Balanced BSTs

Abstract Data Type	Operations	Data Structures
Dictionary		

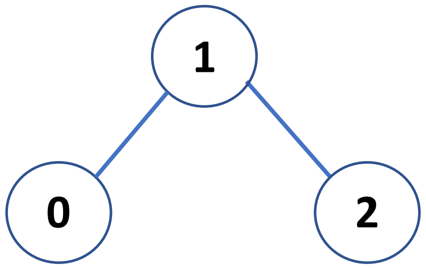
Abstract Data Type	Operations	Data Structures
Dictionary	**Search, Insert, Delete**	

Abstract Data Type	Operations	Data Structures
Dictionary	**Search, Insert, Delete**	BSTs

Recall the problem with BSTs…

Recall the problem with BSTs…

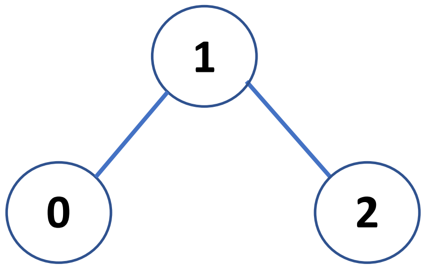
Suppose we start with
a “balanced” BST



Recall the problem with BSTs…

Suppose we start with a “balanced” BST

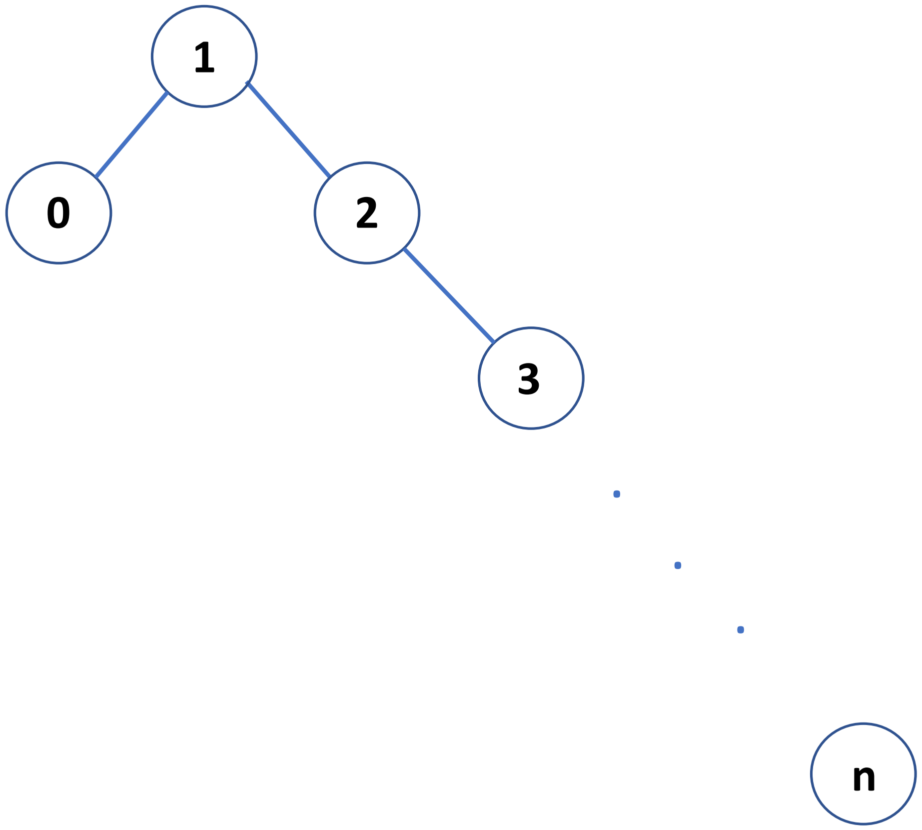
Insert keys 3, 4, 5, ..., n



Recall the problem with BSTs…

Suppose we start with a “balanced” BST

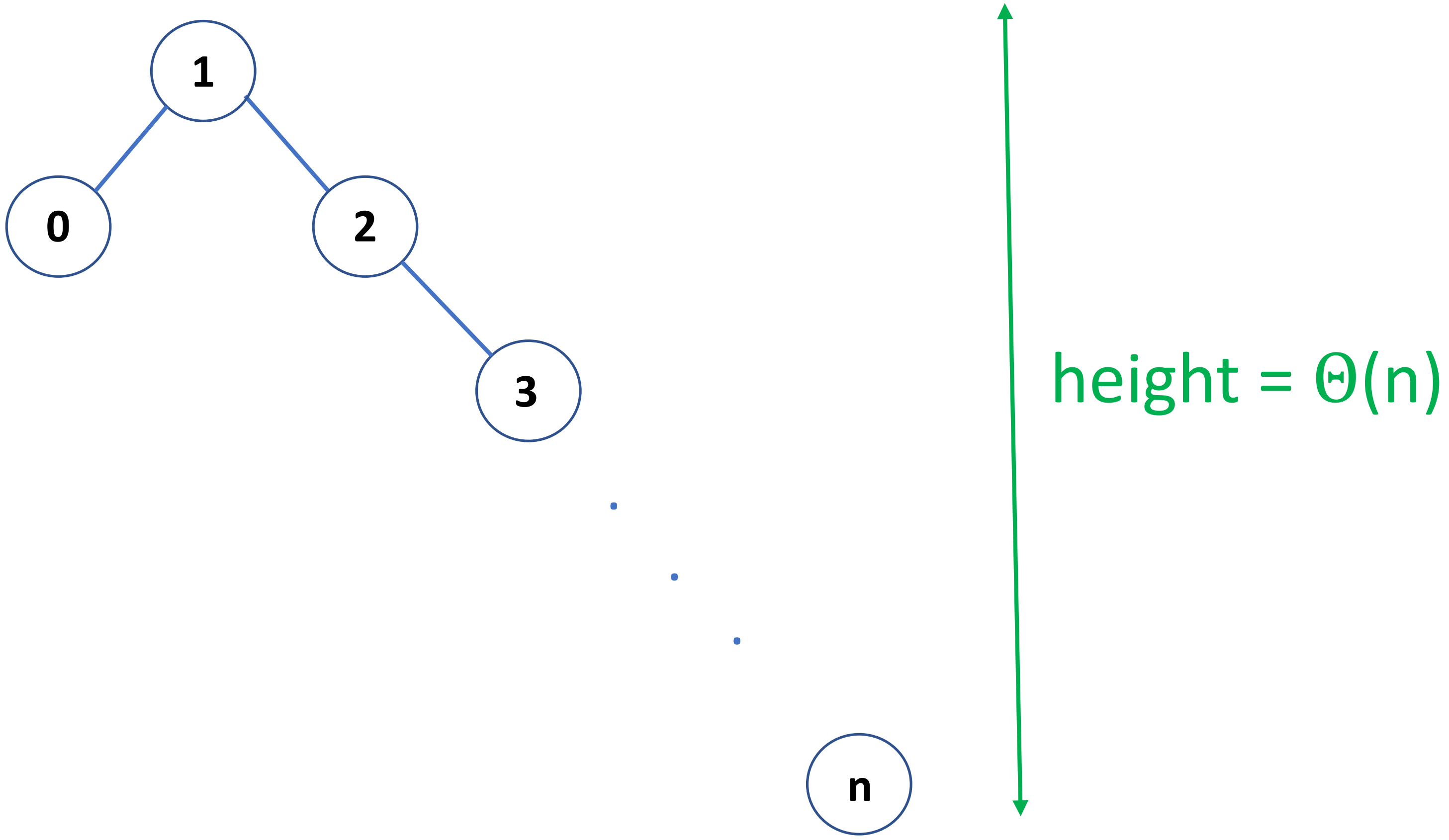
Insert keys 3, 4, 5, ..., n



Recall the problem with BSTs…

Suppose we start with
a “balanced” BST

Insert keys 3, 4, 5, …, n

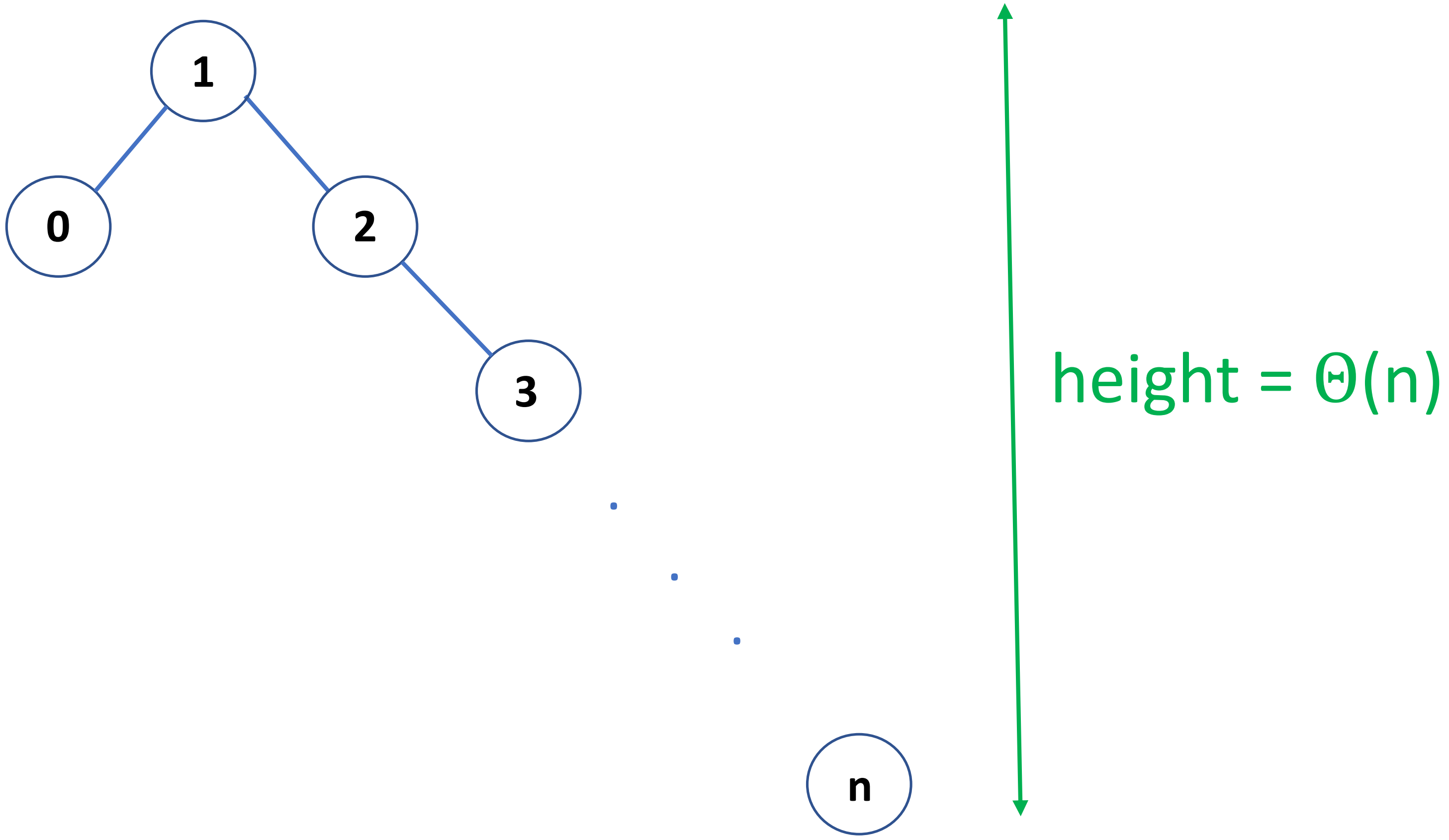


Recall the problem with BSTs…

Suppose we start with a “balanced” BST

Insert keys 3, 4, 5, …, n

Search takes Θ(n) !



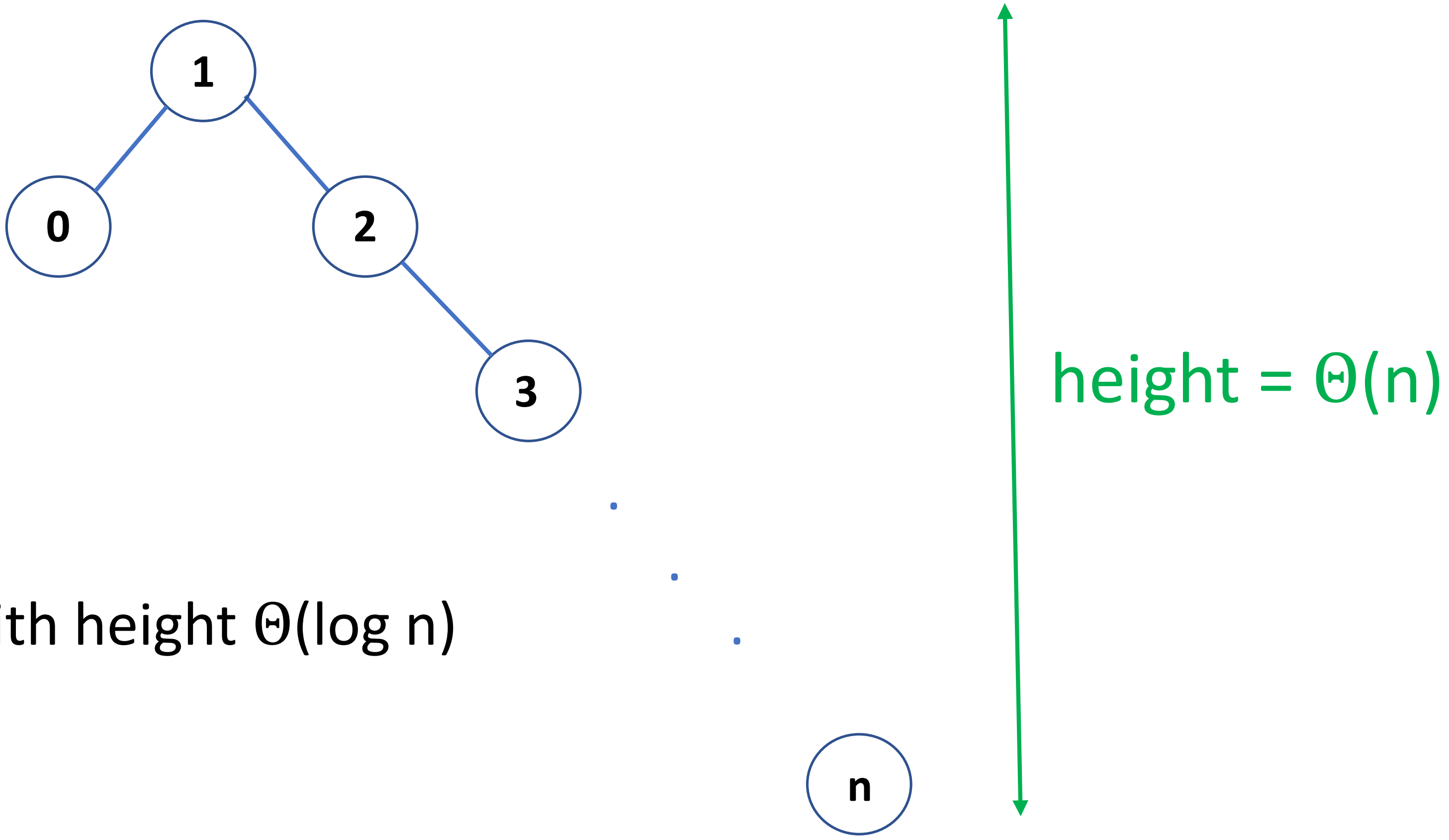
Recall the problem with BSTs…

Suppose we start with
a “balanced” BST

Insert keys 3, 4, 5, …, n

Search takes Θ(n) !

Intuitively, we want BST trees with height Θ(log n)



Recall the problem with BSTs…

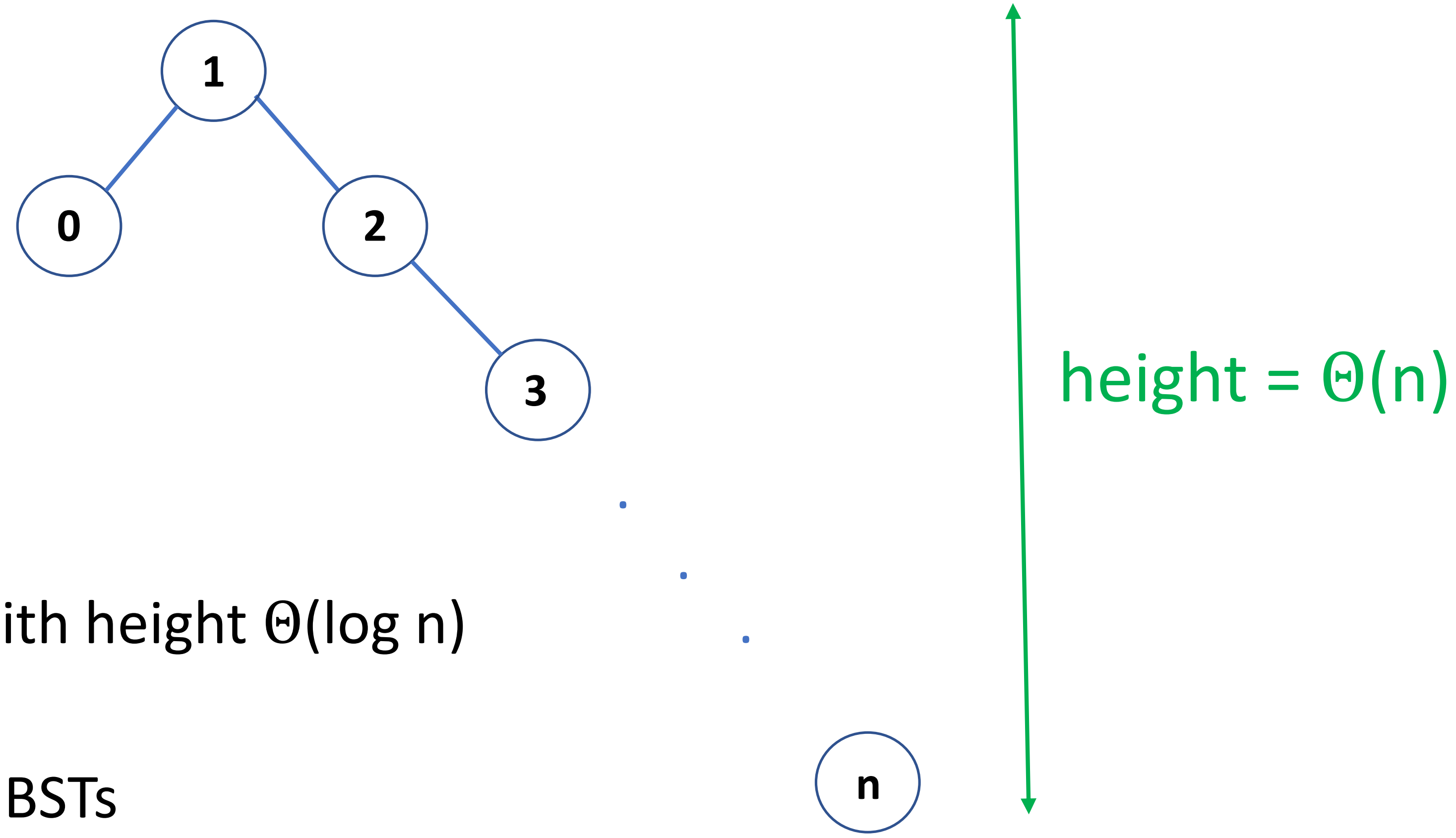
Suppose we start with
a "balanced" BST

Insert keys 3, 4, 5, ..., n

Search takes Θ(n) !

Intuitively, we want BST trees with height Θ(log n)

We achieve this using balanced BSTs



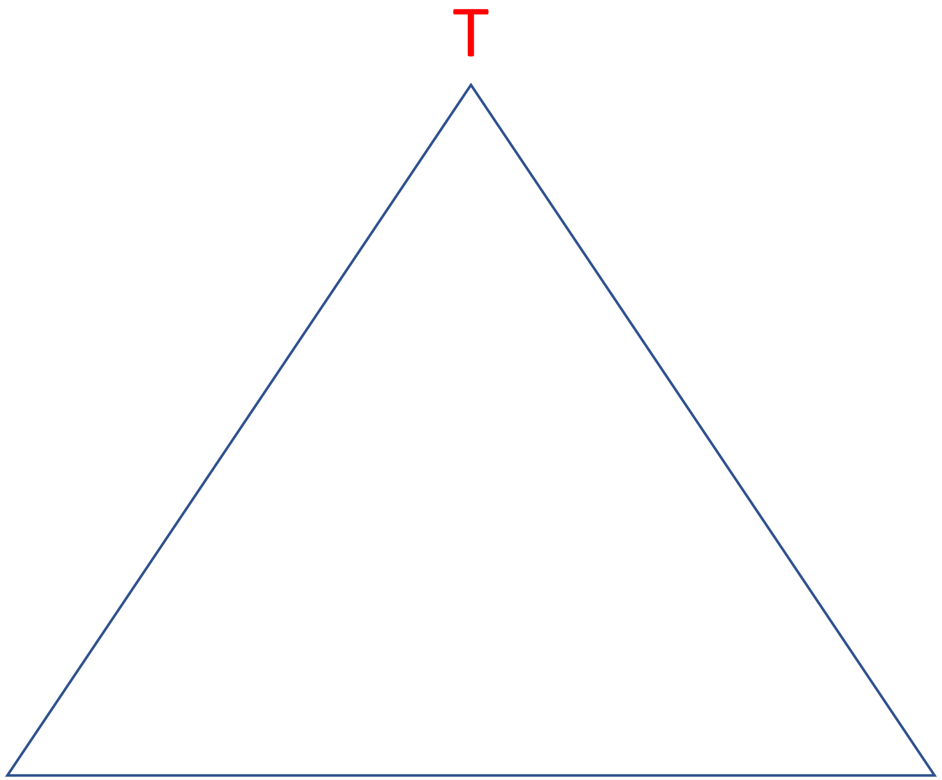
Abstract Data Type	Operations	Data Structures
Dictionary	**Search, Insert, Delete**	BSTs Balanced BSTs

Abstract Data Type	Operations	Data Structures
Dictionary	**Search, Insert, Delete**	BSTs Balanced BSTs : - 2-3 trees - Red-Black Trees - B-Trees - AVL Trees

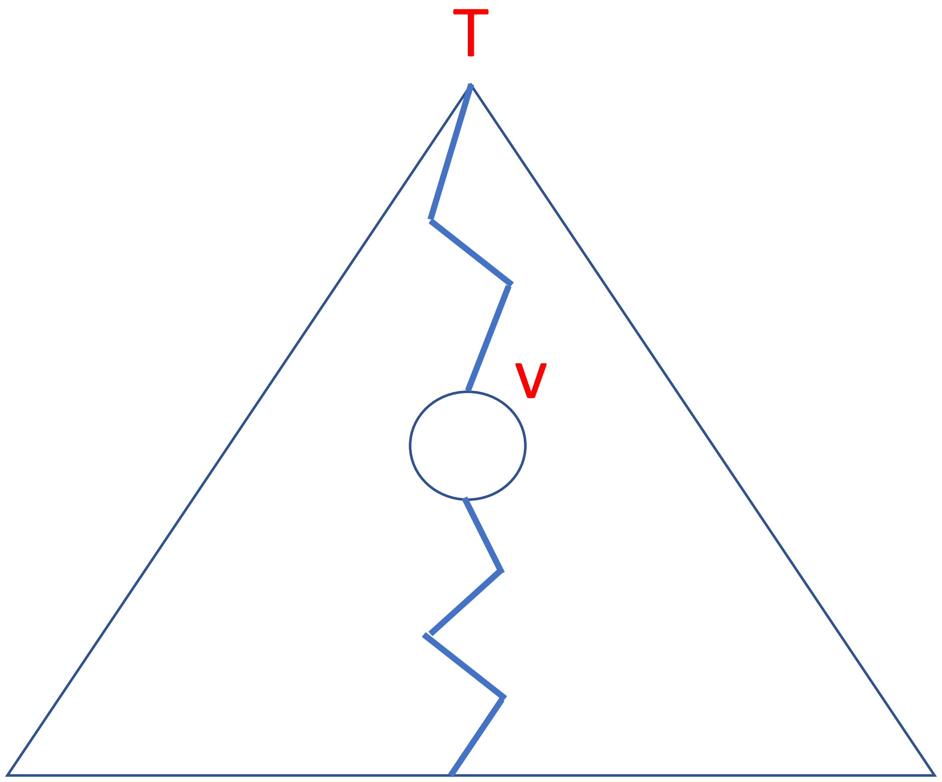
Abstract Data Type	Operations	Data Structures
Dictionary	**Search, Insert, Delete**	BSTs Balanced BSTs : - 2-3 trees - Red-Black Trees - B-Trees - AVL Trees

AVL Trees

Reminder : Height of a Node

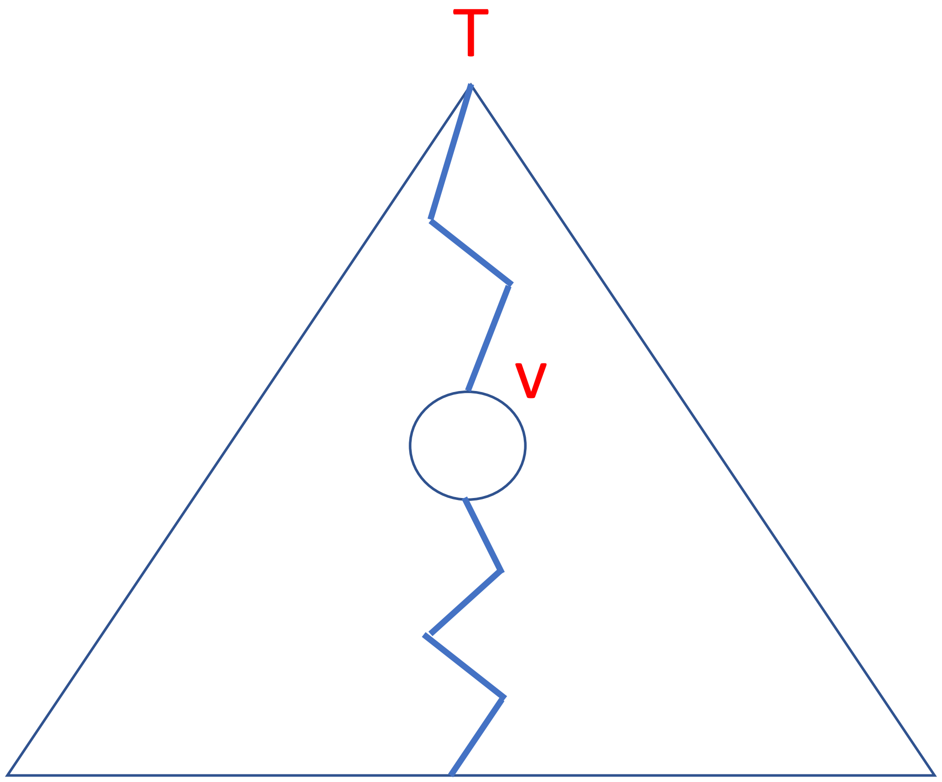


Reminder : Height of a Node



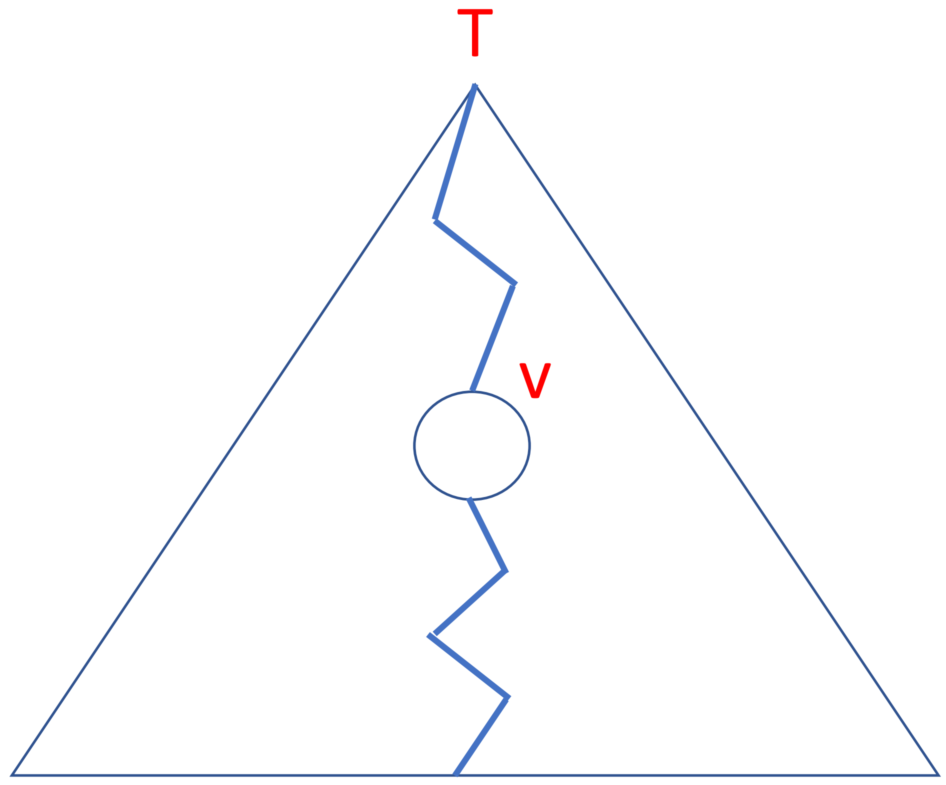
Reminder : Height of a Node

- height(v) :



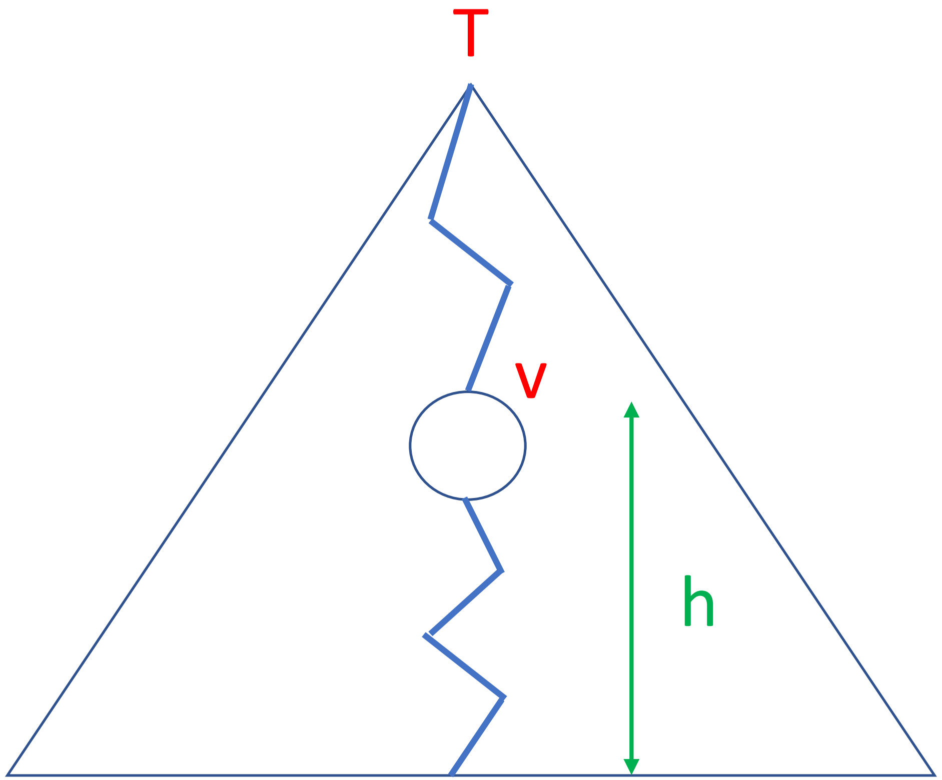
Reminder : Height of a Node

- height(v) : Number of edges in the longest path from v to a leaf



Reminder : Height of a Node

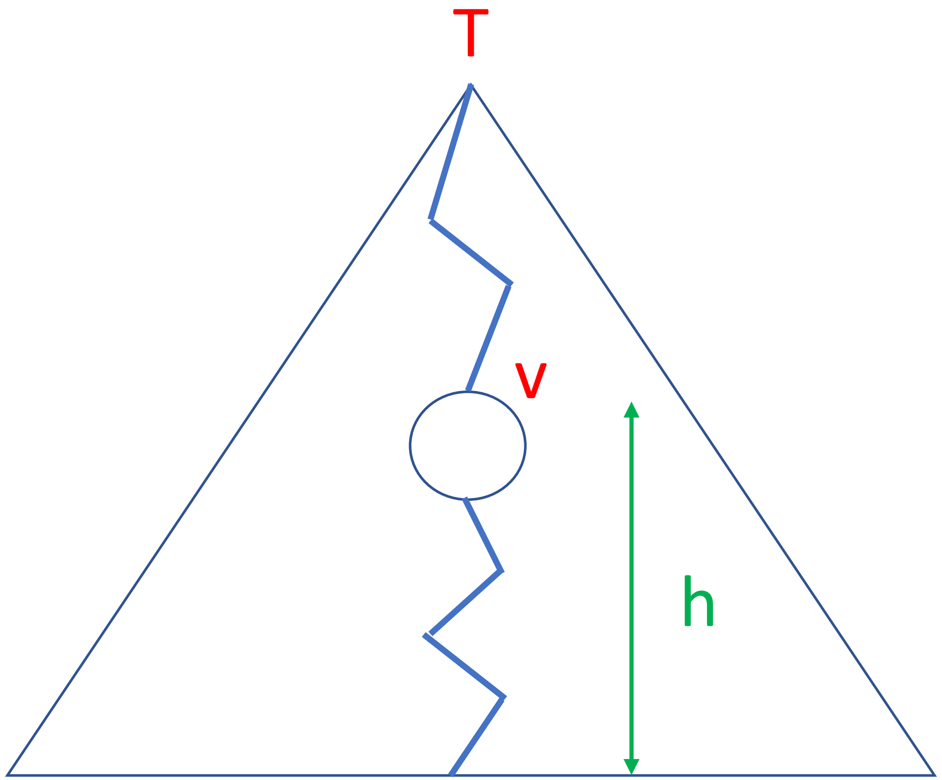
- height(v) : Number of edges in the longest path from v to a leaf



Reminder : Height of a Node

- height(v) : Number of edges in the longest path from v to a leaf

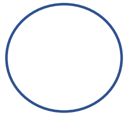
- height(T): height of the root node of T

