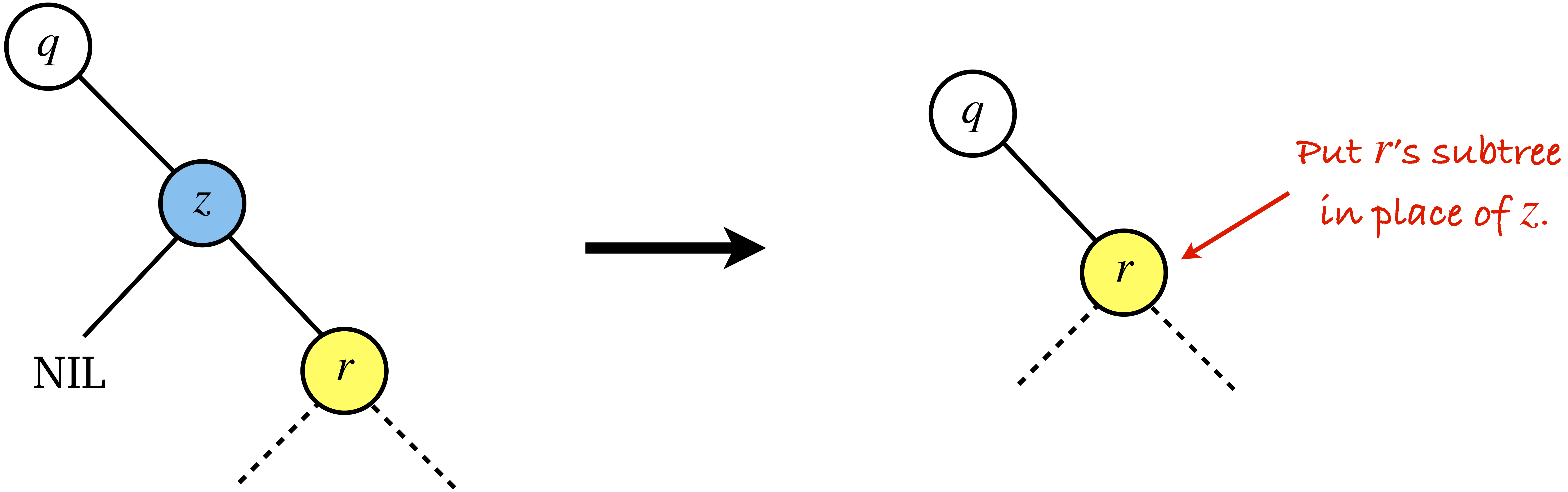
Modifying a BST: Deletion

Case 2: z has one child. (WLOG assume z is a right child.)



Modifying a BST: Deletion

Case 2: z has one child. (WLOG assume z is a right child.)

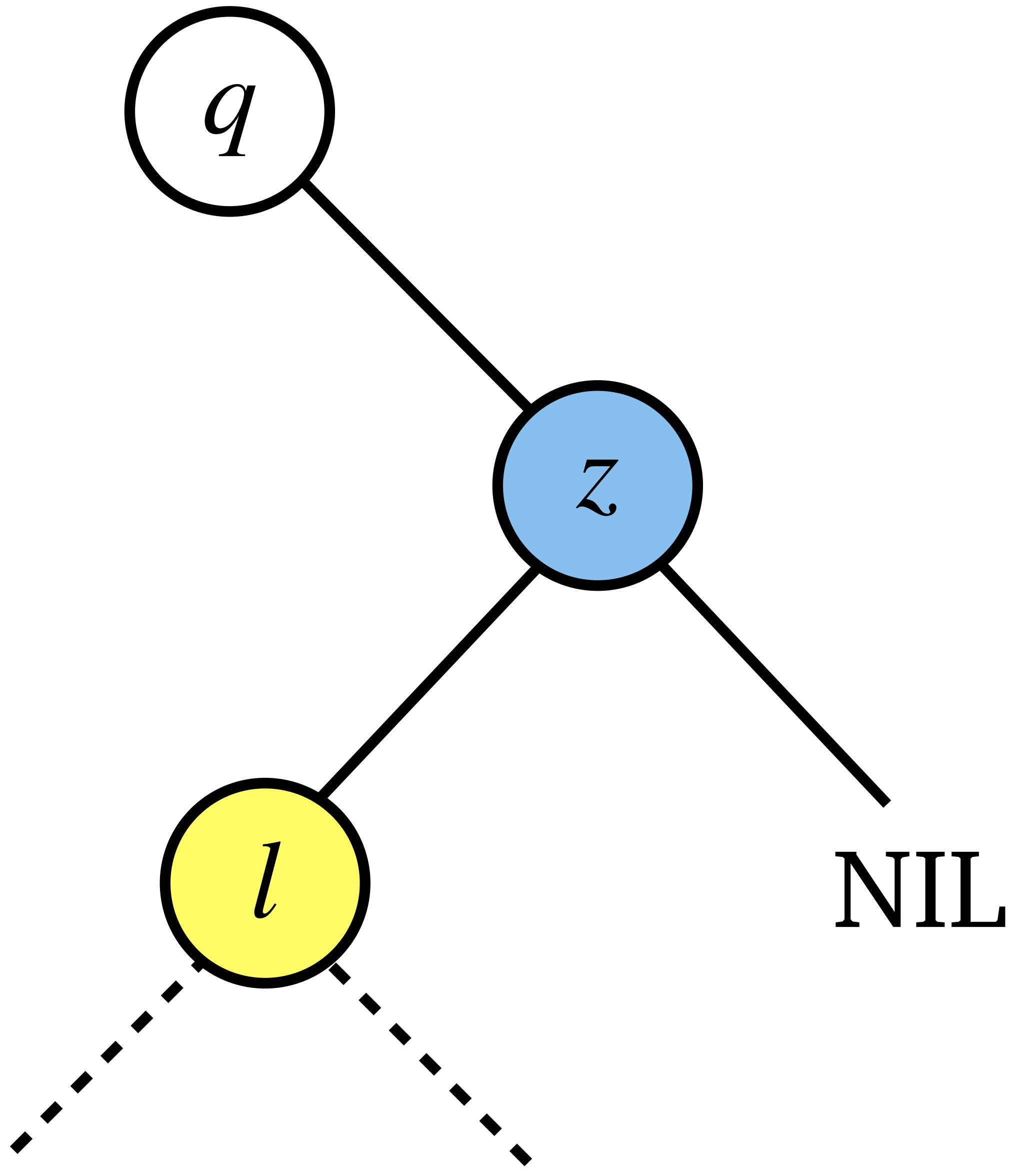


Modifying a BST: Deletion

Case 2: z has one child. (WLOG assume z is a right child.)

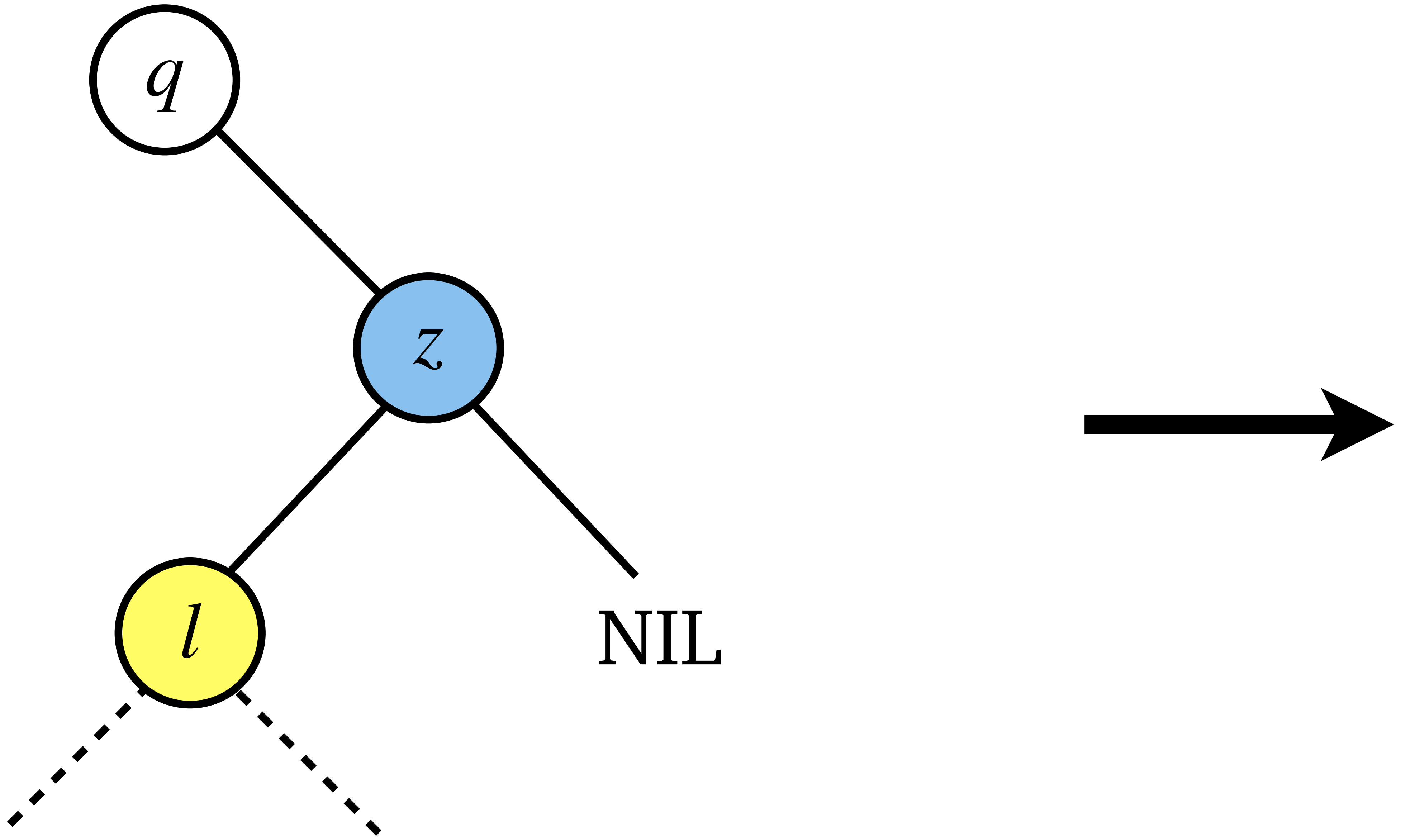
Modifying a BST: Deletion

Case 2: z has one child. (WLOG assume z is a right child.)



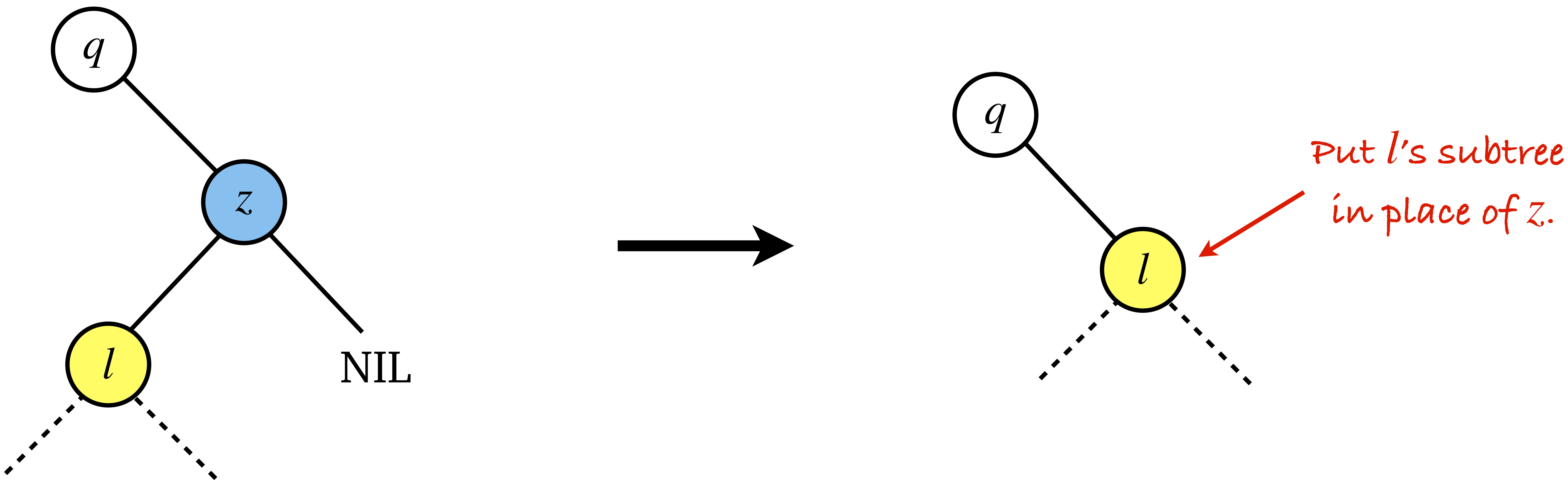
Modifying a BST: Deletion

Case 2: z has one child. (WLOG assume z is a right child.)



Modifying a BST: Deletion

Case 2: z has one child. (WLOG assume z is a right child.)



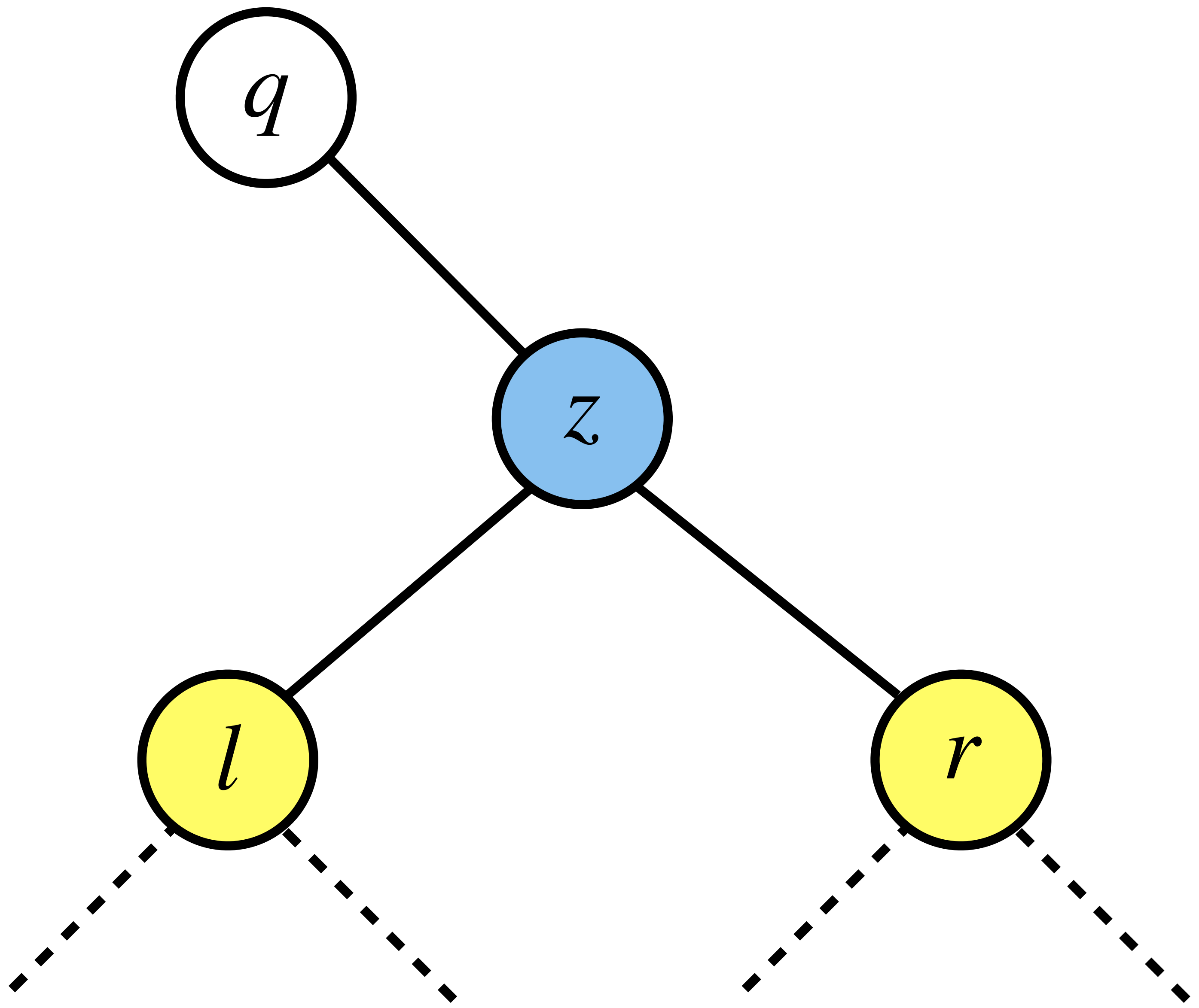
Modifying a BST: Deletion

Modifying a BST: Deletion

Case 3: z has two children. (WLOG assume z is a right child.)

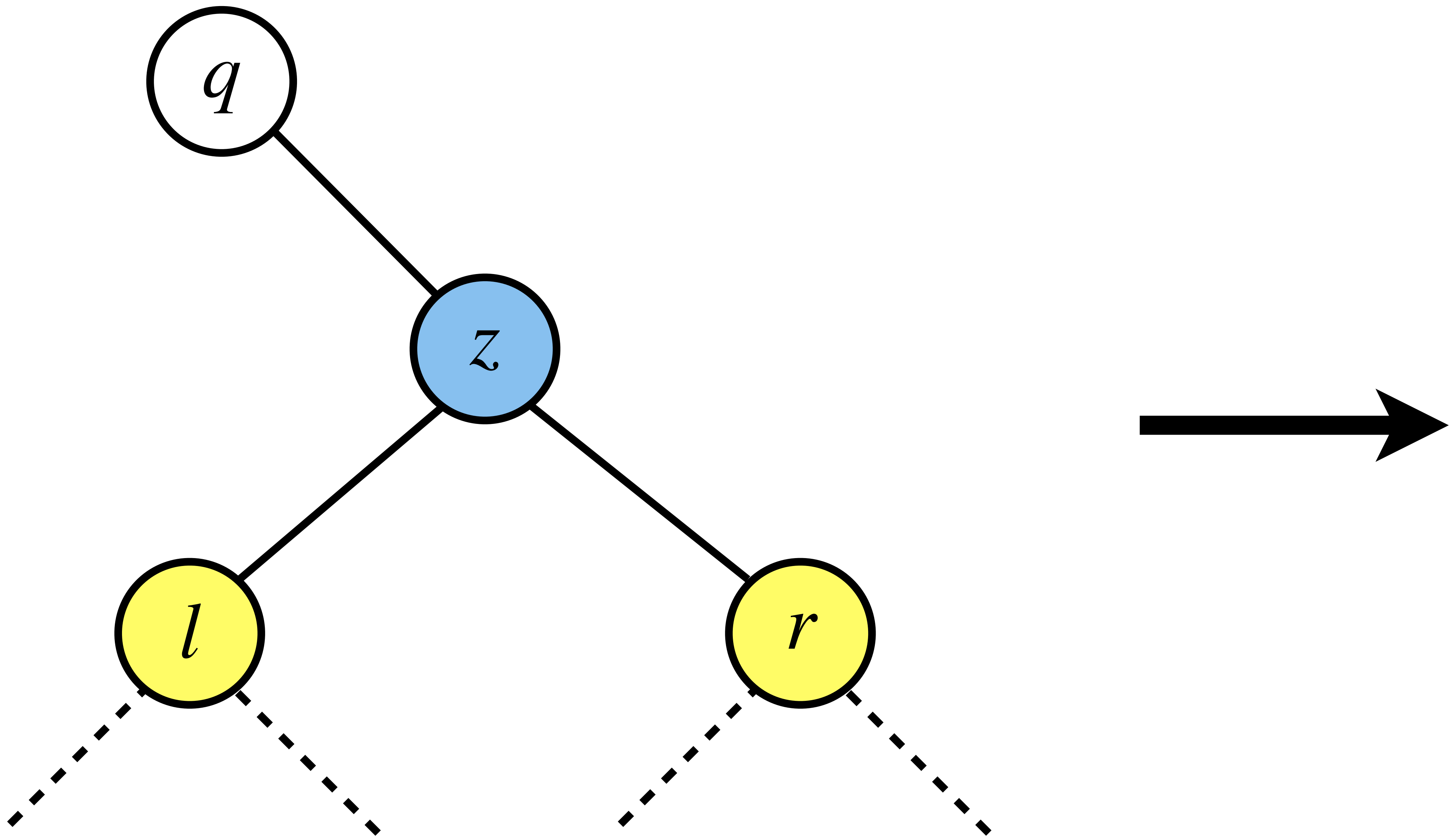
Modifying a BST: Deletion

Case 3: z has two children. (WLOG assume z is a right child.)



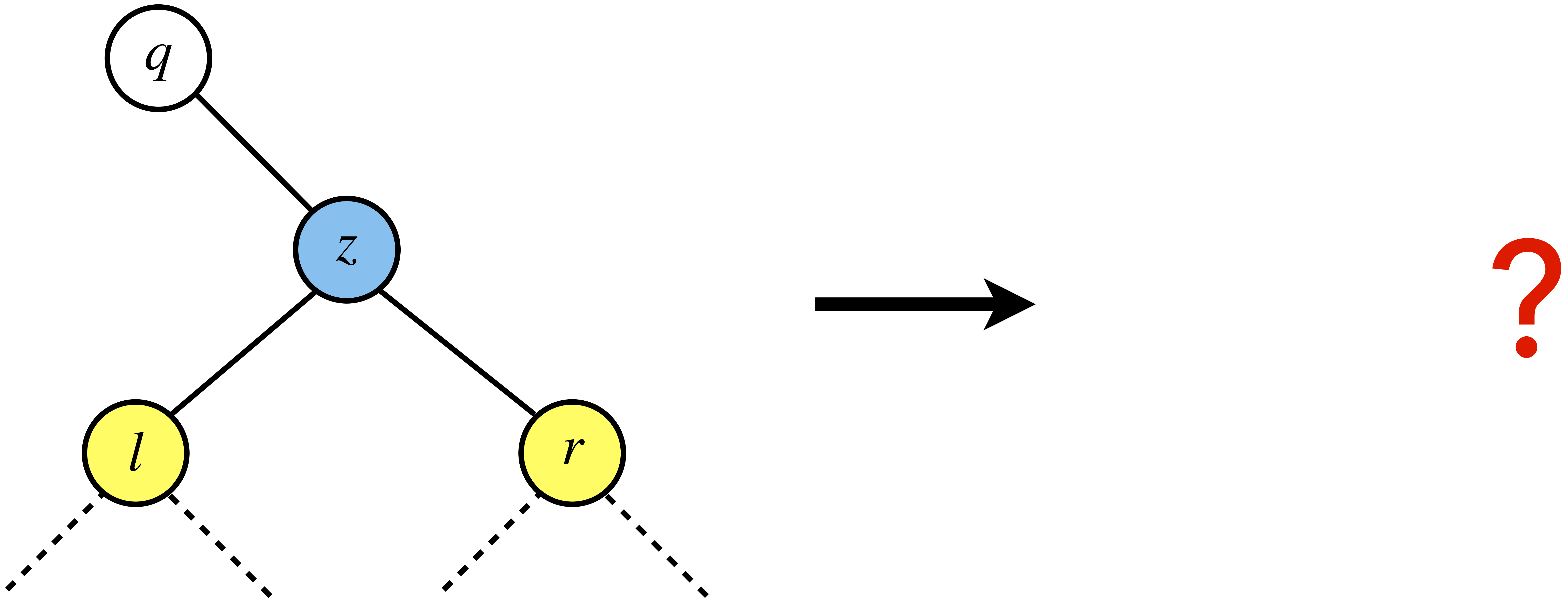
Modifying a BST: Deletion

Case 3: z has two children. (WLOG assume z is a right child.)