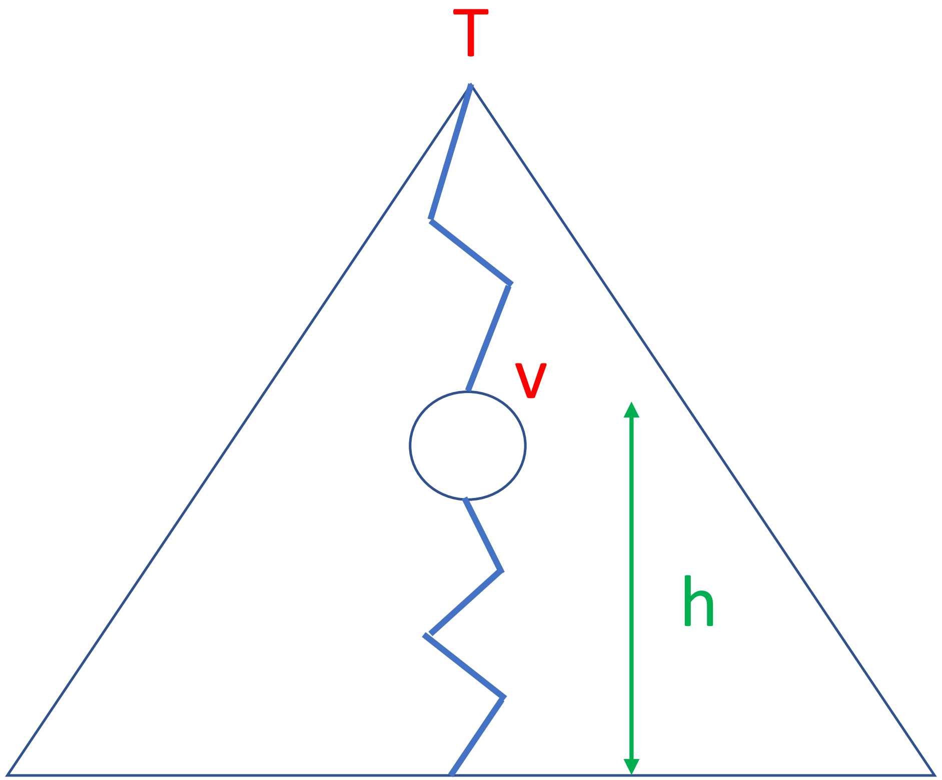


Reminder : Height of a Node

- height(v) : Number of edges in the longest path from v to a leaf

- height(T): height of the root node of T

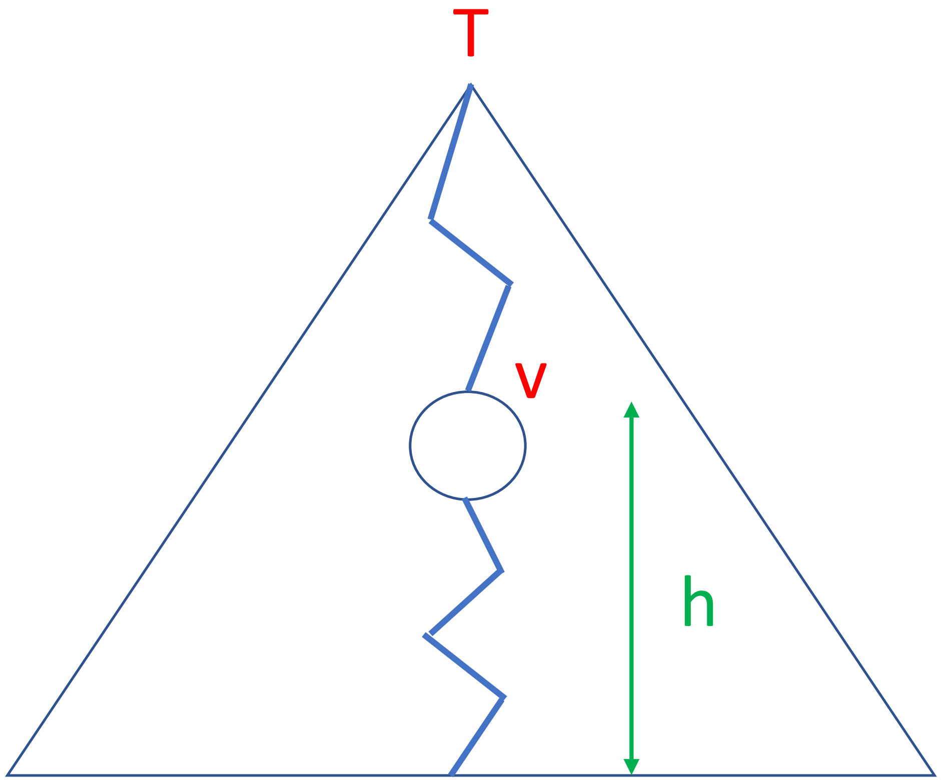
height() = 0

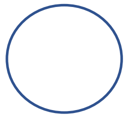


Reminder : Height of a Node

- height(v) : Number of edges in the longest path from v to a leaf

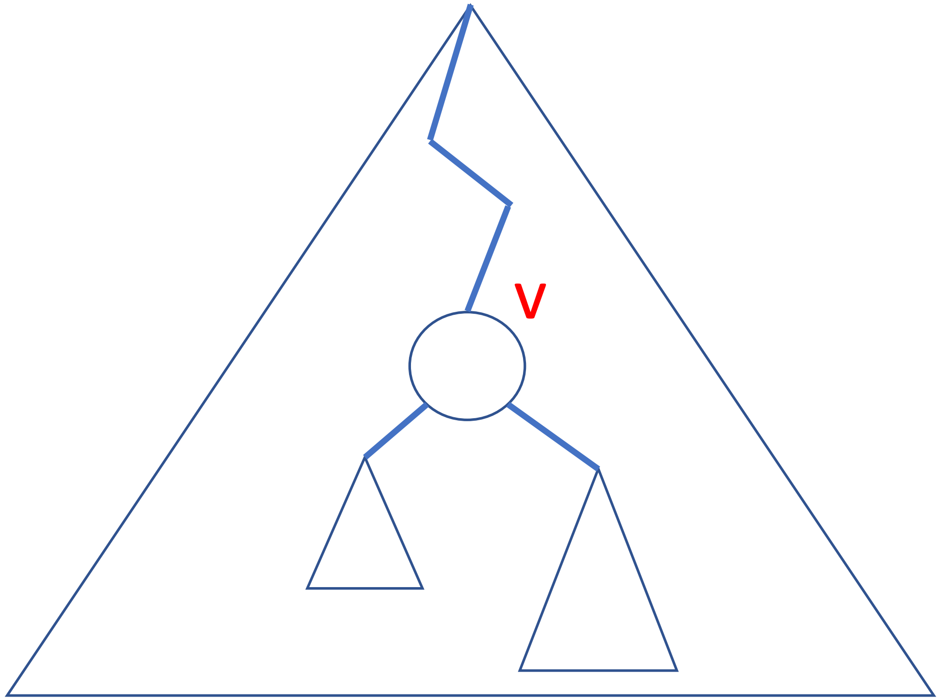
- height(T): height of the root node of T



height() = 0

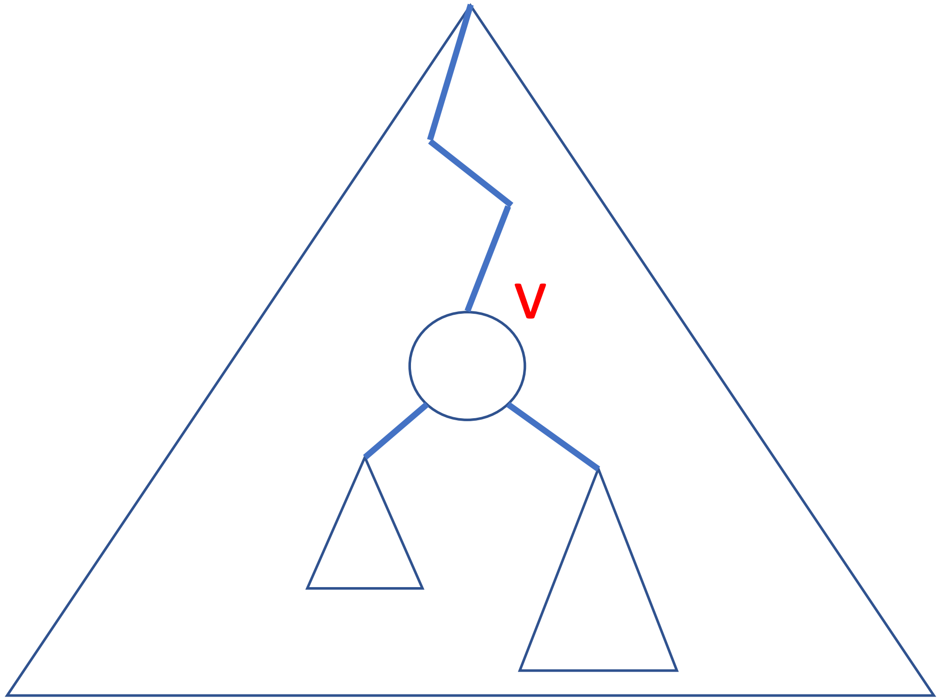
height(empty tree) = -1

Balance Factor (BF) of a node v



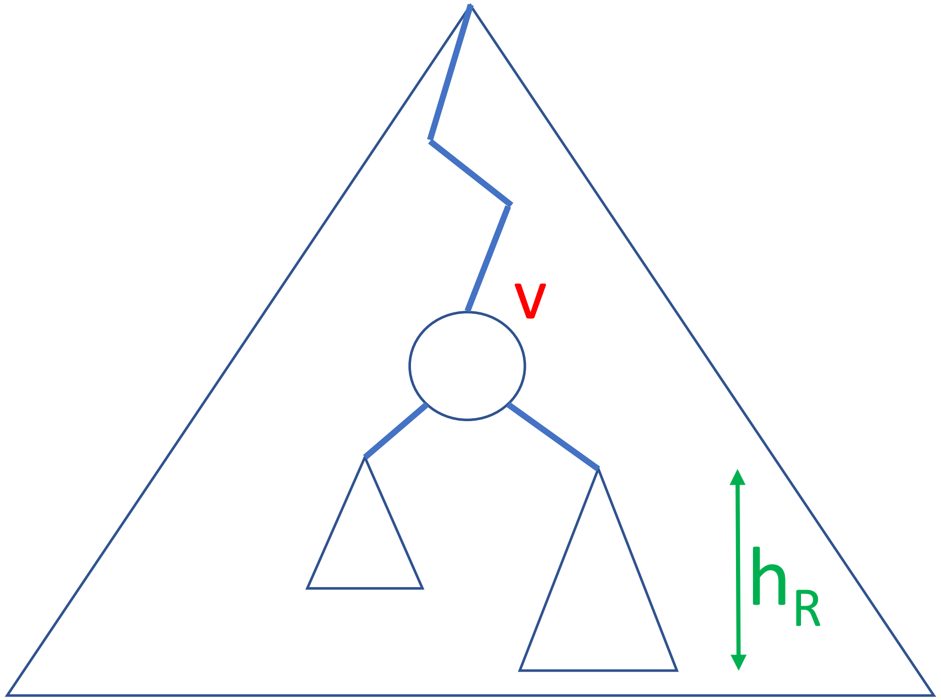
Balance Factor (BF) of a node v

BF(v) =



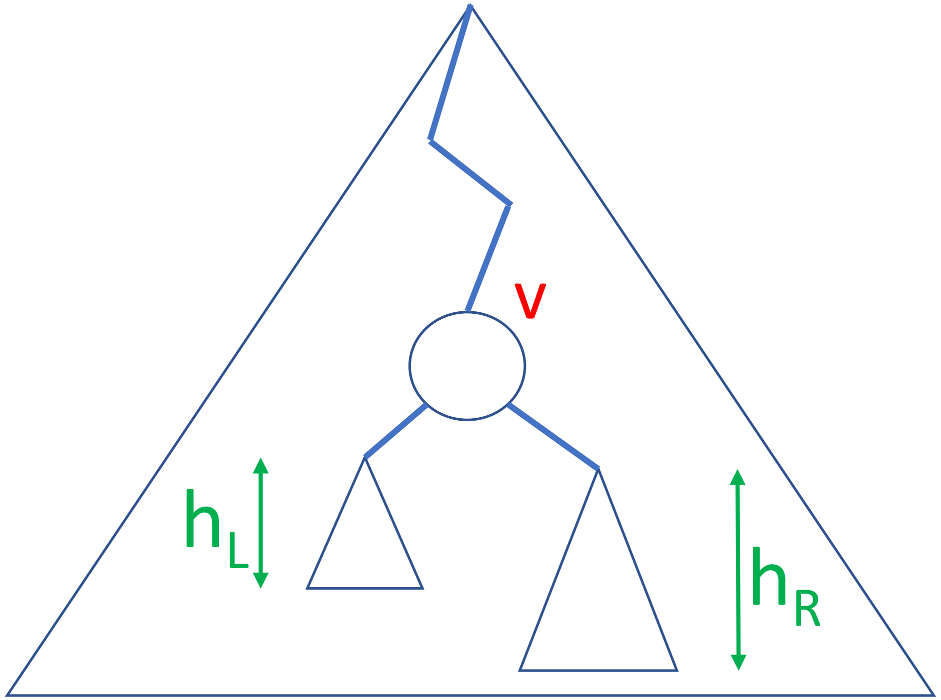
Balance Factor (BF) of a node v

BF(v) = height(right subtree of v)



Balance Factor (BF) of a node v

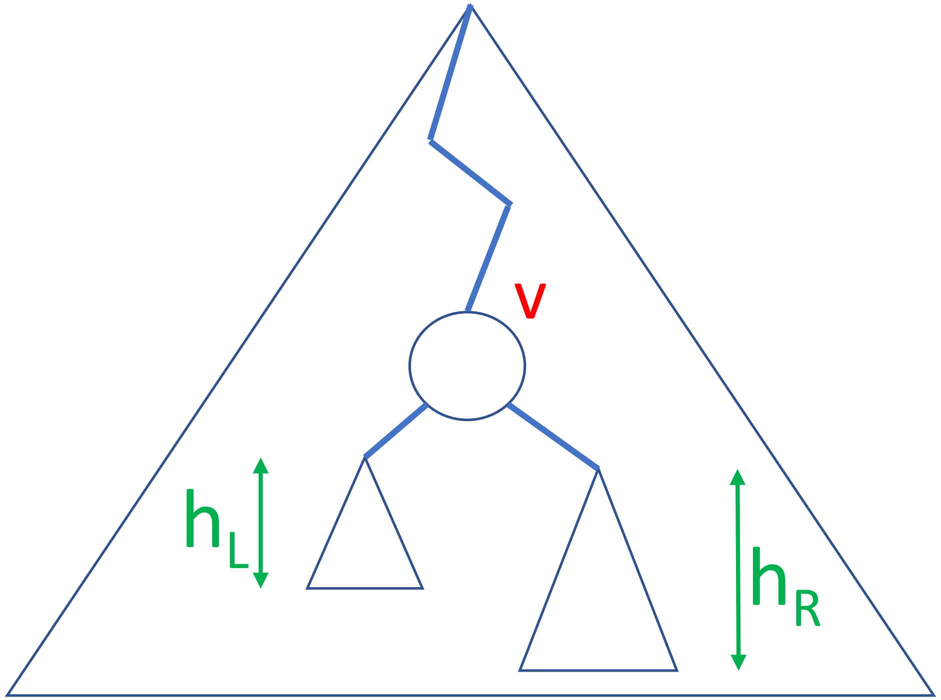
BF(v) = height(right subtree of v) – height(left subtree of v)



Balance Factor (BF) of a node v

BF(v) = height(right subtree of v) – height(left subtree of v)

BF(v) = $h_R - h_L$

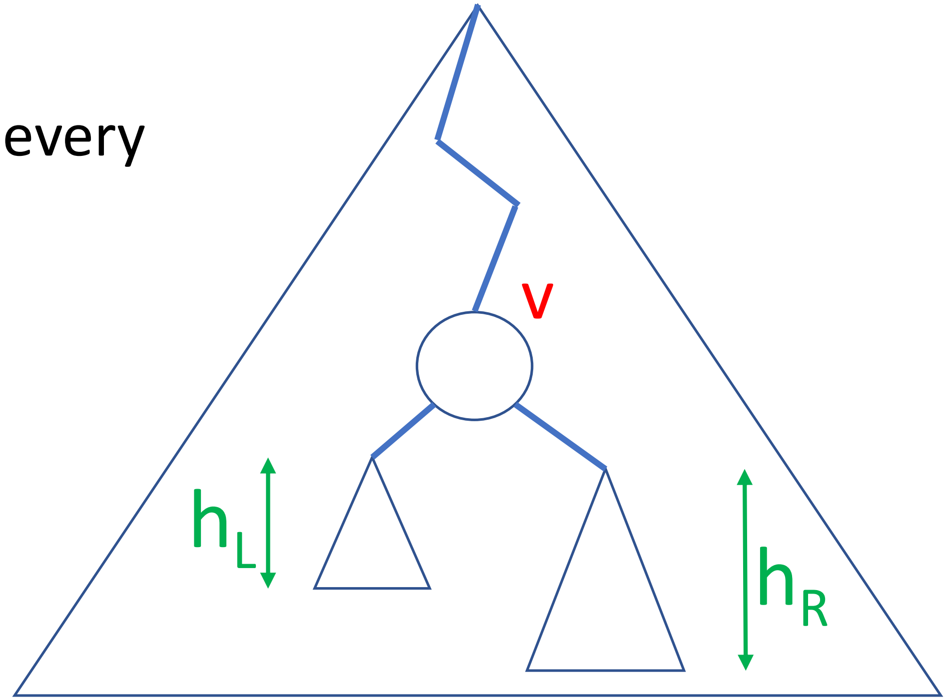


Balance Factor (BF) of a node v

BF(v) = height(right subtree of v) – height(left subtree of v)

$BF(v) = h_R - h_L$

We would like the BF of every node to be close to 0



Adelson-Velski-Landis Trees

Adelson-Velski-Landis Trees

An AVL tree T is a BST where for every node $v \in T$:

Adelson-Velski-Landis Trees

An AVL tree T is a BST where for every node $v \in T$:

$$-1 \;\leq\; BF(v) \;\leq\; +1$$

Adelson-Velski-Landis Trees

An AVL tree T is a BST where for every node $v \in T$:

$$-1 \leq BF(v) \leq +1$$

$$BF(v) = \begin{cases} +1 \\ 0 \\ -1 \end{cases}$$

Adelson-Velski-Landis Trees

An AVL tree T is a BST where for every node v ∈ T :

$$-1 \leq BF(v) \leq +1$$

$$BF(v) = \begin{cases} +1 & \text{(v is "right-heavy")} \\ 0 & \\ -1 & \end{cases}$$

Adelson-Velski-Landis Trees

An AVL tree T is a BST where for every node $v \in T$:

$$-1 \leq BF(v) \leq +1$$

$$BF(v) = \begin{cases} +1 & \text{(}v\text{ is “right-heavy”)} \\ 0 & \text{(}v\text{ is “balanced”)} \\ -1 & \end{cases}$$

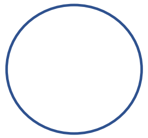
Adelson-Velski-Landis Trees

An AVL tree T is a BST where for every node $v \in T$:

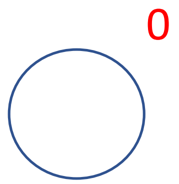
$$-1 \leq BF(v) \leq +1$$

$$BF(v) = \begin{cases} +1 & (v \text{ is "right-heavy"}) \\ 0 & (v \text{ is "balanced"}) \\ -1 & (v \text{ is "left-heavy"}) \end{cases}$$

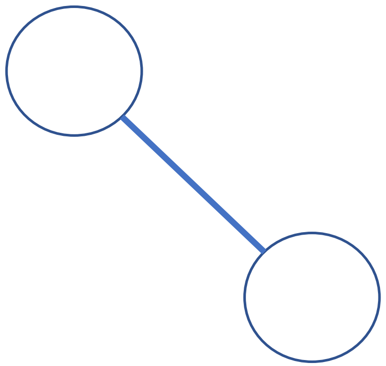
Is this an AVL tree?



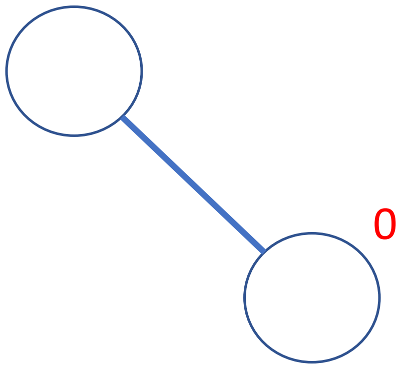
Is this an AVL tree? **Yes!**



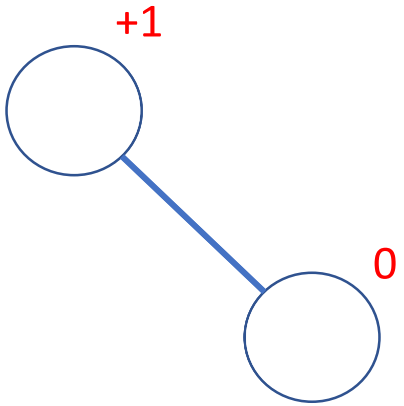
Is this an AVL tree?



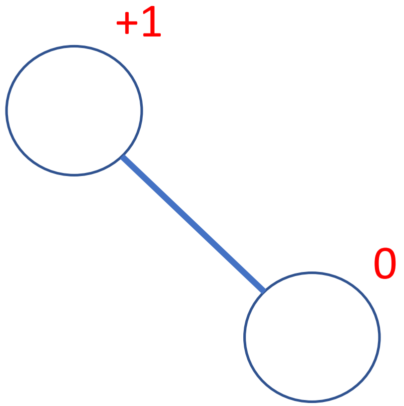
Is this an AVL tree?



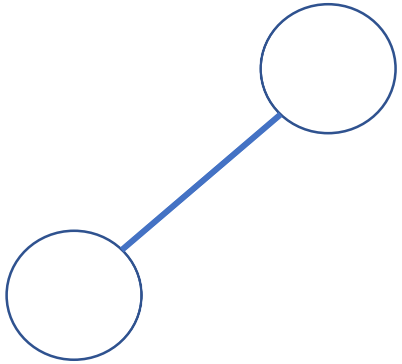
Is this an AVL tree?



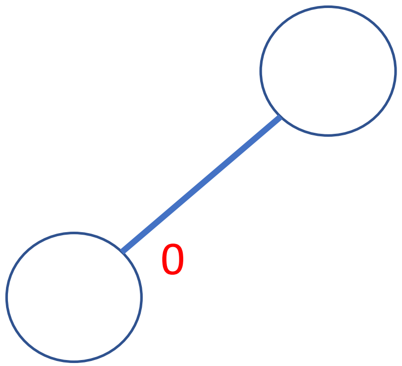
Is this an AVL tree? **Yes!**



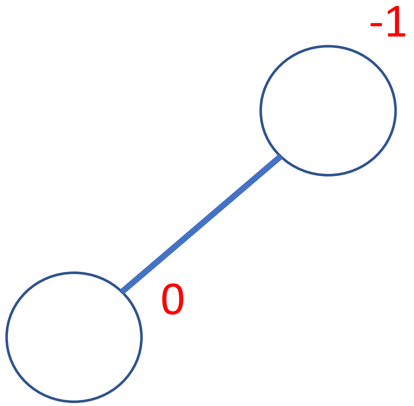
Is this an AVL tree?



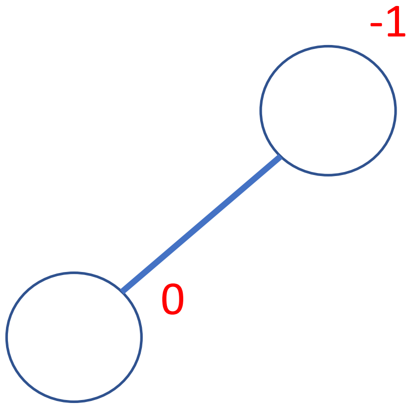
Is this an AVL tree?



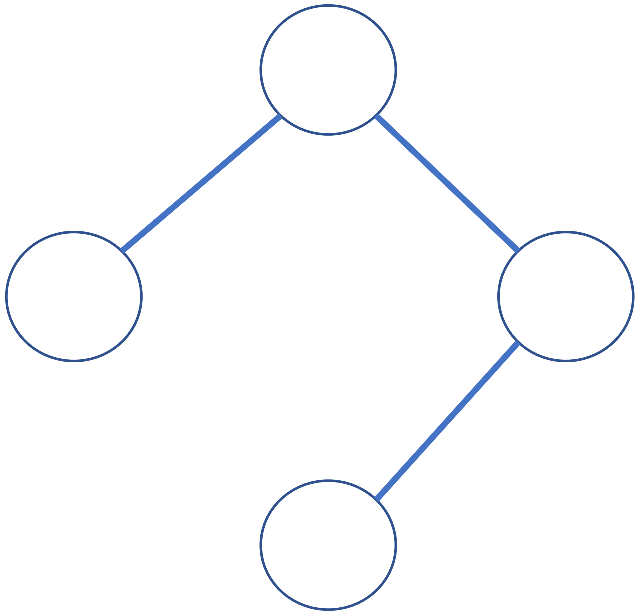
Is this an AVL tree?



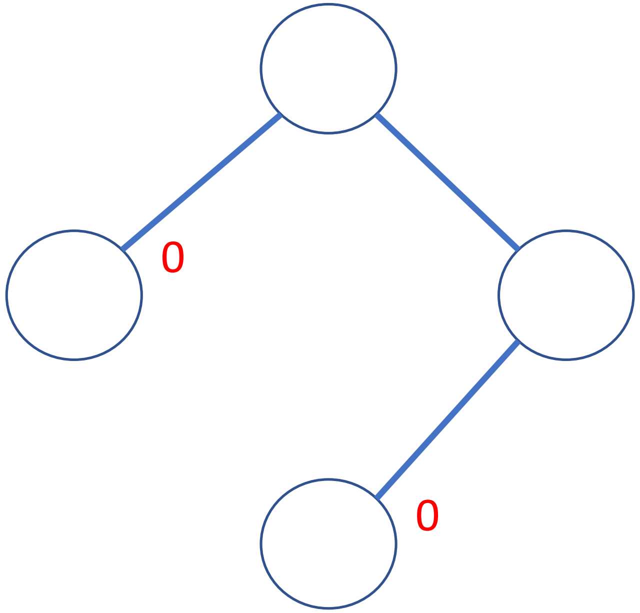
Is this an AVL tree? **Yes!**



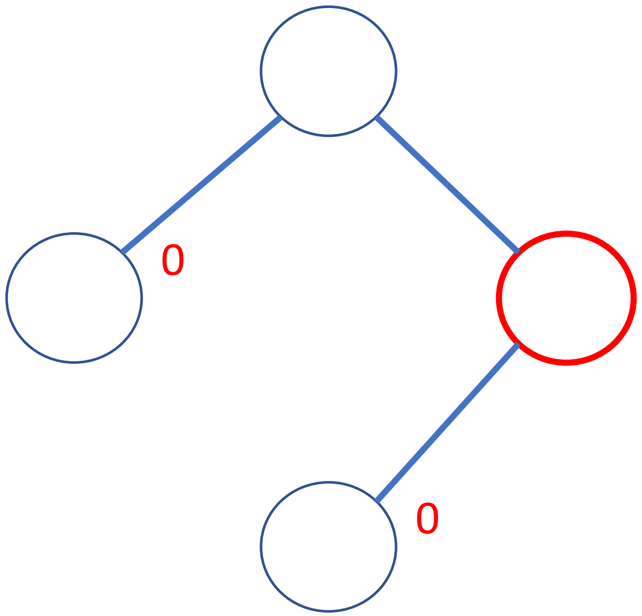
Is this an AVL tree?



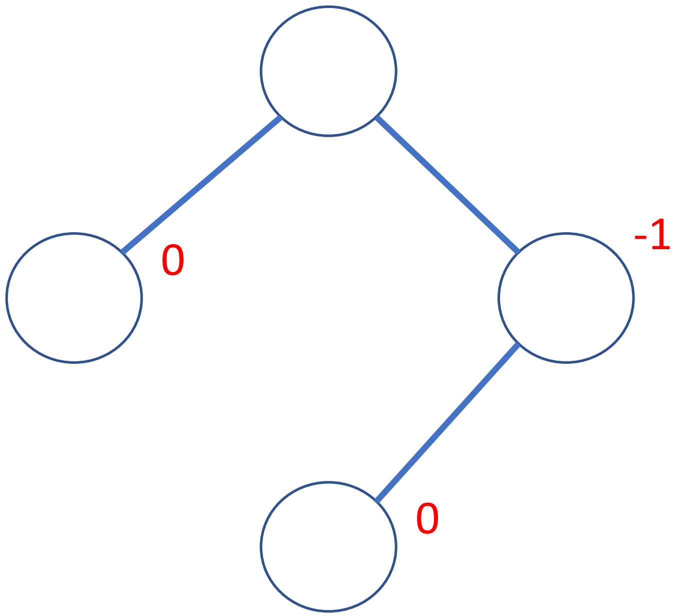
Is this an AVL tree?



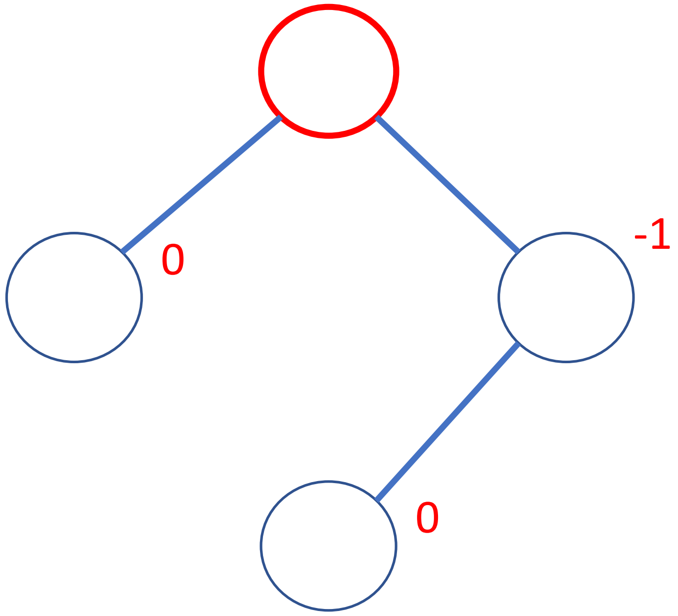
Is this an AVL tree?



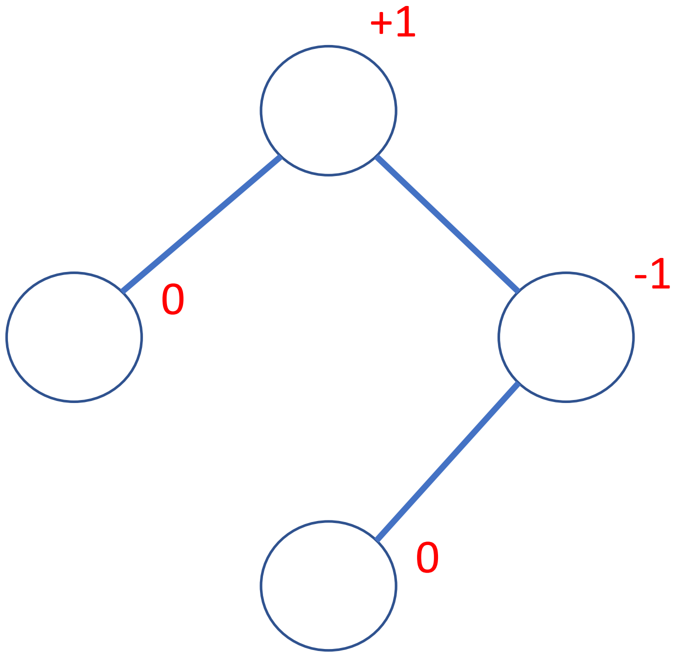
Is this an AVL tree?



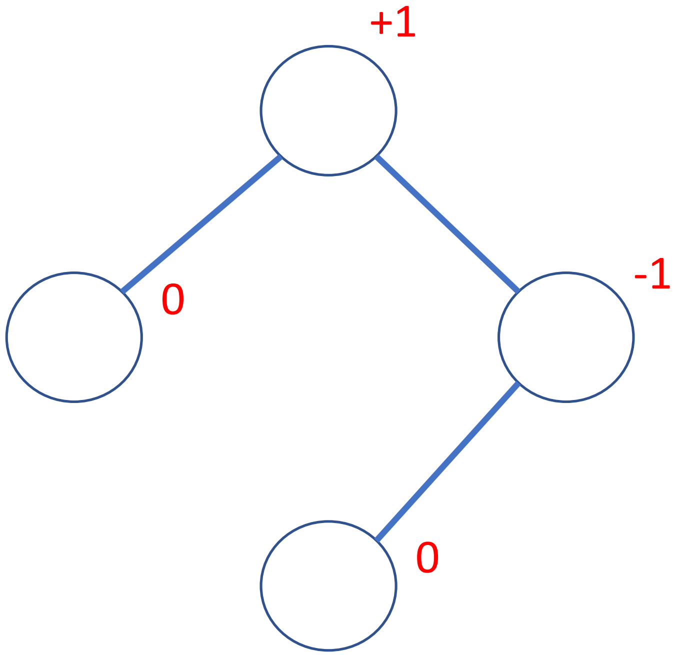
Is this an AVL tree?



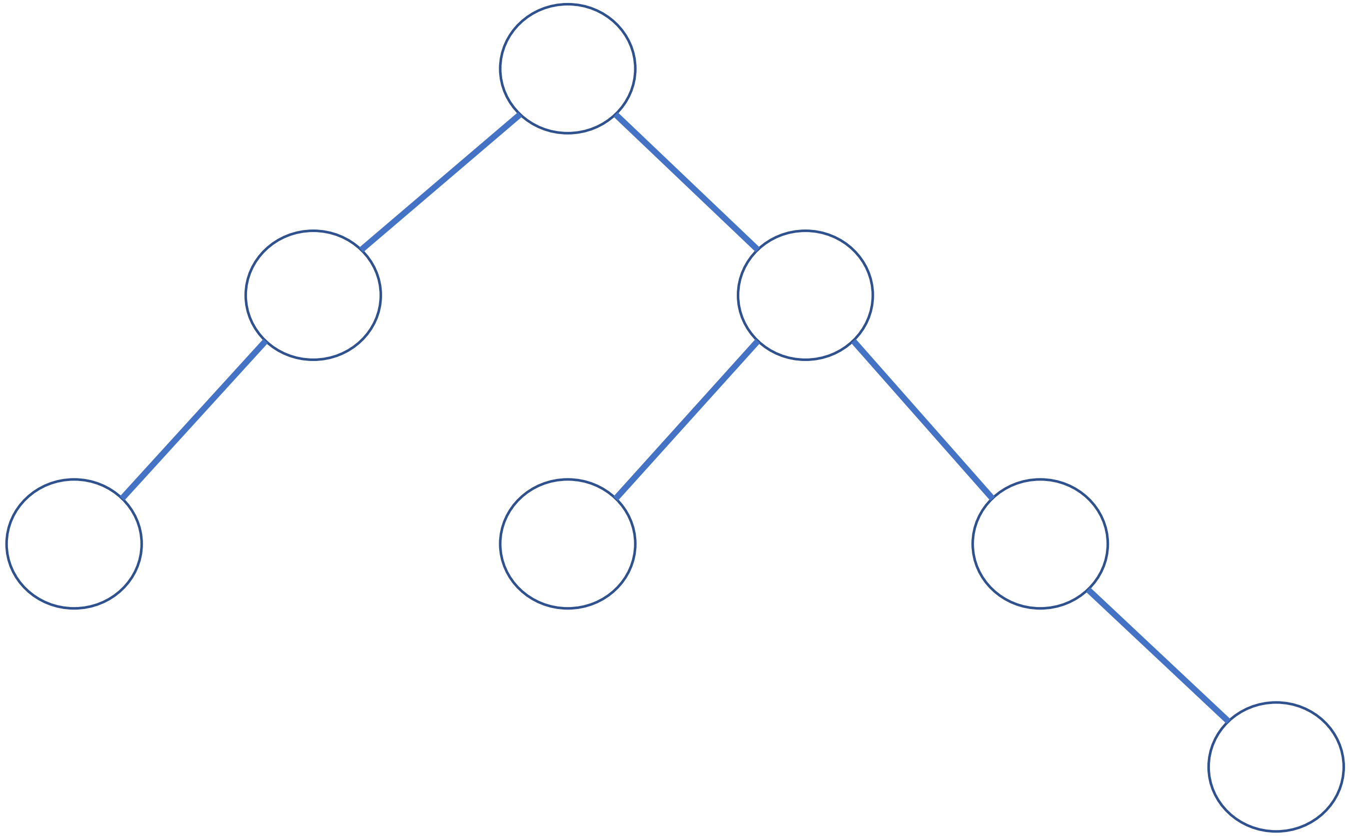
Is this an AVL tree?



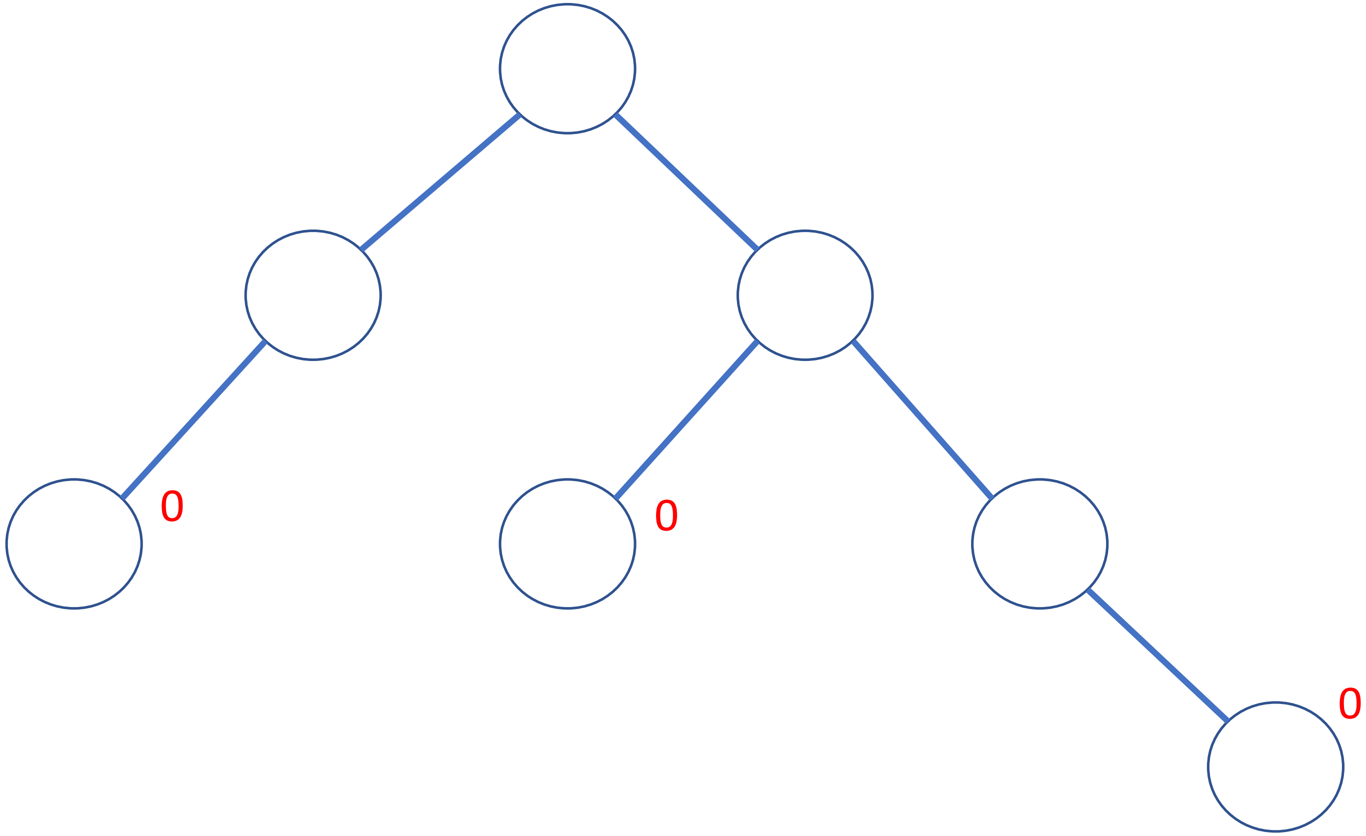
Is this an AVL tree? **Yes!**



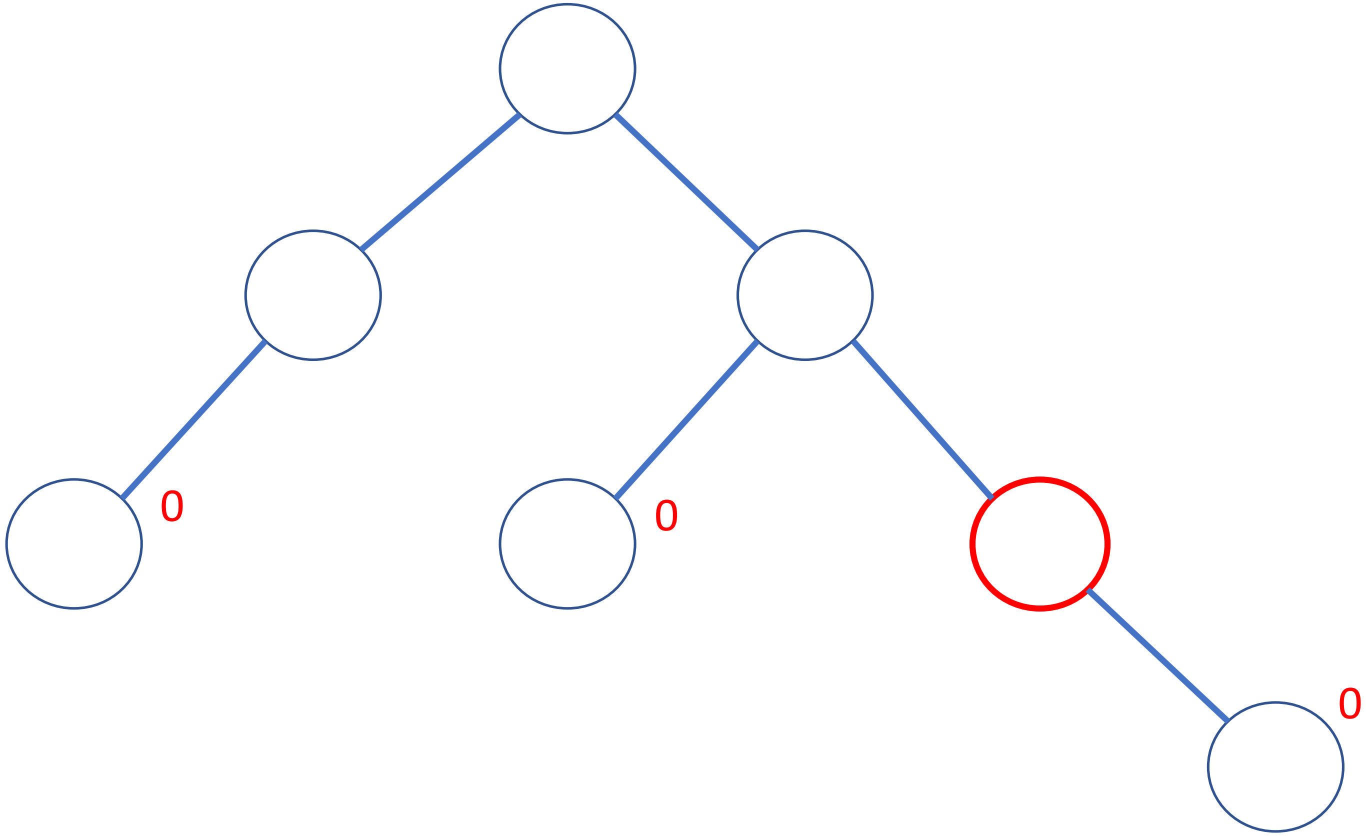
Is this an AVL tree?