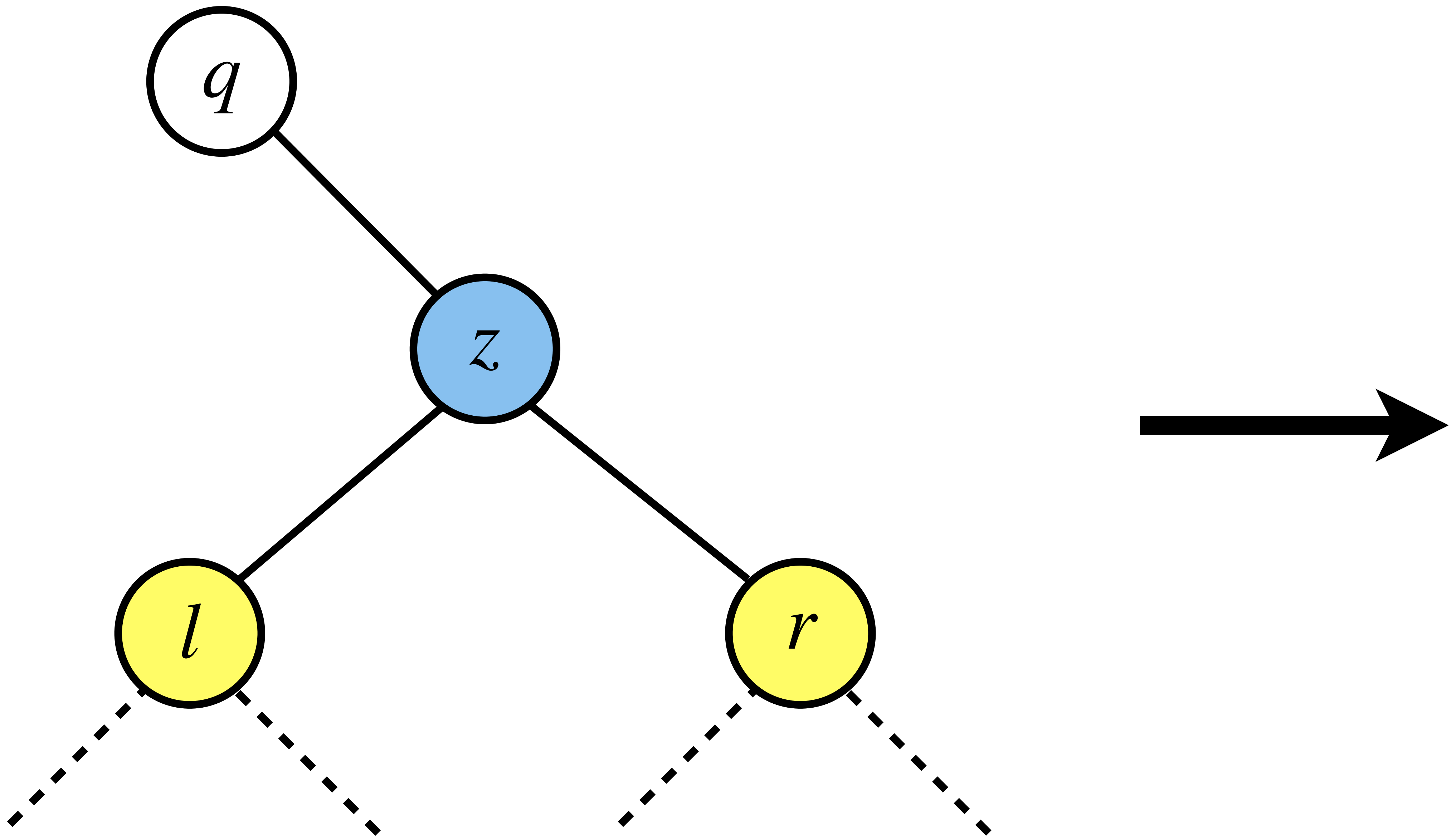
Modifying a BST: Deletion

Case 3: z has two children. (WLOG assume z is a right child.)



Modifying a BST: Deletion

Case 3: z has two children. (WLOG assume z is a right child.)



Two sub-cases:

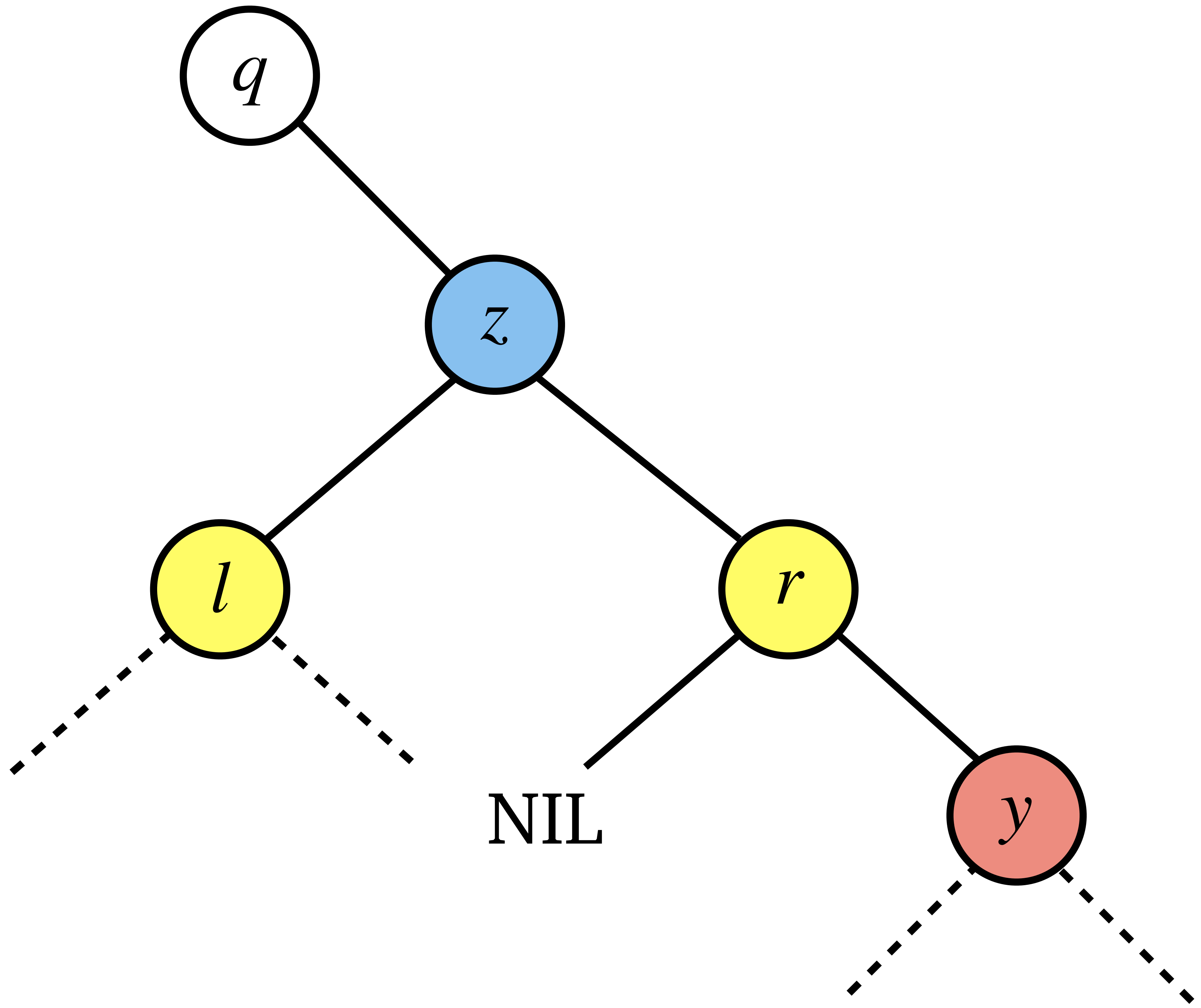
- r has no left child.
- r has a left child.

Modifying a BST: Deletion

Case 3a: z has two children where its right child has no left child.

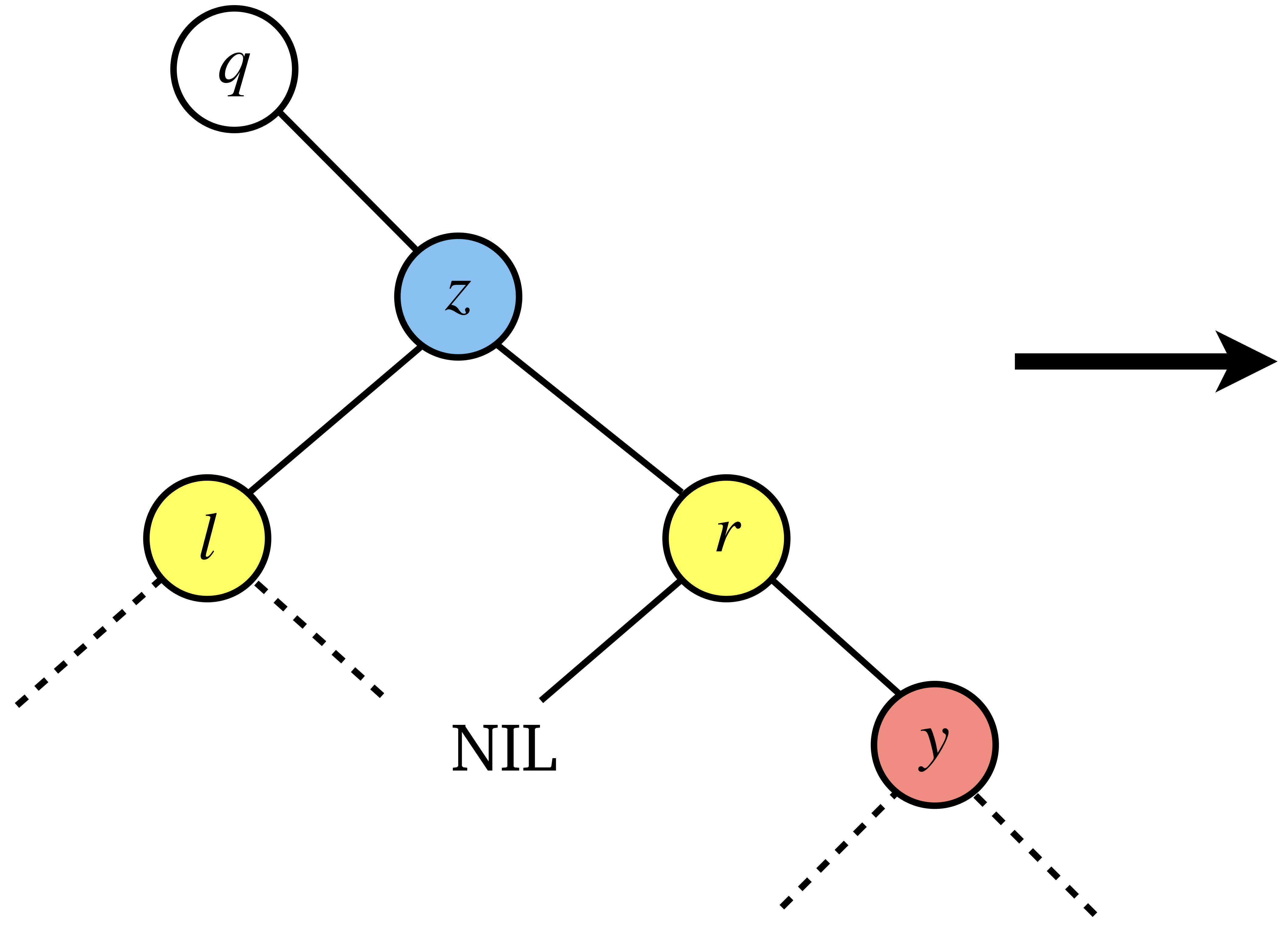
Modifying a BST: Deletion

Case 3**a**: z has two children where its right child has no left child.



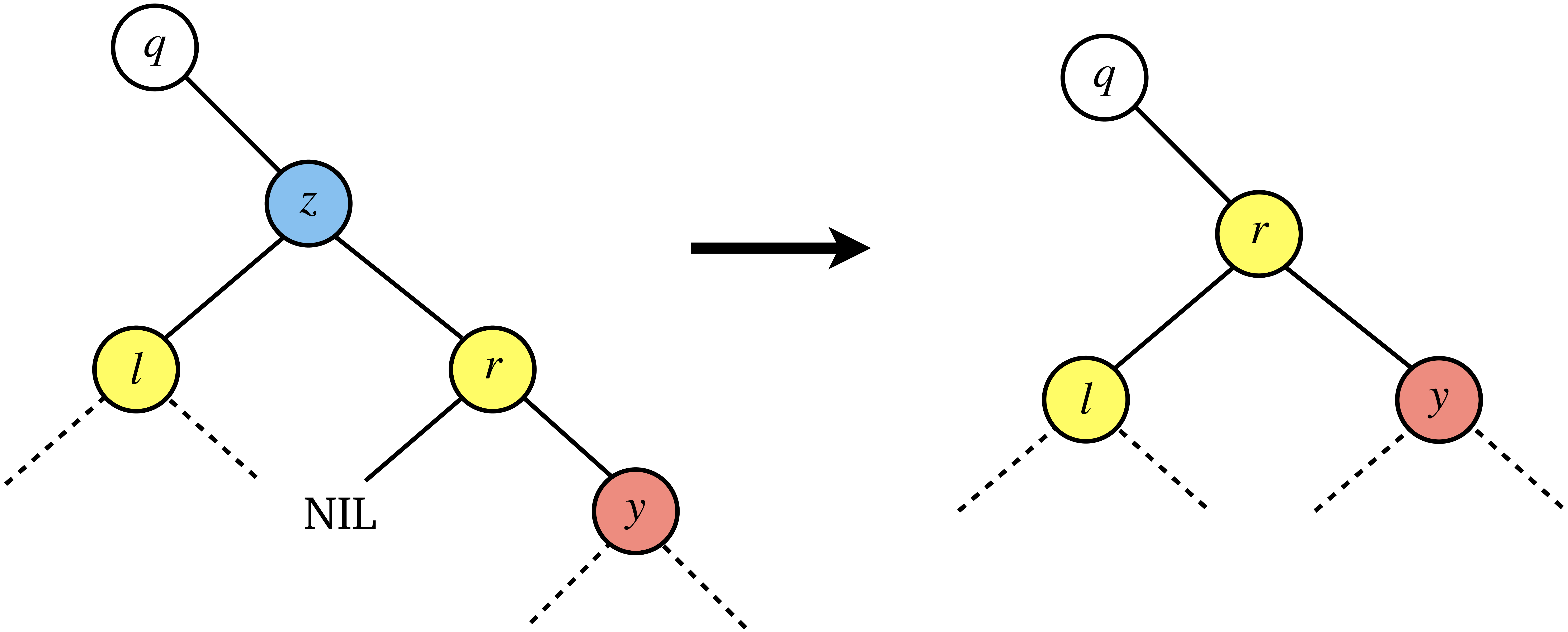
Modifying a BST: Deletion

Case 3**a**: z has two children where its right child has no left child.



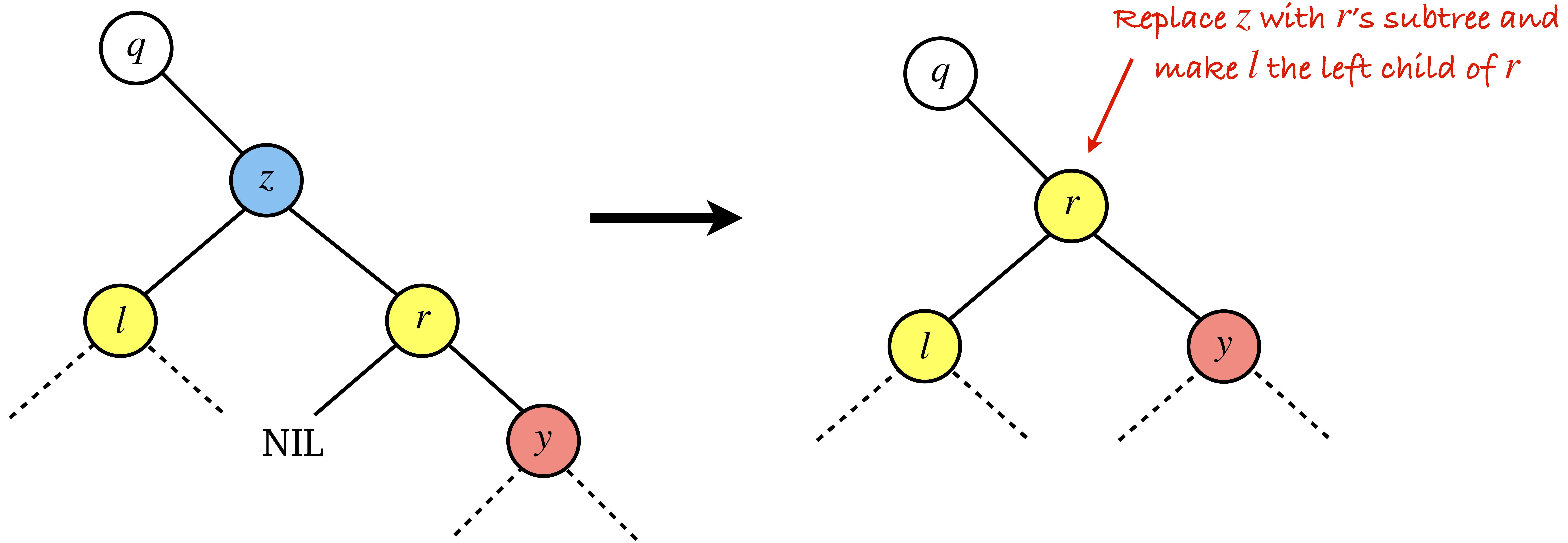
Modifying a BST: Deletion

Case 3a: z has two children where its right child has no left child.



Modifying a BST: Deletion

Case 3**a**: z has two children where its right child has no left child.



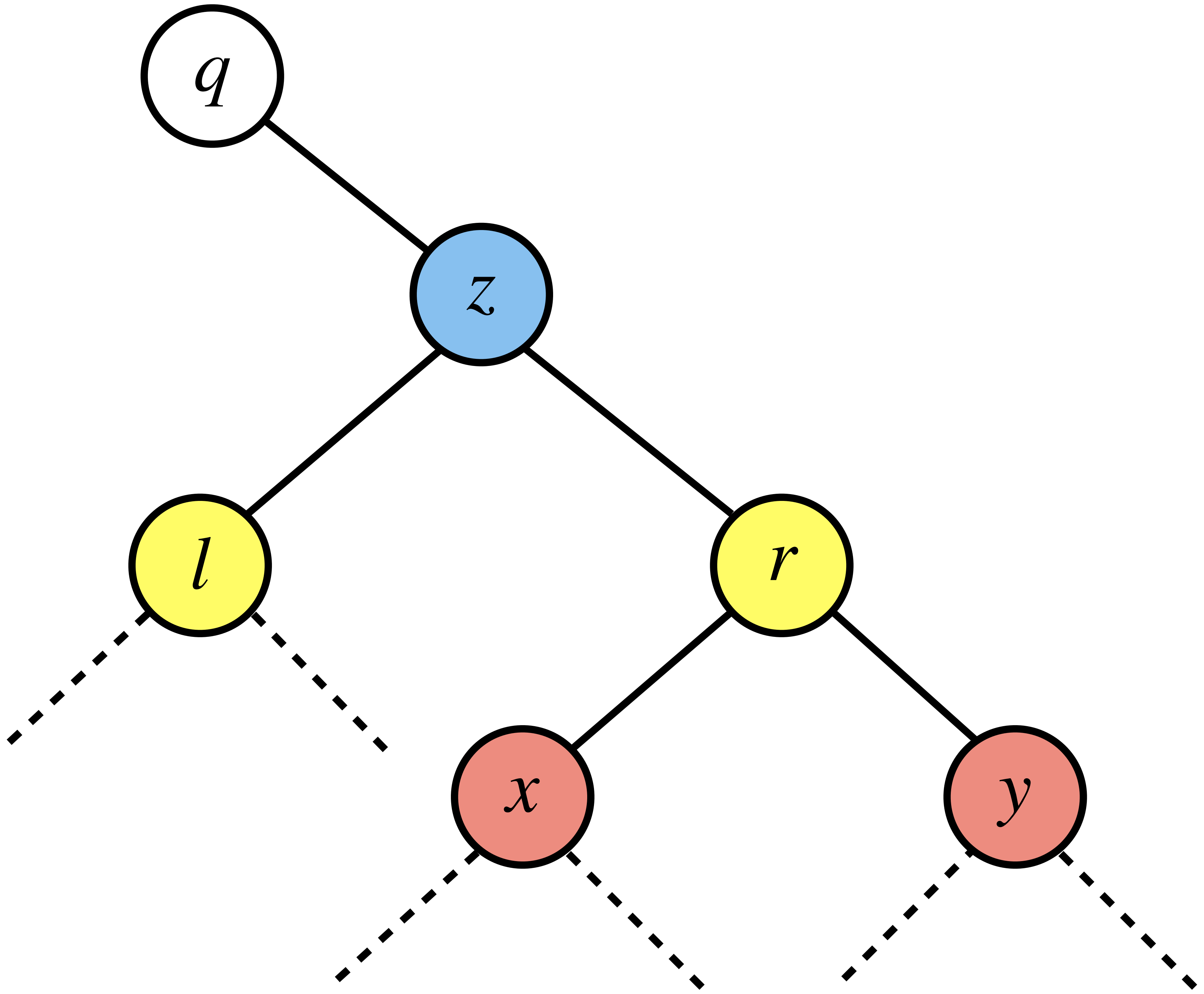
Modifying a BST: Deletion

Modifying a BST: Deletion

Case 3b: z has two children where its right child has a left child.

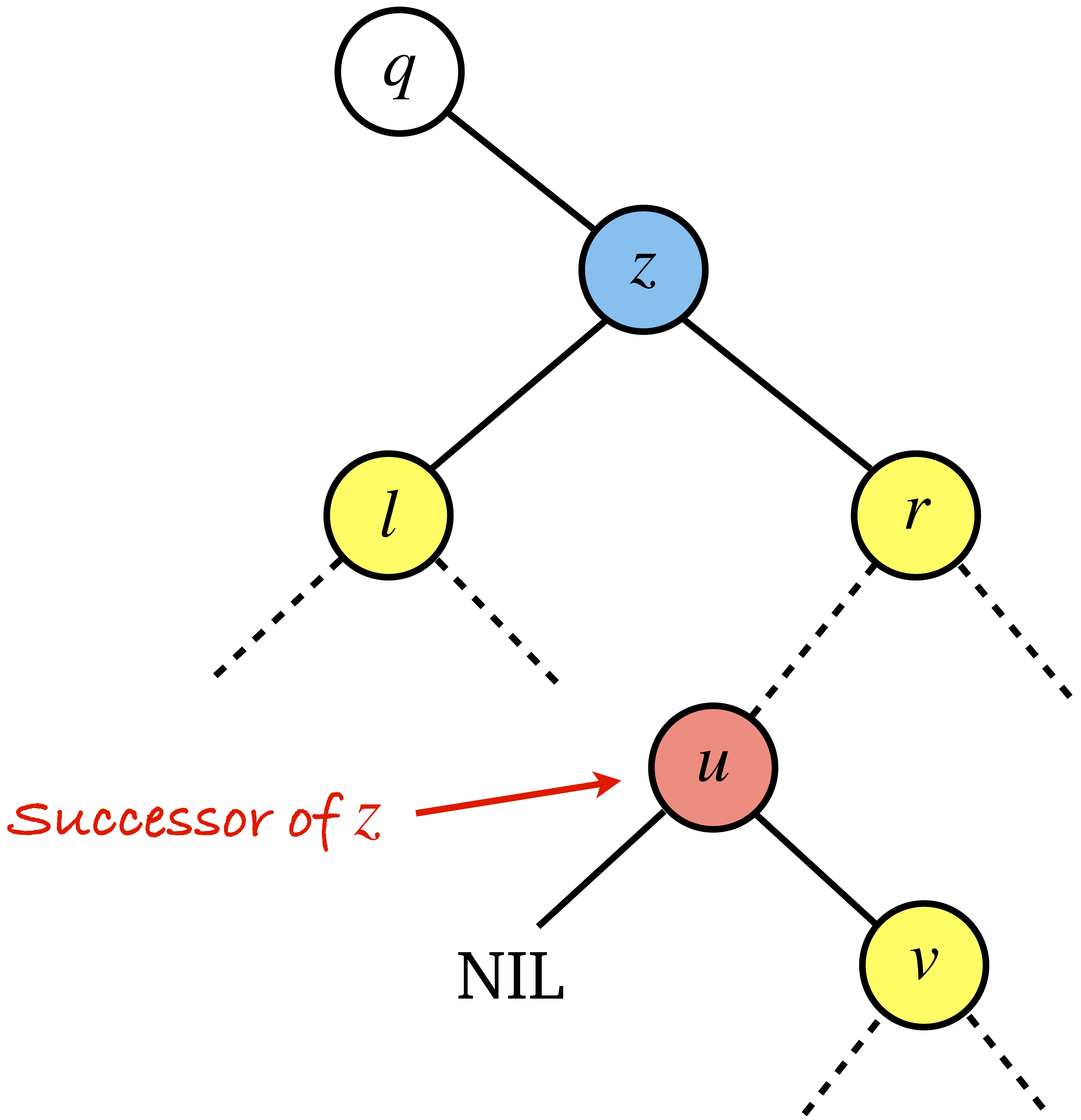
Modifying a BST: Deletion

Case 3b: z has two children where its right child has a left child.



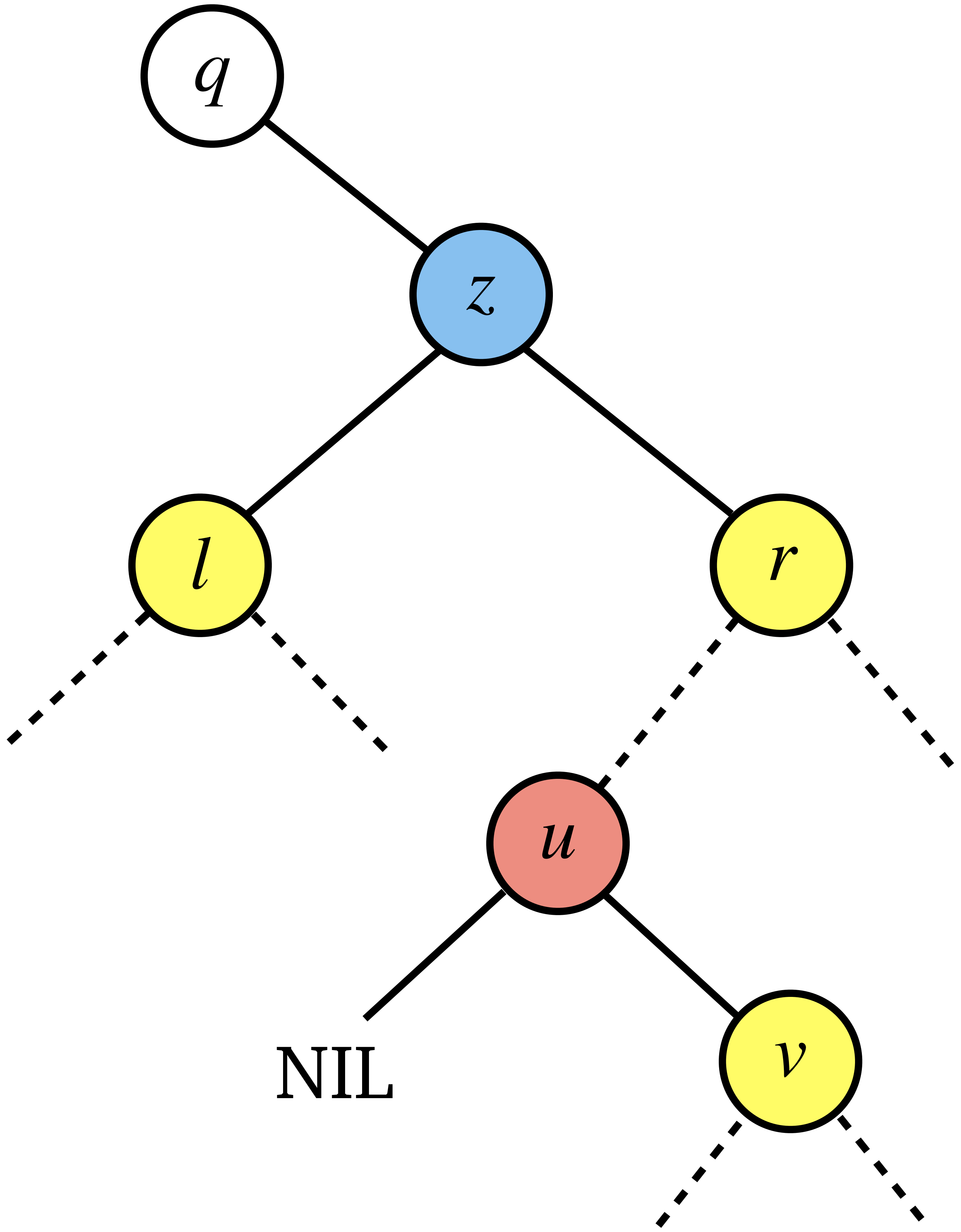
Modifying a BST: Deletion

Case 3**b**: z has two children where its right child has a left child.



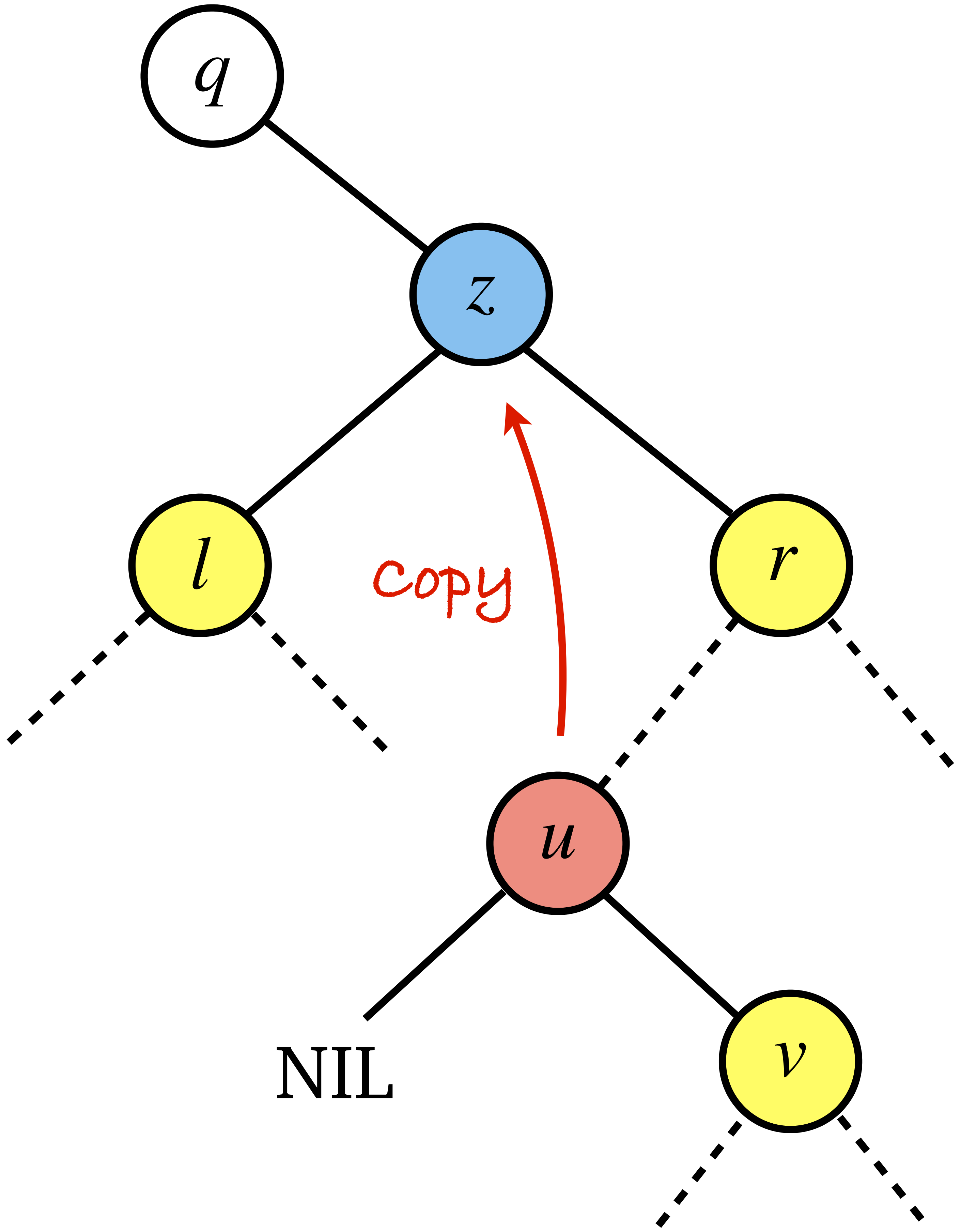
Modifying a BST: Deletion

Case 3**b**: z has two children where its right child has a left child.



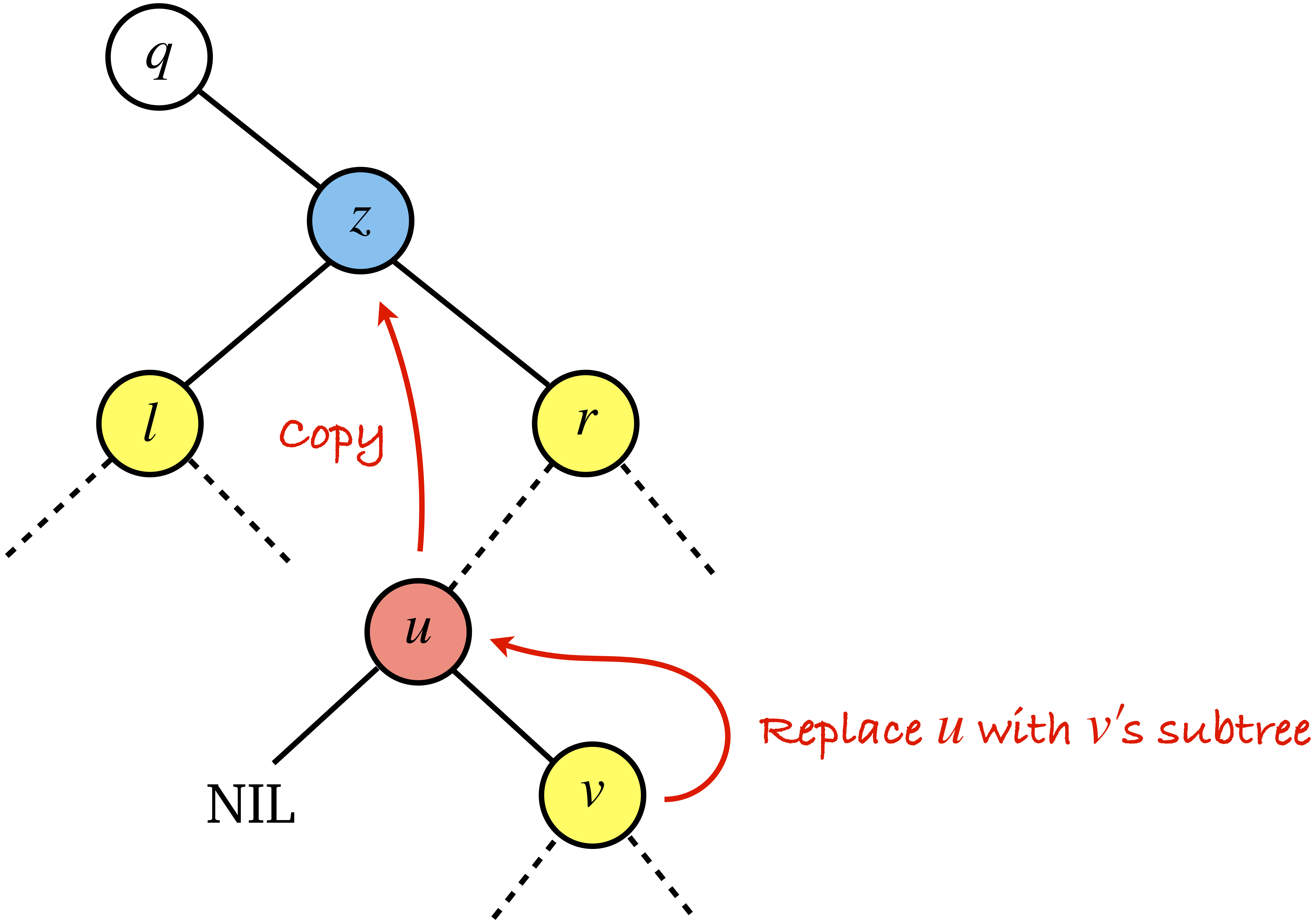
Modifying a BST: Deletion

Case 3**b:** z has two children where its right child has a left child.



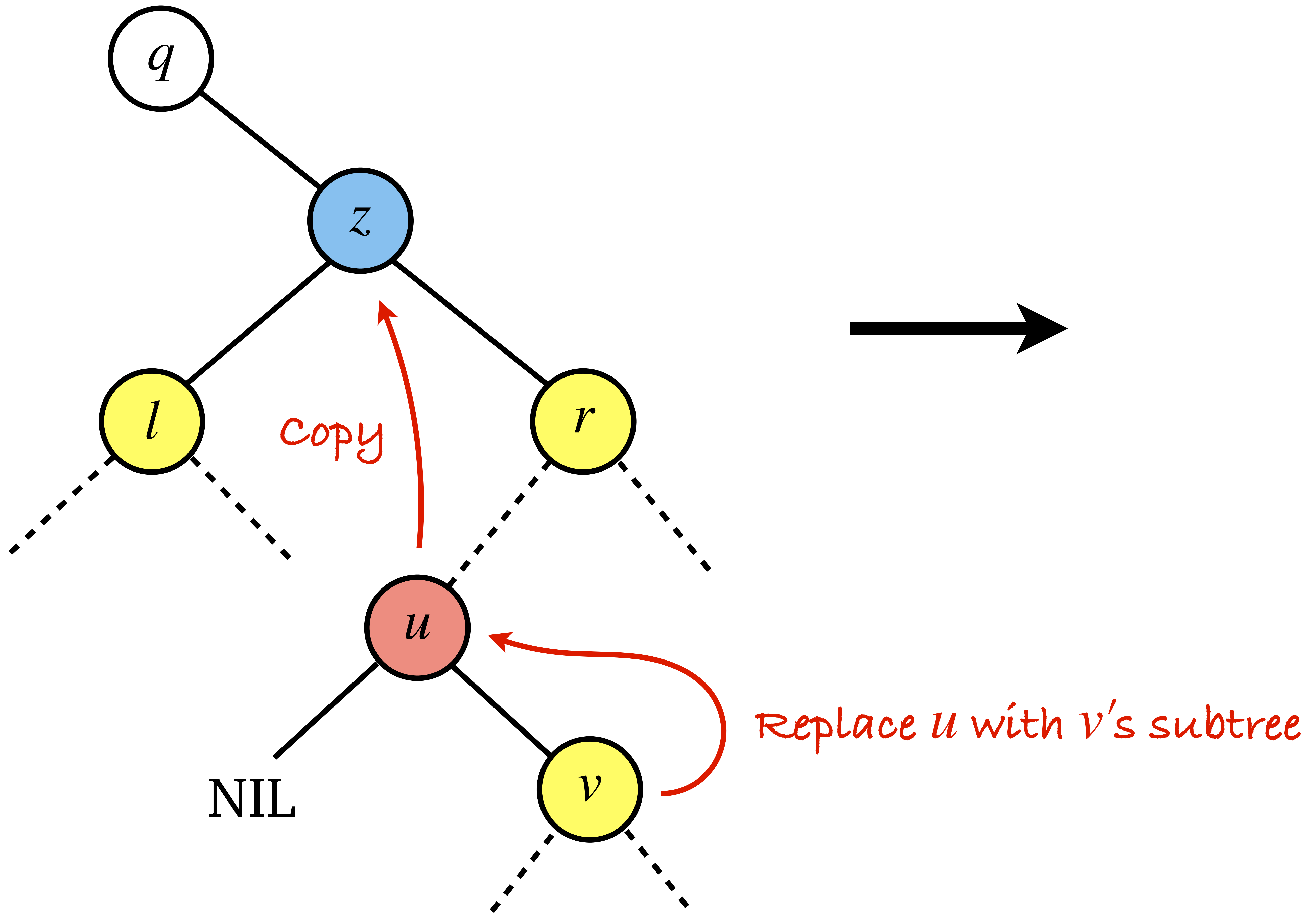
Modifying a BST: Deletion

Case 3**b:** z has two children where its right child has a left child.



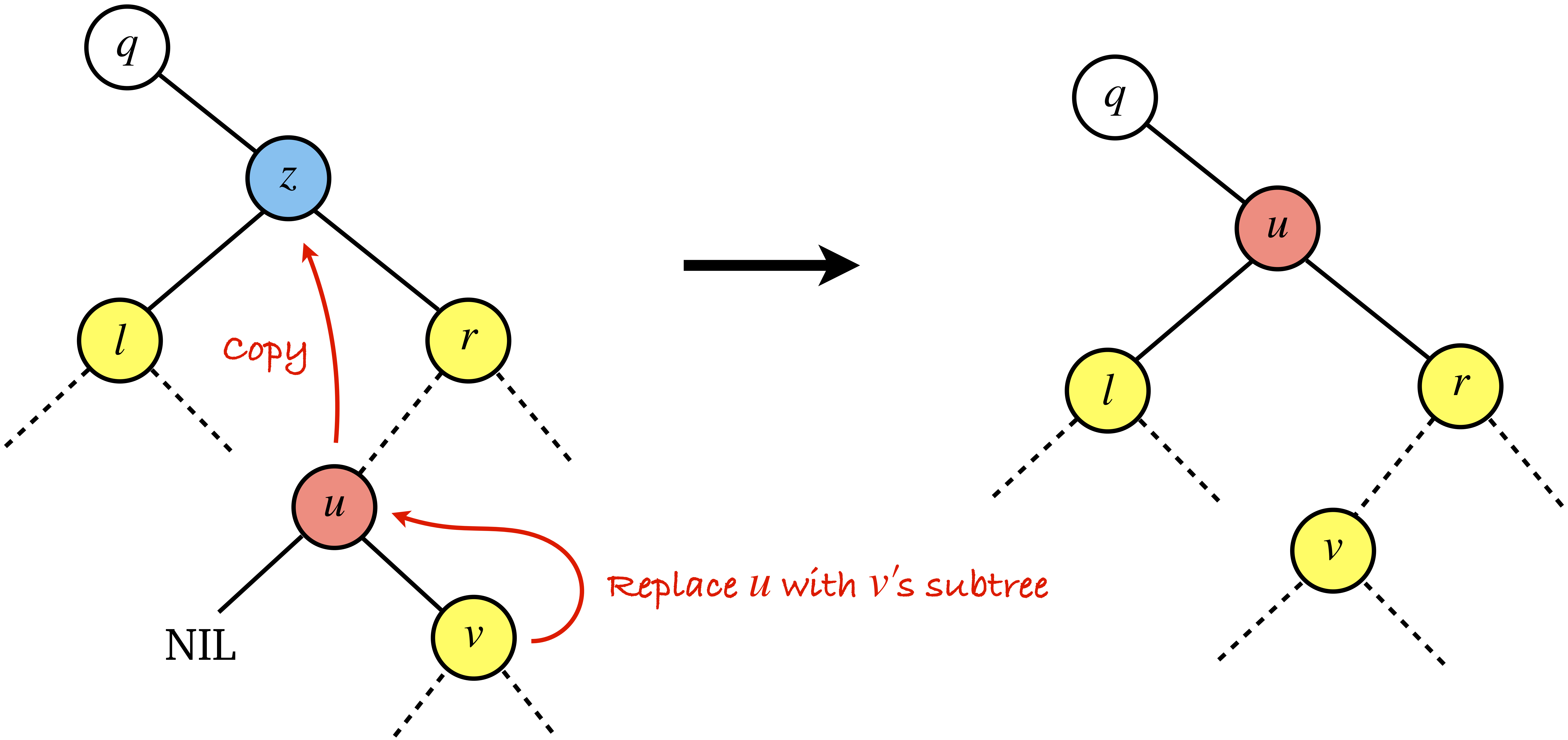
Modifying a BST: Deletion

Case 3**b**: z has two children where its right child has a left child.



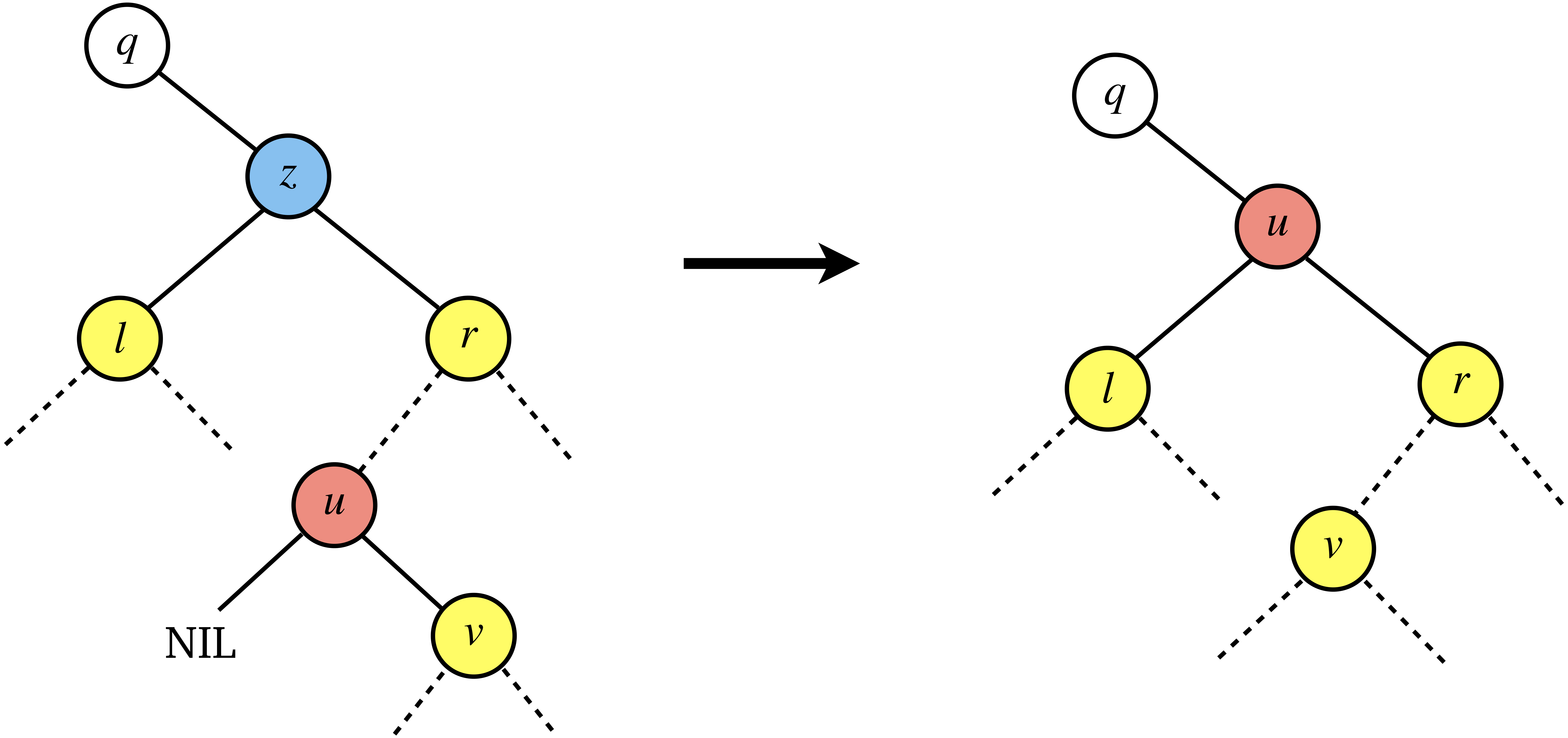
Modifying a BST: Deletion

Case 3**b**: z has two children where its right child has a left child.