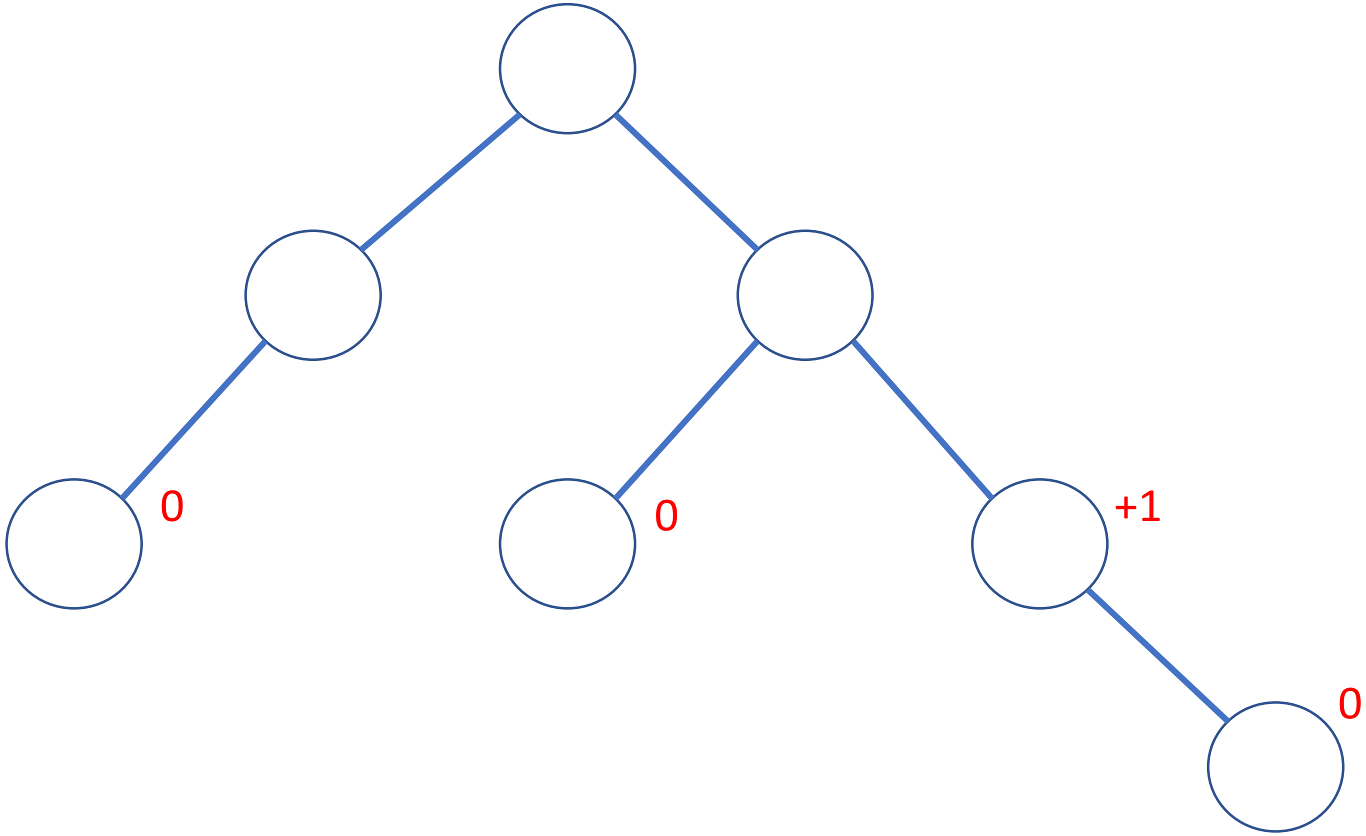
Is this an AVL tree?



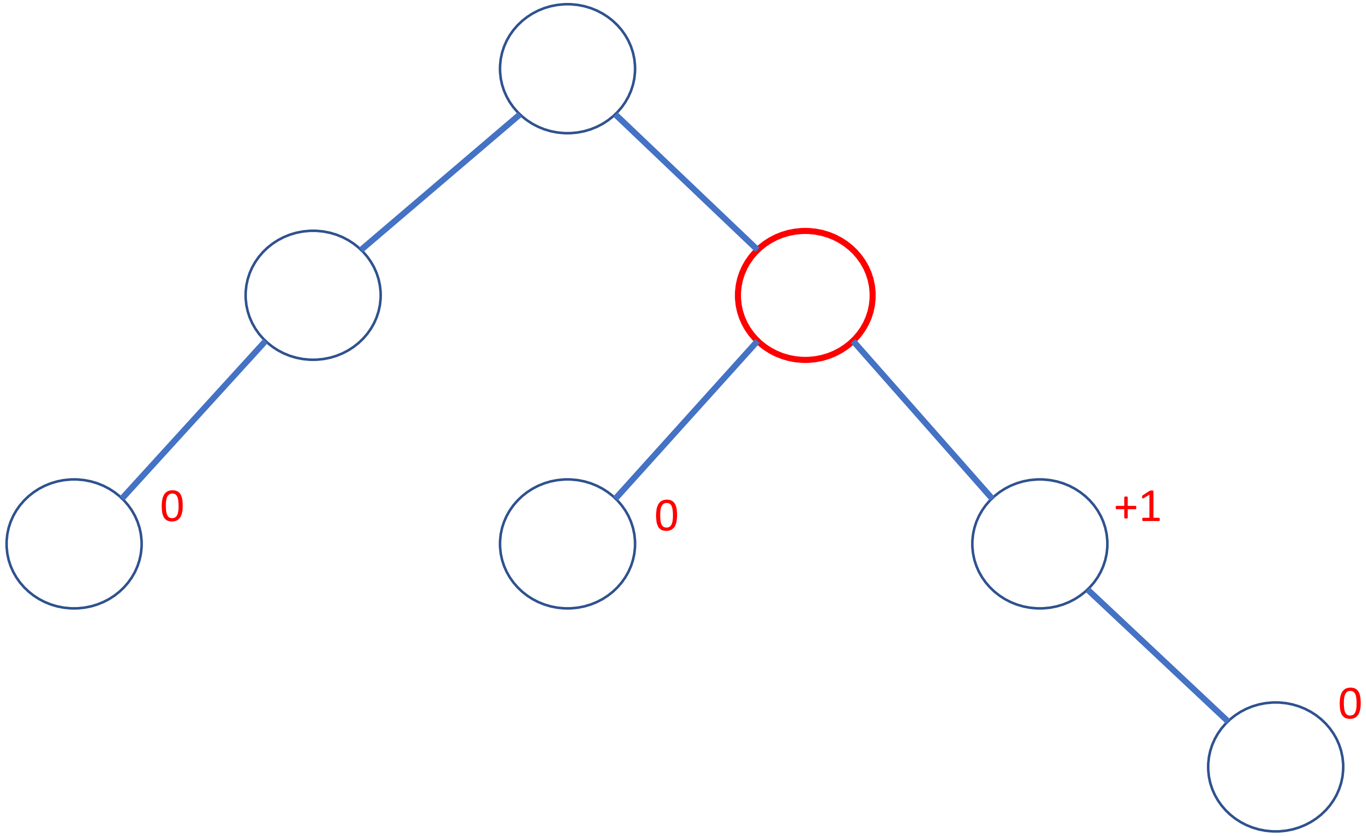
Is this an AVL tree?



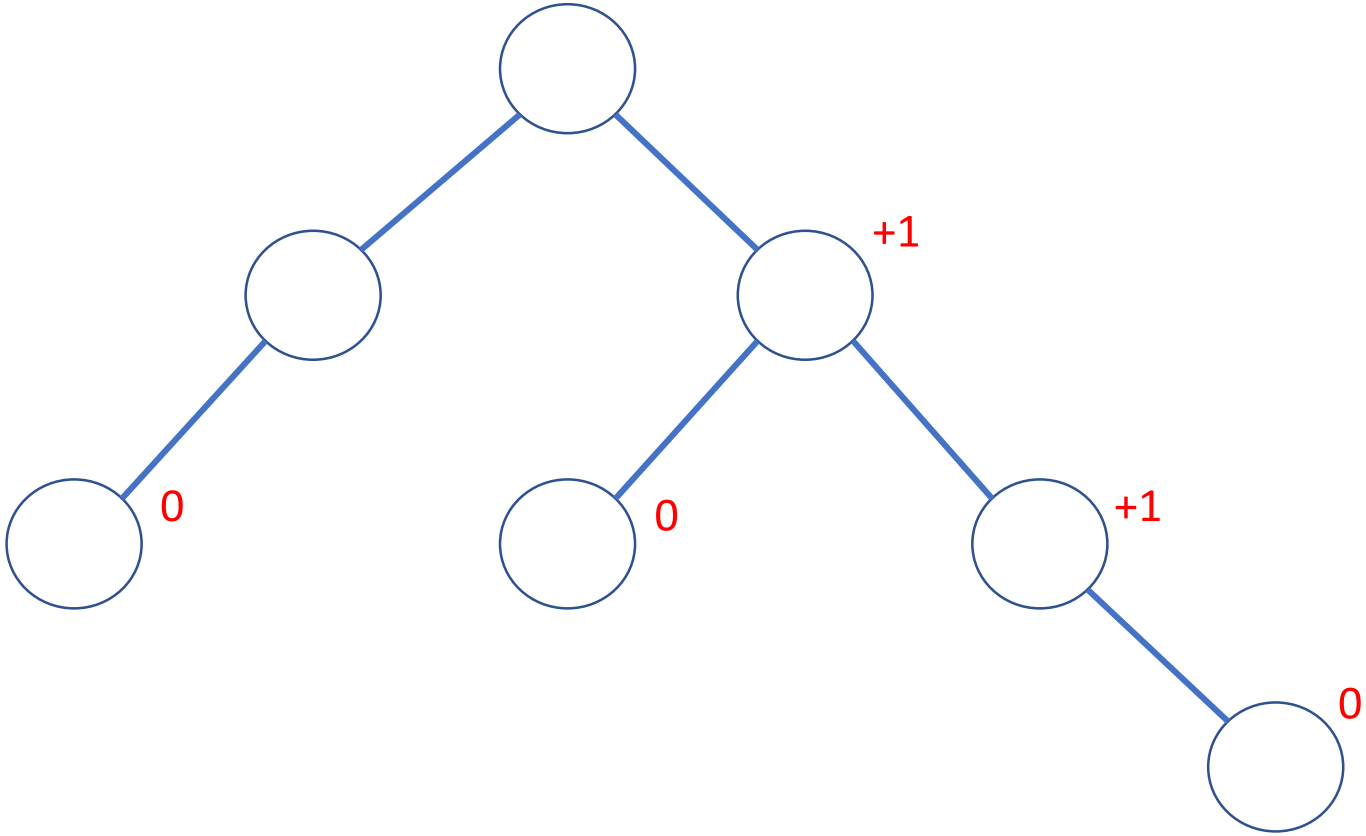
Is this an AVL tree?



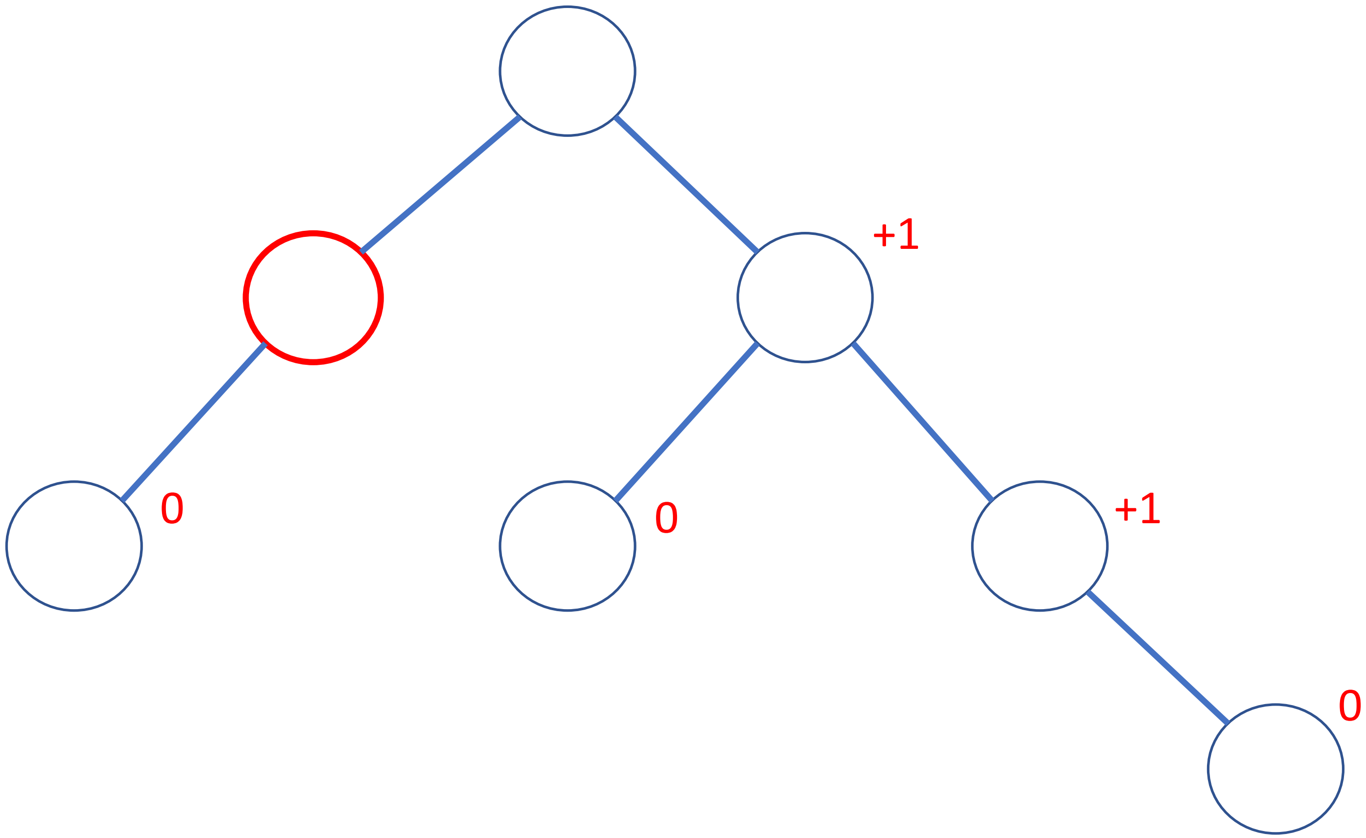
Is this an AVL tree?



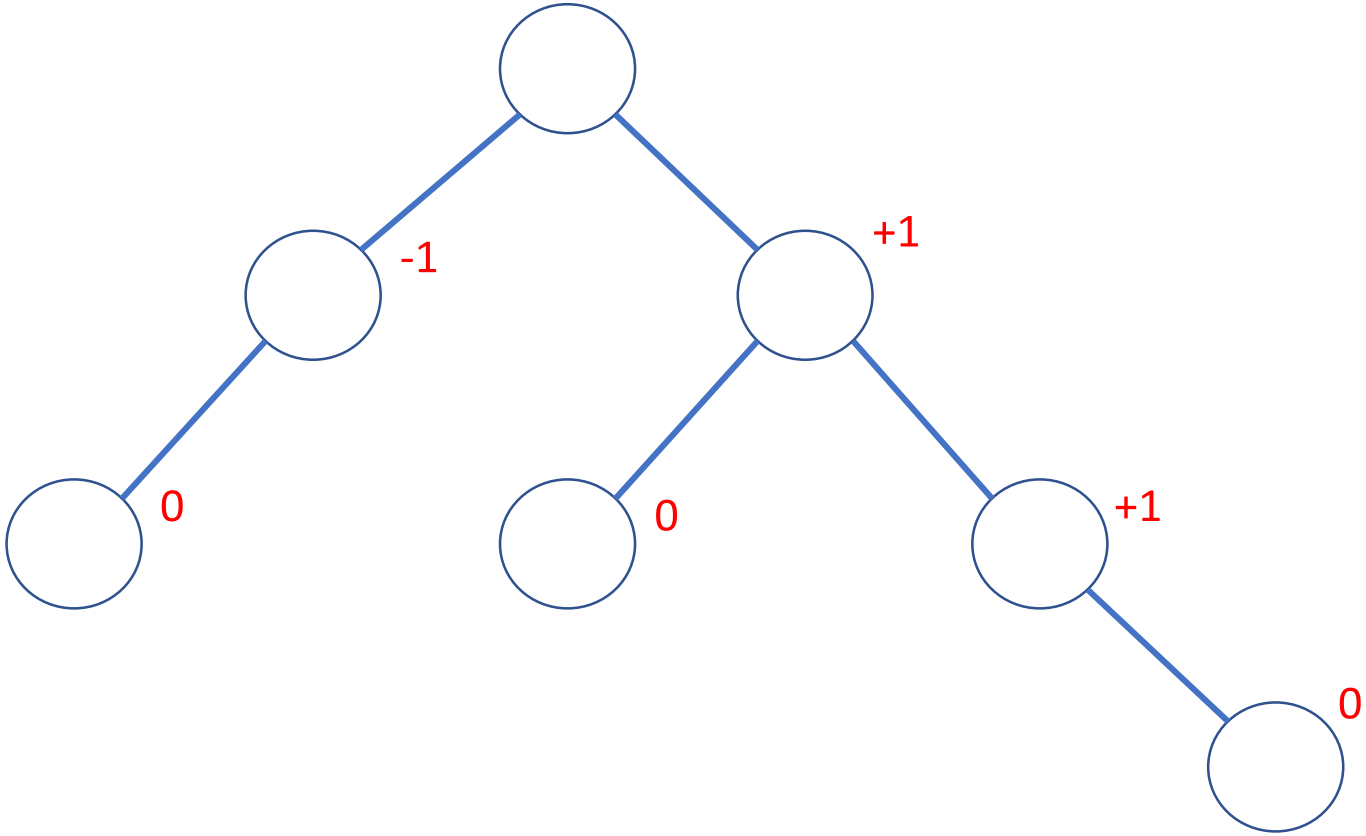
Is this an AVL tree?



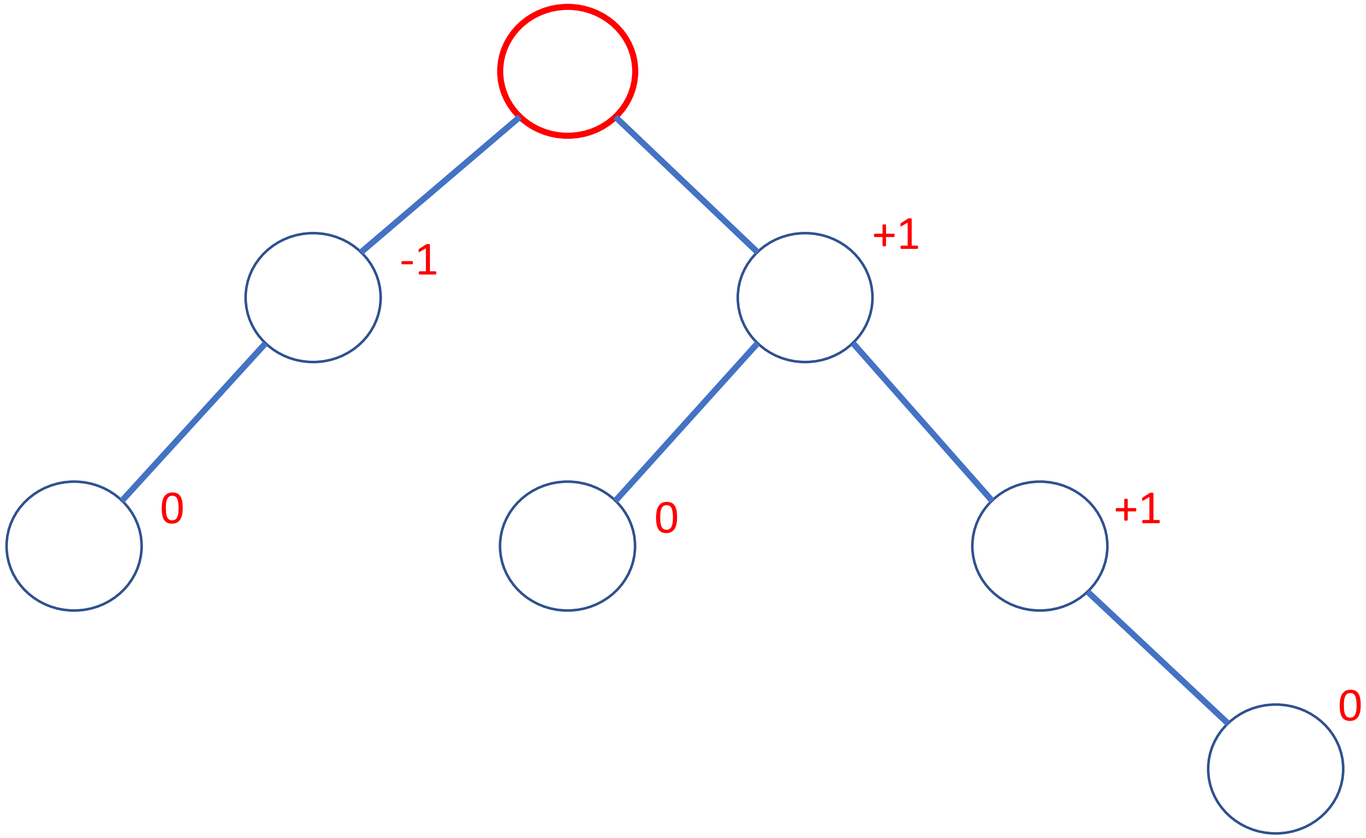
Is this an AVL tree?



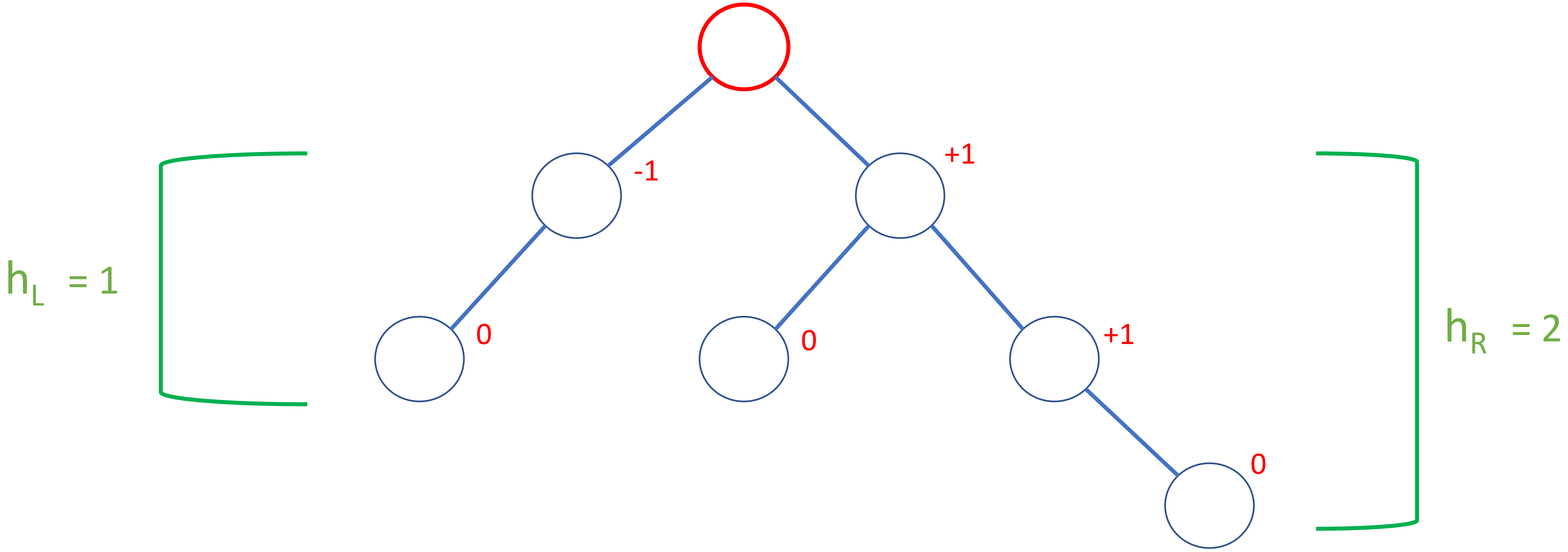
Is this an AVL tree?



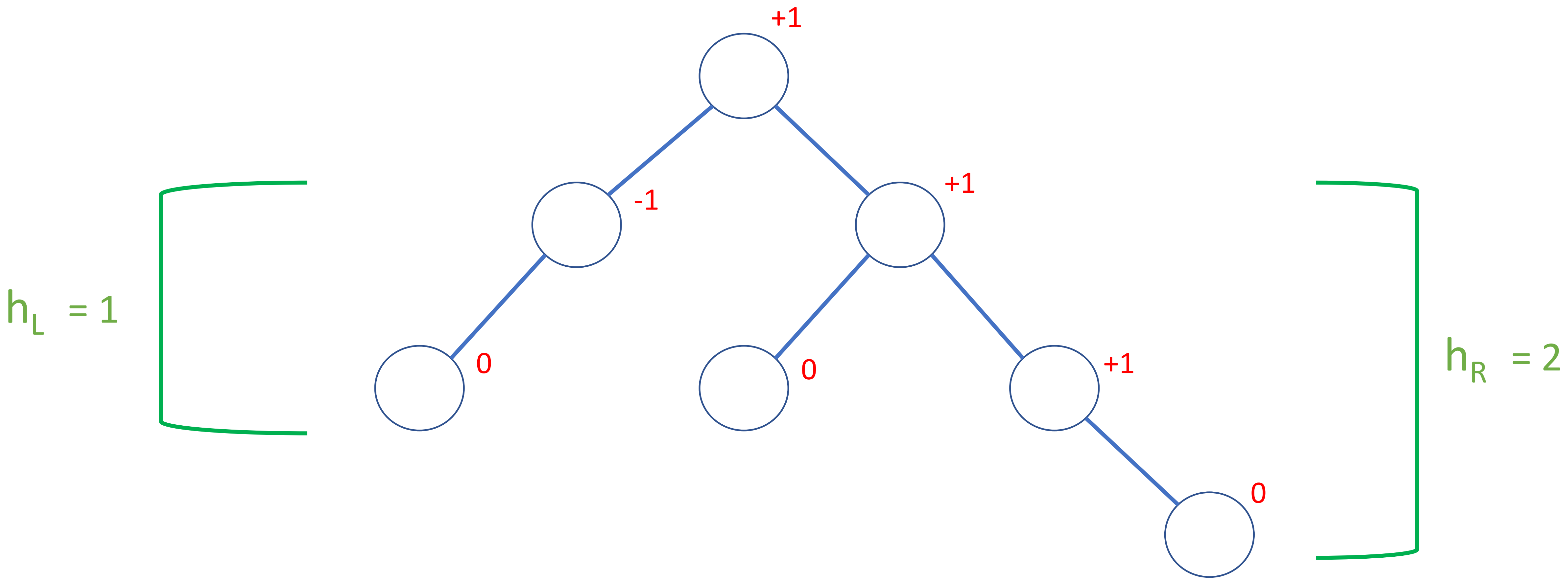
Is this an AVL tree?



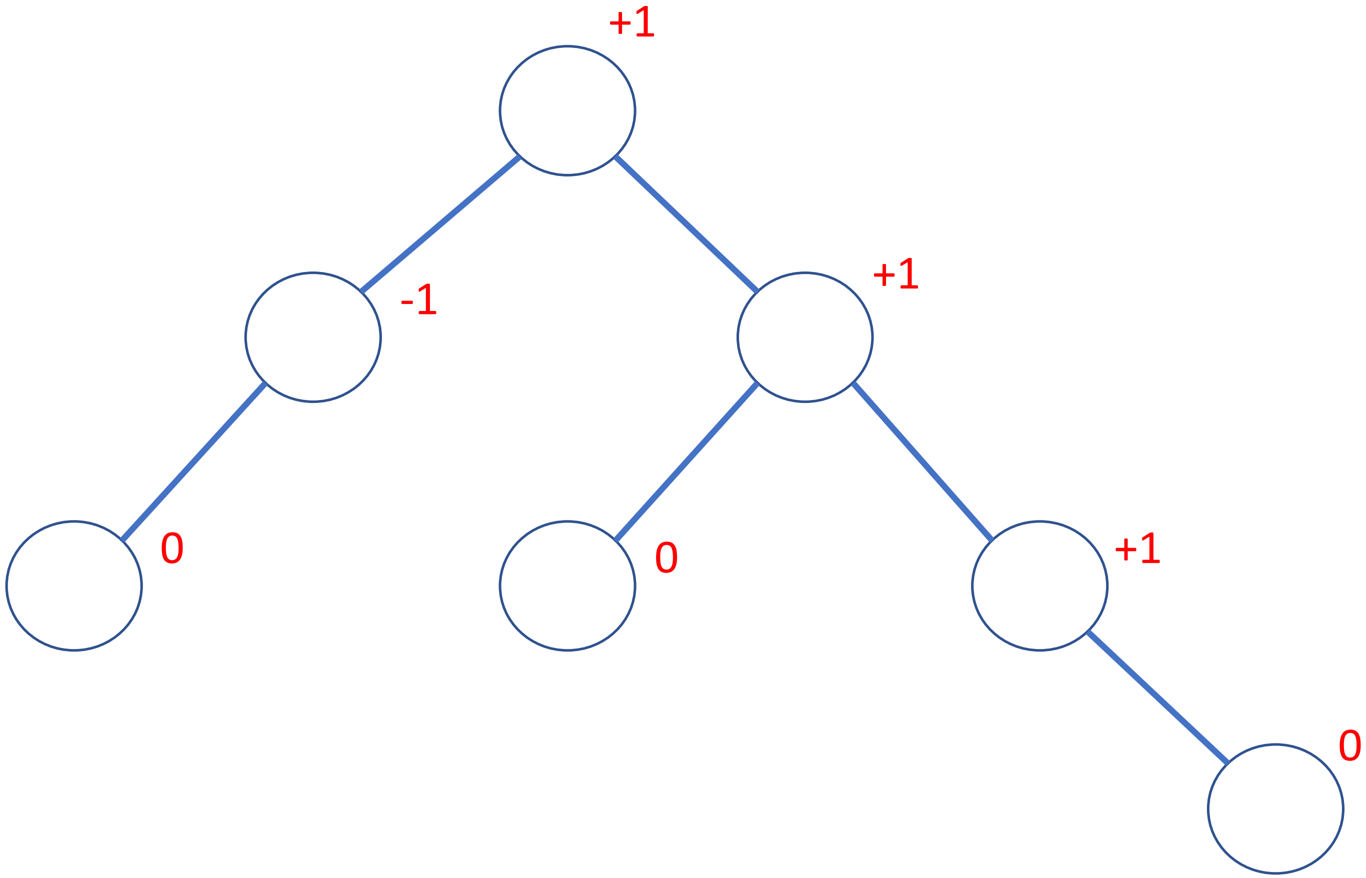
Is this an AVL tree?



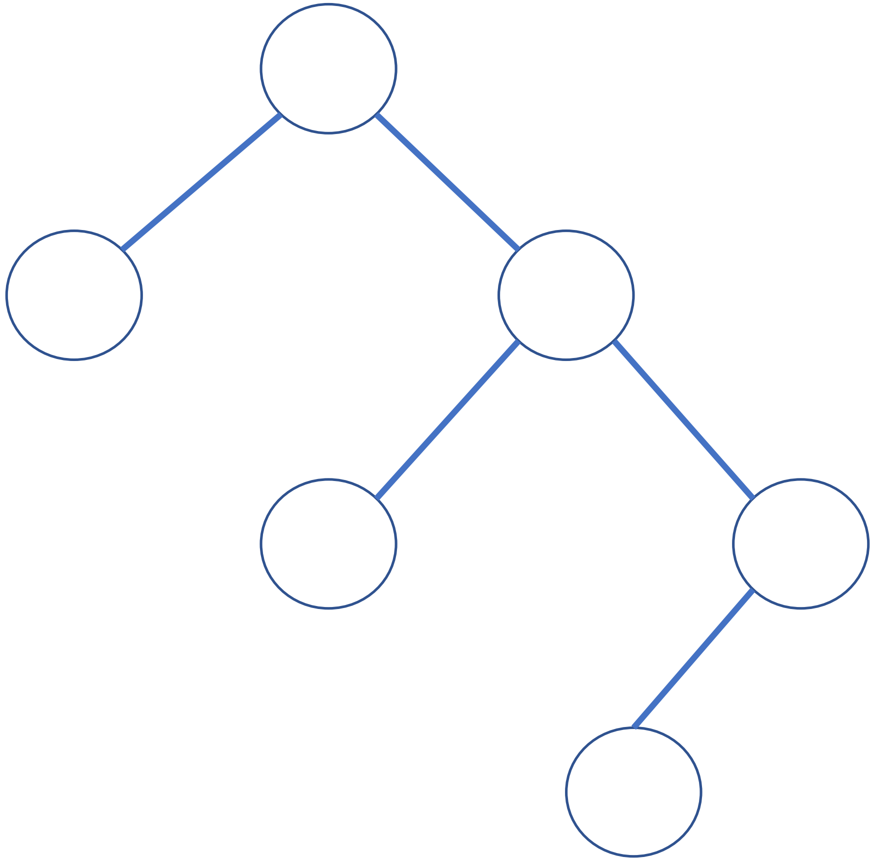
Is this an AVL tree?