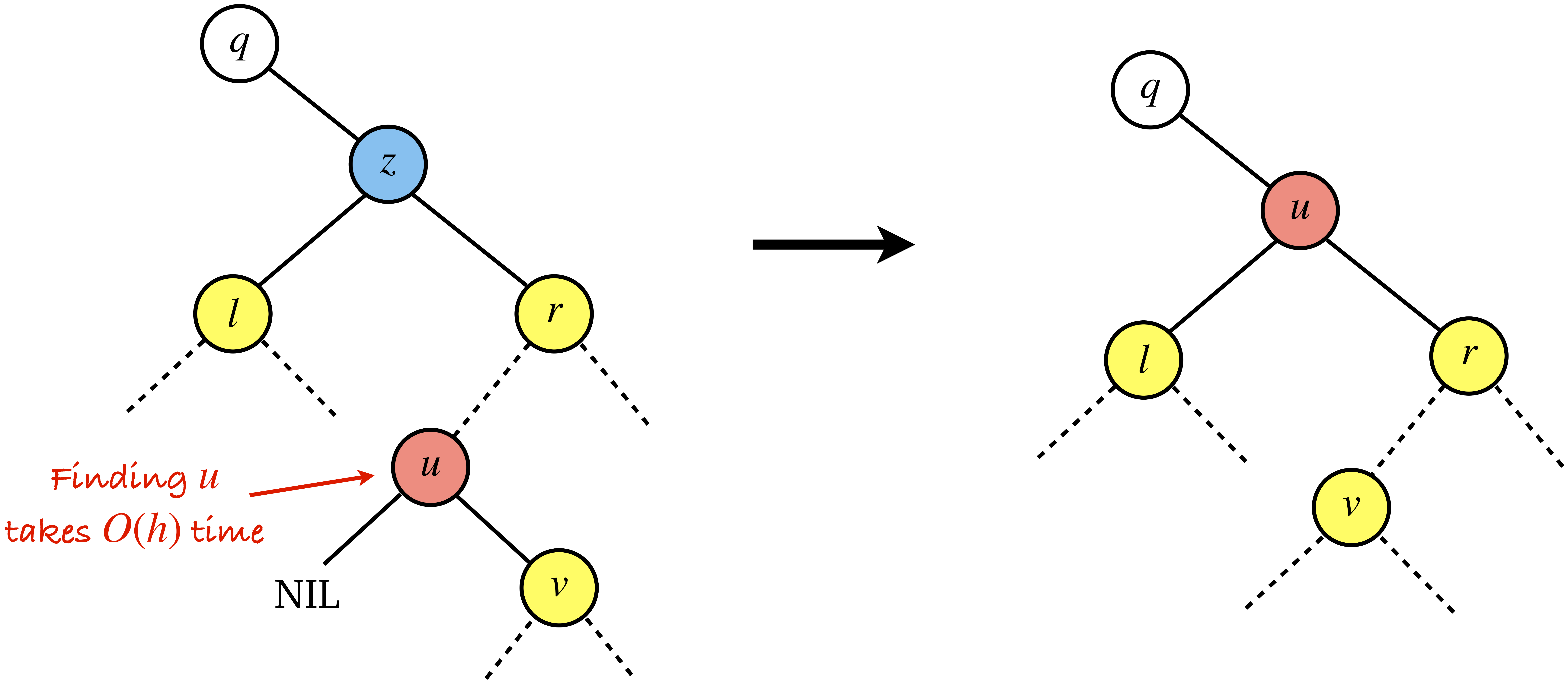
Modifying a BST: Deletion

Case 3**b**: z has two children where its right child has a left child.



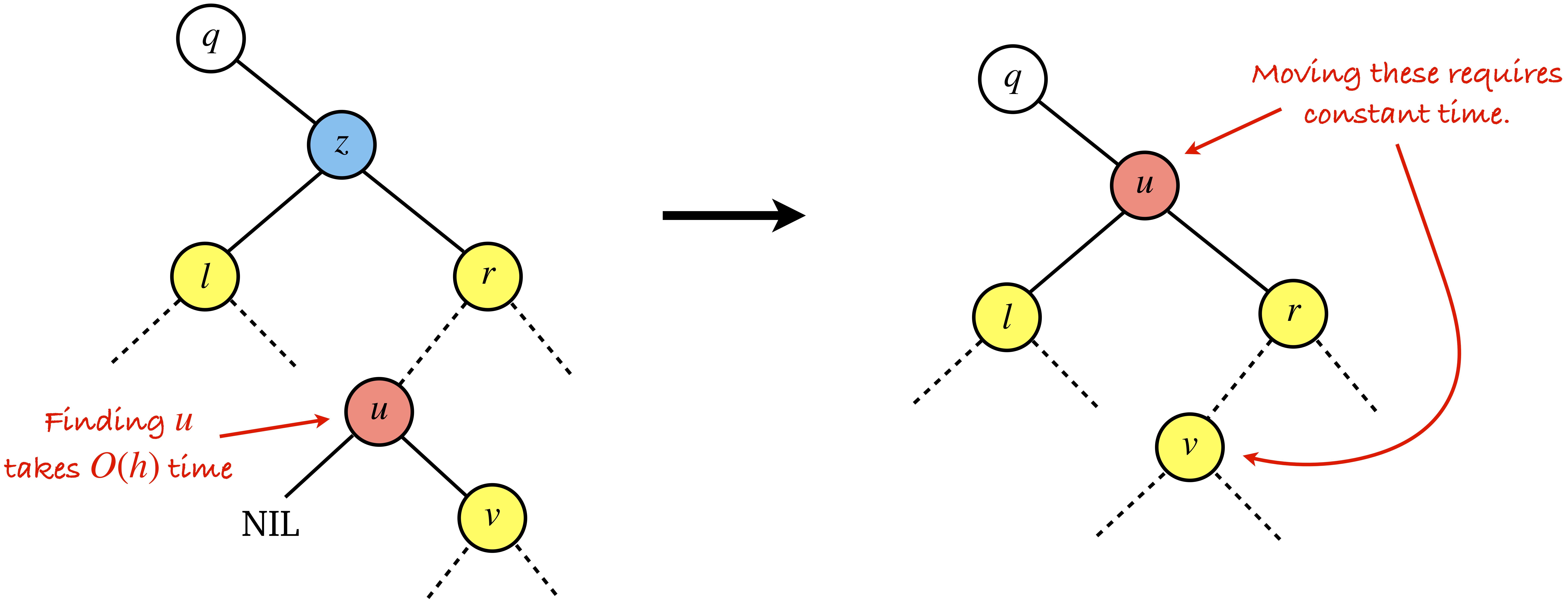
Modifying a BST: Deletion

Case 3**b**: z has two children where its right child has a left child.



Modifying a BST: Deletion

Case 3**b**: z has two children where its right child has a left child.



Are BSTs Good Enough?

Are BSTs Good Enough?

BSTs can perform Insert, Delete, Search, etc., in $O(h)$ time.

Are BSTs Good Enough?

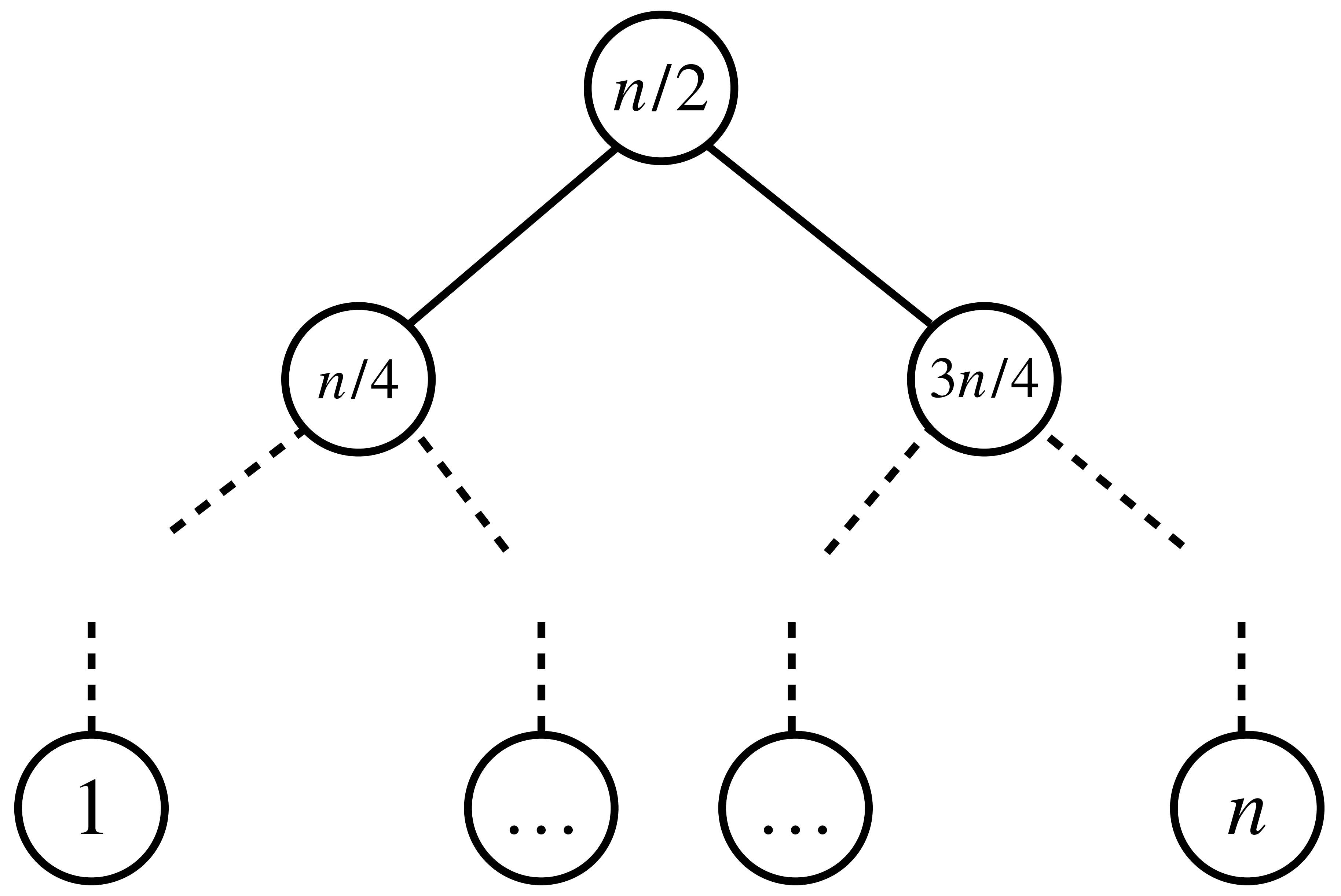
BSTs can perform Insert, Delete, Search, etc., in $O(h)$ time.

But,

Are BSTs Good Enough?

BSTs can perform Insert, Delete, Search, etc., in $O(h)$ time.

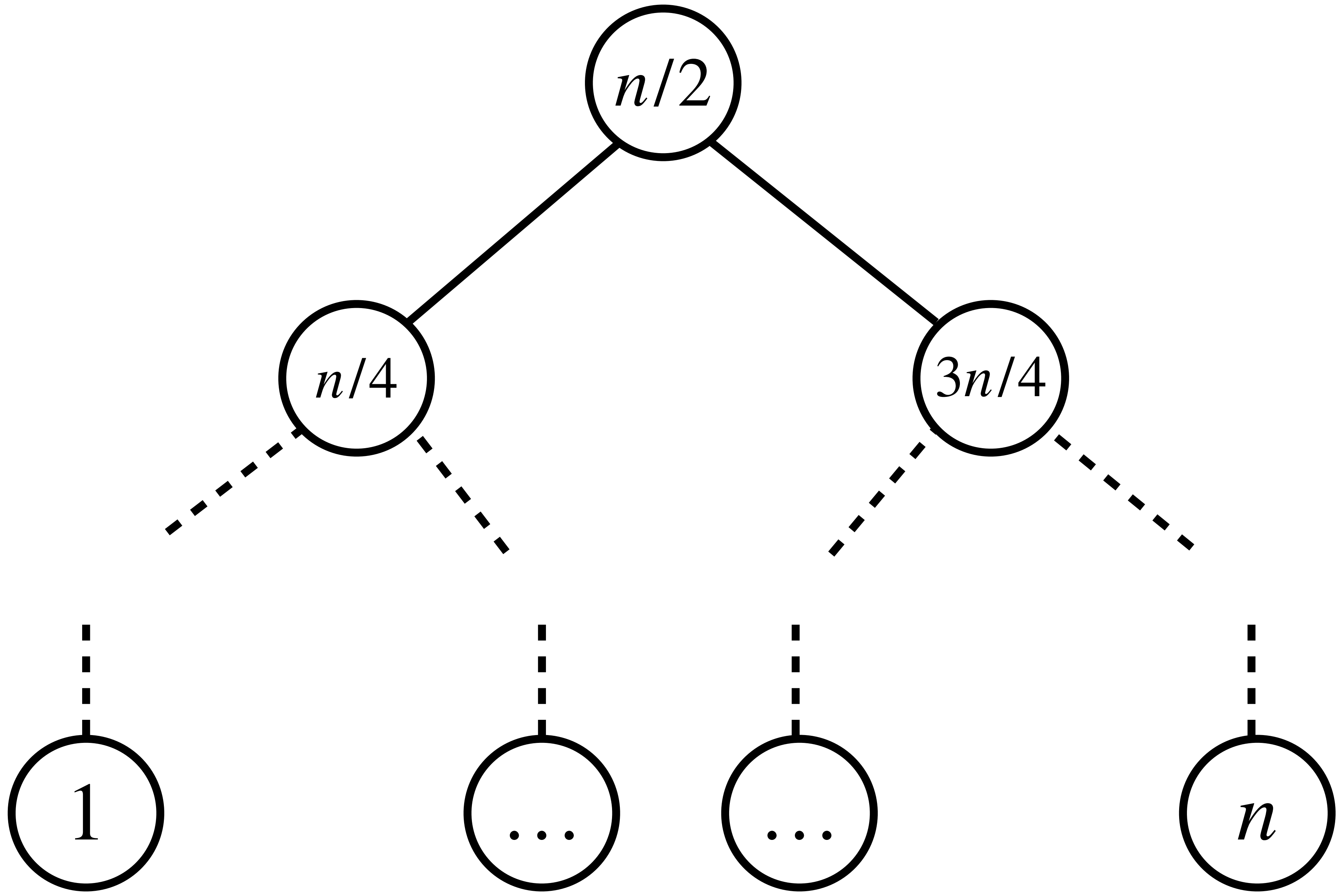
But,



Are BSTs Good Enough?

BSTs can perform Insert, Delete, Search, etc., in $O(h)$ time.

But,

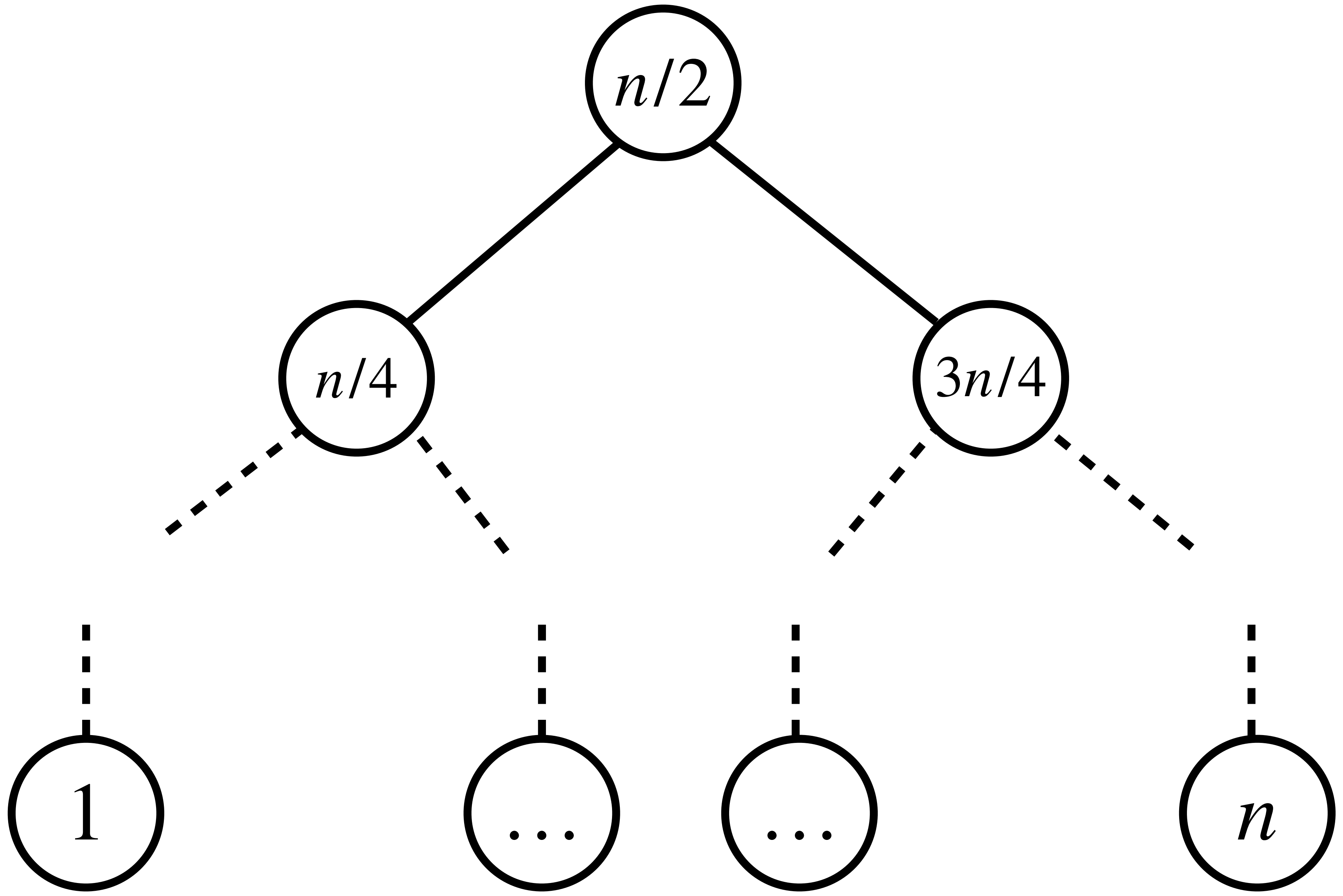


Best case: $h = O(\log n)$

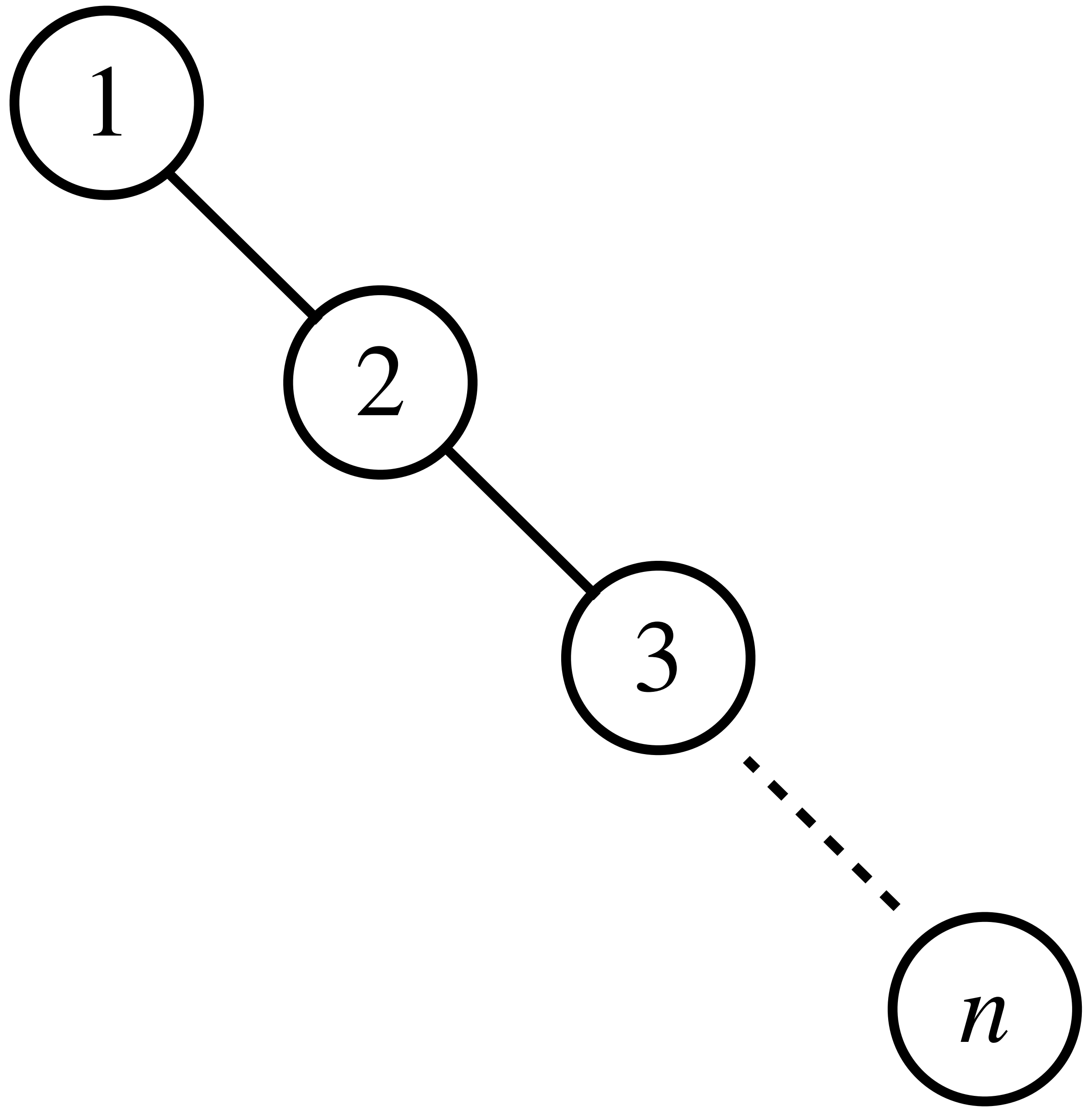
Are BSTs Good Enough?

BSTs can perform Insert, Delete, Search, etc., in $O(h)$ time.

But,



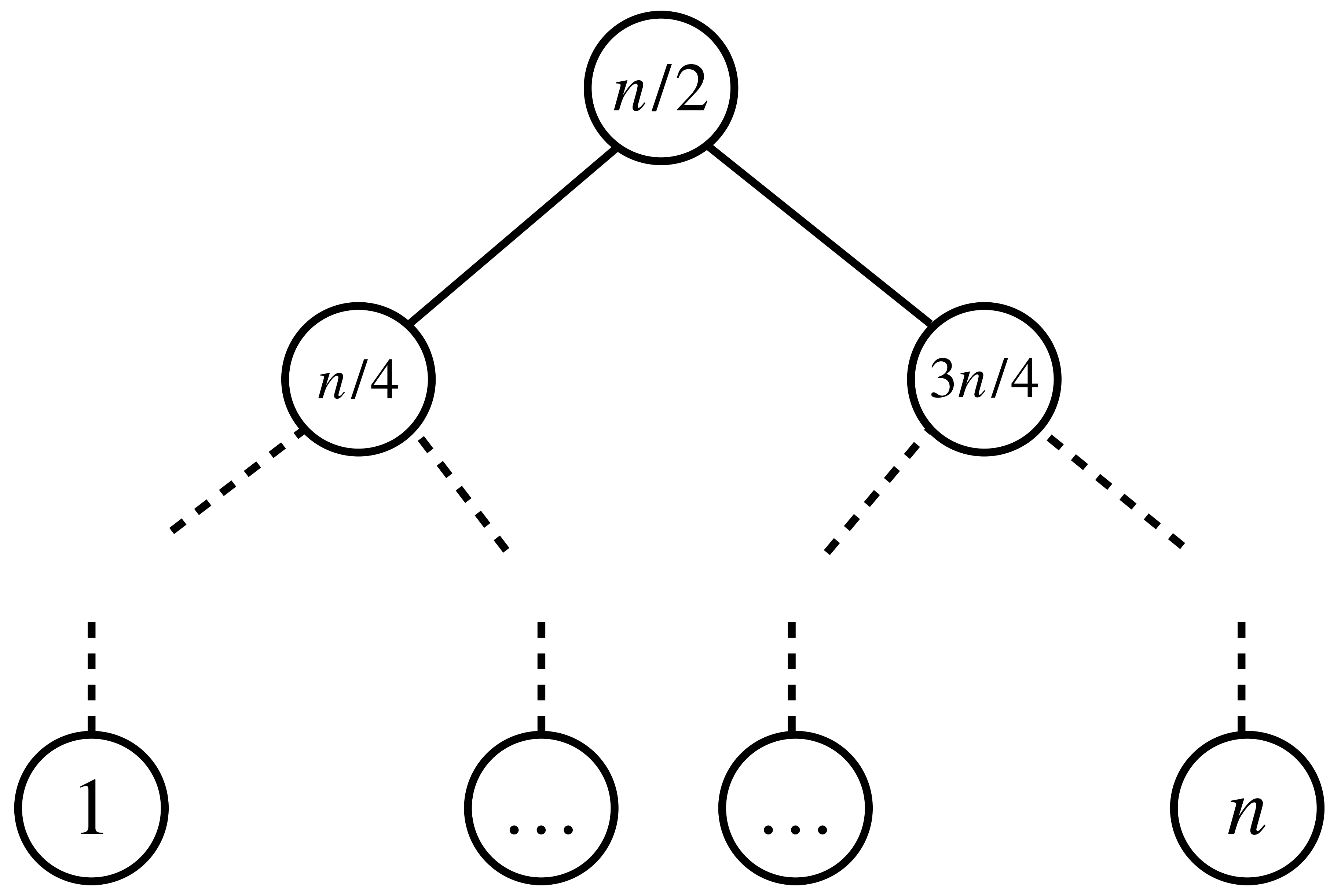
Best case: $h = O(\log n)$



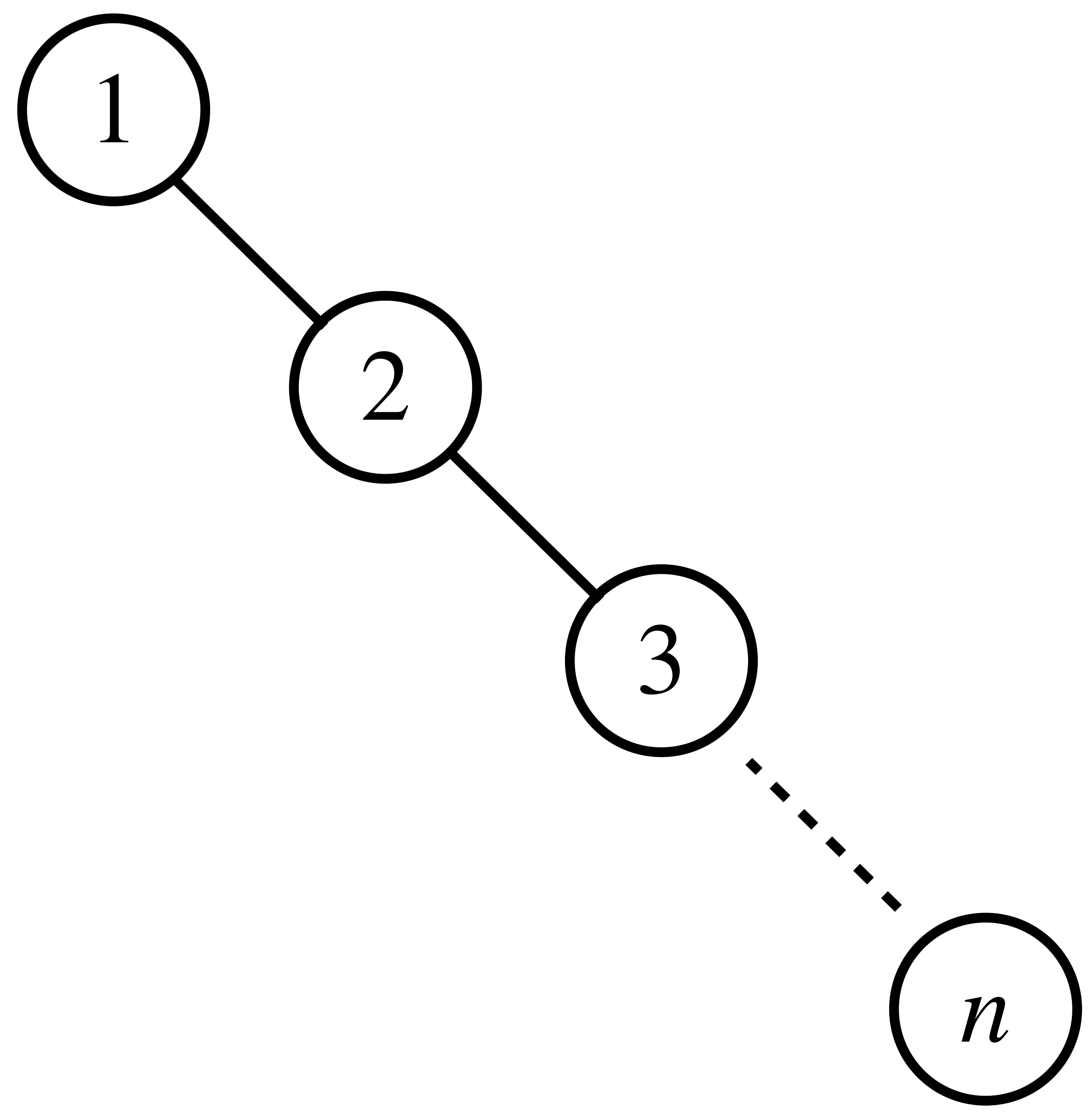
Are BSTs Good Enough?

BSTs can perform Insert, Delete, Search, etc., in $O(h)$ time.

But,



Best case: $h = O(\log n)$

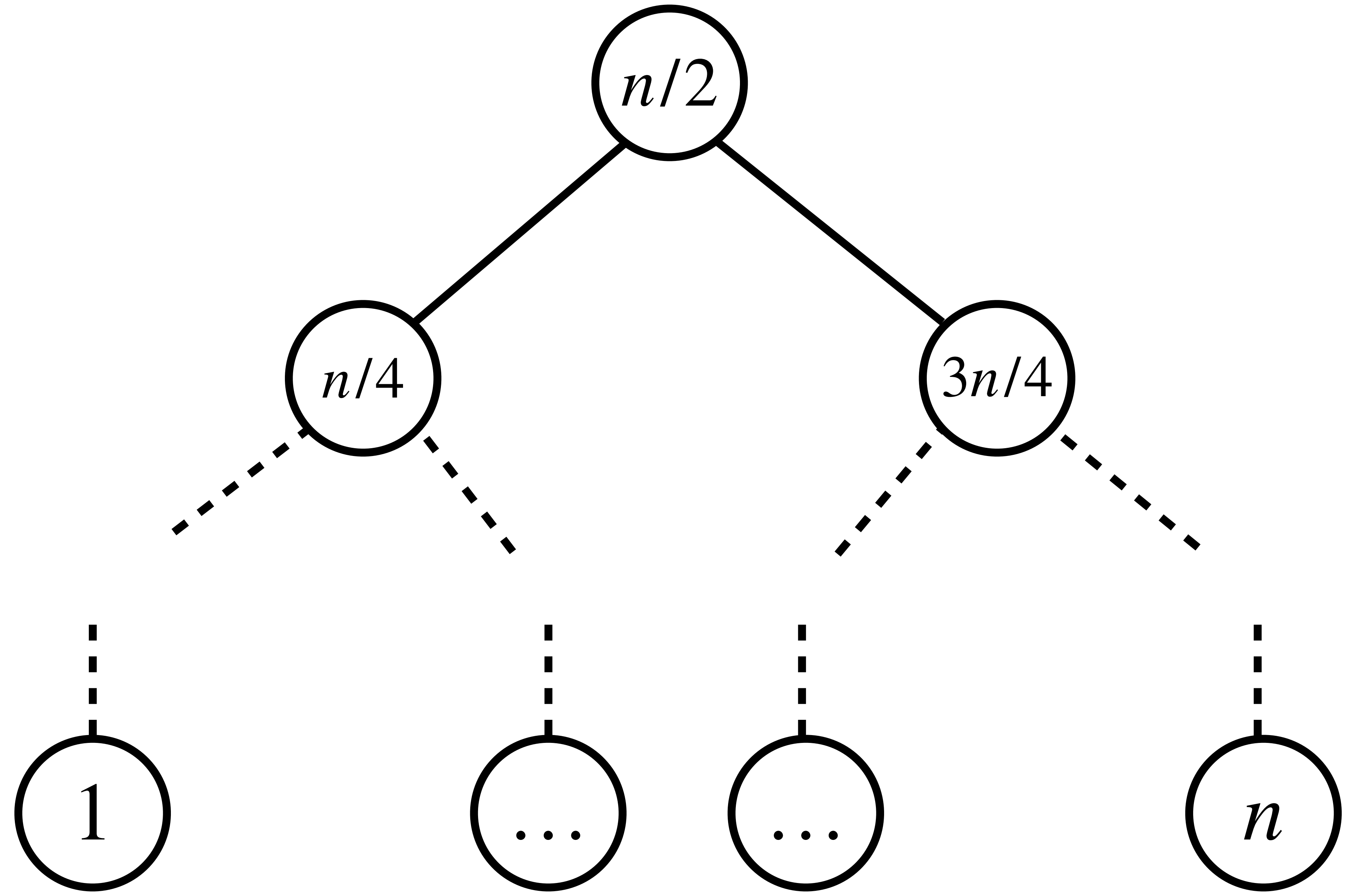


Worst case: $h = O(n)$

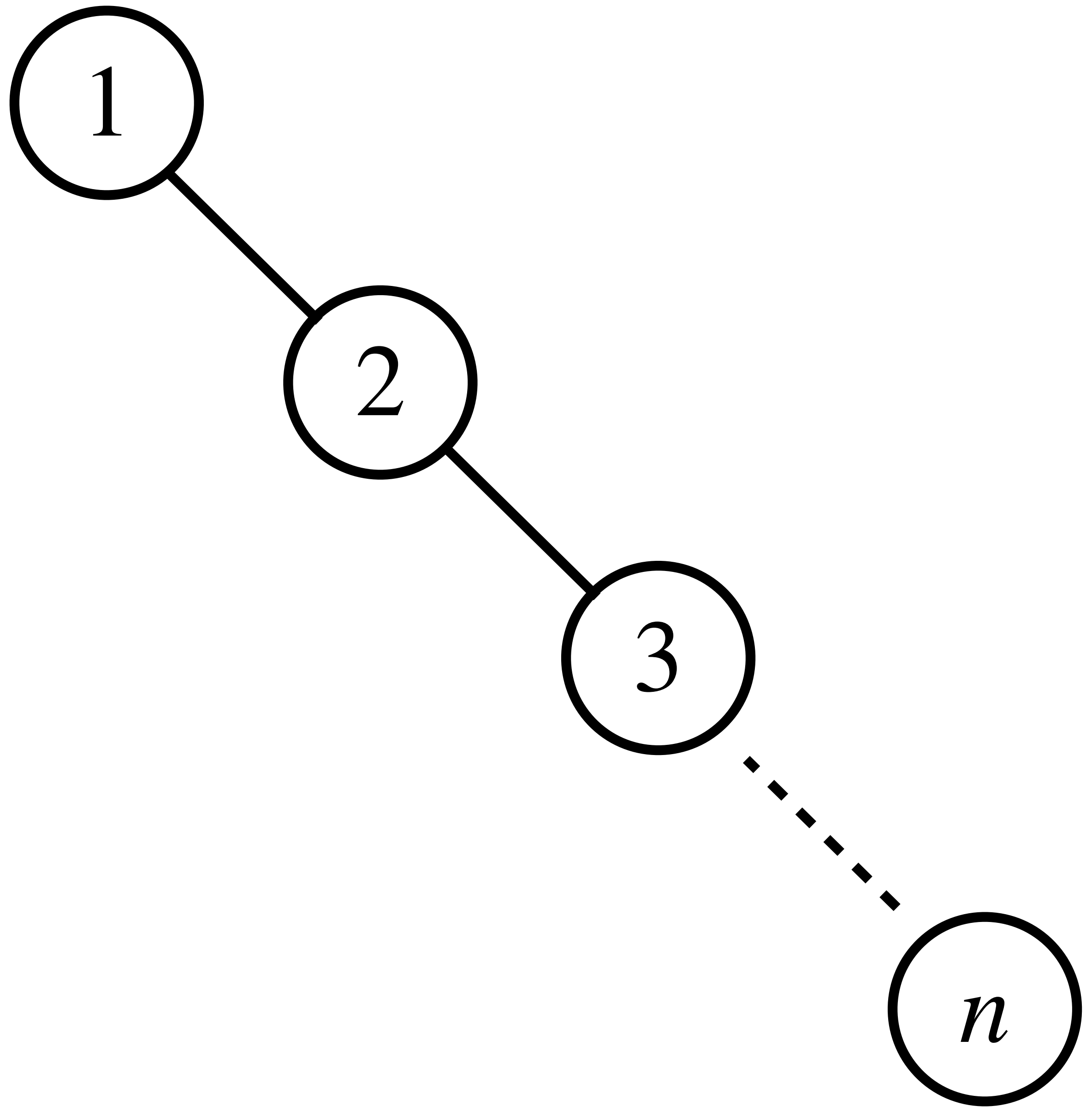
Are BSTs Good Enough?

BSTs can perform Insert, Delete, Search, etc., in $O(h)$ time.

But,



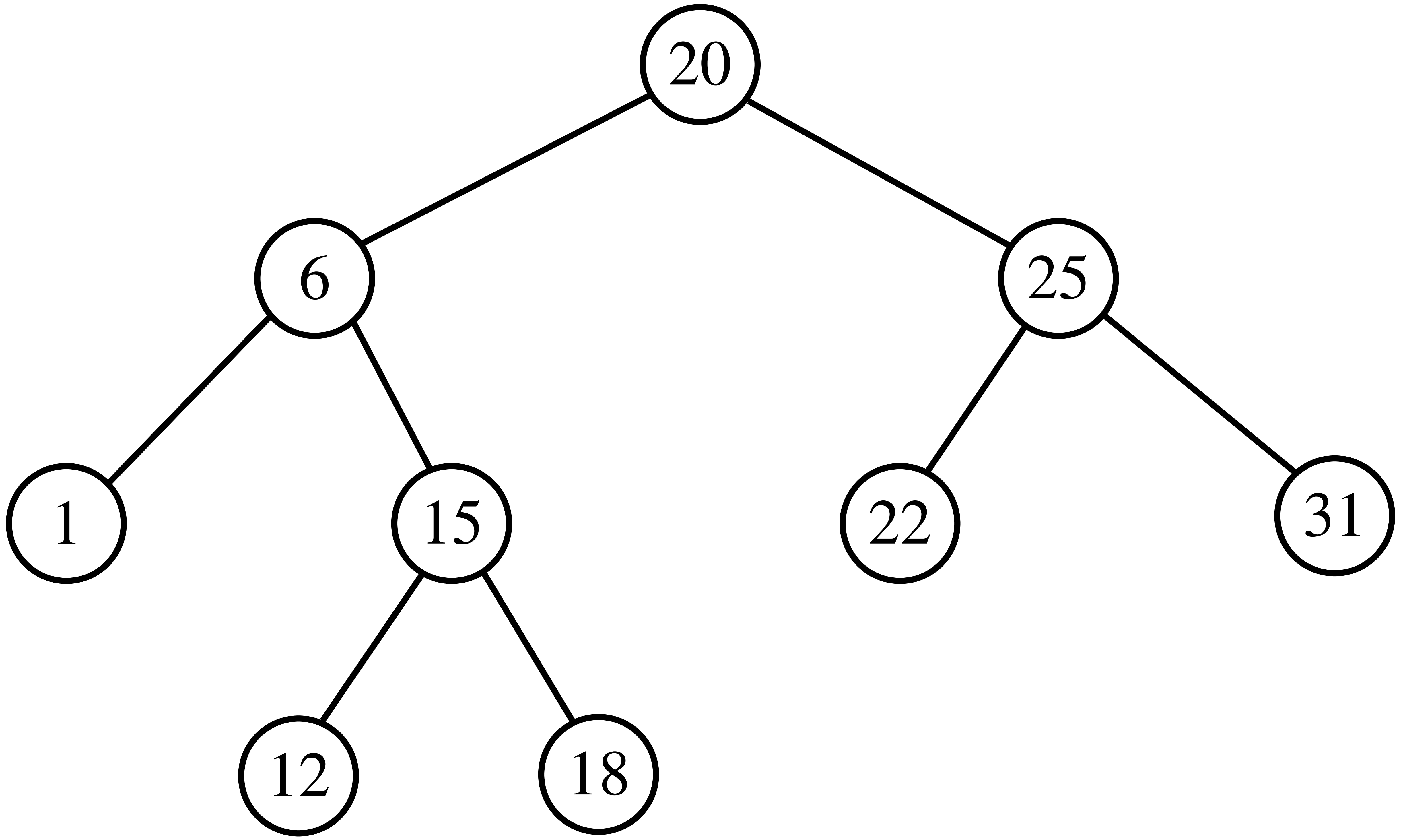
Best case: $h = O(\log n)$



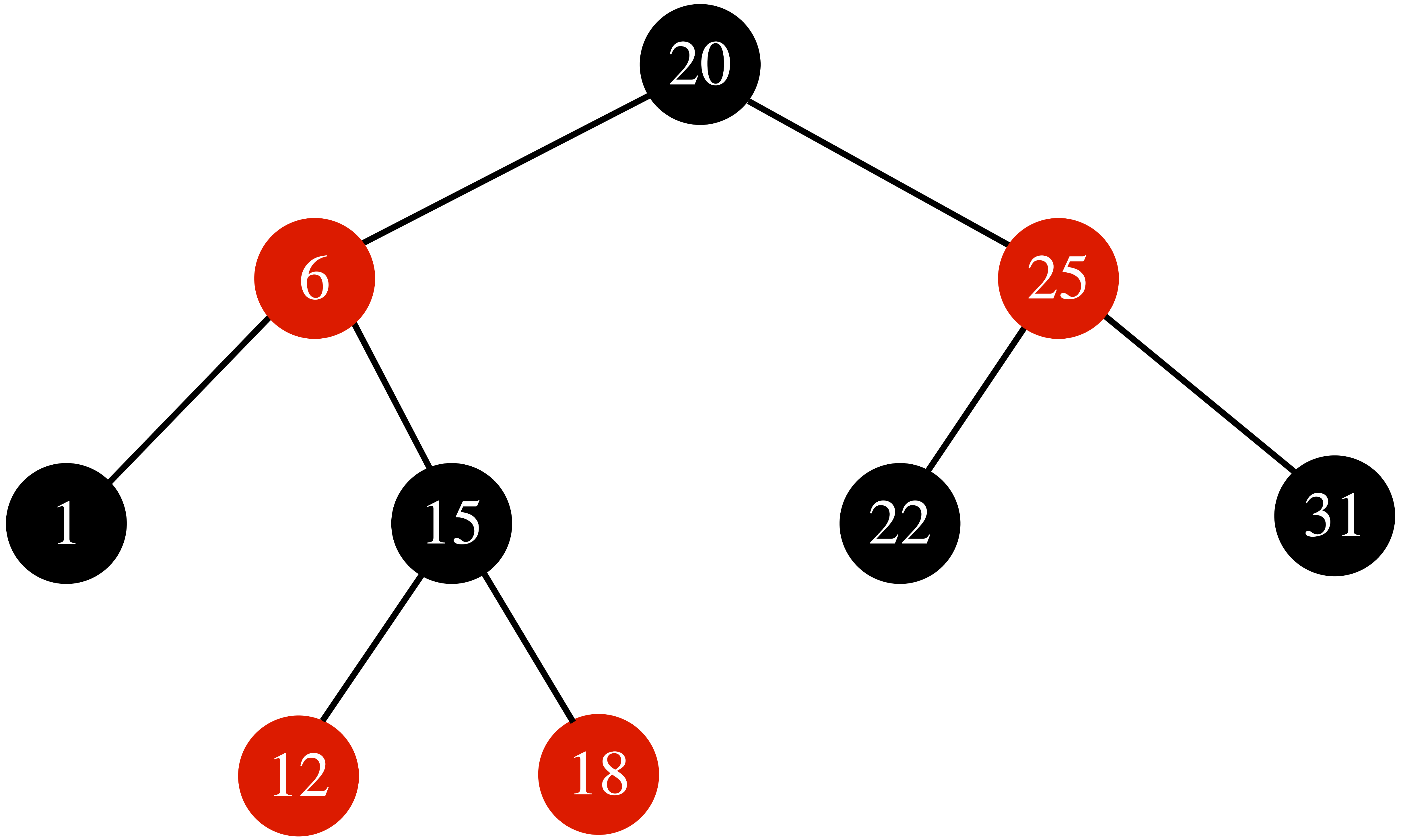
Worst case: $h = O(n)$

Red-black (RB) trees provide a way keep height $O(\log n)$.