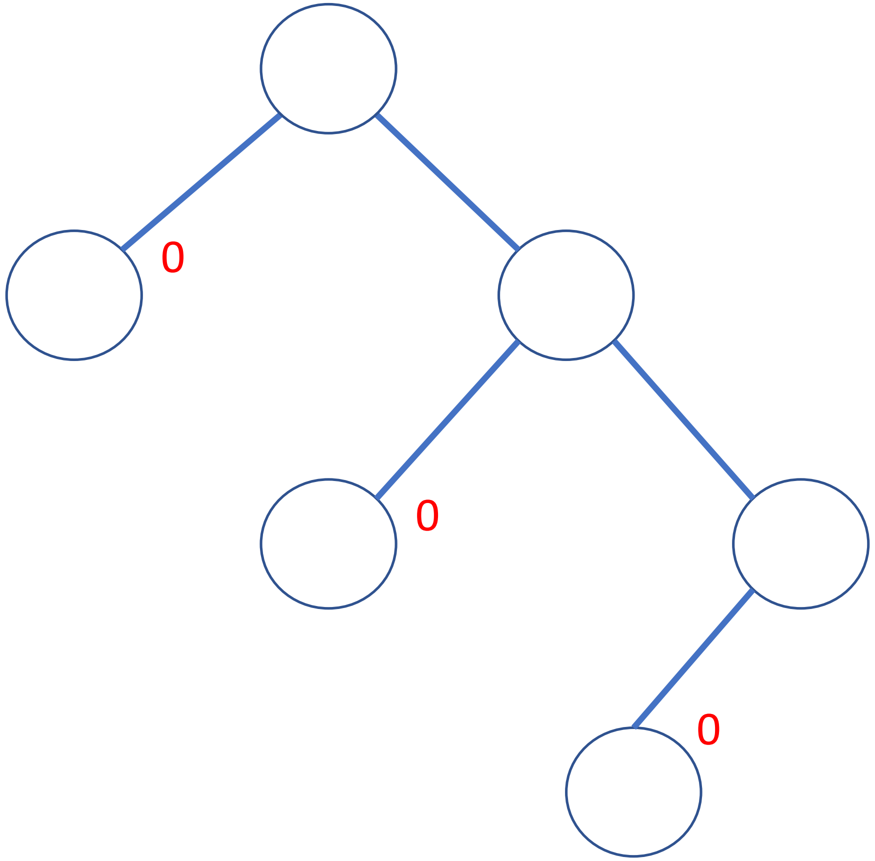
Is this an AVL tree? **Yes!**



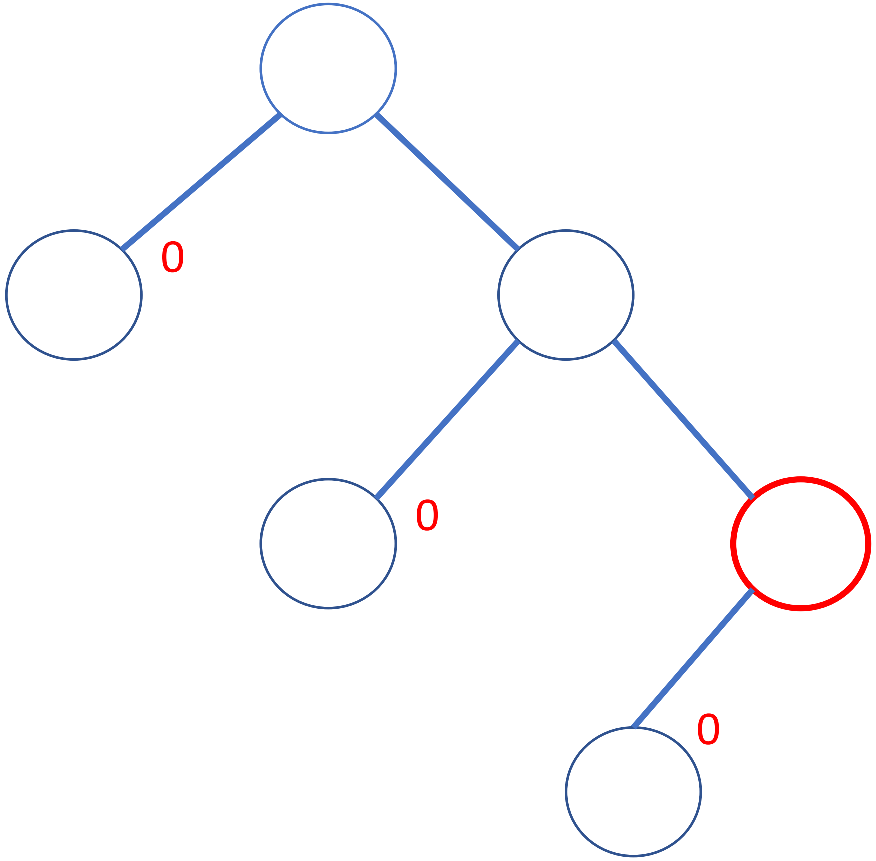
Is this an AVL tree?



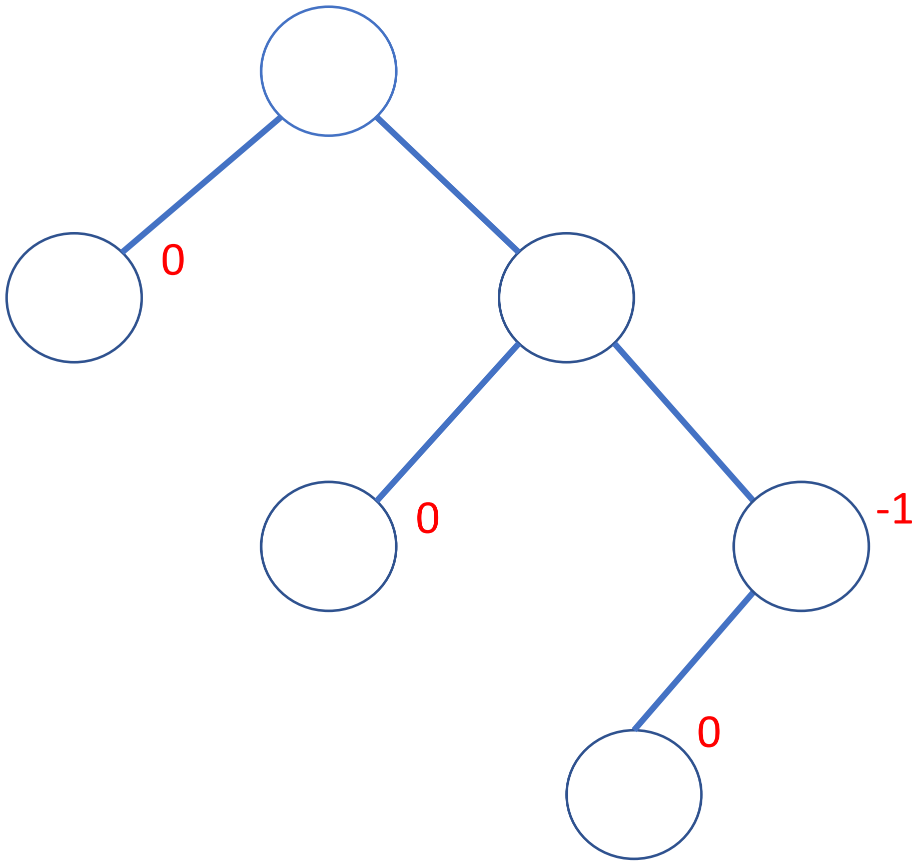
Is this an AVL tree?



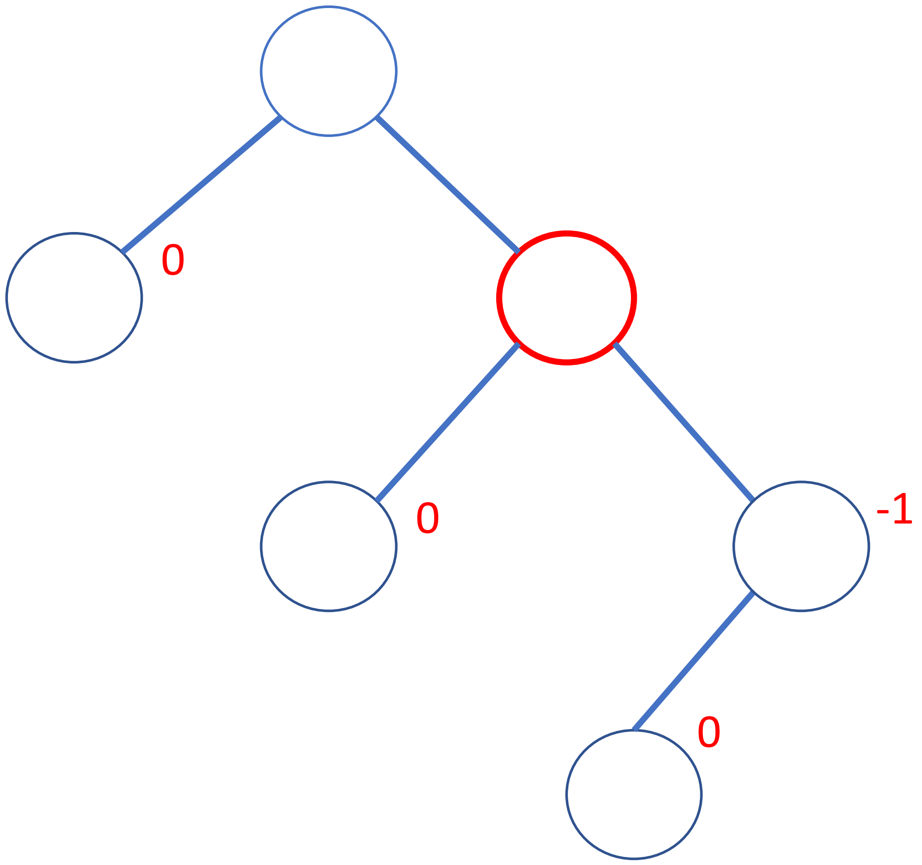
Is this an AVL tree?



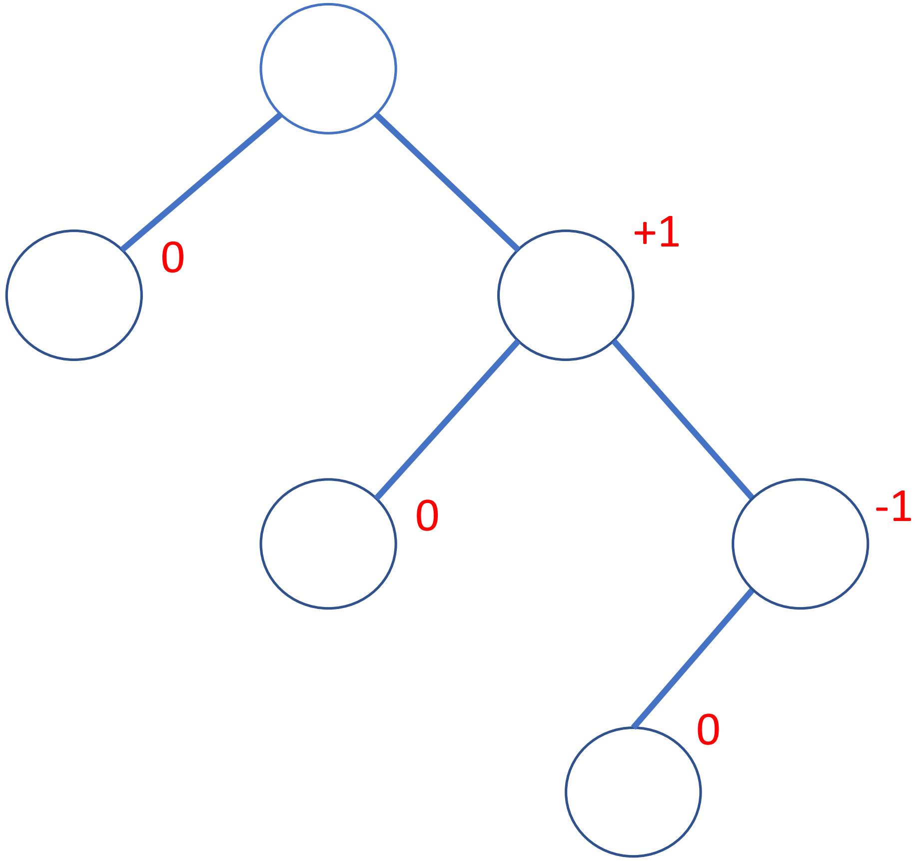
Is this an AVL tree?



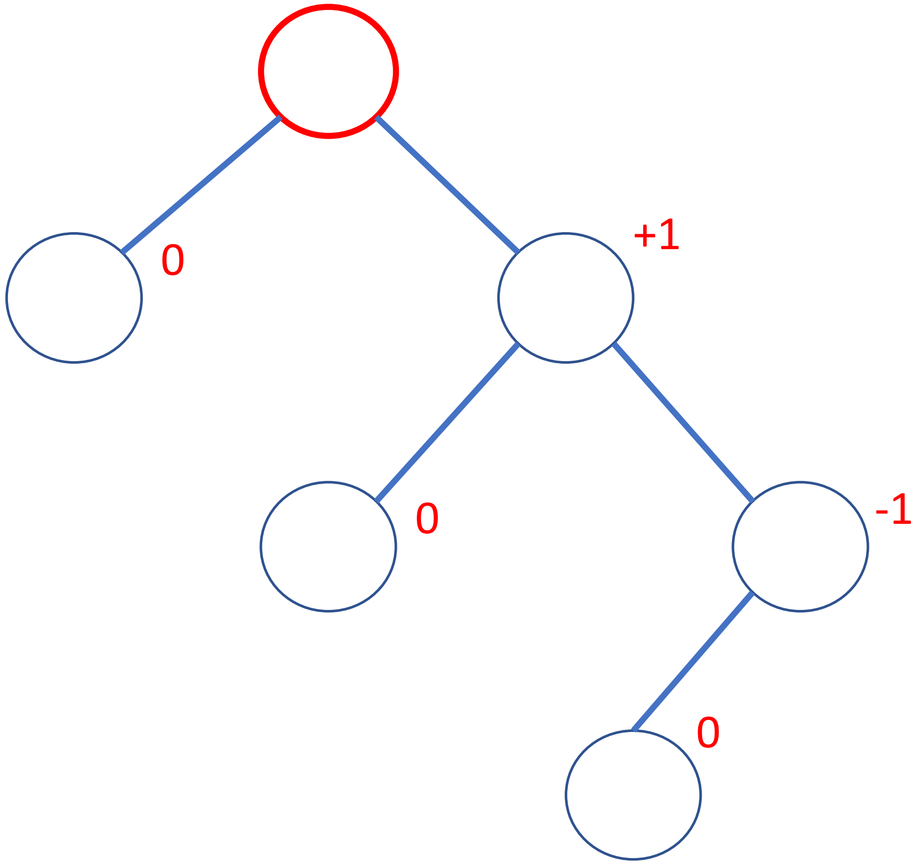
Is this an AVL tree?



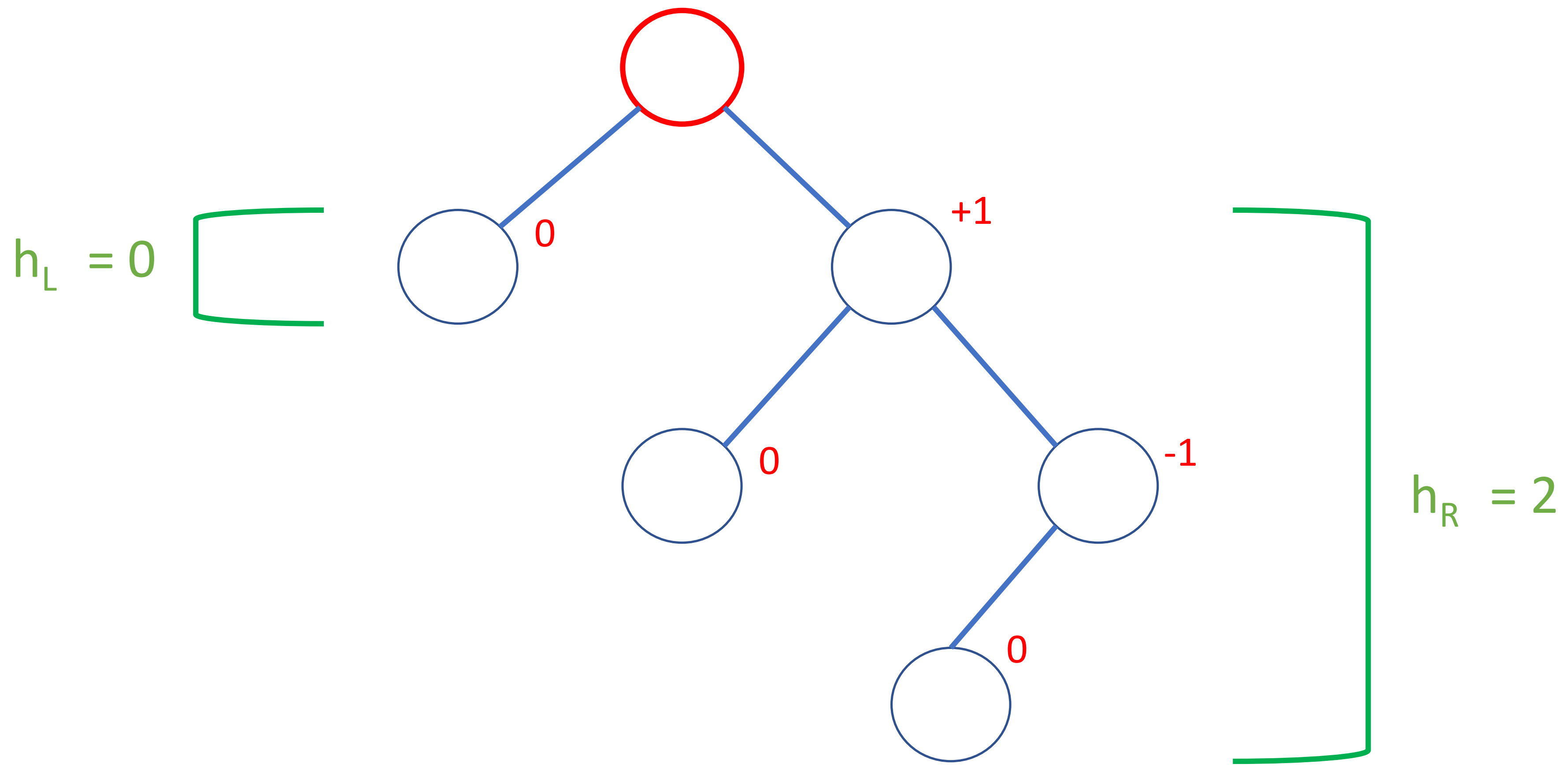
Is this an AVL tree?



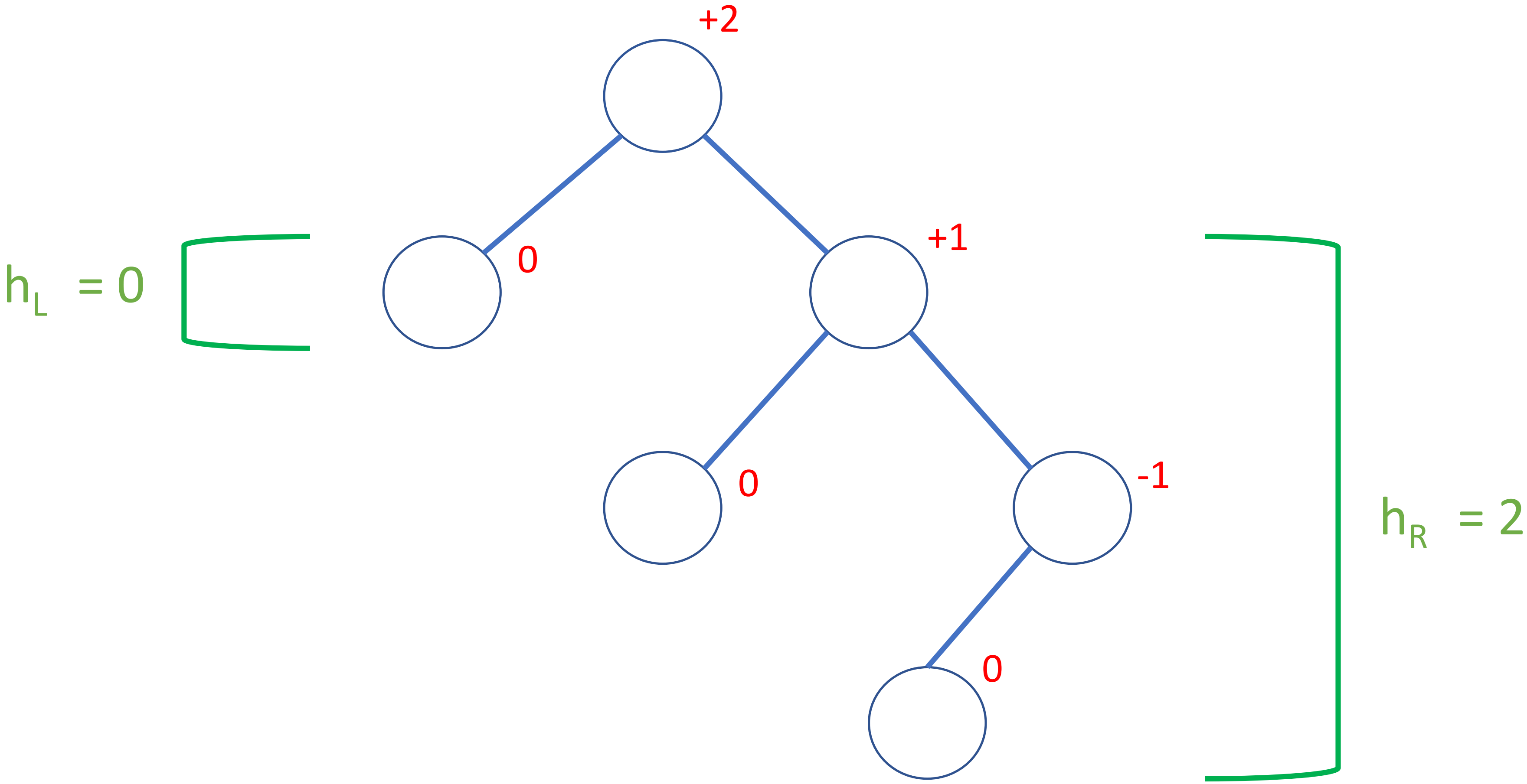
Is this an AVL tree?



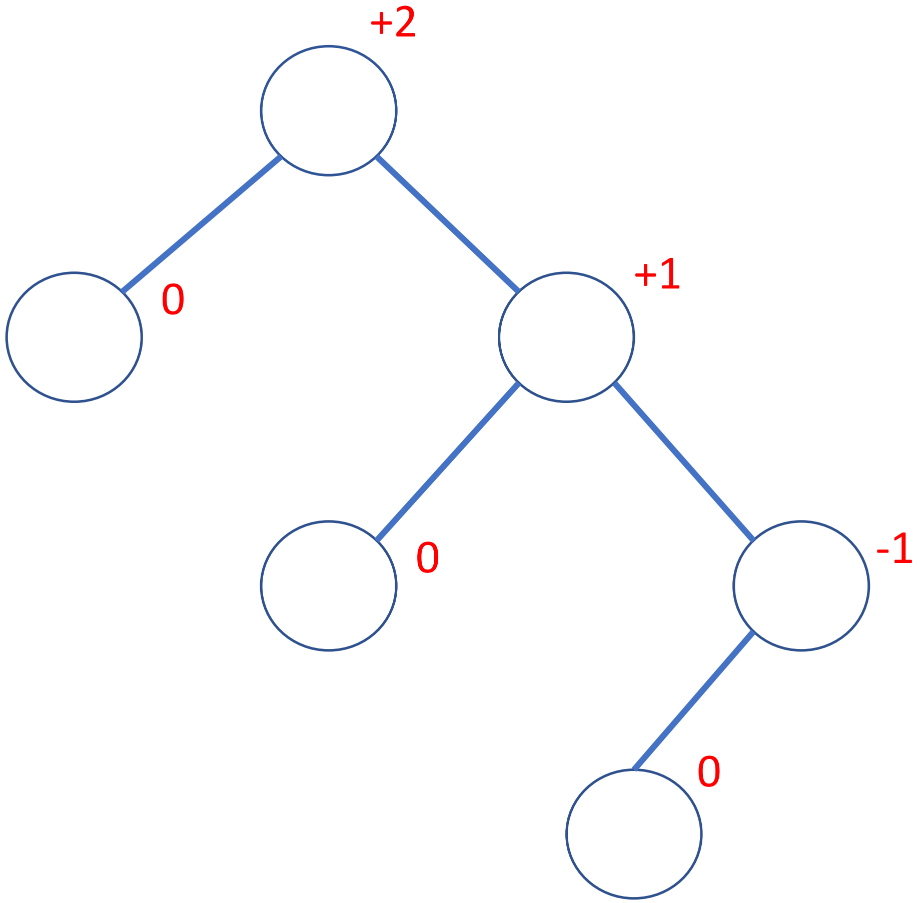
Is this an AVL tree?



Is this an AVL tree?



Is this an AVL tree? **No!**



Properties of AVL trees

Properties of AVL trees

- AVL trees of n nodes has height $\Theta(\log n)$ [height $\leq$ $1.44 \log_2(n+2)$]

Properties of AVL trees

- AVL trees of n nodes has height $\Theta(\log n)$ [height $\leq$ $1.44 \log_2(n+2)$]
- Can do inserts, deletes while maintaining the tree balance in $\Theta(\log n)$ time

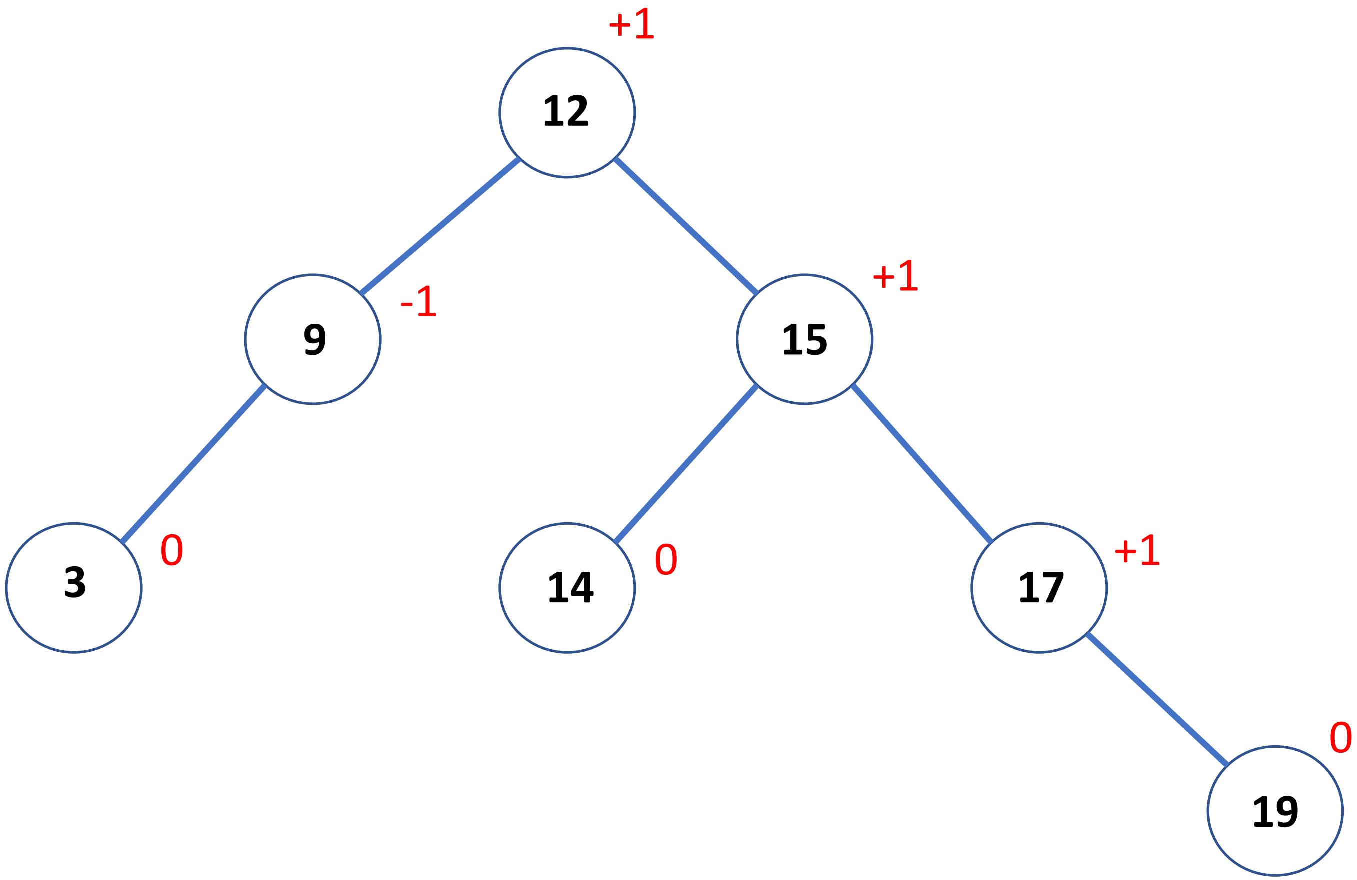
Properties of AVL trees

- AVL trees of n nodes has height $\Theta(\log n)$ [height $\leq$ $1.44 \log_2(n+2)$]
- Can do inserts, deletes while maintaining the tree balance in $\Theta(\log n)$ time

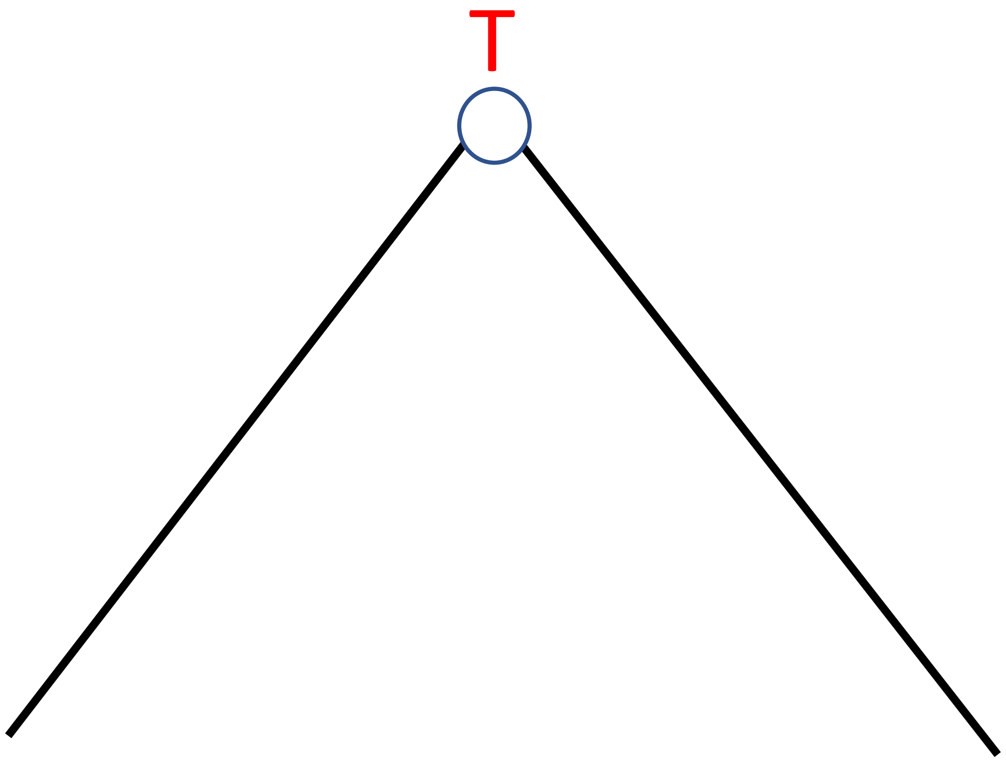
Elegant, relatively clean and simple, works well in practice

AVL tree operations

- **Search**(T, x)
- **Insert**(T, x)
- **Delete**(T, x)