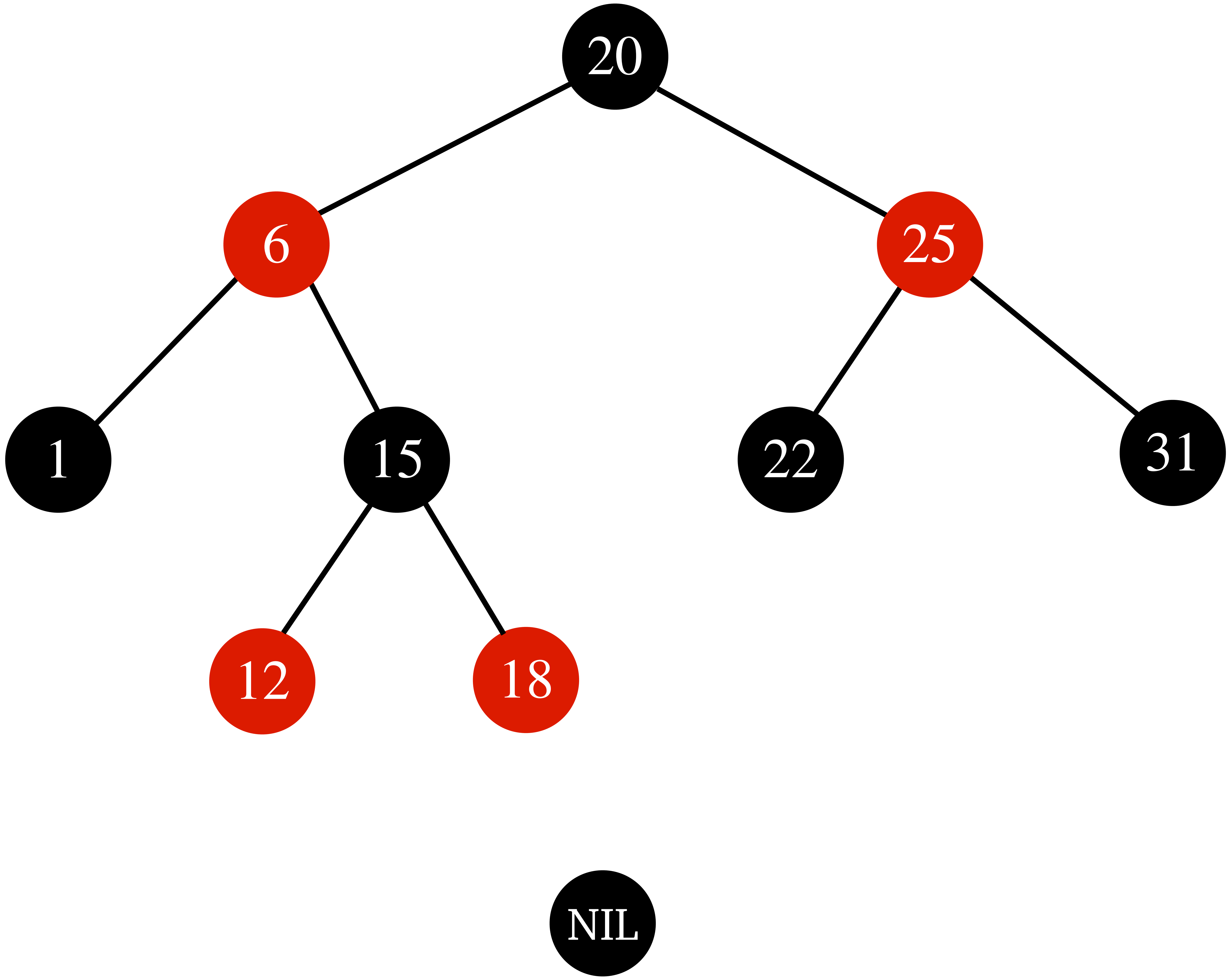
RB-Trees: How do they look like?



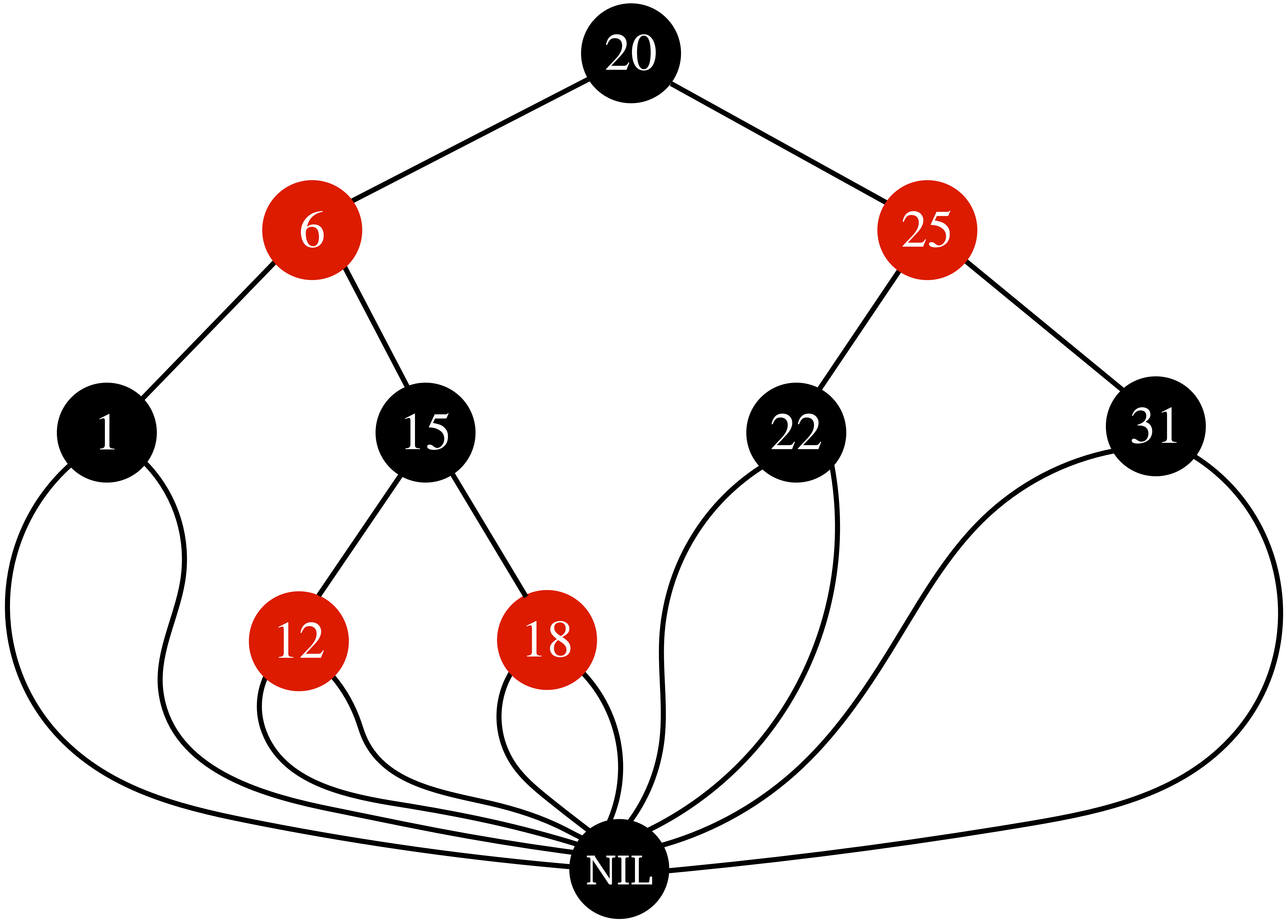
RB-Trees: How do they look like?



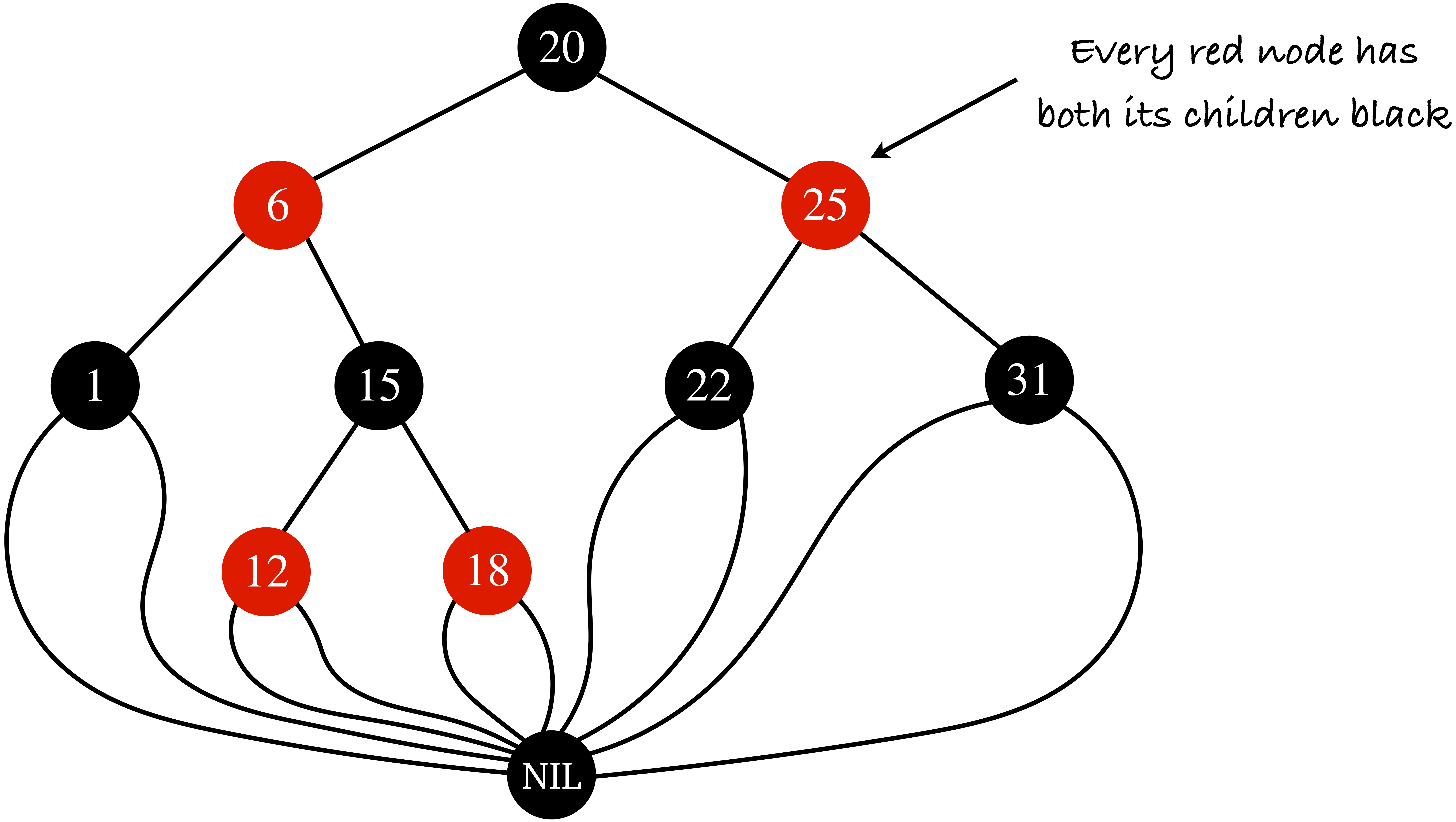
RB-Trees: How do they look like?



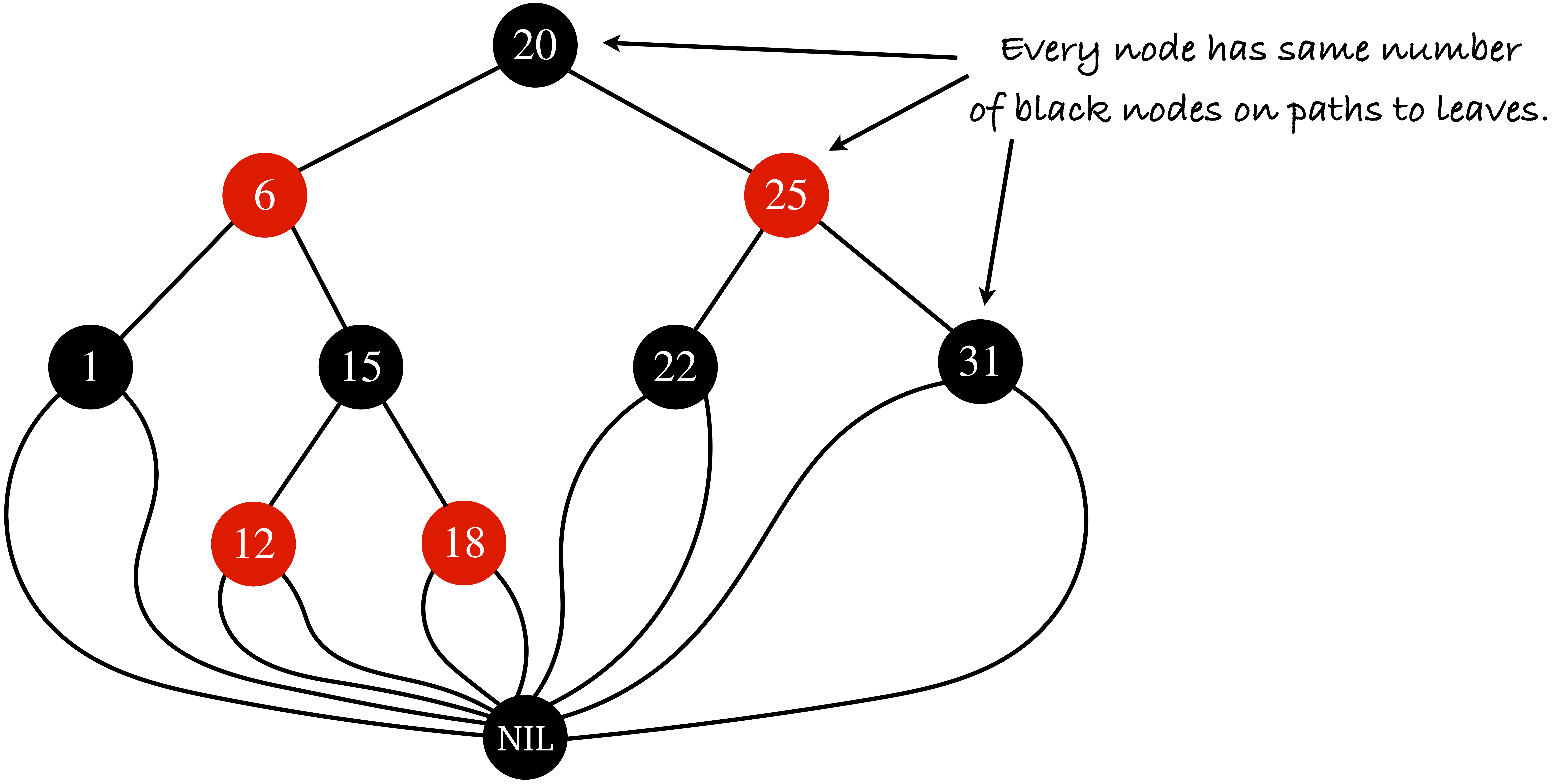
RB-Trees: How do they look like?



RB-Trees: How do they look like?



RB-Trees: How do they look like?



RB-Trees: Formal Description

RB-Trees: Formal Description

RB-trees are BSTs which satisfy the following properties:

RB-Trees: Formal Description

RB-trees are BSTs which satisfy the following properties:

- Every node has a colour either red or **black**.

RB-Trees: Formal Description

RB-trees are BSTs which satisfy the following properties:

- Every node has a colour either **red** or **black**.
- Root is **black**.

RB-Trees: Formal Description

RB-trees are BSTs which satisfy the following properties:

- Every node has a colour either **red** or **black**.
- Root is **black**.
- Leaf nodes are NIL nodes which are **black** in colour.

RB-Trees: Formal Description

RB-trees are BSTs which satisfy the following properties:

- Every node has a colour either red or **black**.
- Root is **black**.
- Leaf nodes are NIL nodes which are **black** in colour.
- Every **internal (non-leaf) node** has two children.

RB-Trees: Formal Description

RB-trees are BSTs which satisfy the following properties:

- Every node has a colour either **red** or **black**.
- Root is **black**.
- Leaf nodes are NIL nodes which are **black** in colour.
- Every **internal (non-leaf) node** has two children.
- If a node is **red**, then both its children are **black**.

RB-Trees: Formal Description

RB-trees are BSTs which satisfy the following properties:

- Every node has a colour either **red** or **black**.
- Root is **black**.
- Leaf nodes are NIL nodes which are **black** in colour.
- Every **internal (non-leaf) node** has two children.
- If a node is **red**, then both its children are **black**.
- For every node, all the paths from the node to descendent leaves contain the

RB-Trees: Formal Description

RB-trees are BSTs which satisfy the following properties:

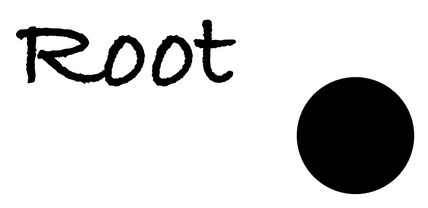
- Every node has a colour either red or **black**.
- Root is **black**.
- Leaf nodes are NIL nodes which are **black** in colour.
- Every **internal (non-leaf) node** has two children.
- If a node is red, then both its children are **black**.
- For every node, all the paths from the node to descendent leaves contain the same number of **black** nodes.

RB-Trees: Formal Description

RB-trees are BSTs which satisfy the following properties:

- Every node has a colour either **red** or **black**.
- Root is **black**.
- Leaf nodes are NIL nodes which are **black** in colour.
- Every **internal (non-leaf) node** has two children.
- If a node is **red**, then both its children are **black**.
- For every node, all the paths from the node to descendent leaves contain the same number of **black** nodes.

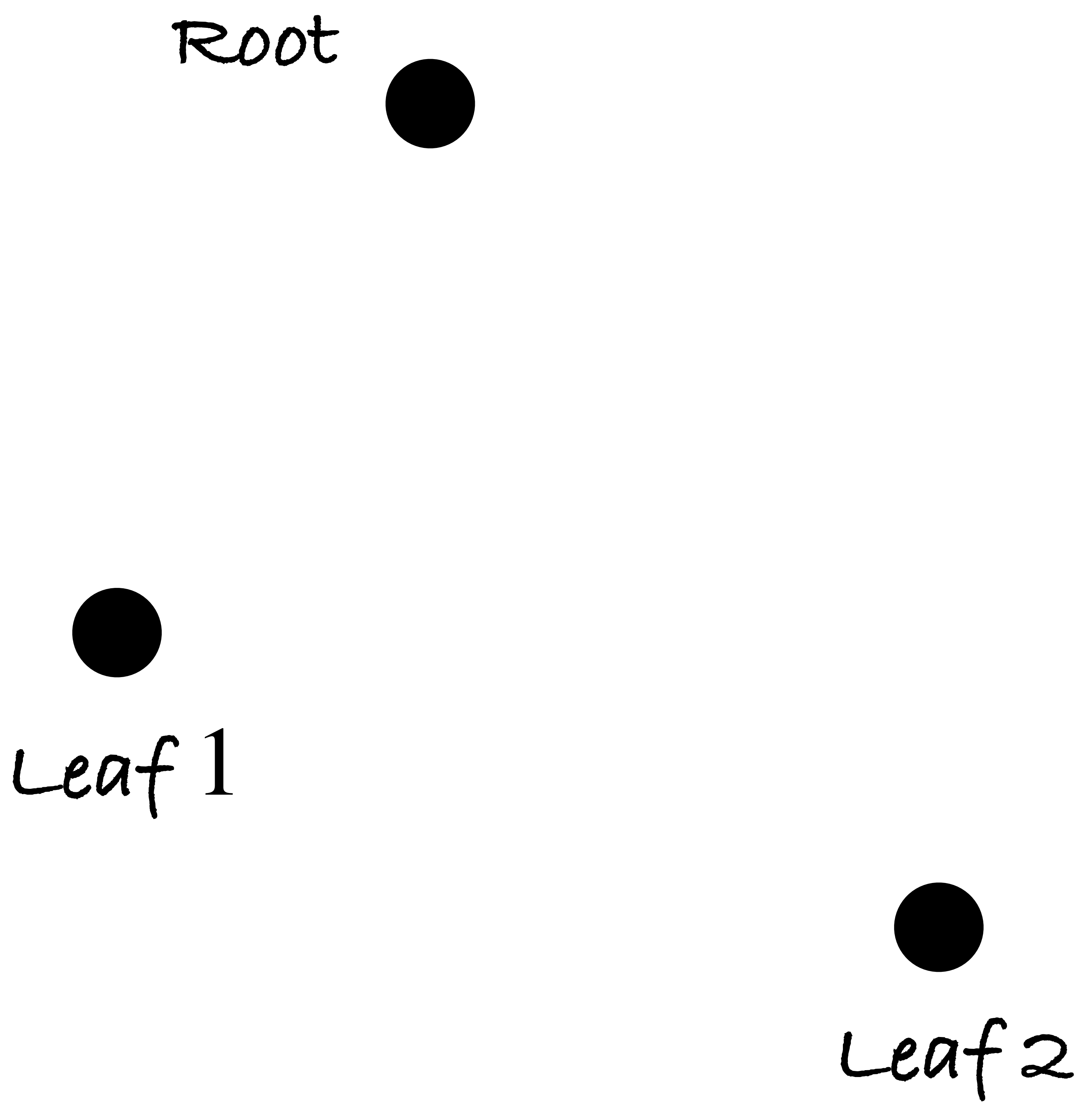
RB-Trees: Formal Description



RB-trees are BSTs which satisfy the following properties:

- Every node has a colour either red or **black**.
- Root is **black**.
- Leaf nodes are NIL nodes which are **black** in colour.
- Every **internal (non-leaf) node** has two children.
- If a node is red, then both its children are **black**.
- For every node, all the paths from the node to descendent leaves contain the same number of **black** nodes.