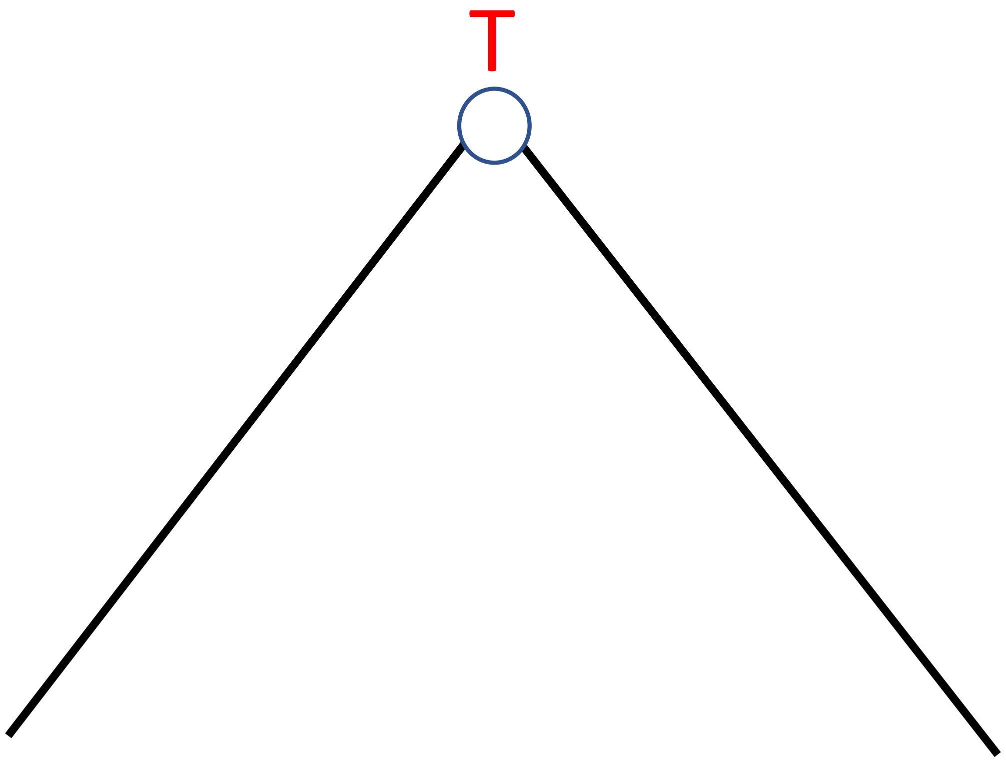


Insert(T, x) : General idea



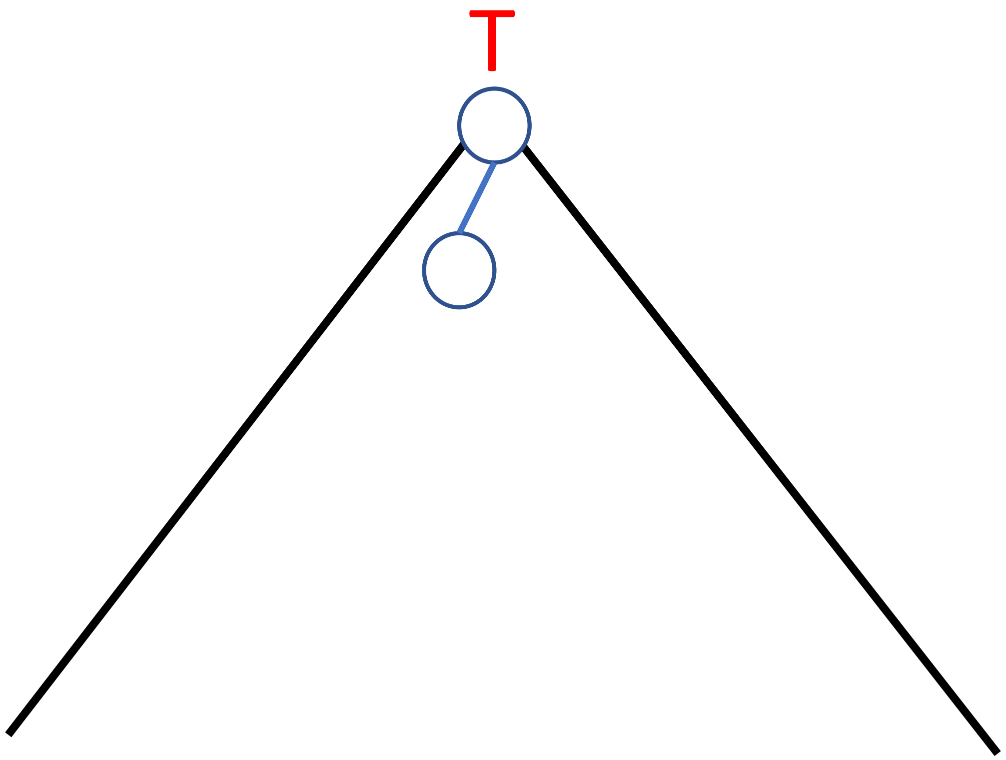
Insert(T, x) : General idea

- Insert x into T as in any BST



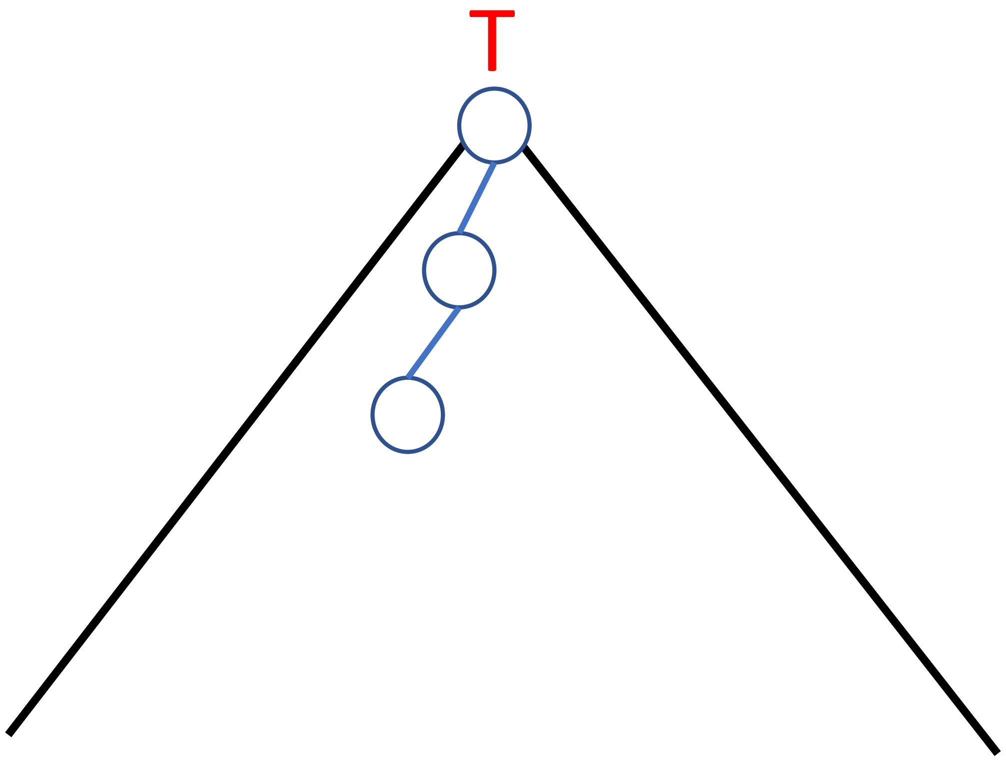
Insert(T, x) : General idea

- Insert x into T as in any BST



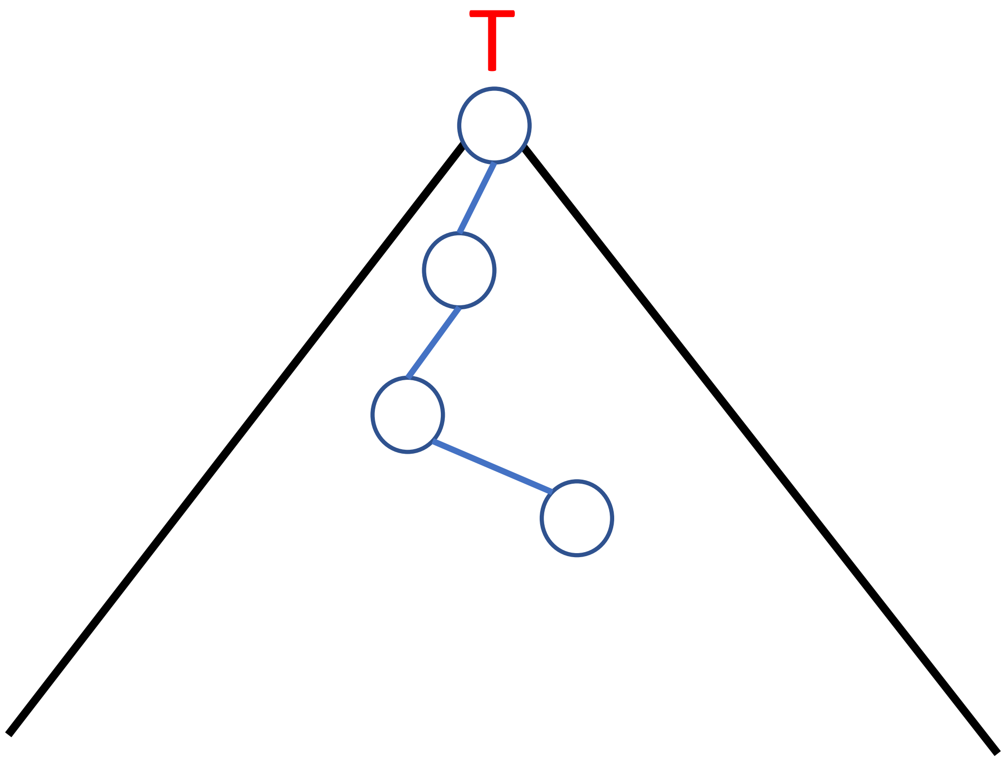
Insert(T, x) : General idea

- Insert x into T as in any BST



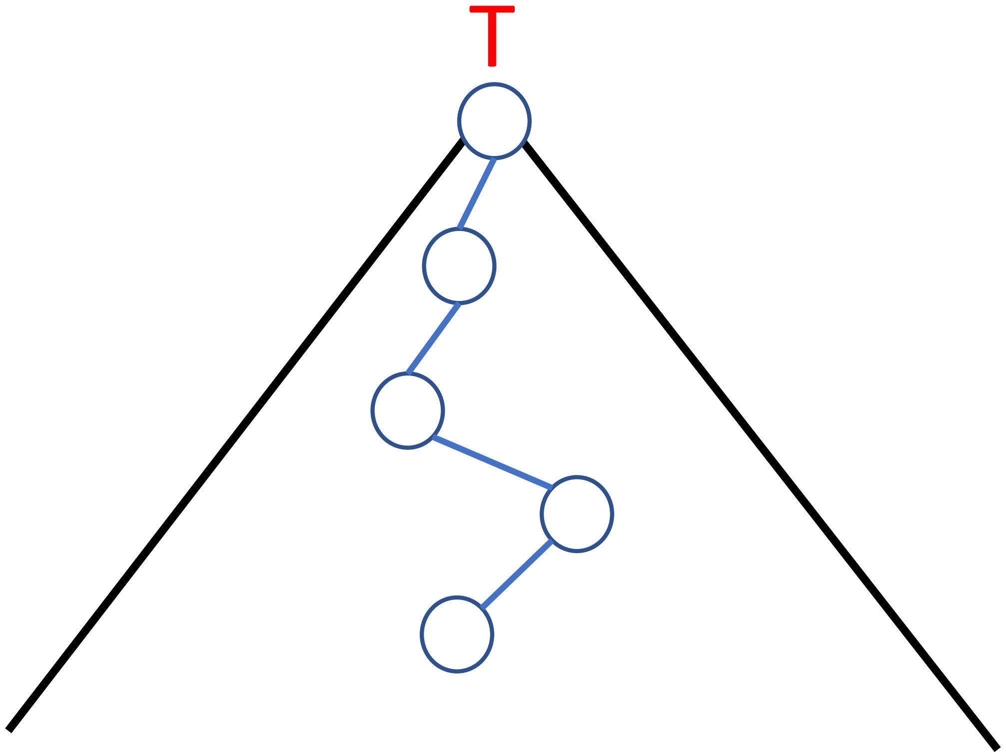
Insert(T, x) : General idea

- Insert x into T as in any BST



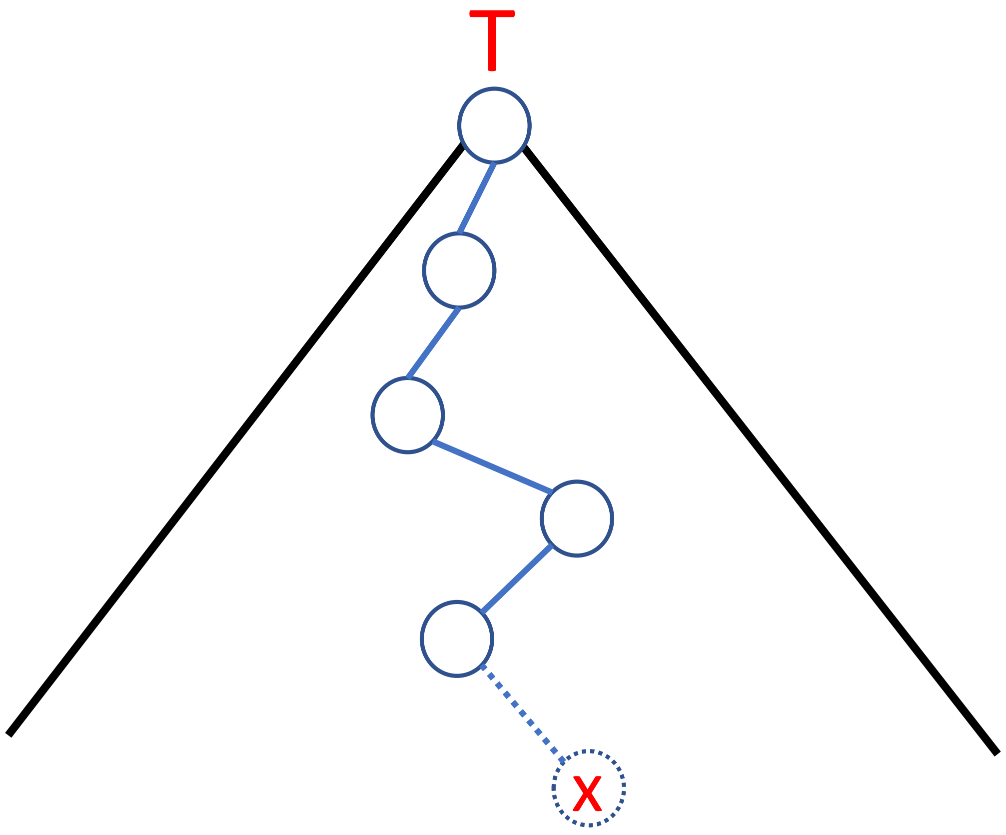
Insert(T, x) : General idea

- Insert x into T as in any BST



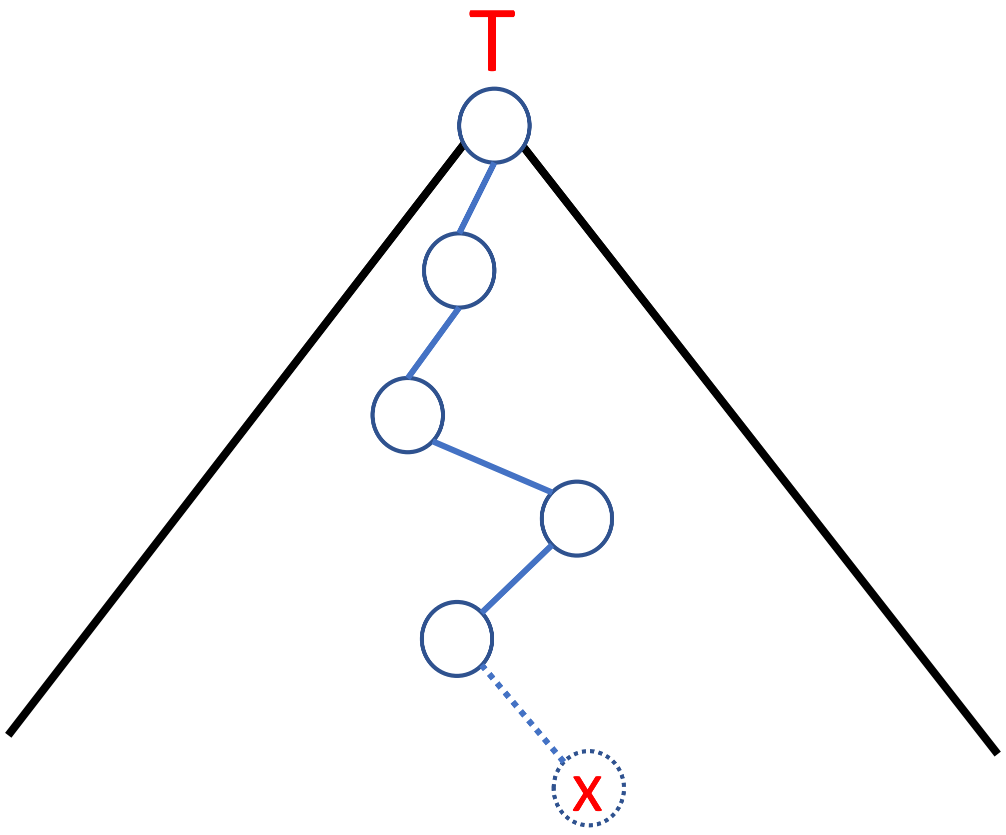
Insert(T, x) : General idea

- Insert x into T as in any BST



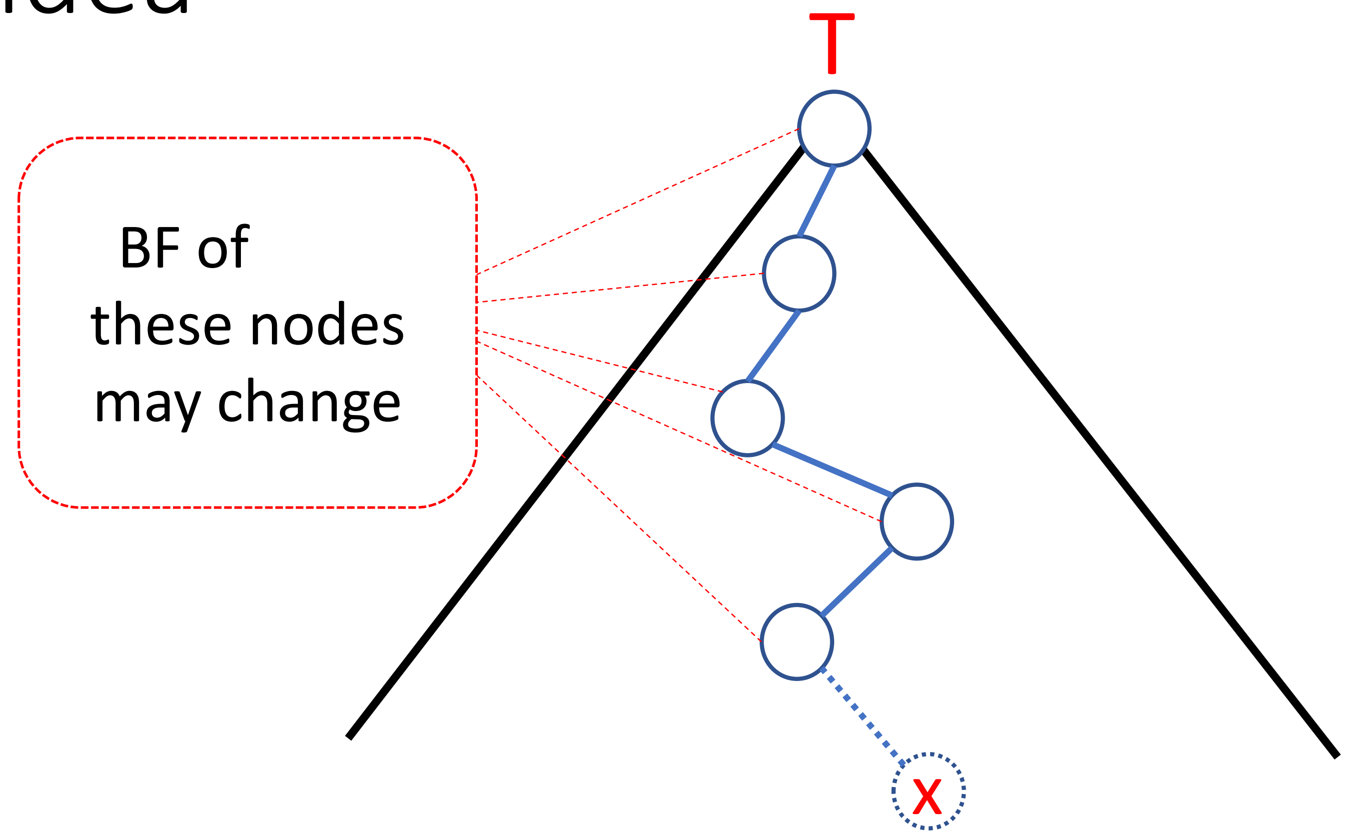
Insert(T, x) : General idea

- Insert x into T as in any BST :
  - x is now a leaf



Insert(T, x) : General idea

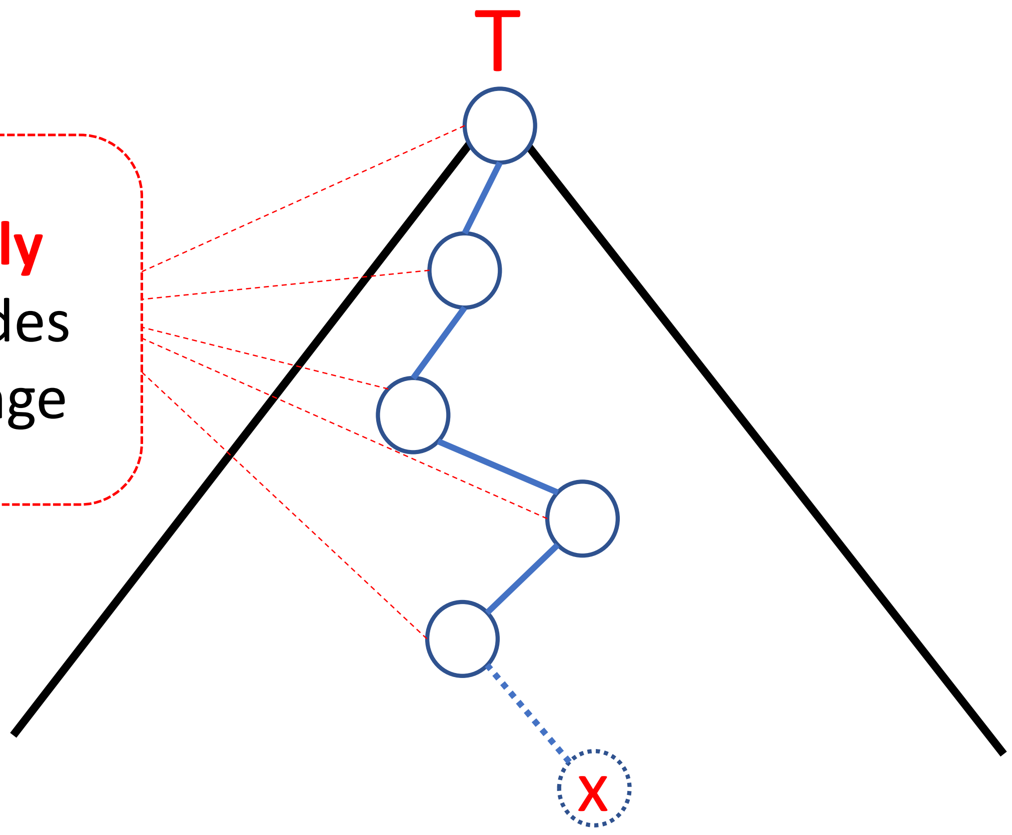
- Insert x into T as in any BST :
  - x is now a leaf



Insert(T, x) : General idea

- Insert x into T as in any BST :
  - x is now a leaf

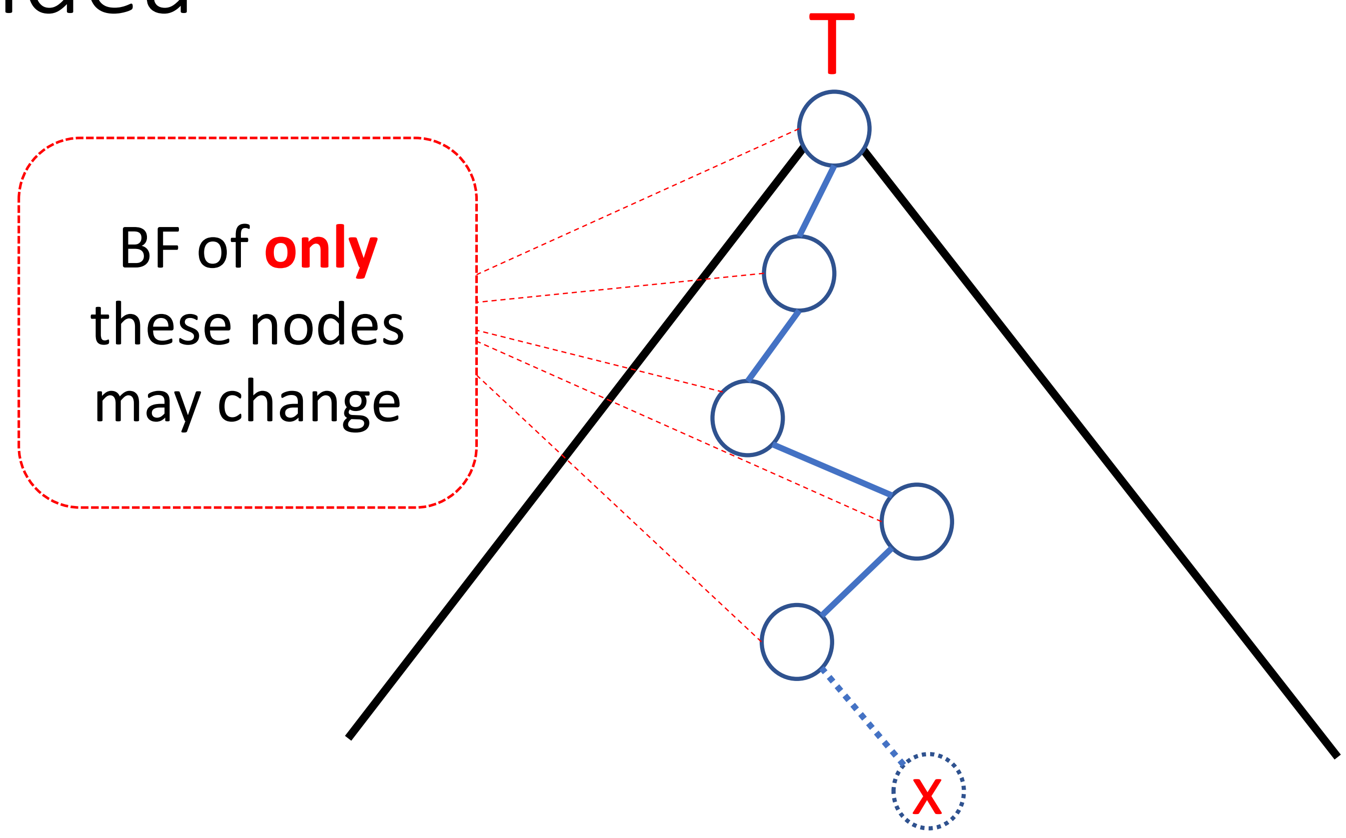
BF of **only** these nodes may change



Insert(T, x) : General idea

- Insert x into T as in any BST :
  - x is now a leaf

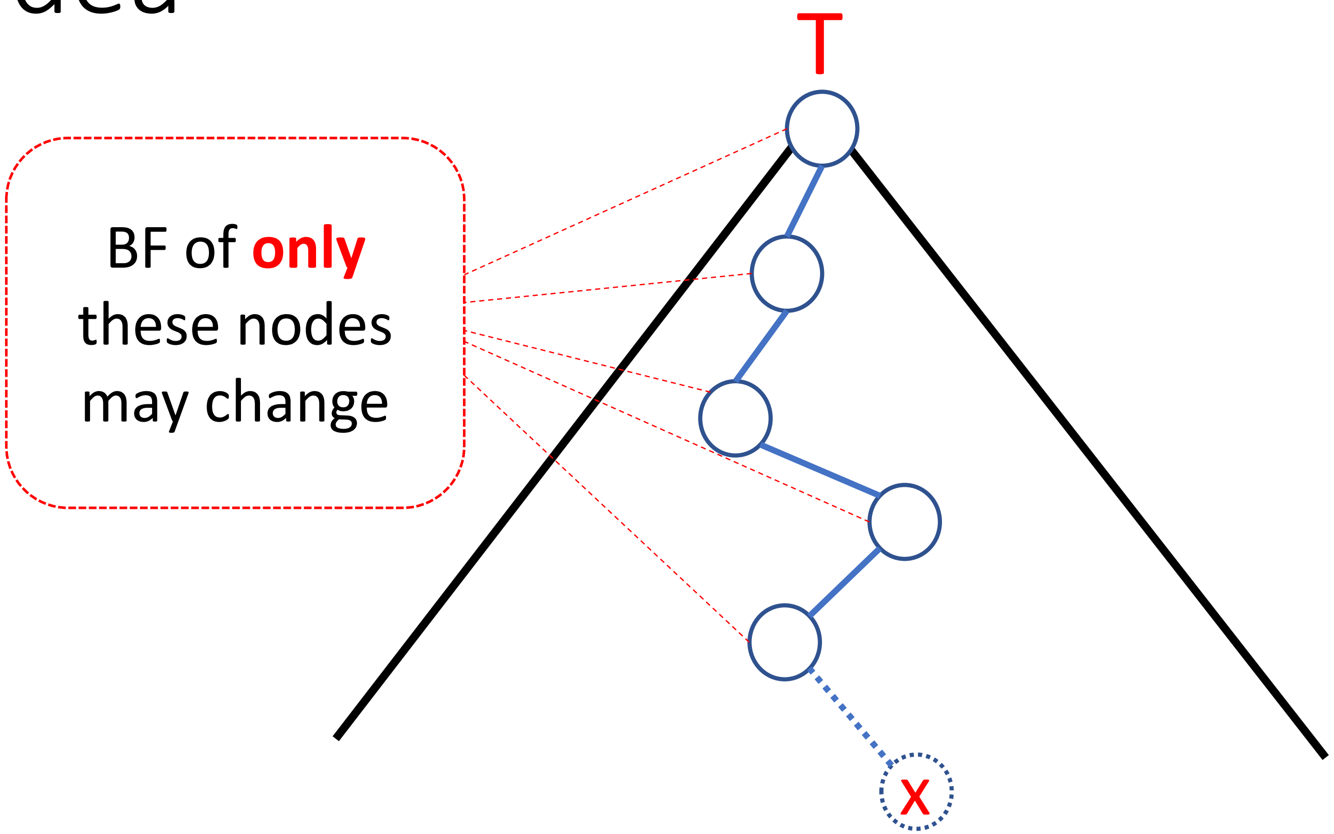
- Go up from x to the root



Insert(T, x) : General idea

- Insert x into T as in any BST :
  - x is now a leaf

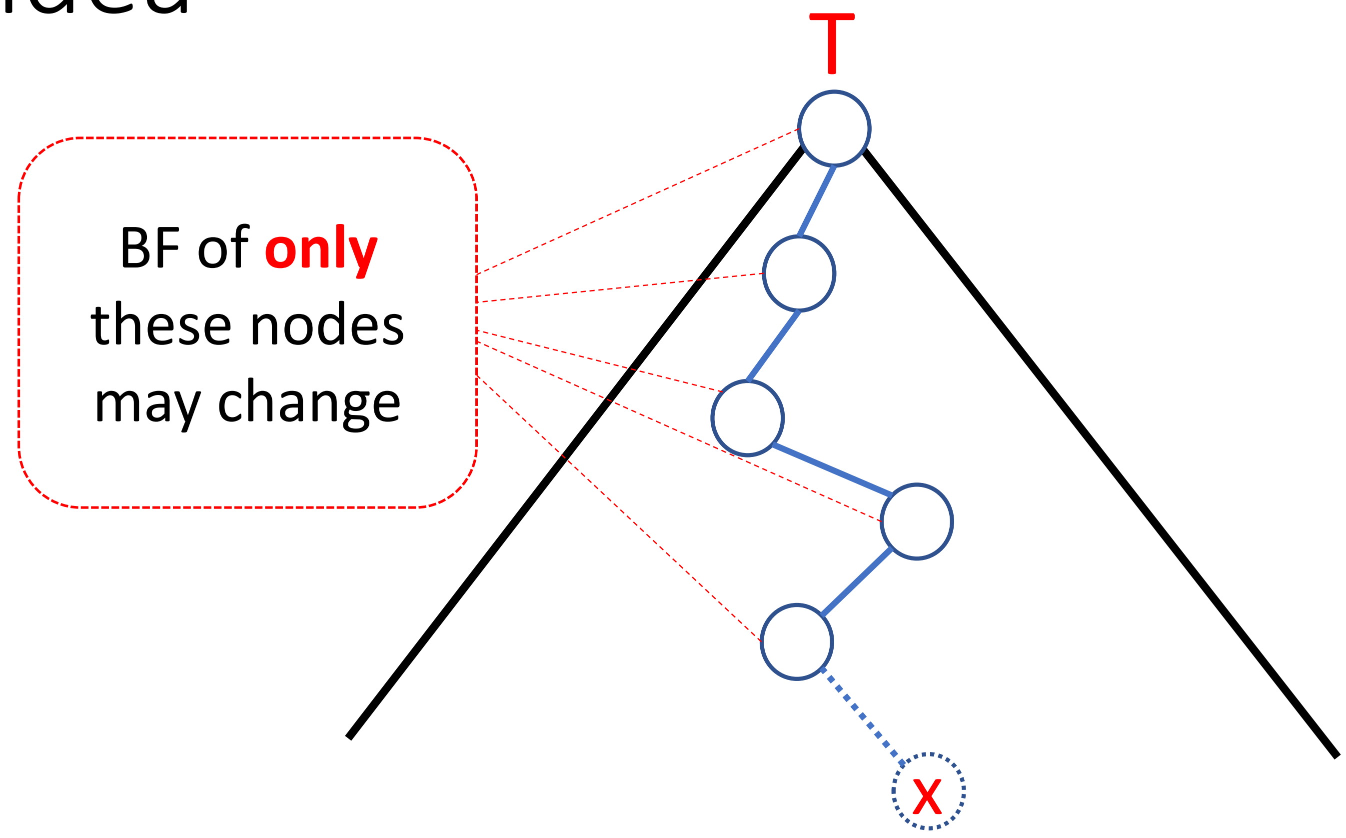
- Go up from x to the root and for each node :



Insert(T, x) : General idea

- Insert x into T as in any BST :
  - x is now a leaf

- Go up from x to the root and for each node :
  - Adjust the BF

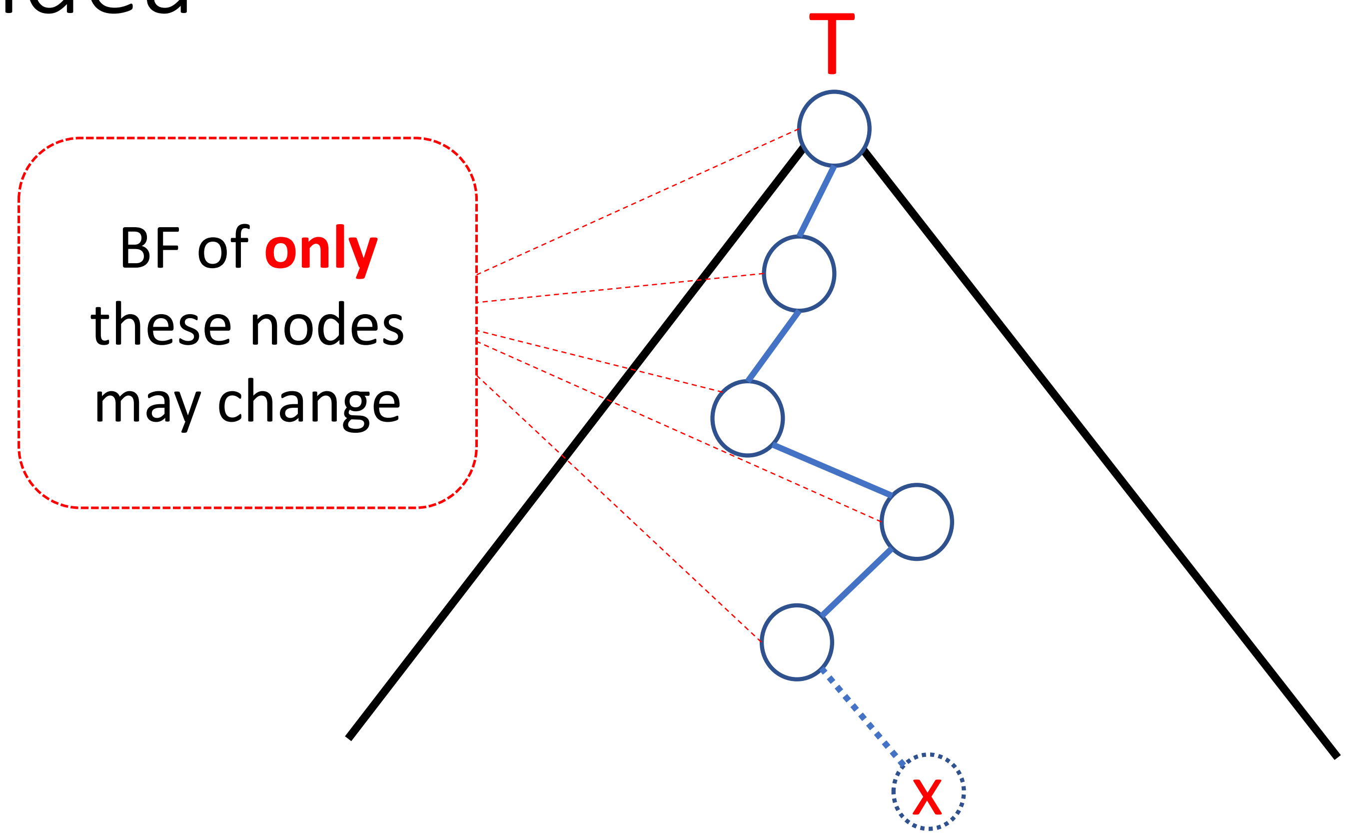


Insert(T, x) : General idea

- Insert x into T as in any BST :
  - x is now a leaf

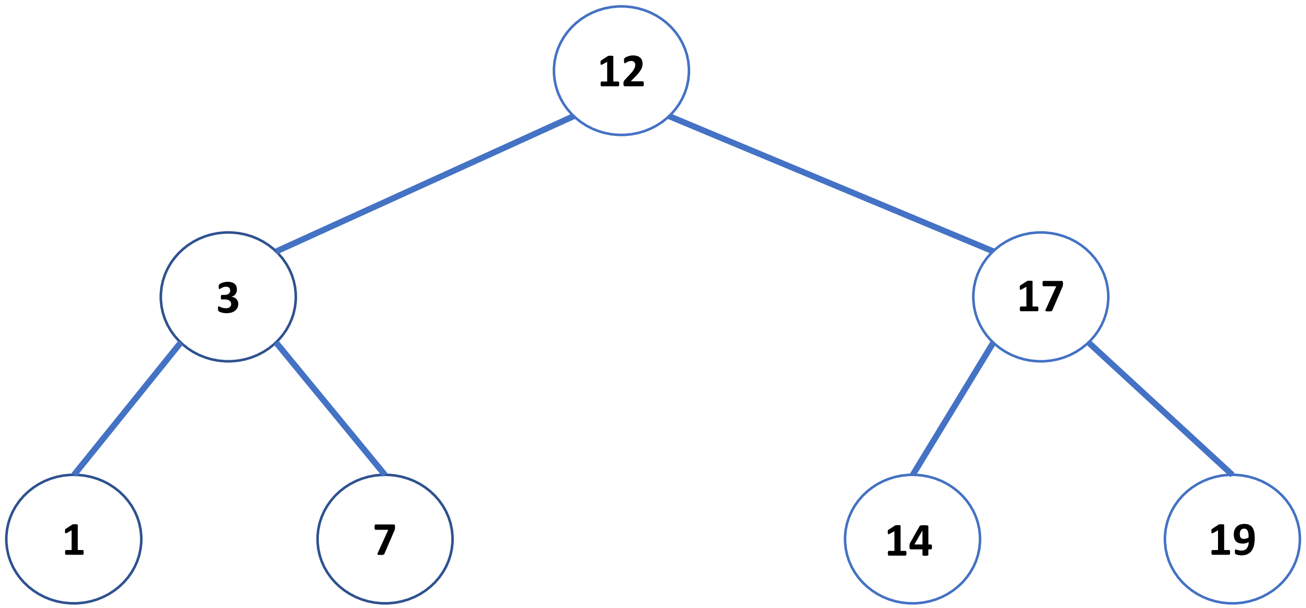
- Go up from x to the root and for each node :
  - Adjust the BF
  - “Rebalance” if necessary

    i.e. if BF > 1 or BF < -1

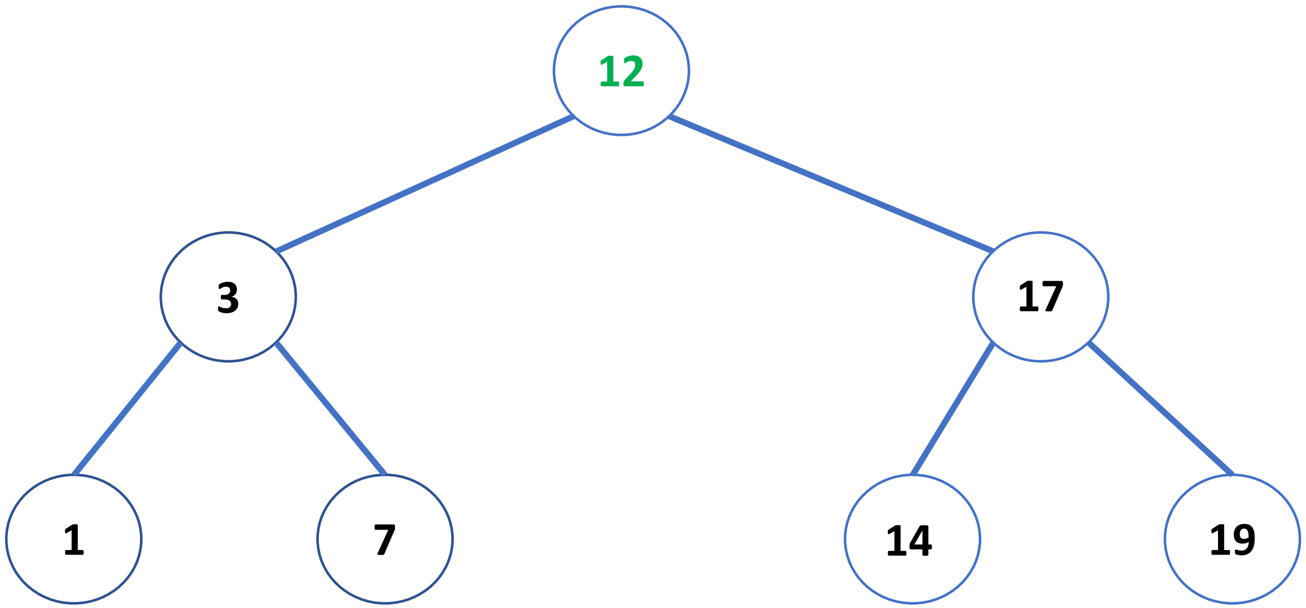


Example: AVL of {1, 3, 7, 12, 14, 17, 19}

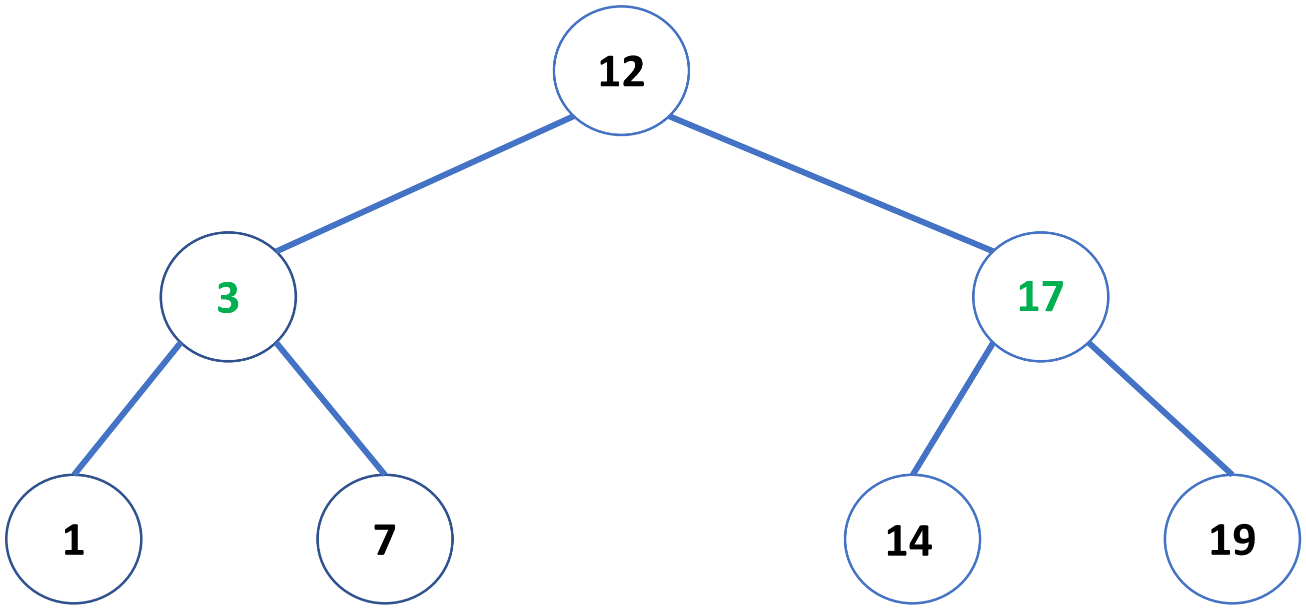
Example: AVL of {1, 3, 7, 12, 14, 17, 19}



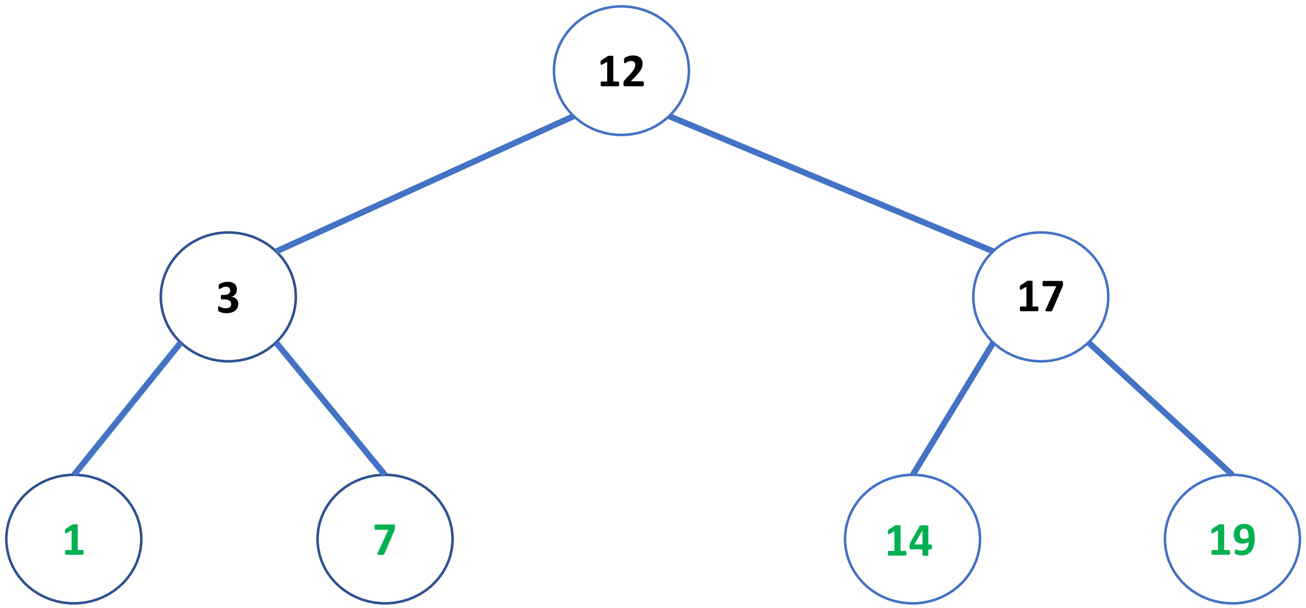
Example: AVL of {1, 3, 7, 12, 14, 17, 19}



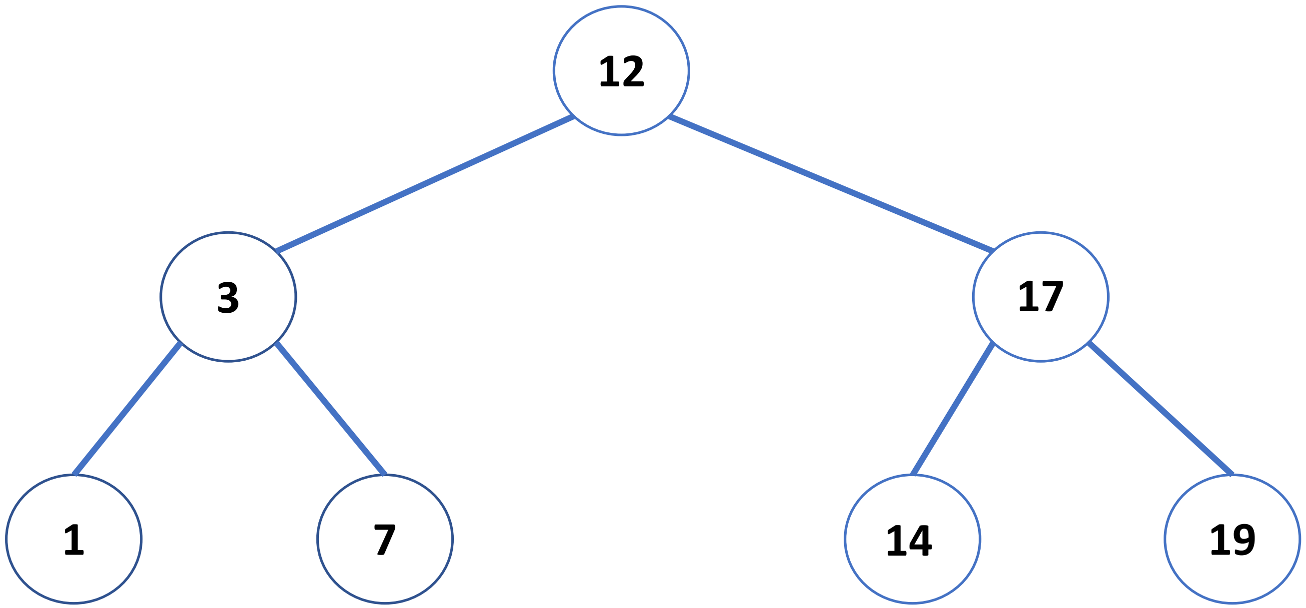
Example: AVL of {1, 3, 7, 12, 14, 17, 19}



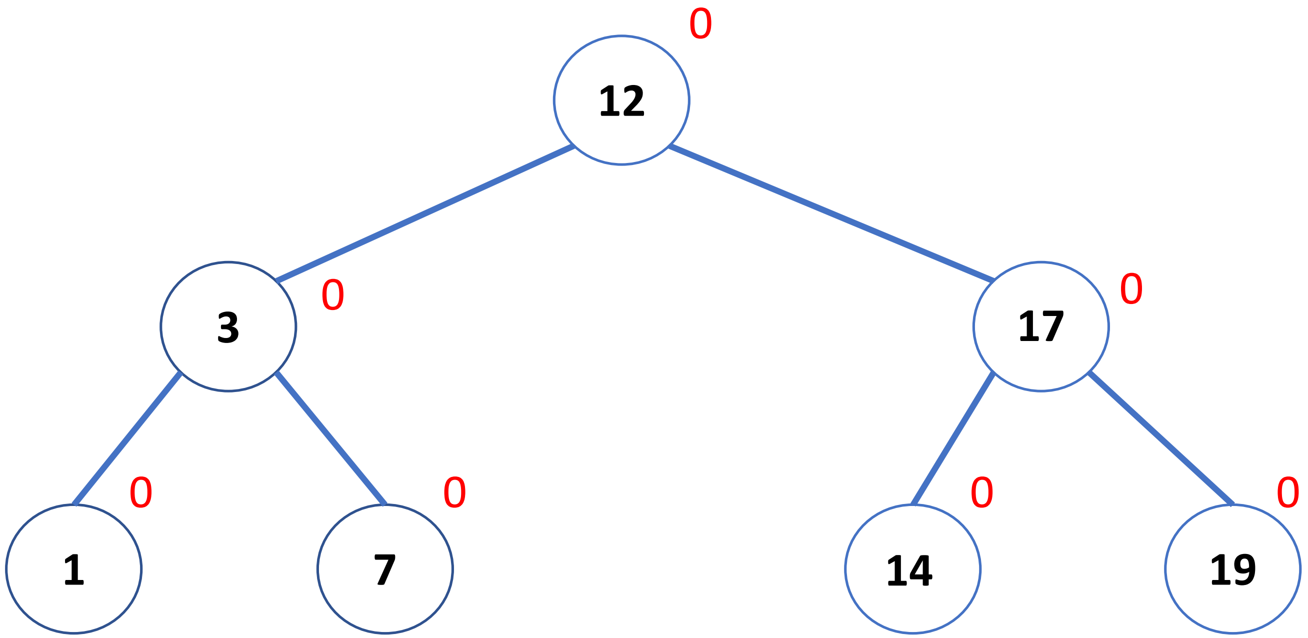
Example: AVL of {1, 3, 7, 12, 14, 17, 19}



Example: AVL of {1, 3, 7, 12, 14, 17, 19}

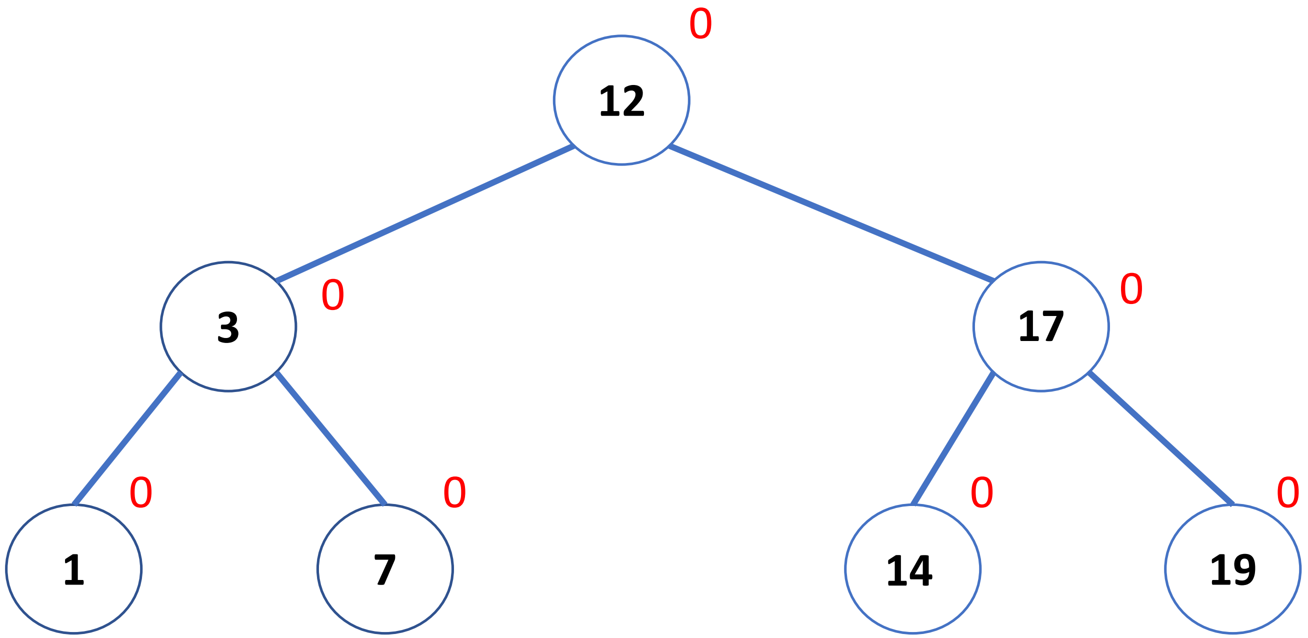


Example: AVL of {1, 3, 7, 12, 14, 17, 19}



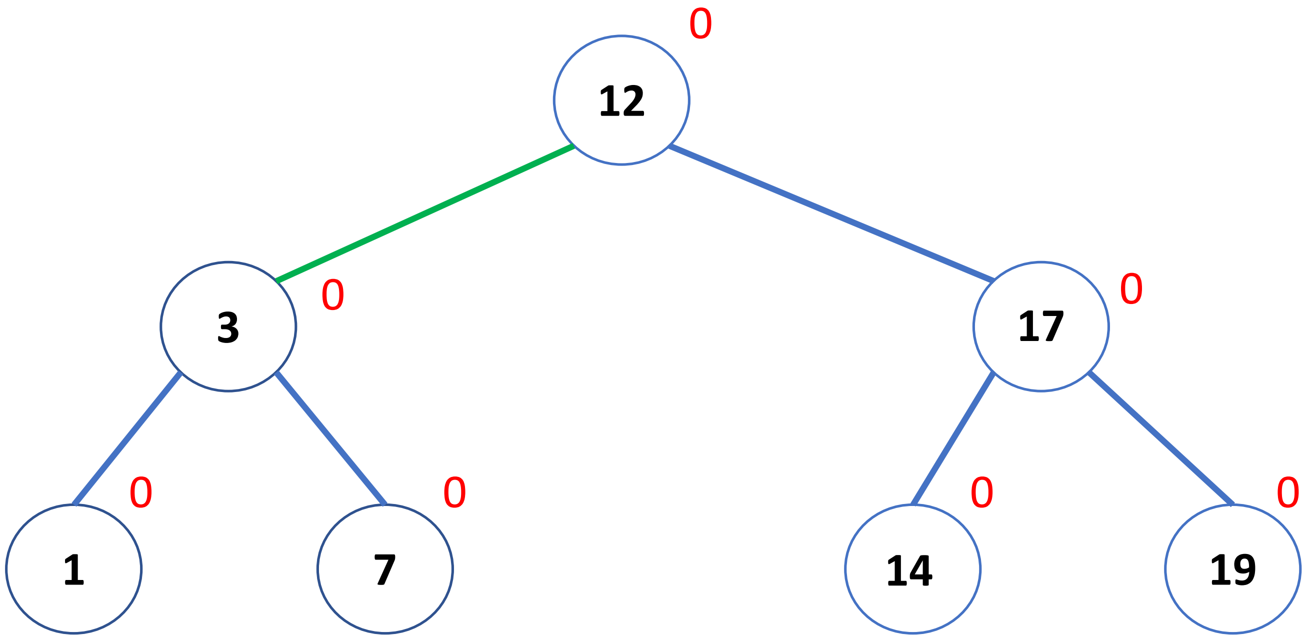
Example: AVL of {1, 3, 7, 12, 14, 17, 19}

Insert(T, 8)



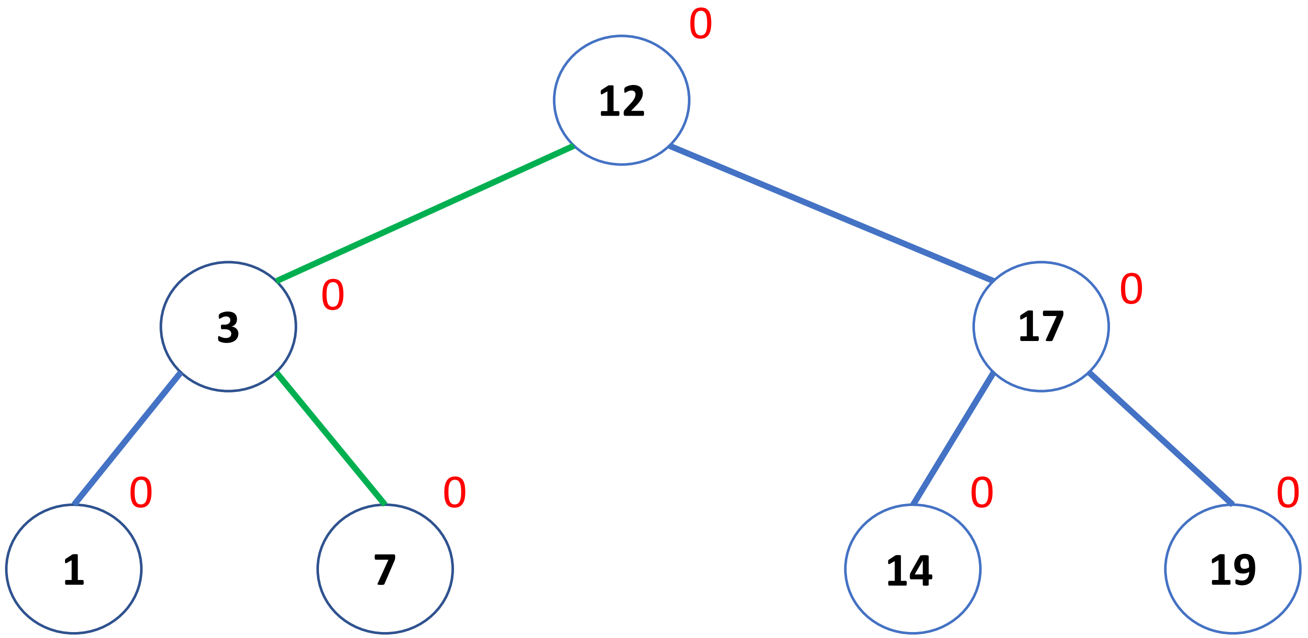
Example: AVL of {1, 3, 7, 12, 14, 17, 19}

Insert(T, 8)



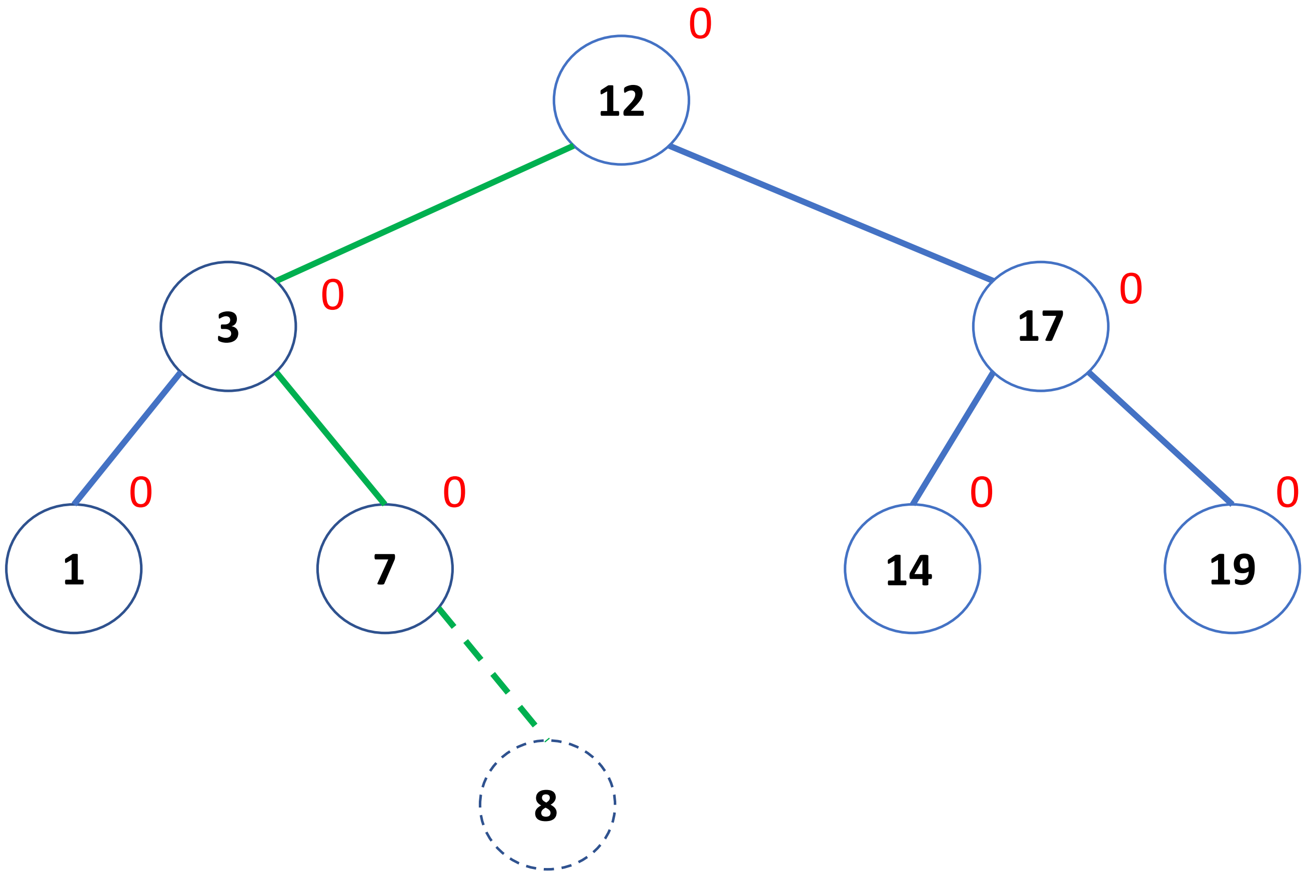
Example: AVL of {1, 3, 7, 12, 14, 17, 19}

Insert(T, 8)



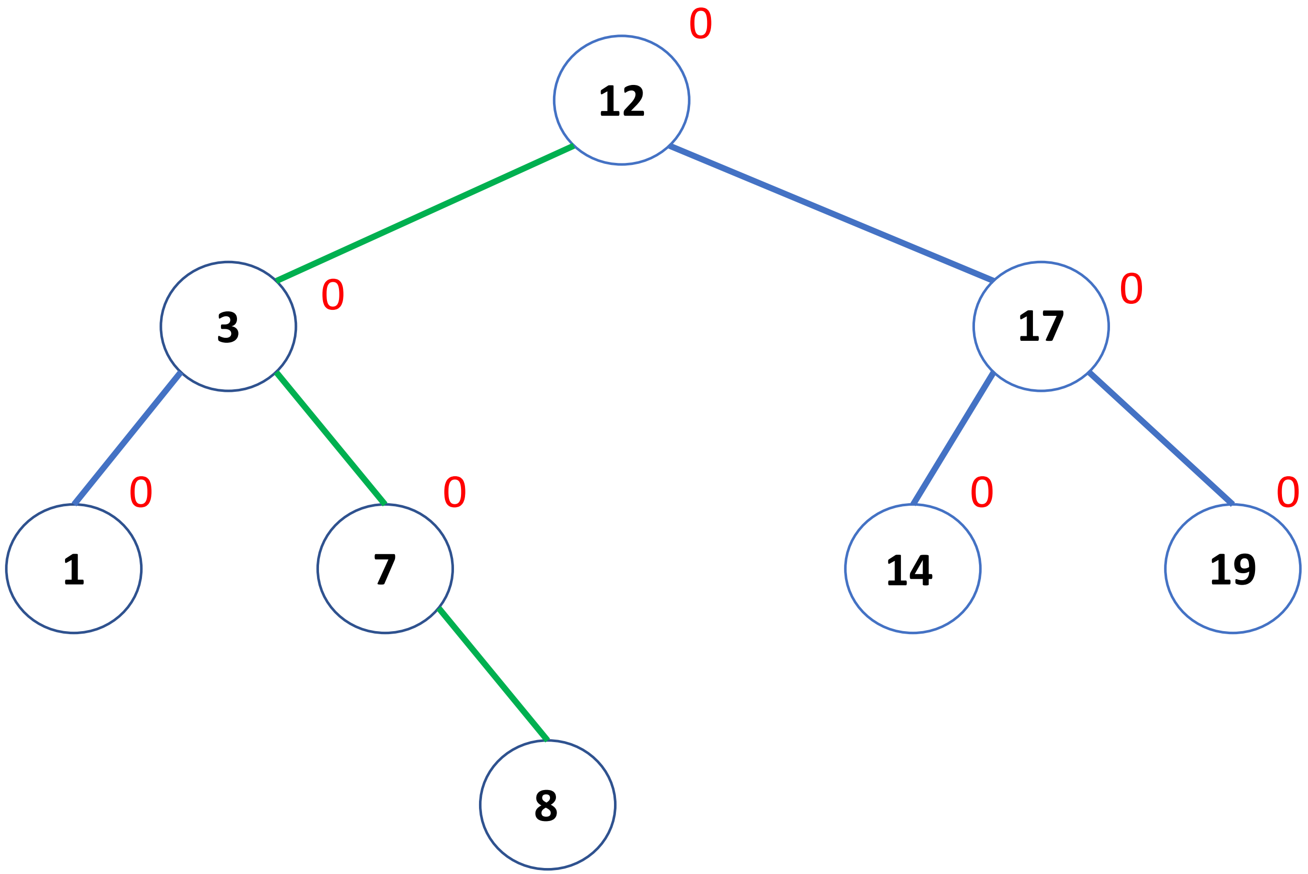
Example: AVL of {1, 3, 7, 12, 14, 17, 19}

Insert(T, 8)



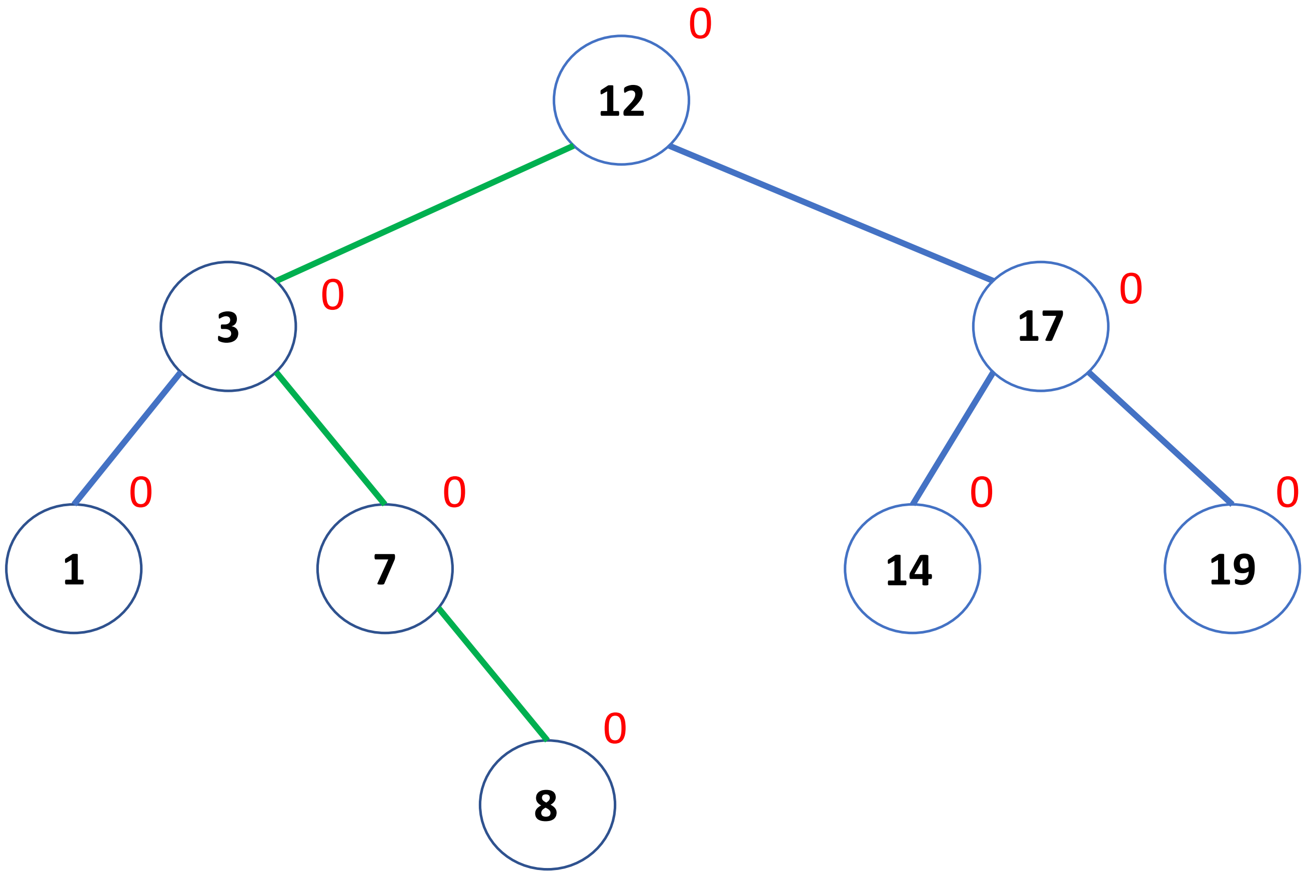
Example: AVL of {1, 3, 7, 12, 14, 17, 19}

Insert(T, 8)



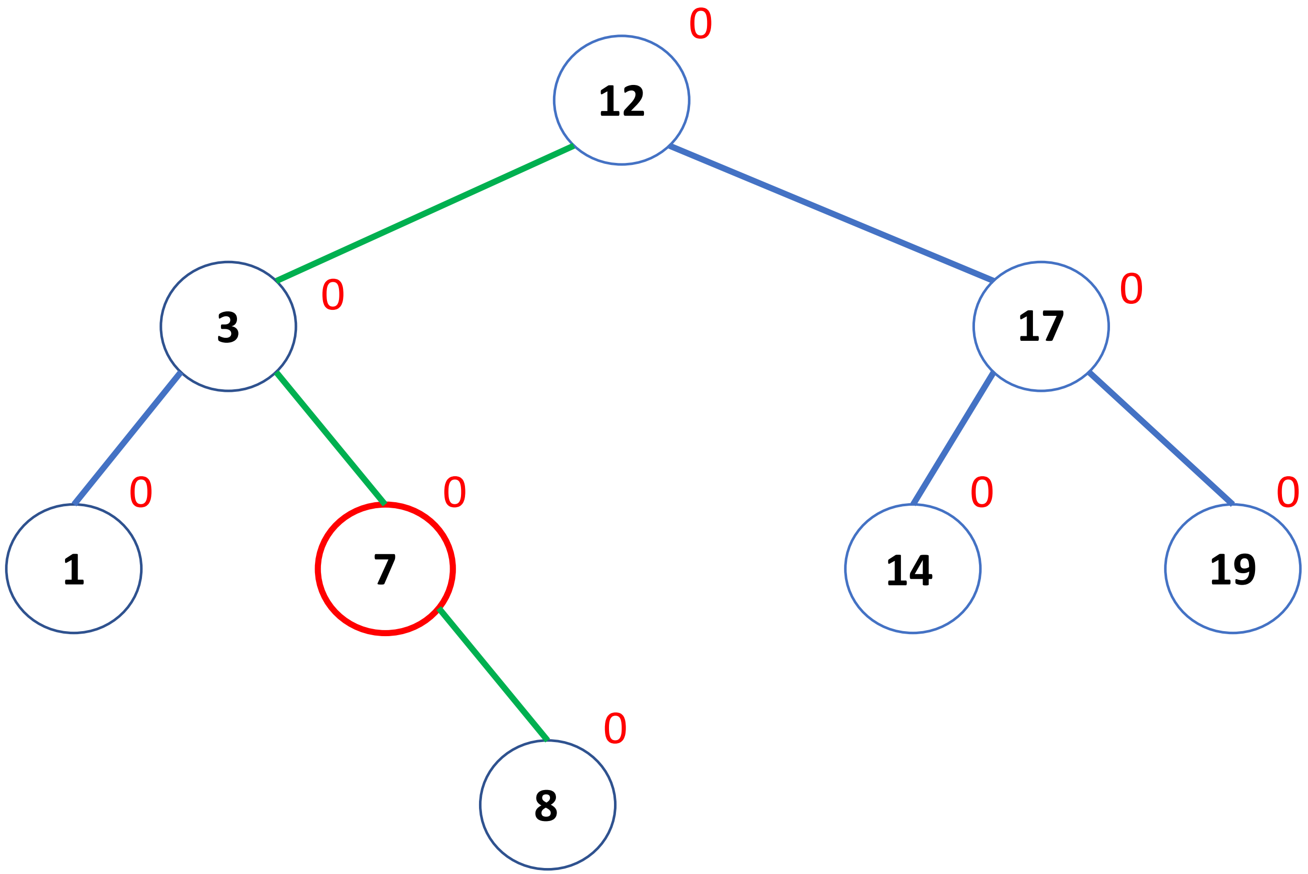
Example: AVL of {1, 3, 7, 12, 14, 17, 19}

Insert(T, 8)



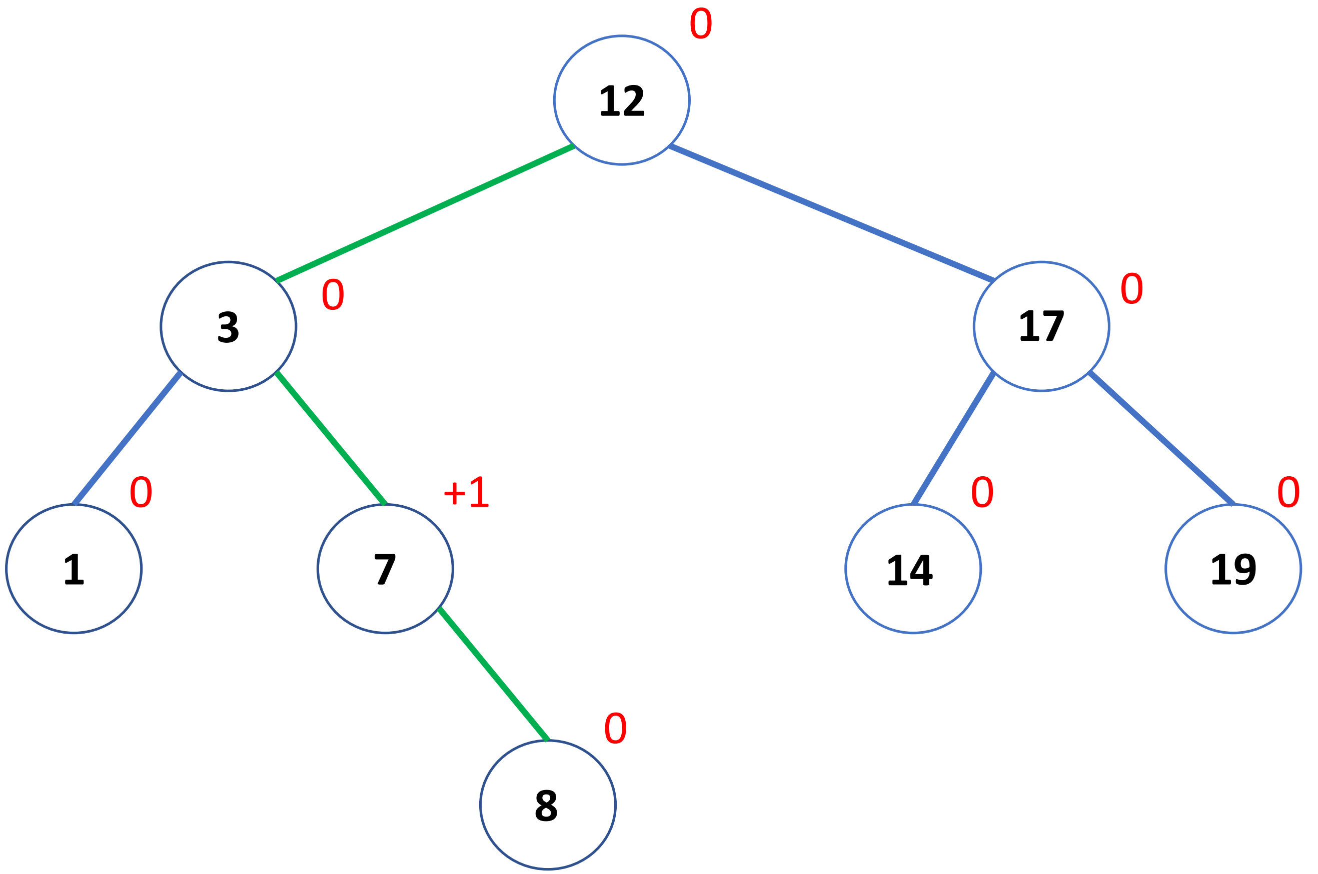
Example: AVL of {1, 3, 7, 12, 14, 17, 19}