RB-Trees: Formal Description



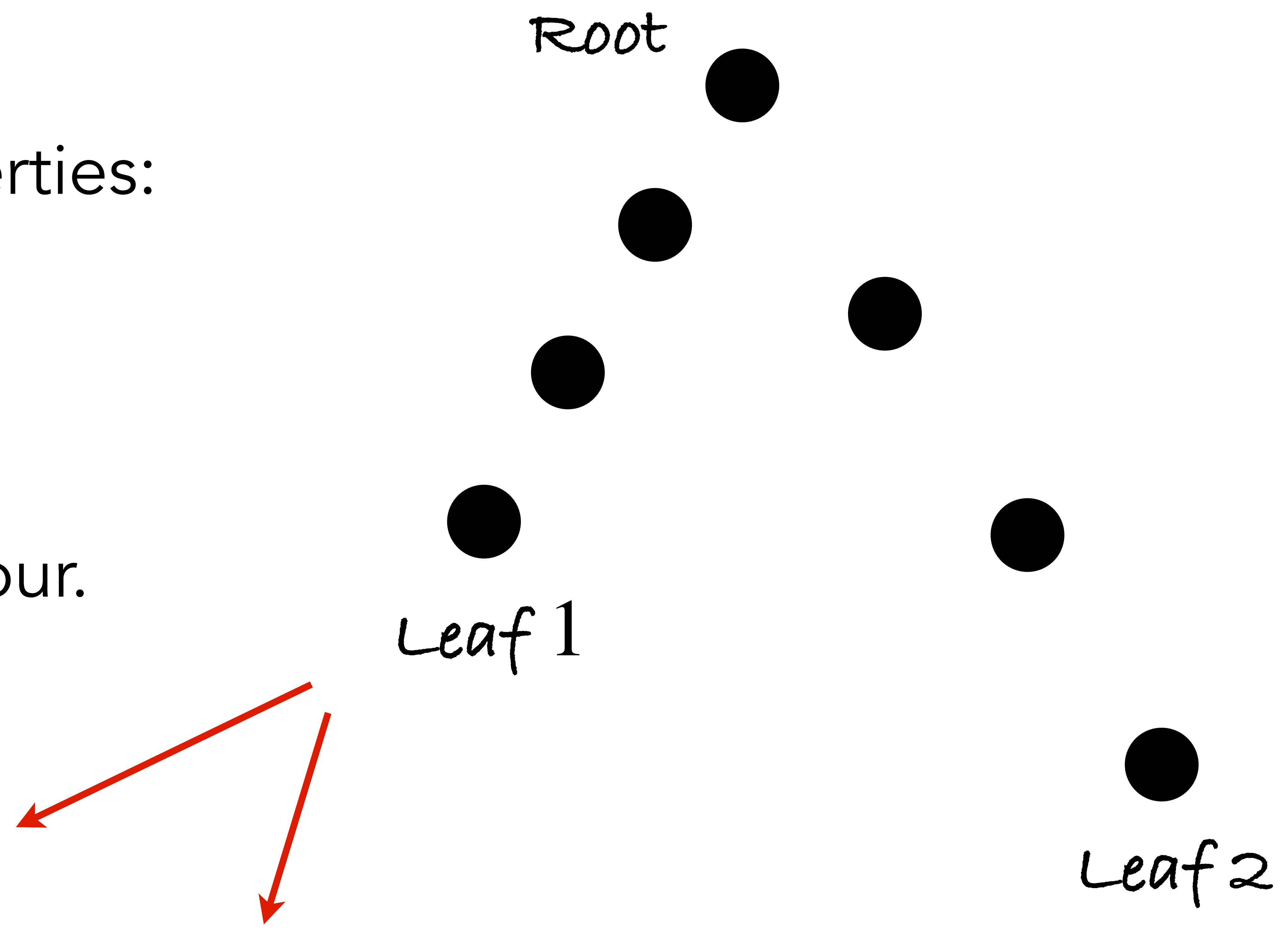
RB-trees are BSTs which satisfy the following properties:

- Every node has a colour either **red** or **black**.
- Root is **black**.
- Leaf nodes are NIL nodes which are **black** in colour.
- Every **internal (non-leaf) node** has two children.
- If a node is **red**, then both its children are **black**.
- For every node, all the paths from the node to descendent leaves contain the same number of **black** nodes.

RB-Trees: Formal Description

RB-trees are BSTs which satisfy the following properties:

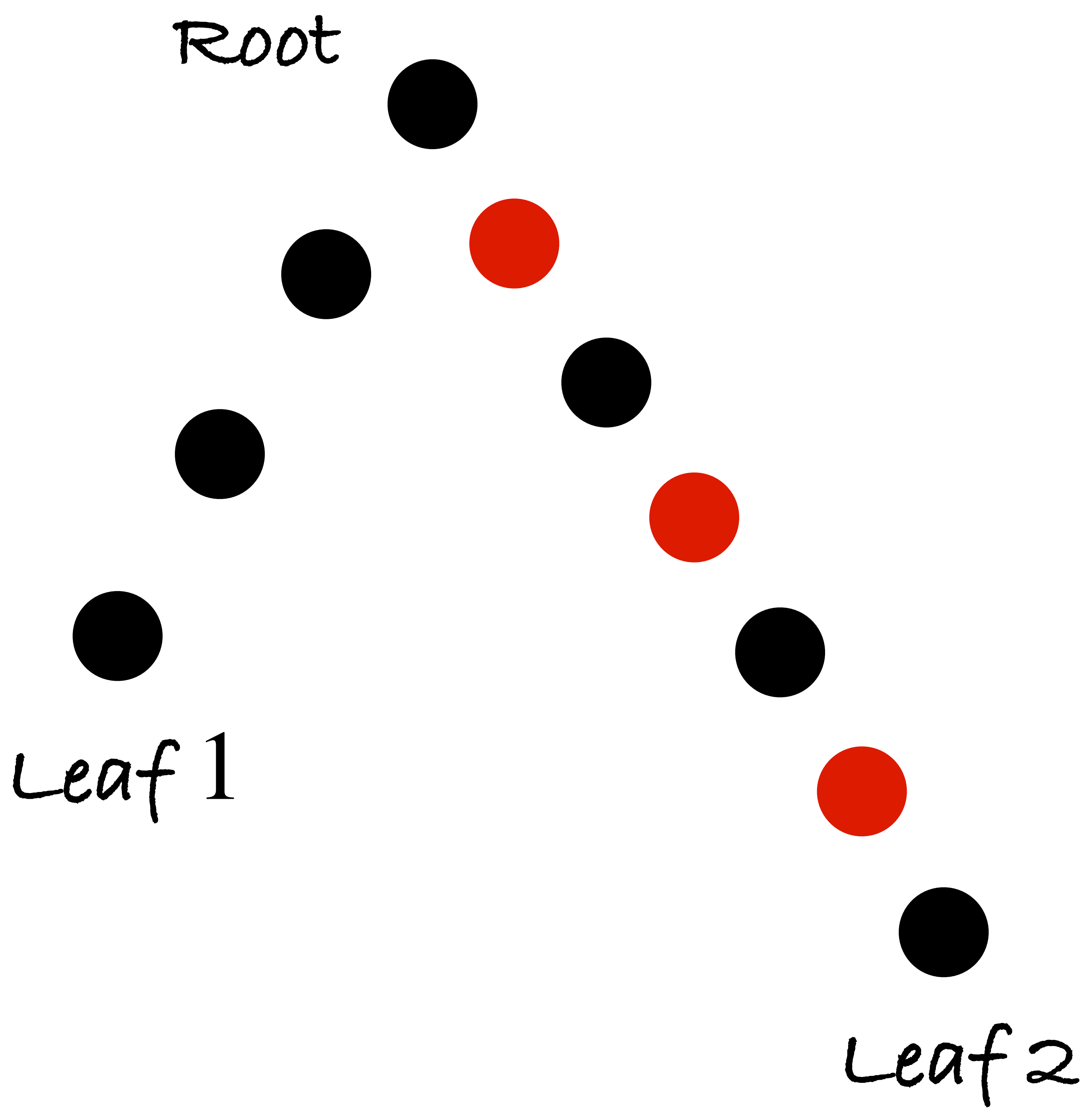
- Every node has a colour either **red** or **black**.
- Root is **black**.
- Leaf nodes are NIL nodes which are **black** in colour.
- Every **internal (non-leaf) node** has two children.
- If a node is **red**, then both its children are **black**.
- For every node, all the paths from the node to descendent leaves contain the same number of **black** nodes.



RB-Trees: Formal Description

RB-trees are BSTs which satisfy the following properties:

- Every node has a colour either **red** or **black**.
- Root is **black**.
- Leaf nodes are NIL nodes which are **black** in colour.
- Every **internal (non-leaf) node** has two children.
- If a node is **red**, then both its children are **black**.
- For every node, all the paths from the node to descendent leaves contain the same number of **black** nodes.



RB-Trees: Formal Description

RB-trees are BSTs which satisfy the following properties:

- Every node has a colour either **red** or **black**.
- Root is **black**.
- Leaf nodes are NIL nodes which are **black** in colour.
- Every **internal node** has two children.
- If a node is **red**, then both its children are **black**.
- For every node, all the paths from the node to descendent leaves contain the same number of **black** nodes.

Both these properties ensure that no path from root to a leaf is more than twice as long as any other.

RB-Trees: Black Height

RB-Trees: Black Height

Def: For any node x, the number of black nodes on any path from x to a **leaf**,

RB-Trees: Black Height

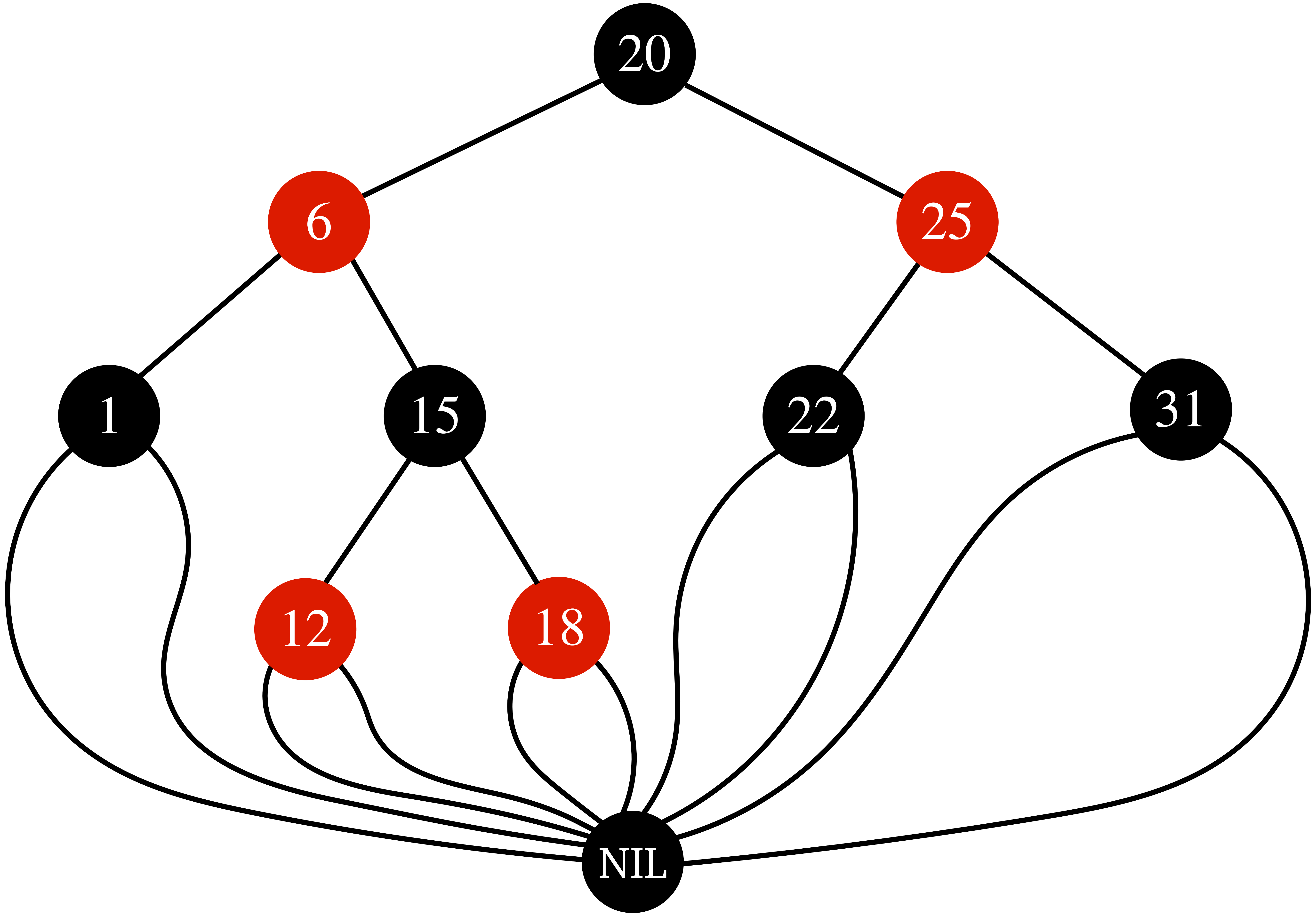
Def: For any node x, the number of black nodes on any path from x to a **leaf**, excluding x,

RB-Trees: Black Height

Def: For any node x, the number of black nodes on any path from x to a **leaf**, excluding x, is called **black height** of x, denoted by $bh(x)$.

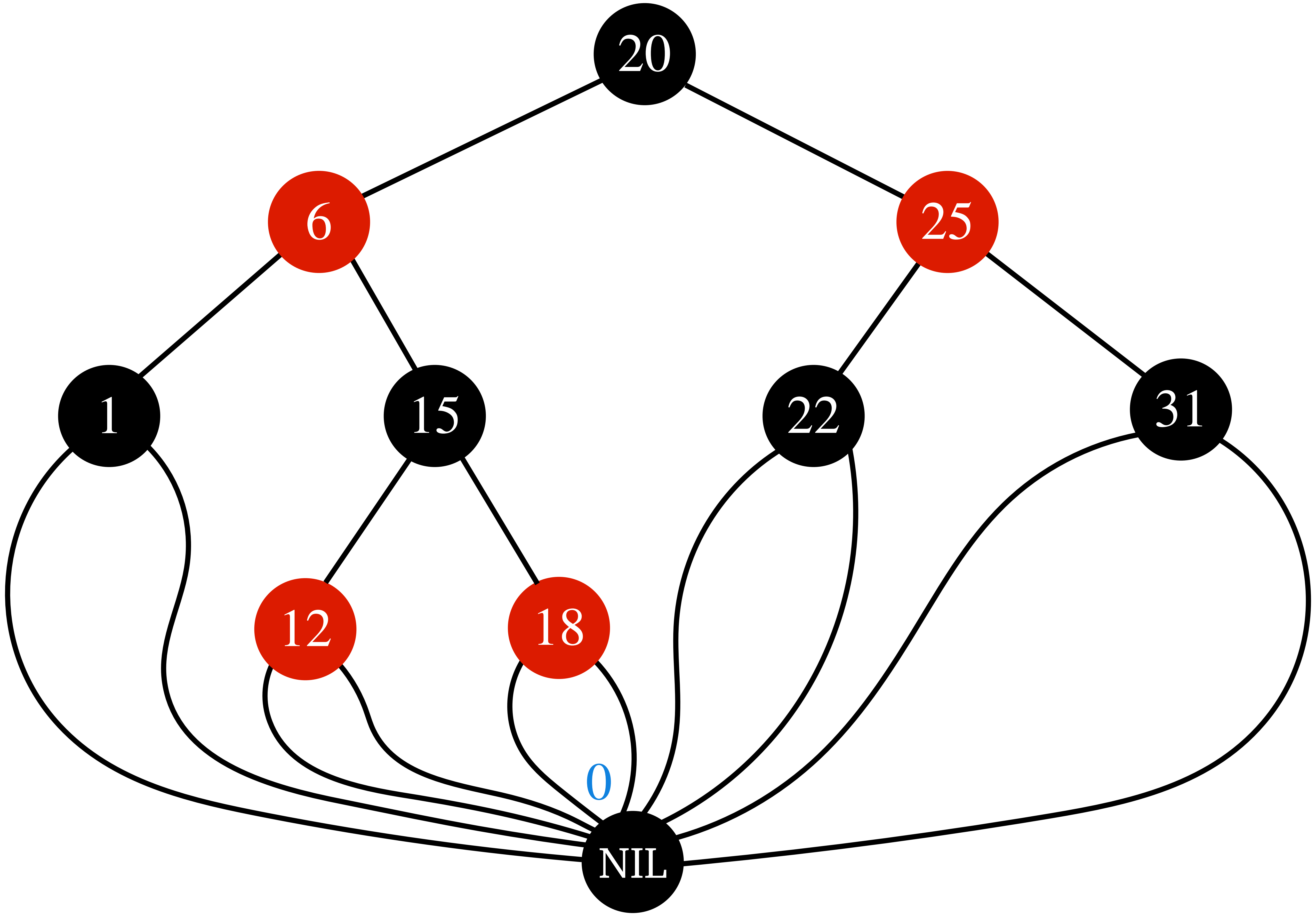
RB-Trees: Black Height

Def: For any node x, the number of black nodes on any path from x to a **leaf**, excluding x, is called **black height** of x, denoted by $bh(x)$.



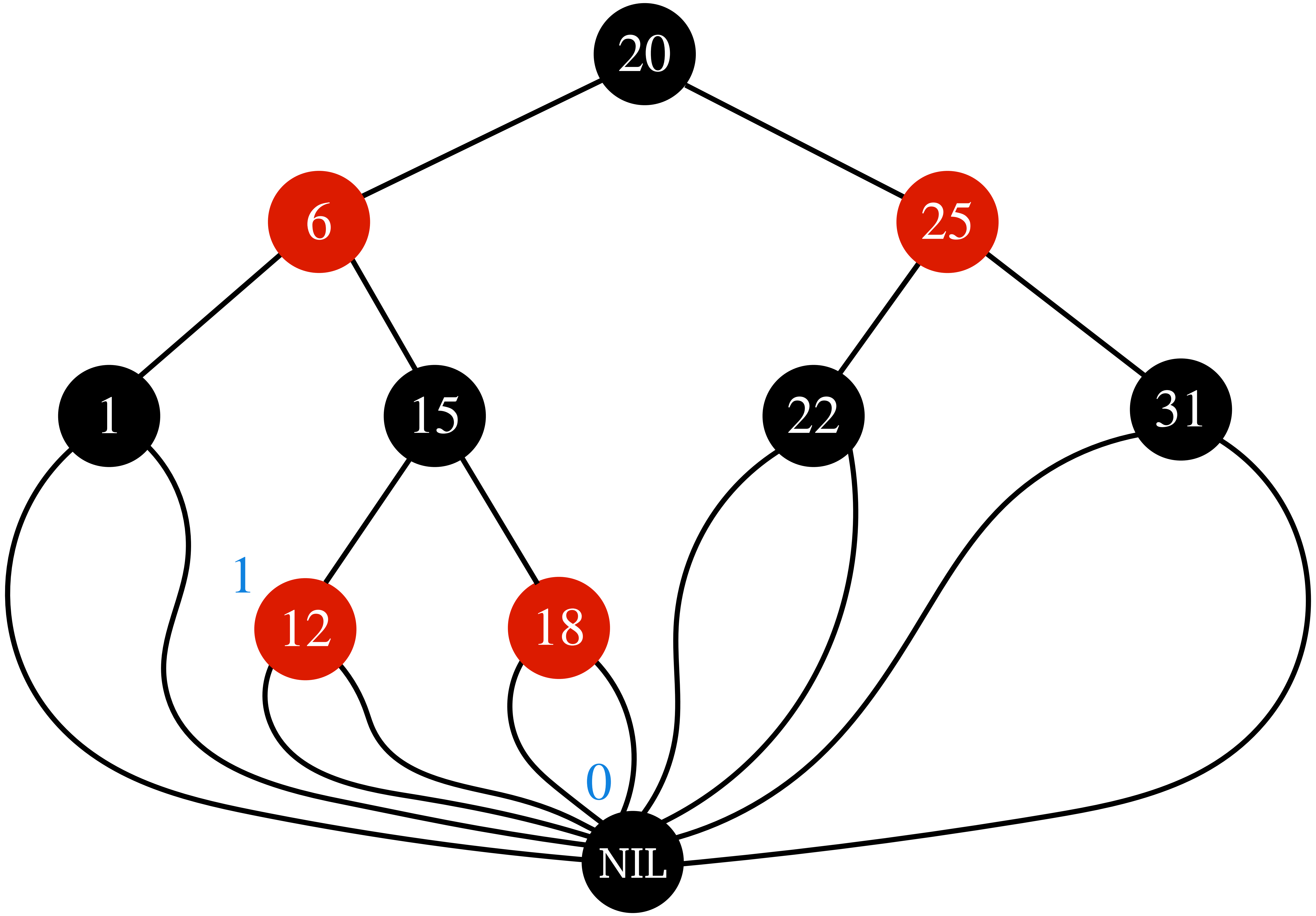
RB-Trees: Black Height

Def: For any node x, the number of black nodes on any path from x to a **leaf**, excluding x, is called **black height** of x, denoted by $bh(x)$.



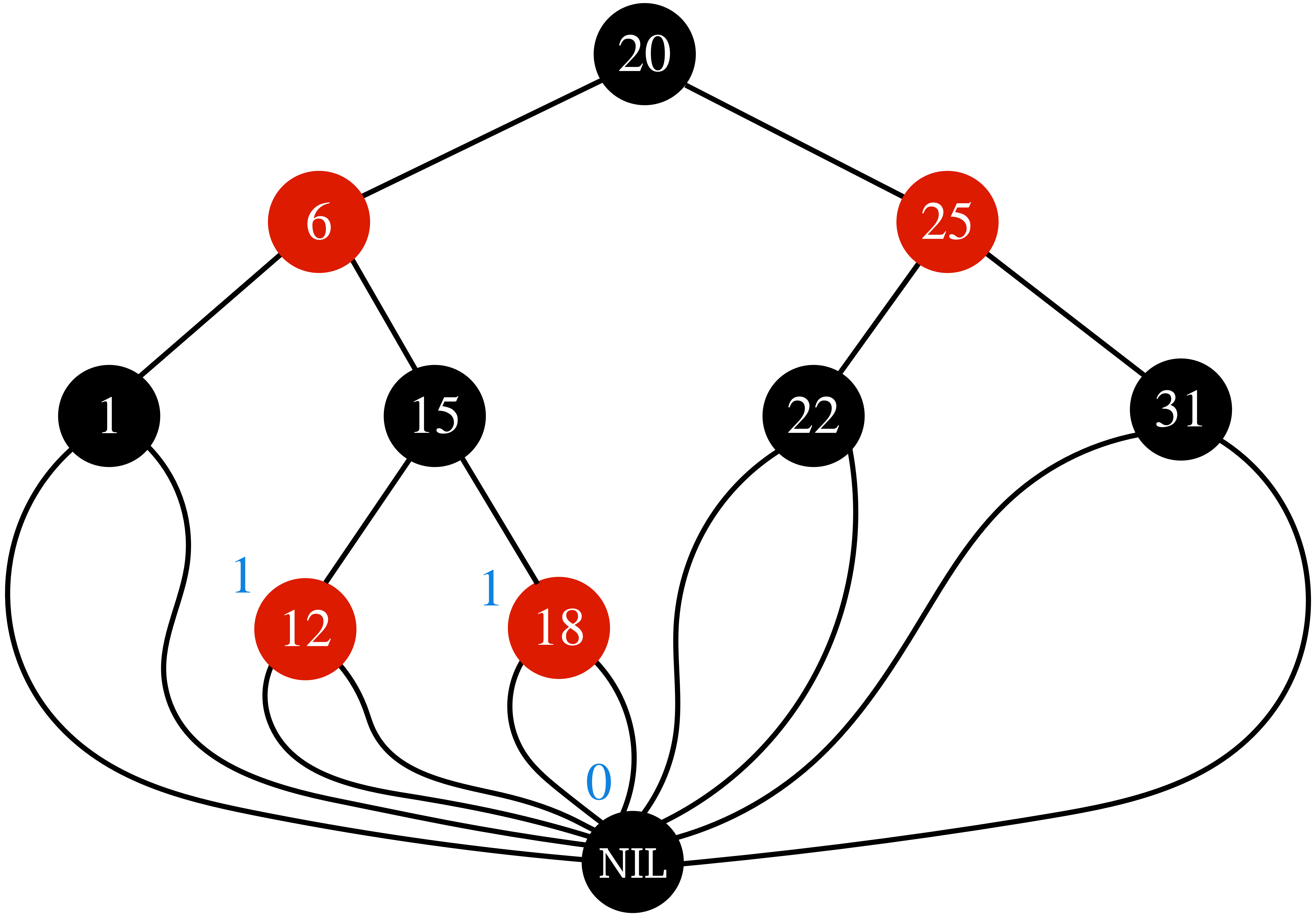
RB-Trees: Black Height

Def: For any node x, the number of black nodes on any path from x to a **leaf**, excluding x, is called **black height** of x, denoted by $bh(x)$.