Insert(T, 8)



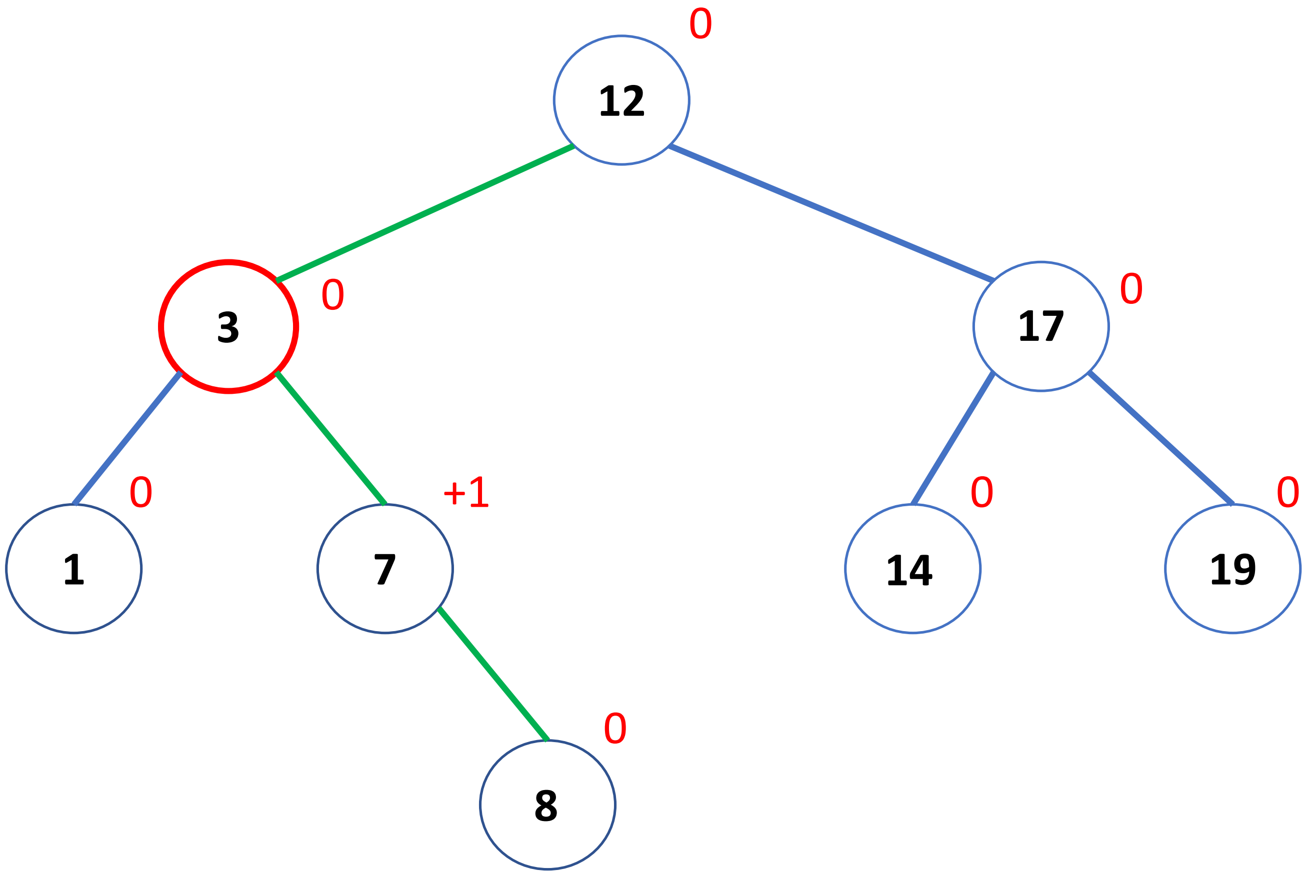
Example: AVL of {1, 3, 7, 12, 14, 17, 19}

Insert(T, 8)



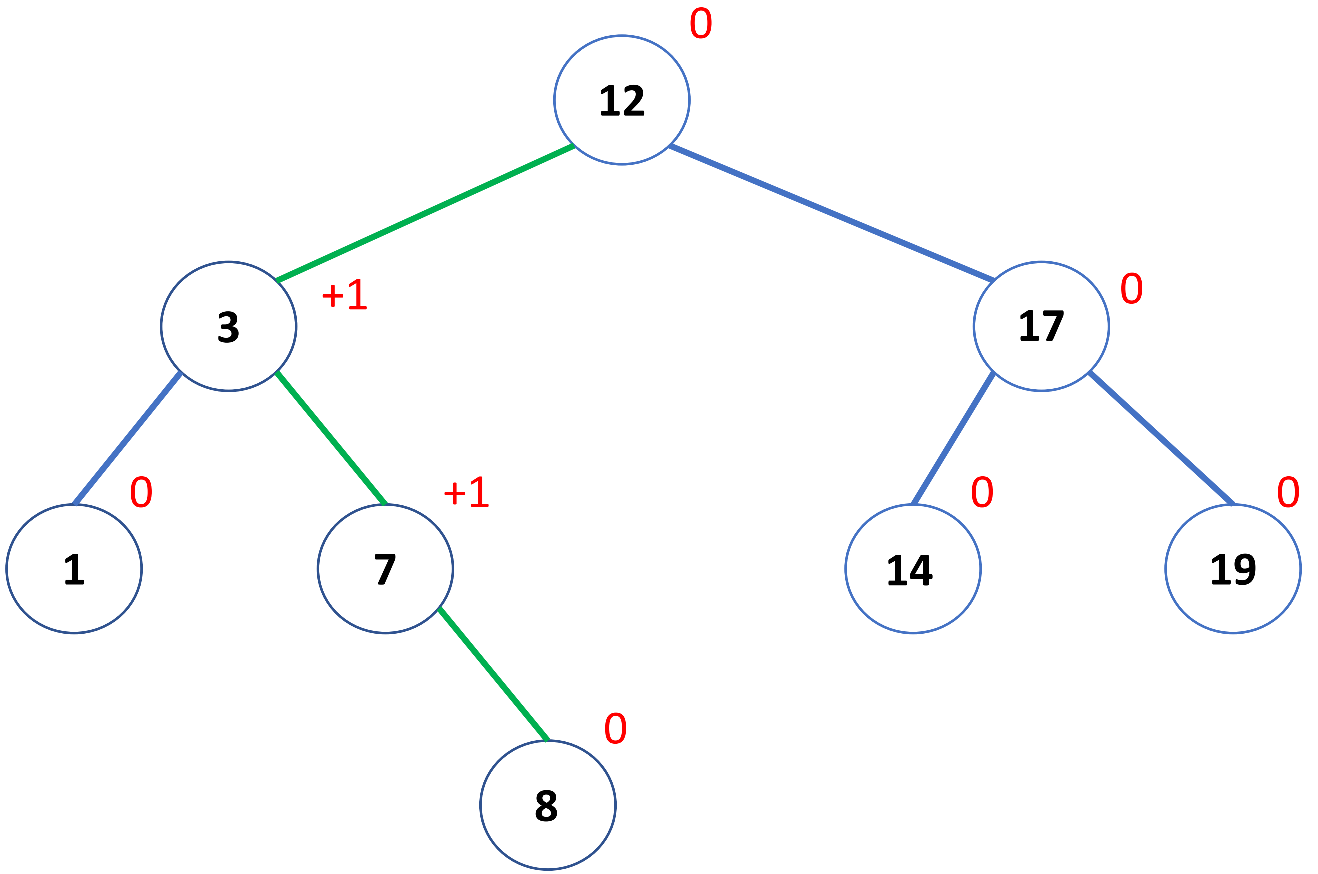
Example: AVL of {1, 3, 7, 12, 14, 17, 19}

Insert(T, 8)



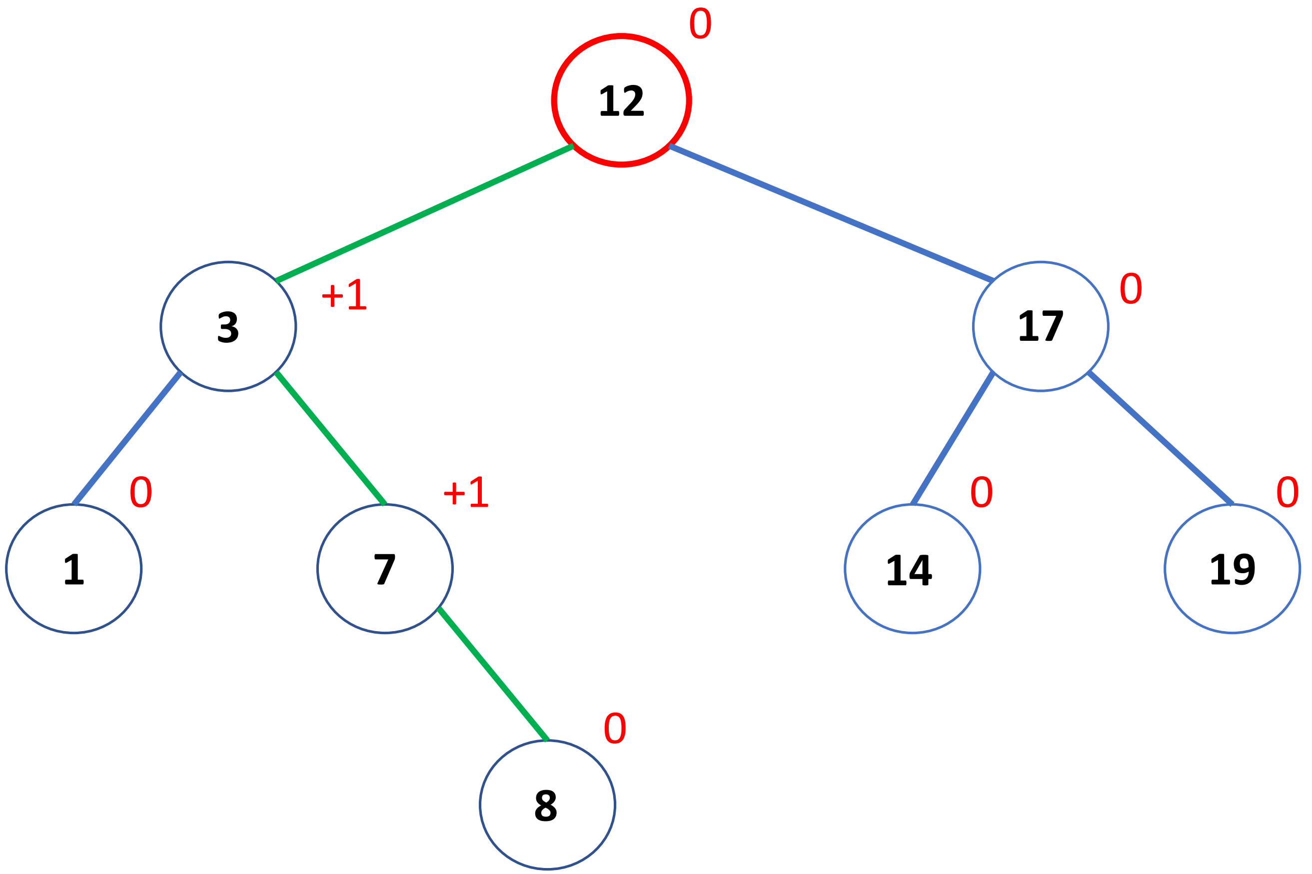
Example: AVL of {1, 3, 7, 12, 14, 17, 19}

Insert(T, 8)



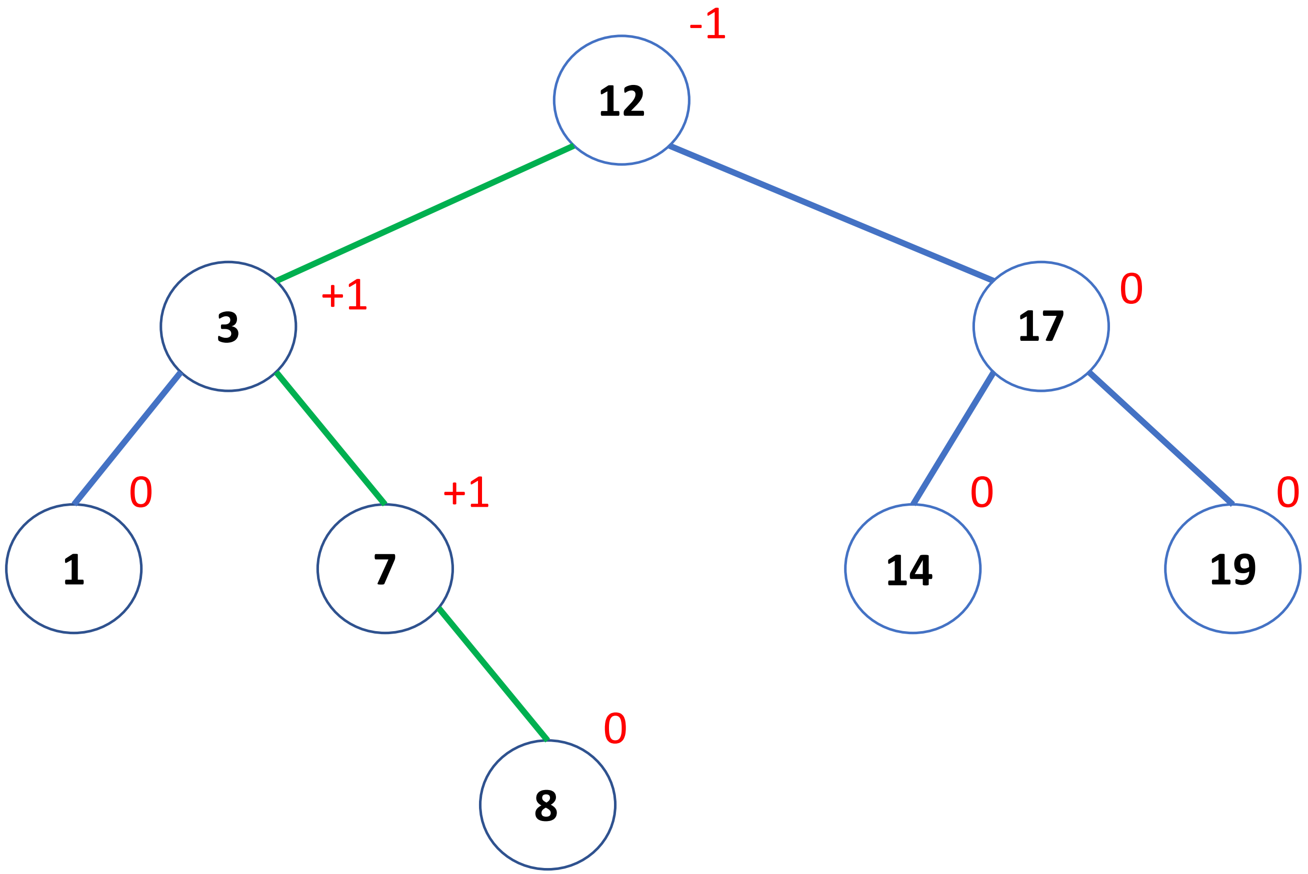
Example: AVL of {1, 3, 7, 12, 14, 17, 19}

Insert(T, 8)



Example: AVL of {1, 3, 7, 12, 14, 17, 19}

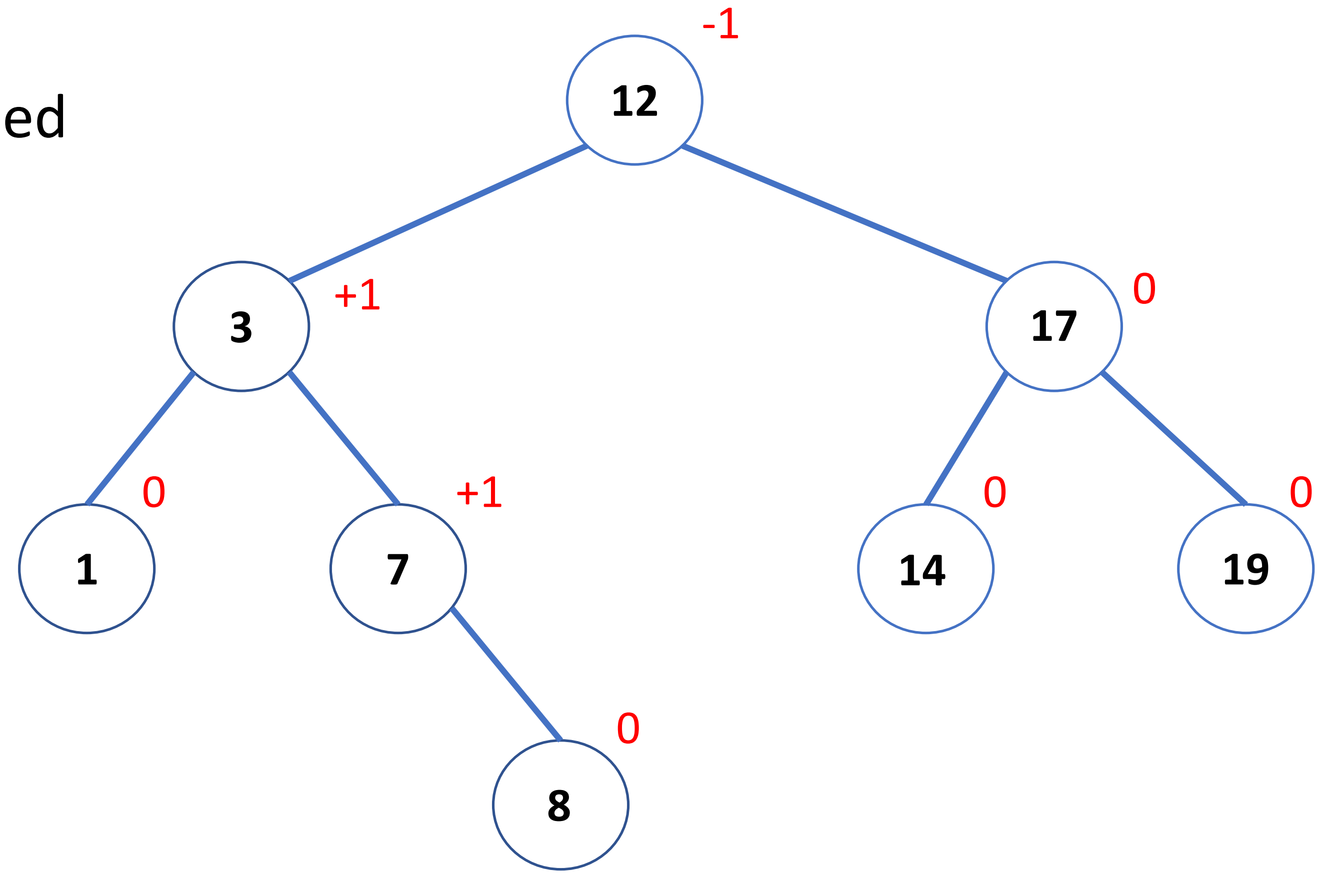
Insert(T, 8)



Example: AVL of {1, 3, 7, 12, 14, 17, 19}

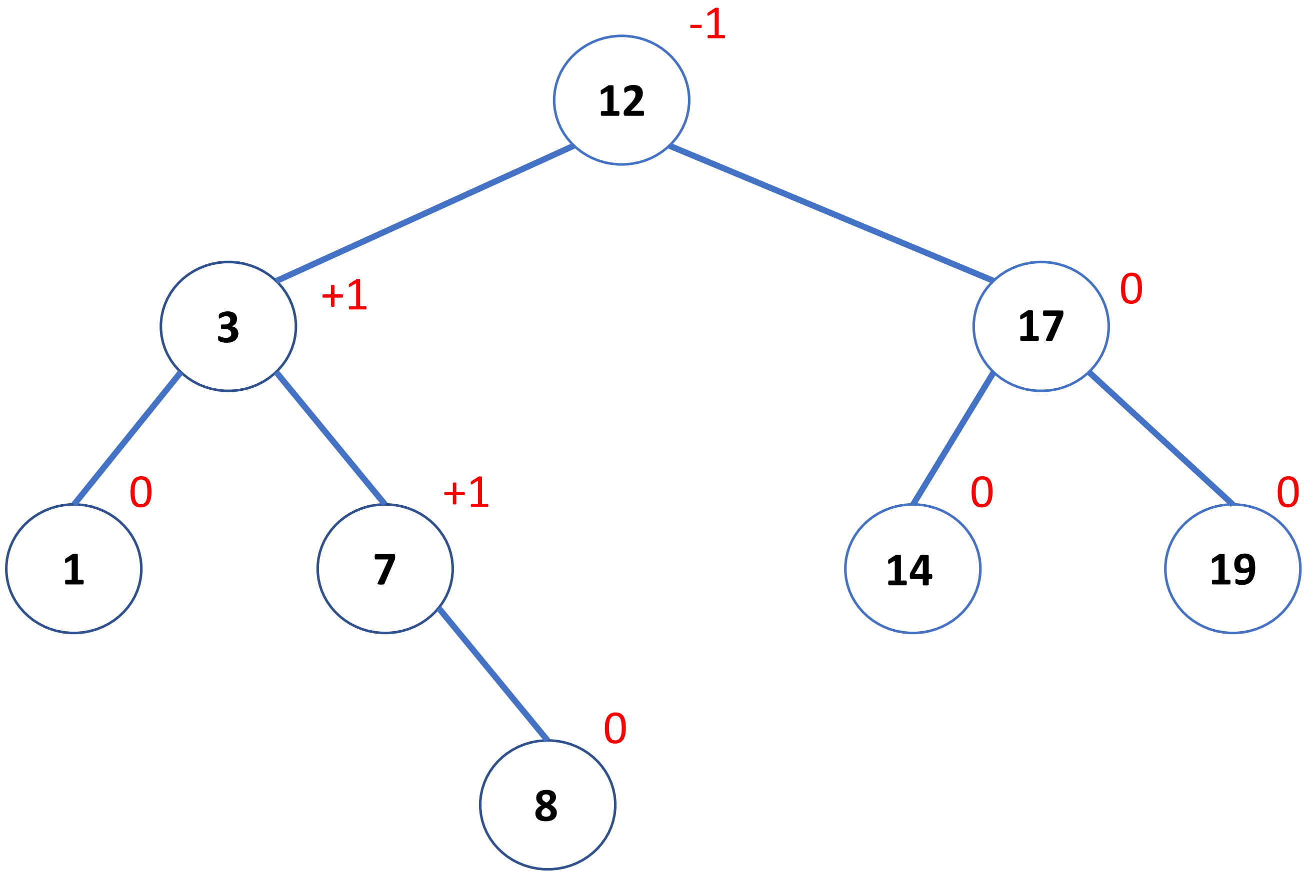
Insert(T, 8) :

No rebalancing needed



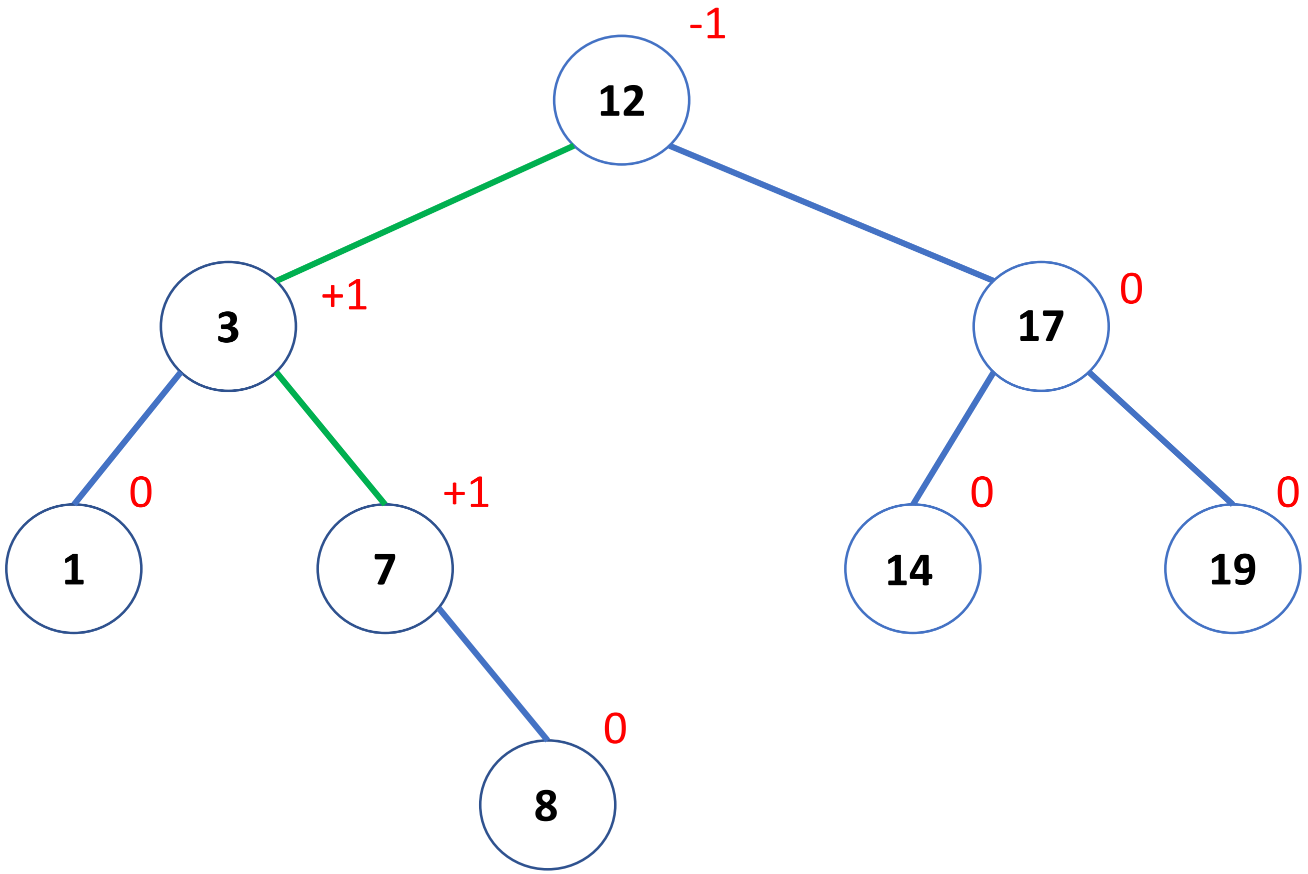
Example: AVL of {1, 3, 7, 8, 12, 14, 17, 19}

Insert(T, 5)



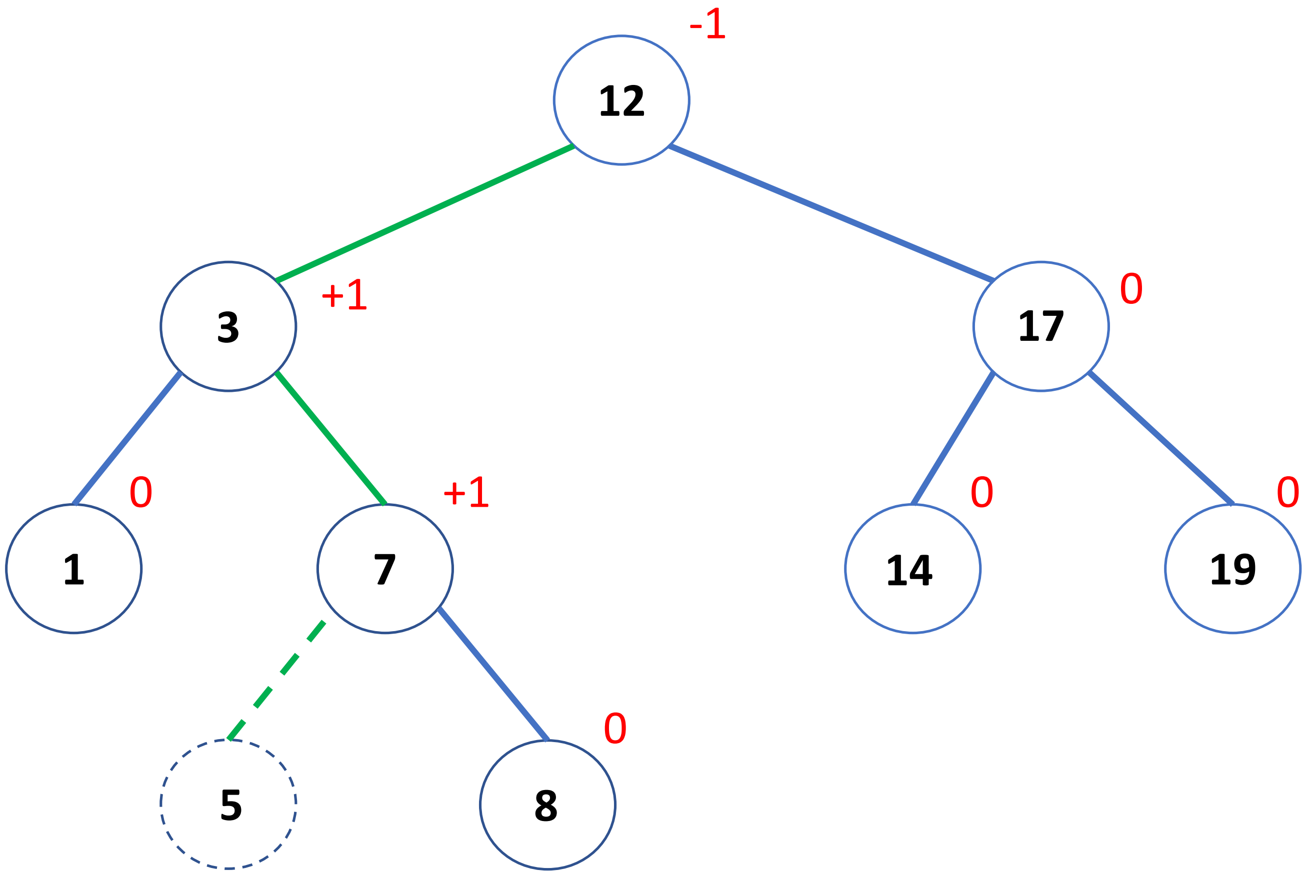
Example: AVL of {1, 3, 7, 8, 12, 14, 17, 19}

Insert(T, 5)



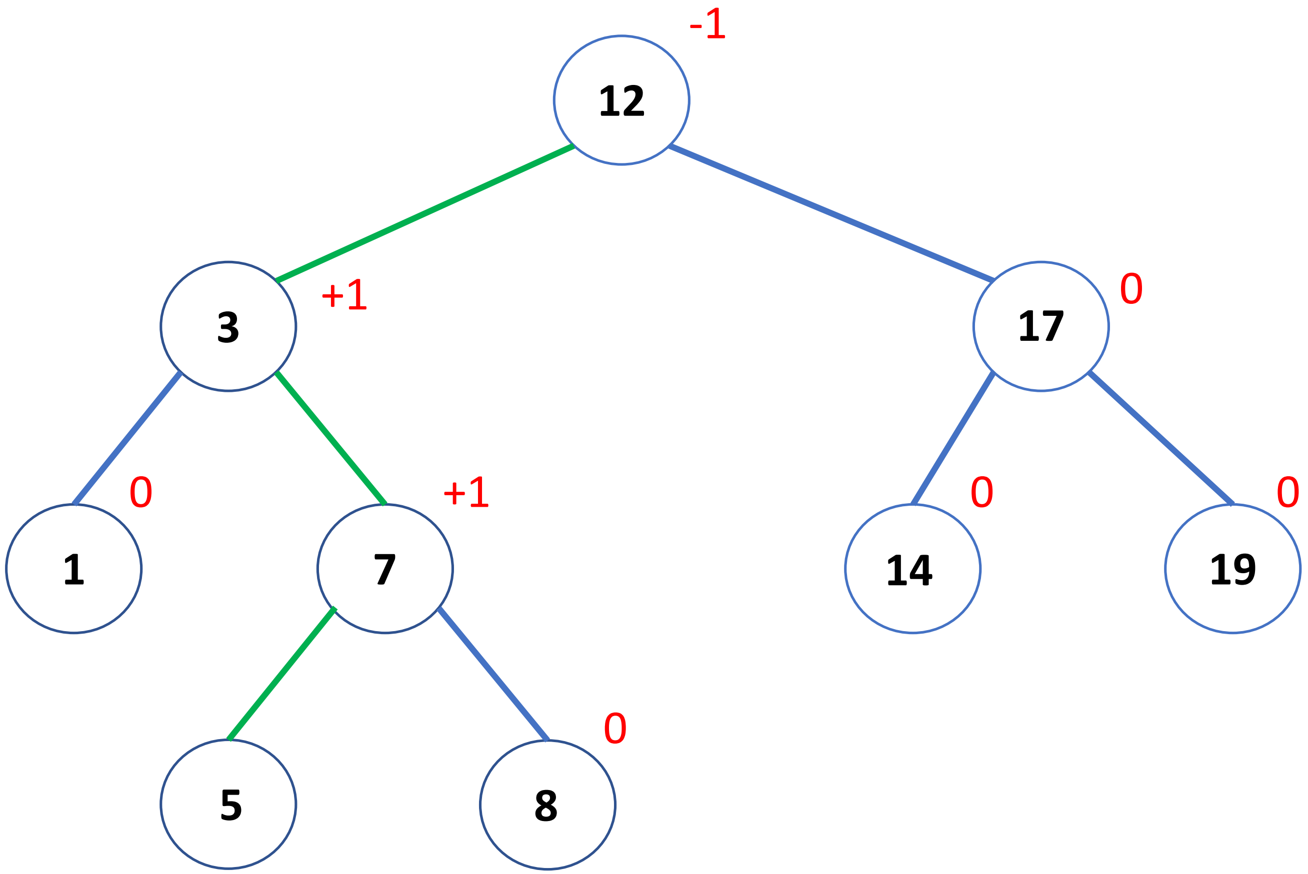
Example: AVL of {1, 3, 7, 8, 12, 14, 17, 19}

Insert(T, 5)



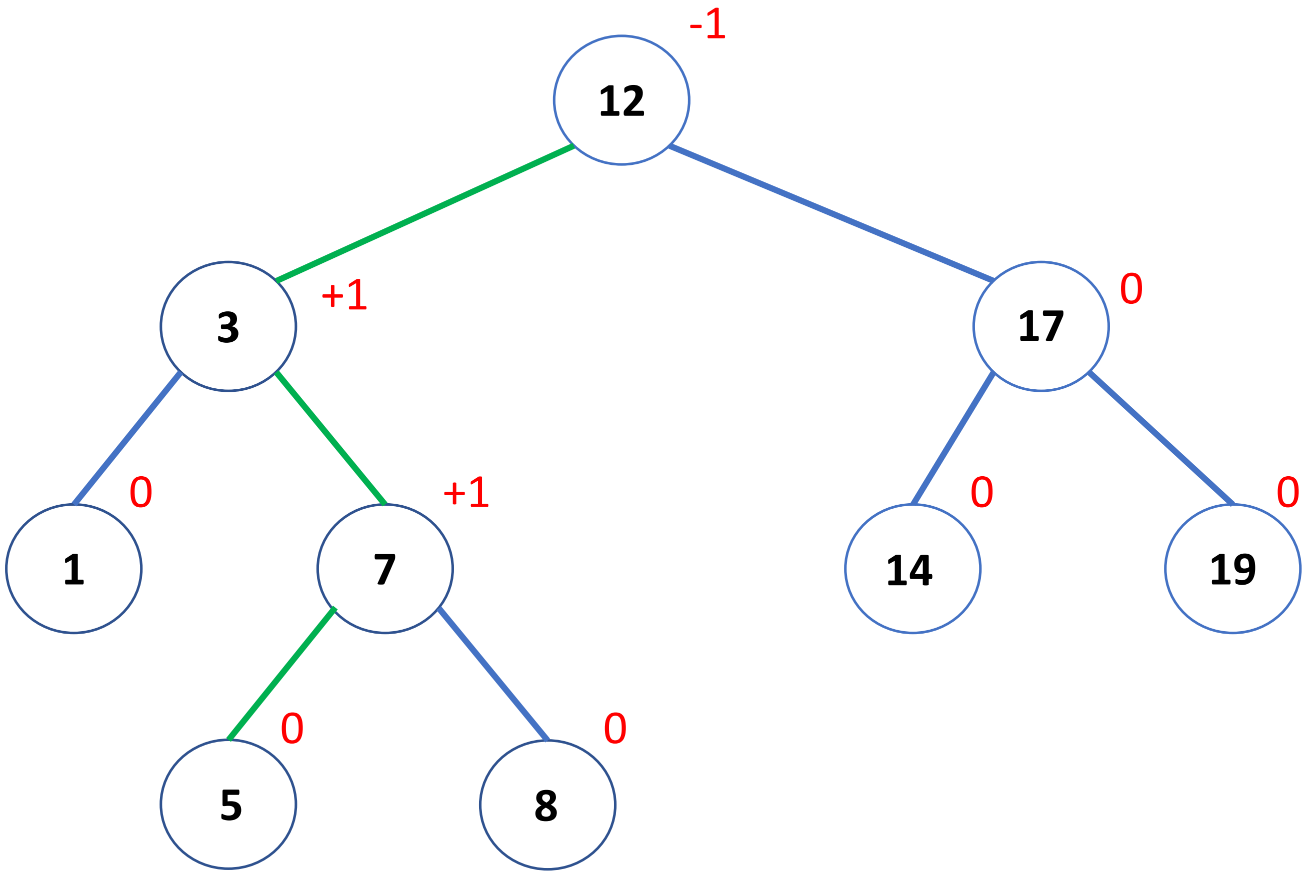
Example: AVL of {1, 3, 7, 8, 12, 14, 17, 19}