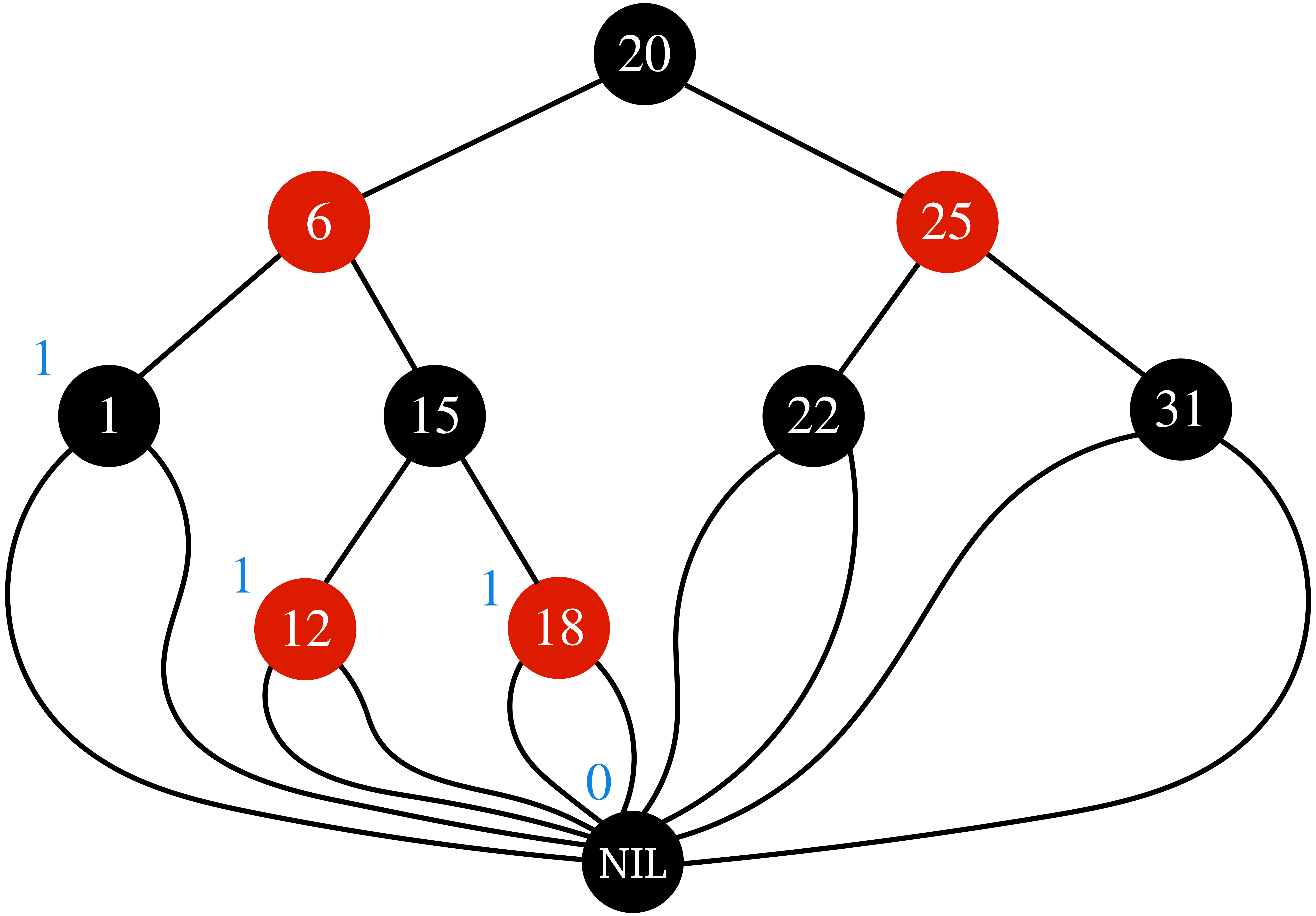
RB-Trees: Black Height

Def: For any node x, the number of black nodes on any path from x to a **leaf**, excluding x, is called **black height** of x, denoted by $bh(x)$.



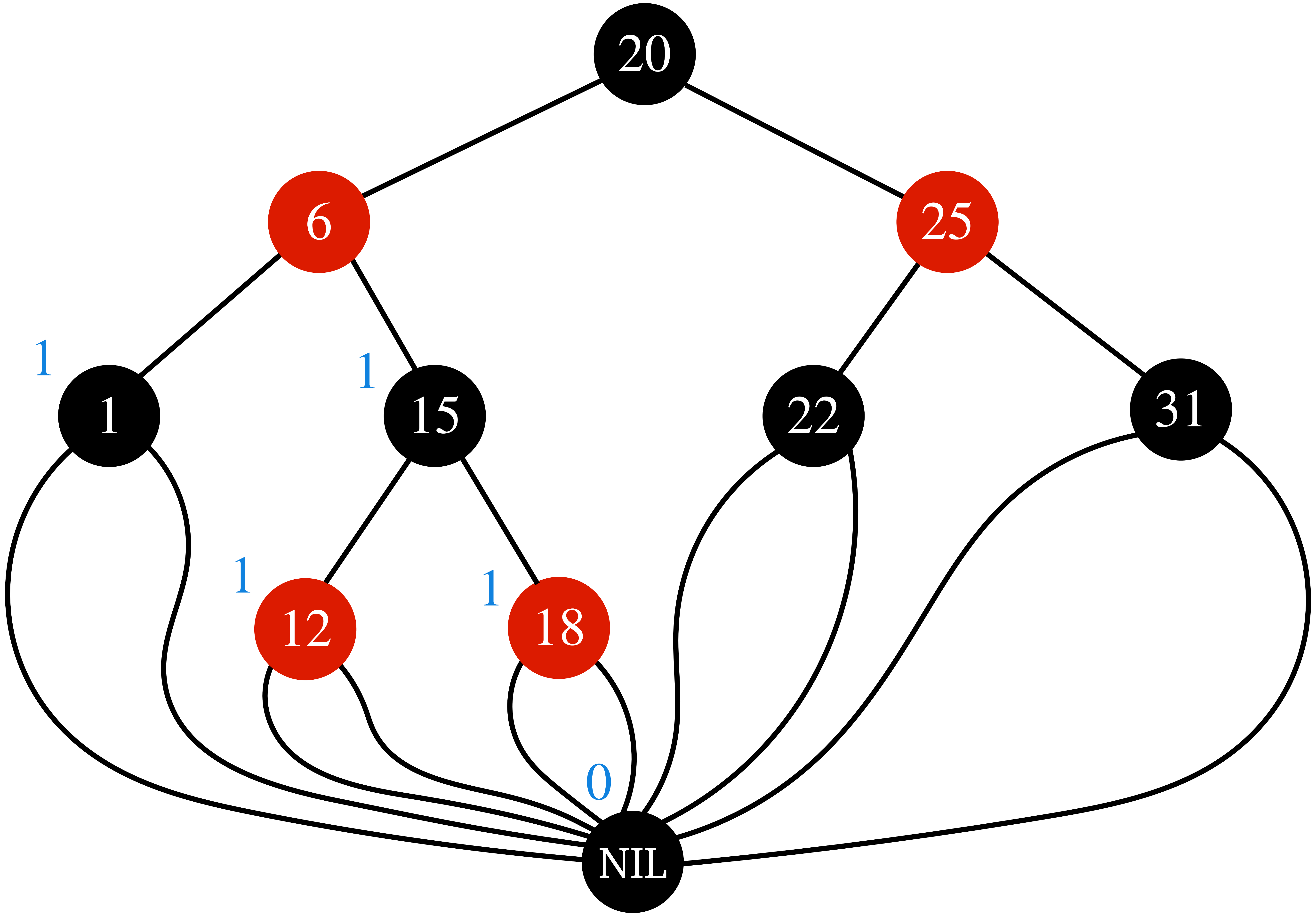
RB-Trees: Black Height

Def: For any node x, the number of black nodes on any path from x to a **leaf**, excluding x, is called **black height** of x, denoted by $bh(x)$.



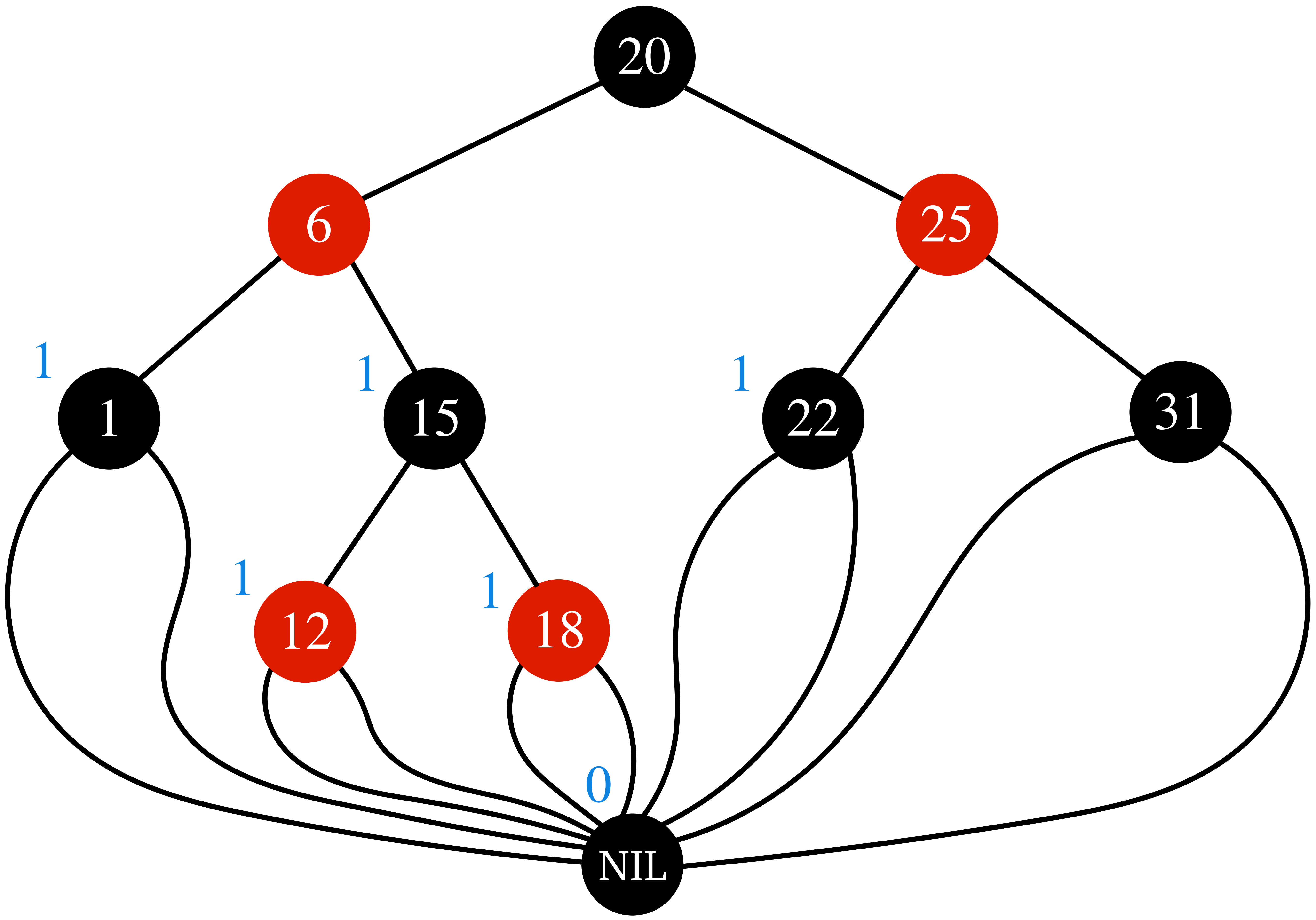
RB-Trees: Black Height

Def: For any node x, the number of black nodes on any path from x to a **leaf**, excluding x, is called **black height** of x, denoted by $bh(x)$.



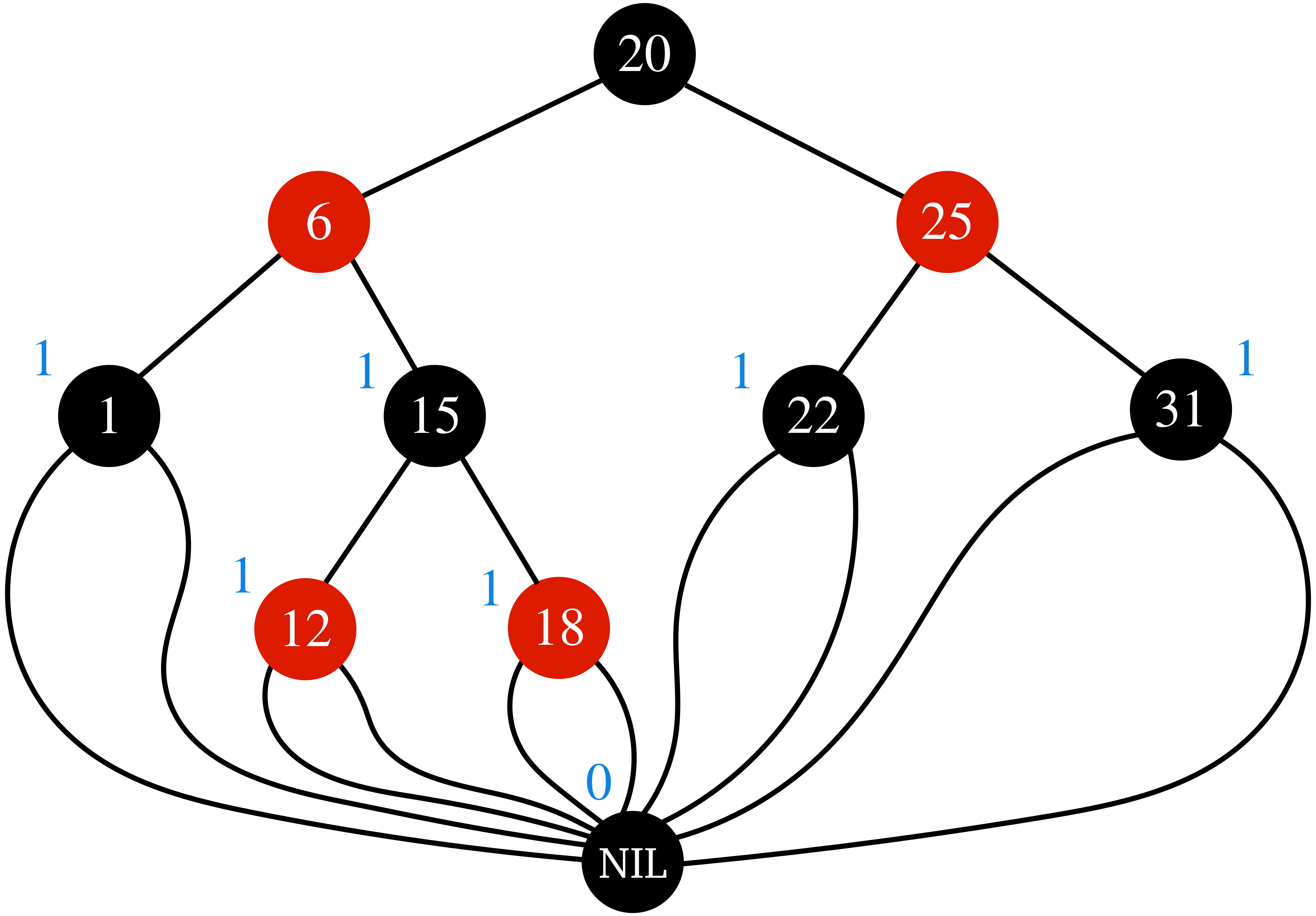
RB-Trees: Black Height

Def: For any node x, the number of black nodes on any path from x to a **leaf**, excluding x, is called **black height** of x, denoted by $bh(x)$.



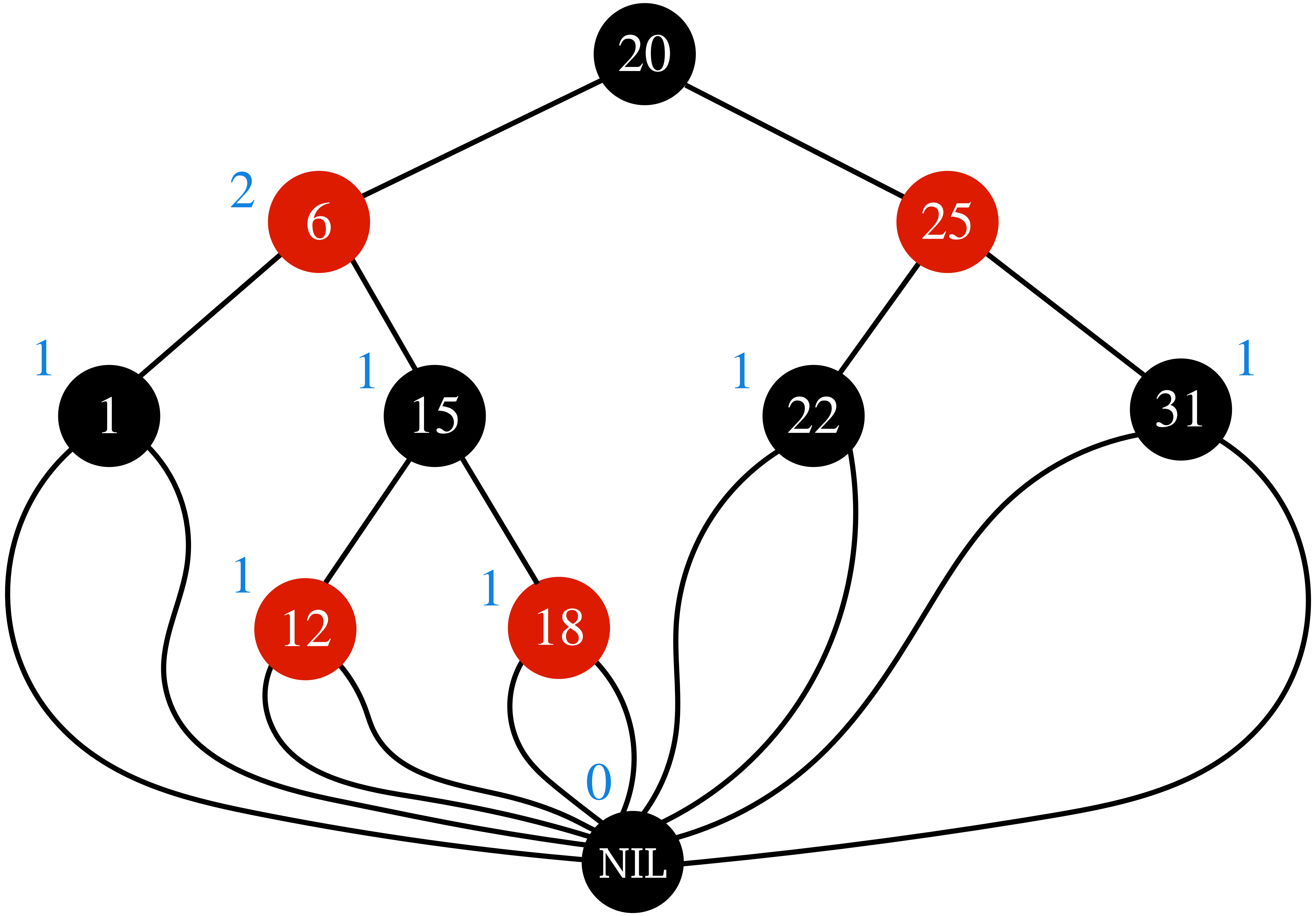
RB-Trees: Black Height

Def: For any node x, the number of black nodes on any path from x to a **leaf**, excluding x, is called **black height** of x, denoted by $bh(x)$.



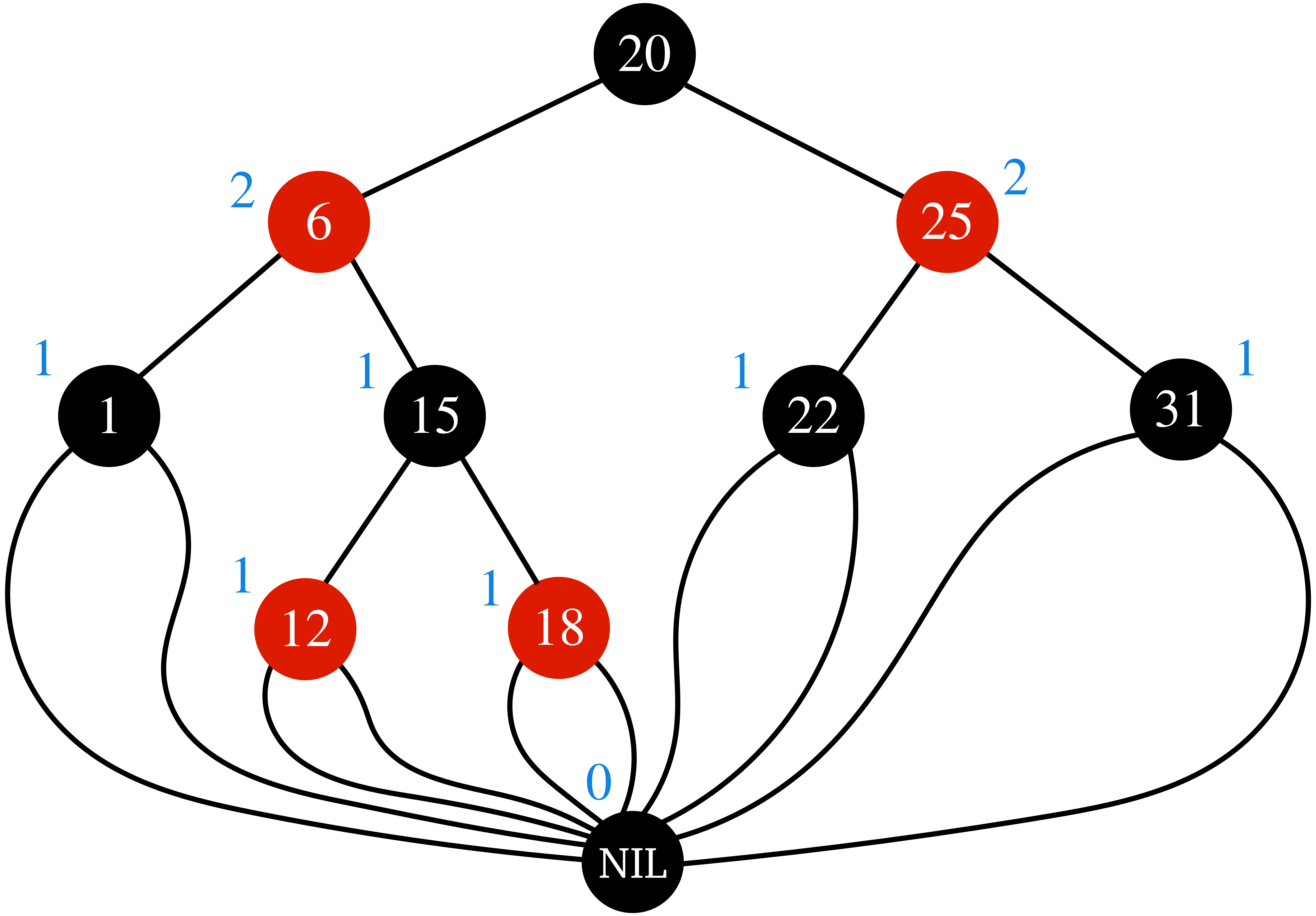
RB-Trees: Black Height

Def: For any node x, the number of black nodes on any path from x to a **leaf**, excluding x, is called **black height** of x, denoted by $bh(x)$.



RB-Trees: Black Height

Def: For any node x, the number of black nodes on any path from x to a **leaf**, excluding x, is called **black height** of x, denoted by $bh(x)$.



RB-Trees: Black Height

Def: For any node x, the number of black nodes on any path from x to a **leaf**, excluding x, is called **black height** of x, denoted by $bh(x)$.