Insert(T, 5)



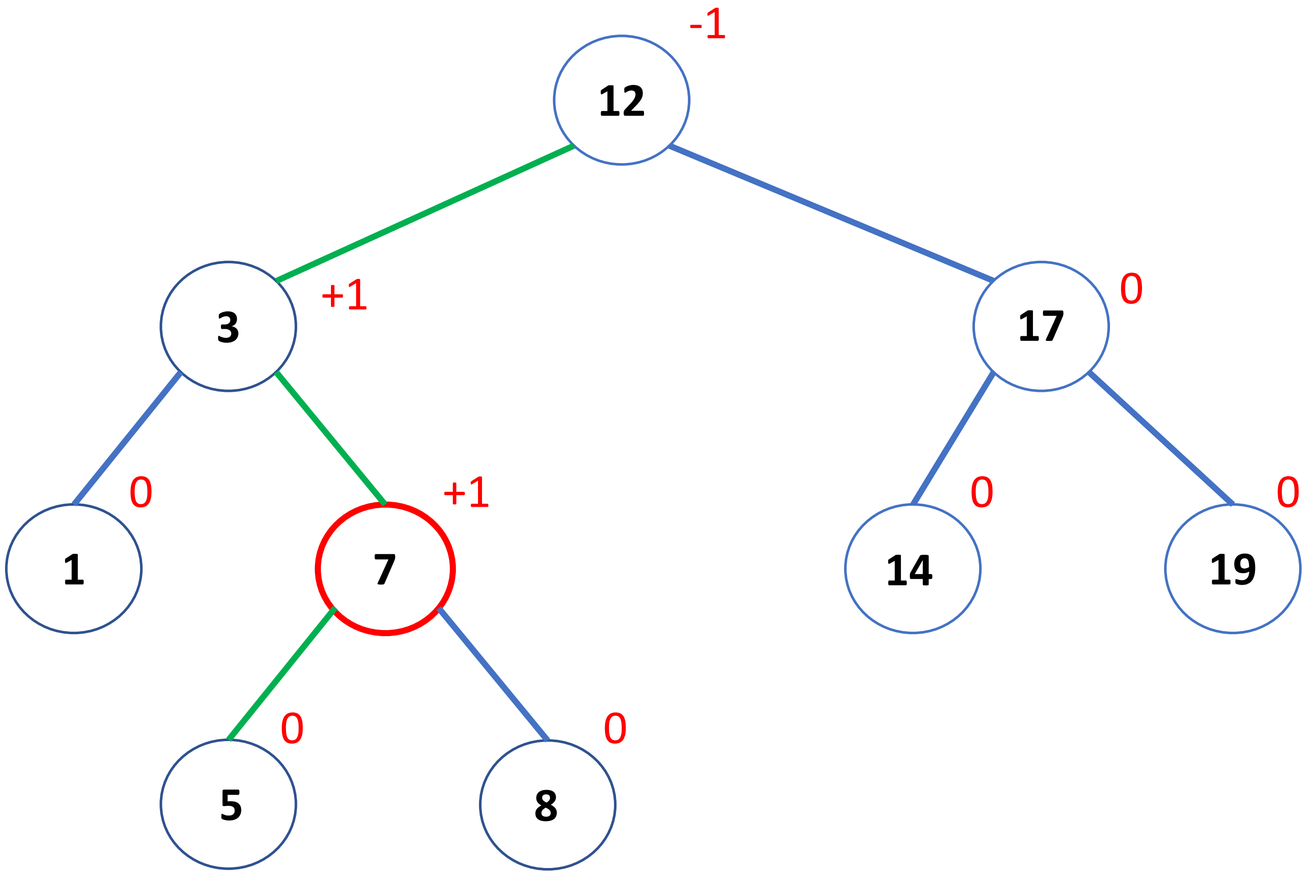
Example: AVL of {1, 3, 7, 8, 12, 14, 17, 19}

Insert(T, 5)



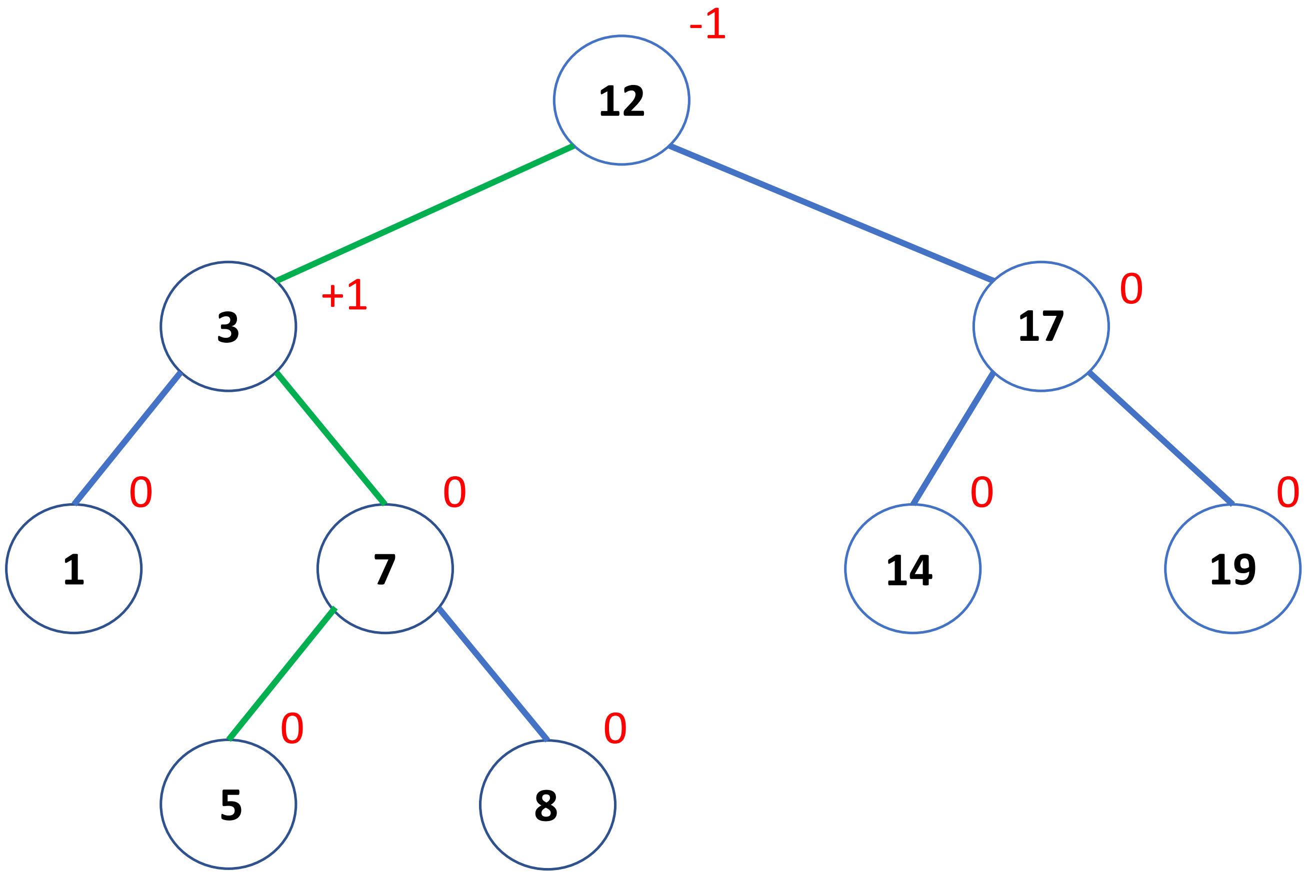
Example: AVL of {1, 3, 7, 8, 12, 14, 17, 19}

Insert(T, 5)



Example: AVL of {1, 3, 7, 8, 12, 14, 17, 19}

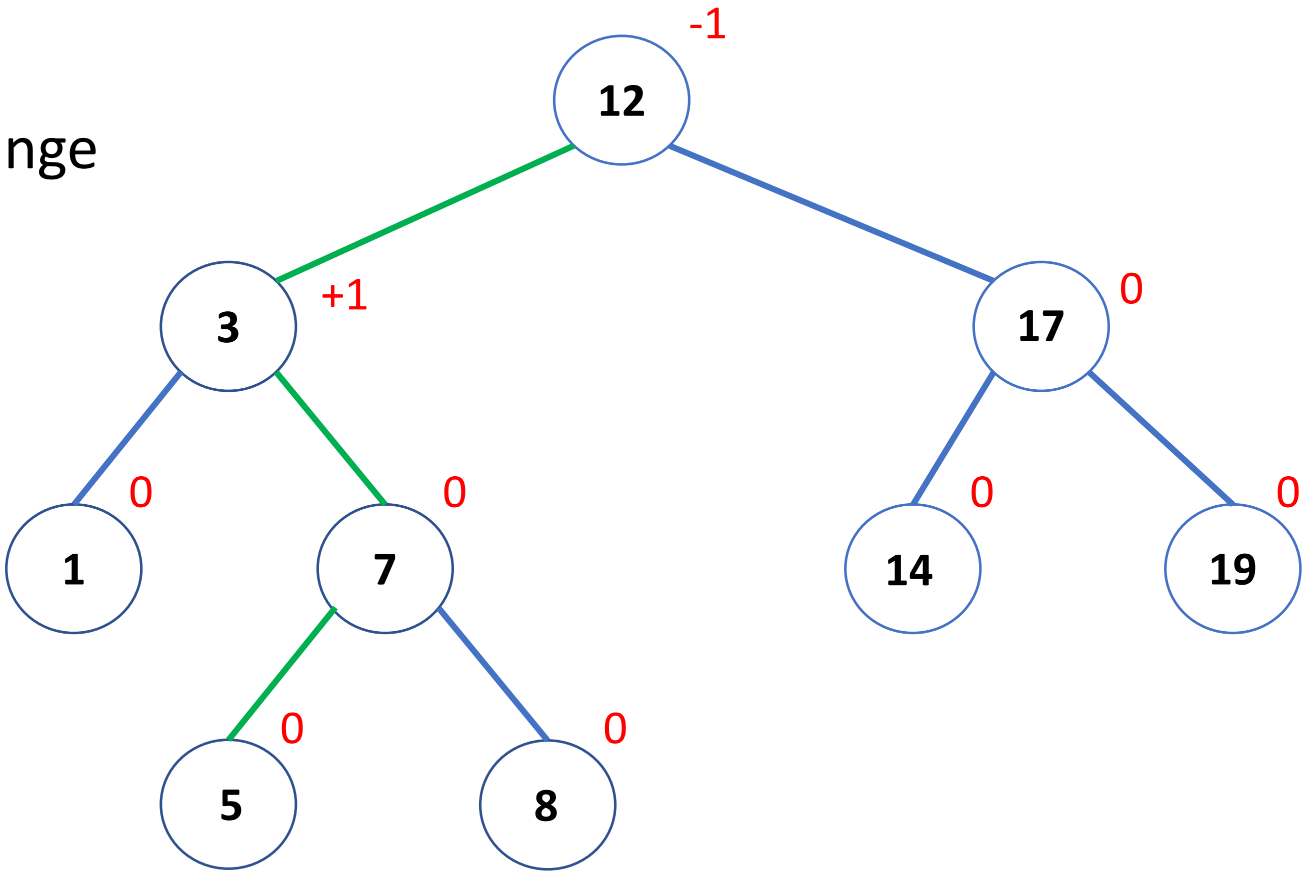
Insert(T, 5)



Example: AVL of {1, 3, 7, 8, 12, 14, 17, 19}

Insert(T, 5):

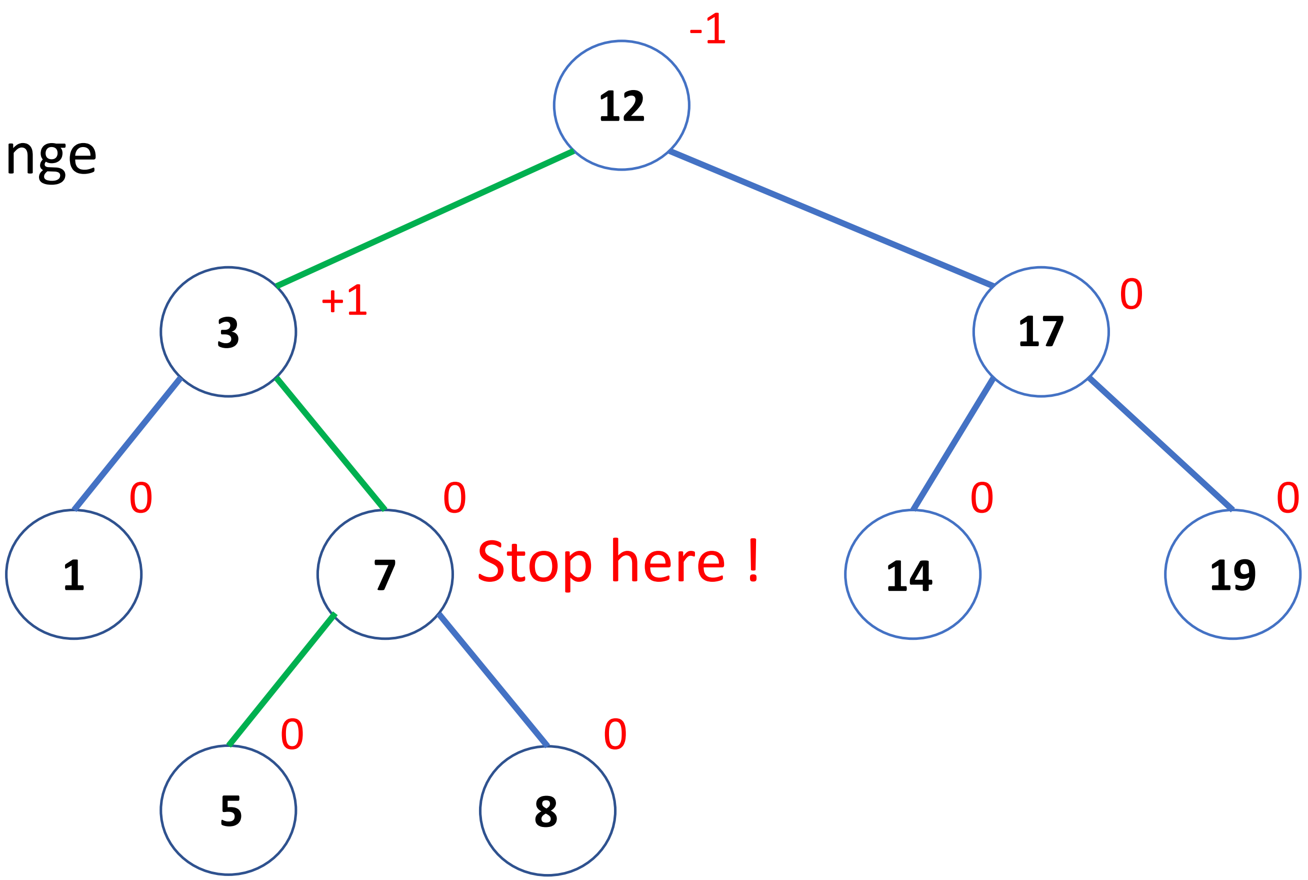
height(7) doesn't change



Example: AVL of {1, 3, 7, 8, 12, 14, 17, 19}

Insert(T, 5):

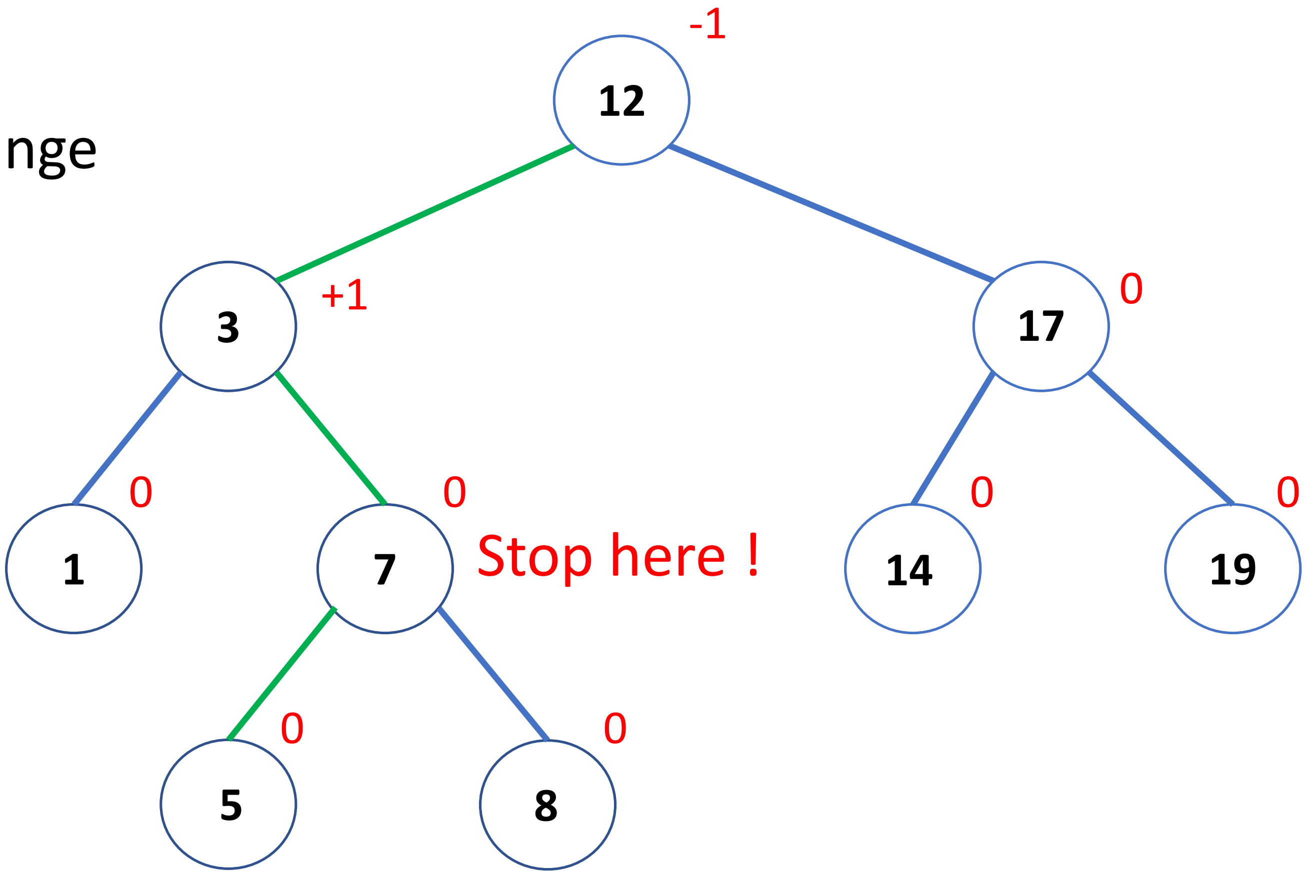
height(7) doesn't change
⇒ Stop at 7 !



Example: AVL of {1, 3, 7, 8, 12, 14, 17, 19}

Insert(T, 5):

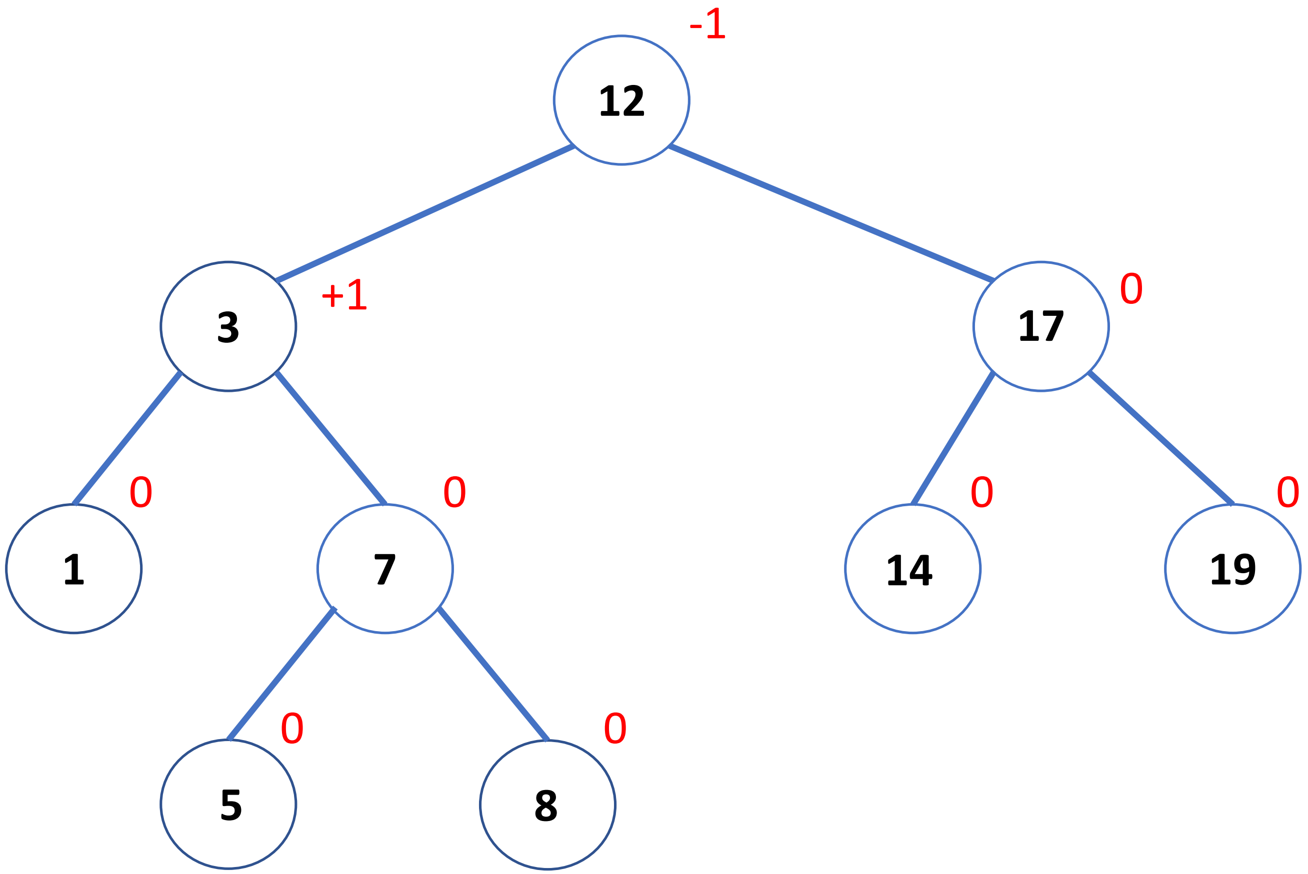
height(7) doesn't change
⇒ Stop at 7 !



No rebalancing needed !

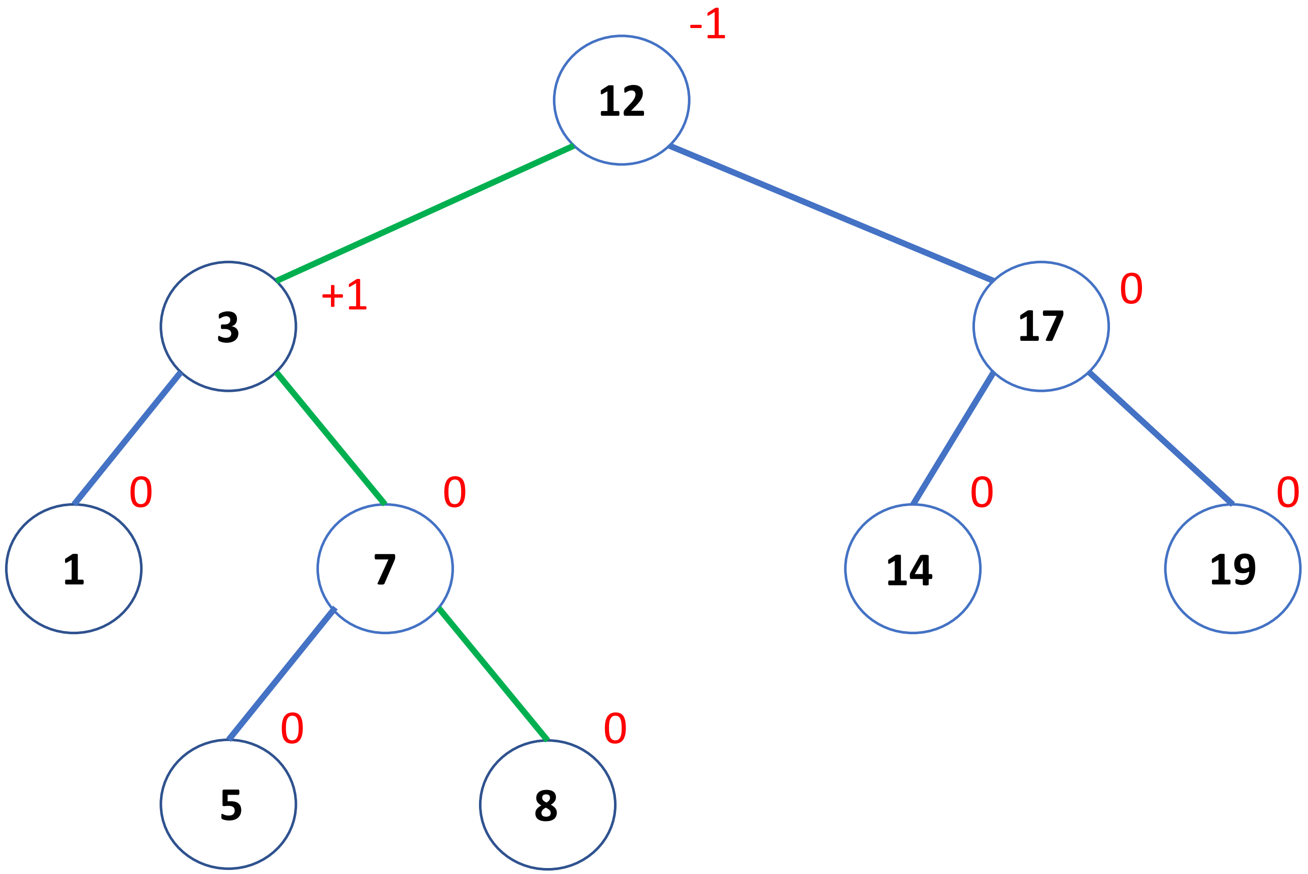
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 10)



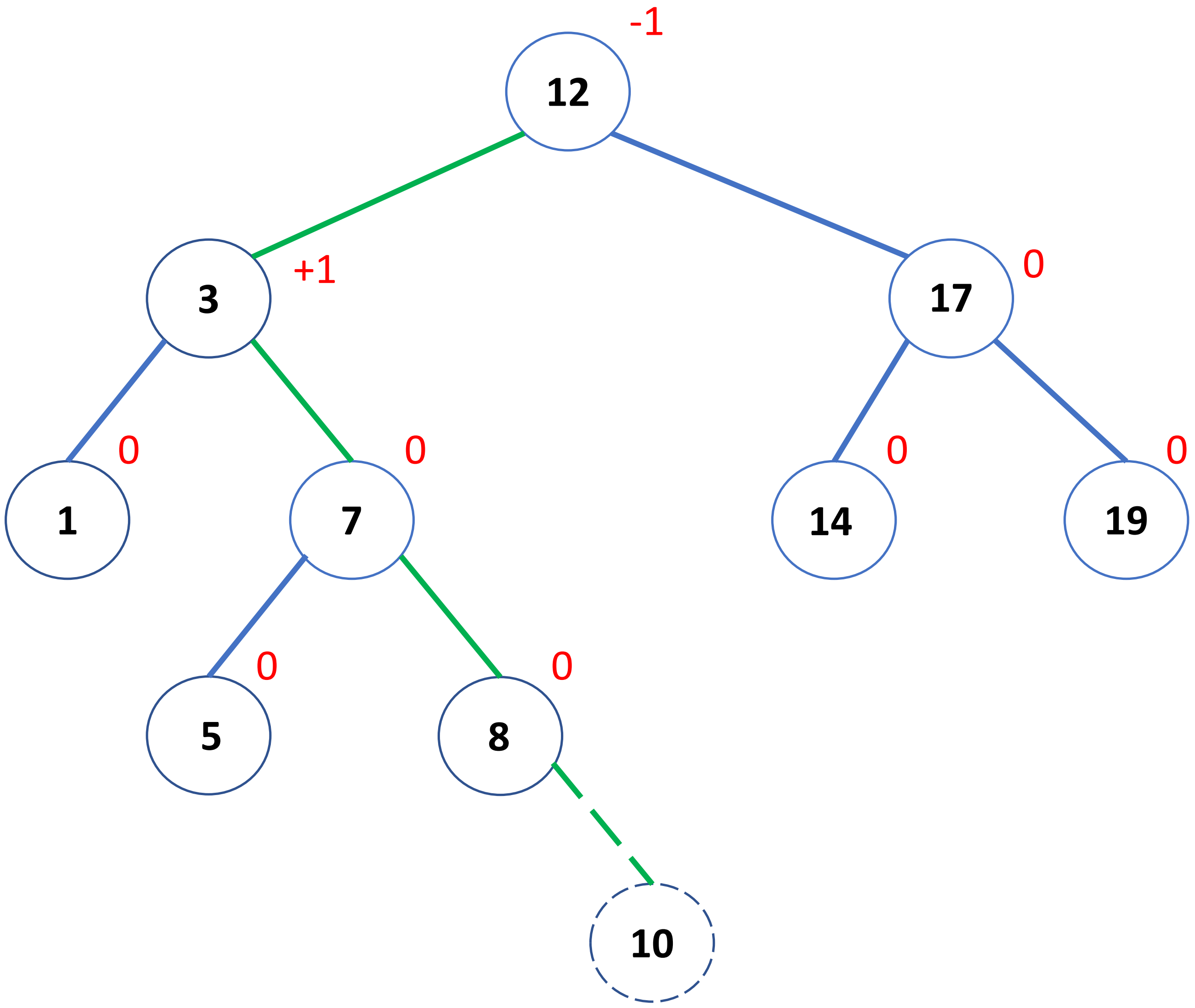
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 10)



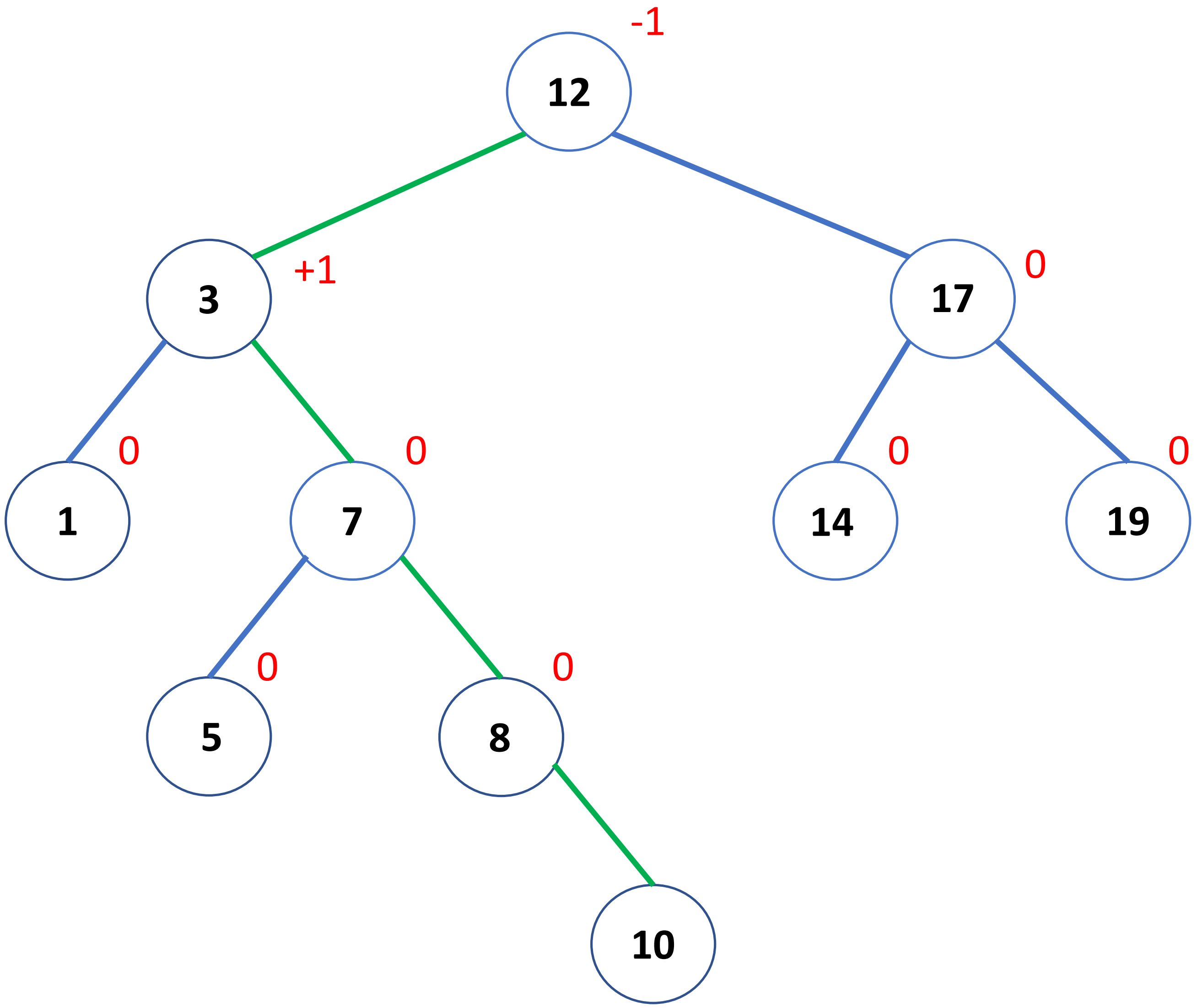
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 10)



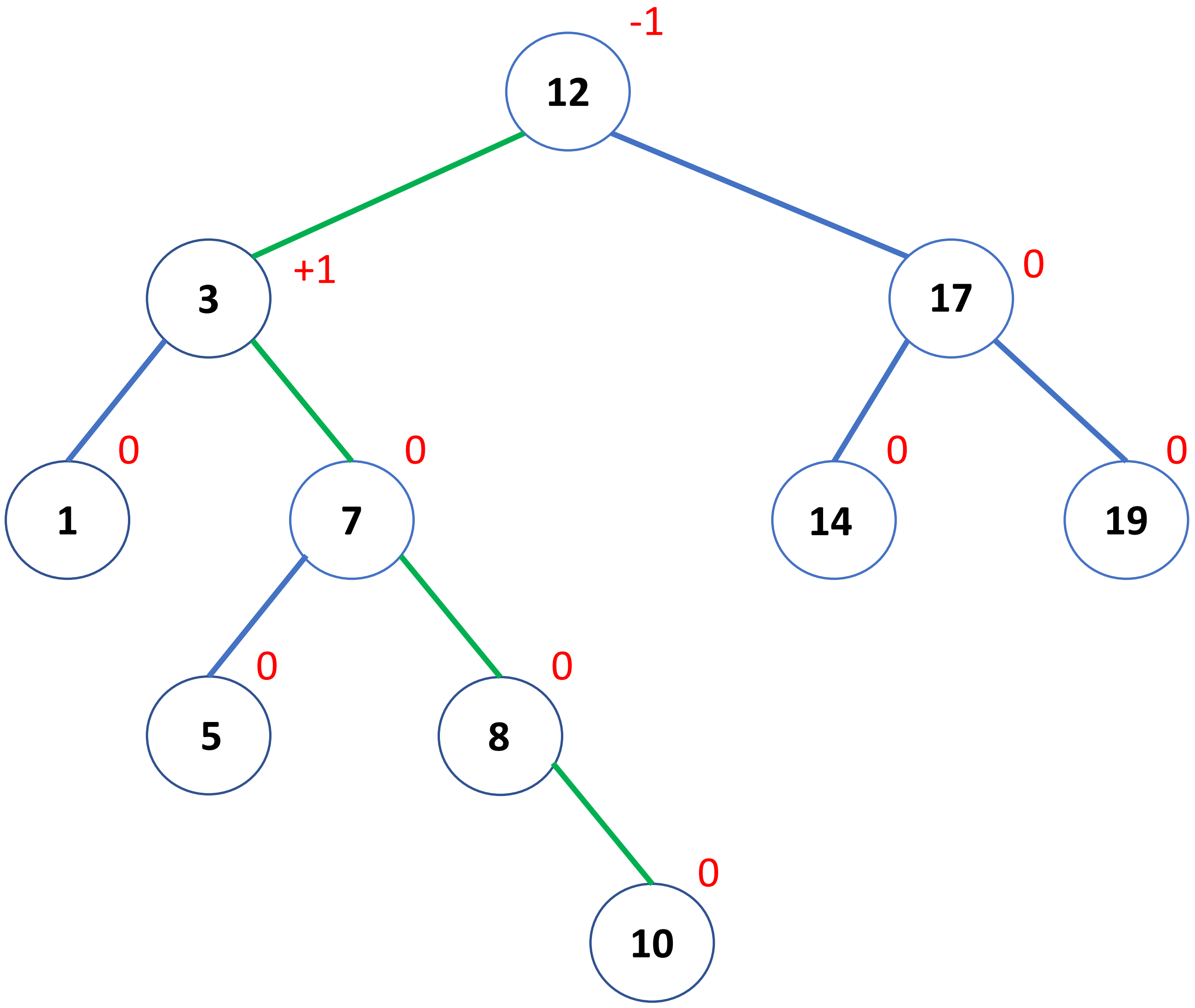
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 10)



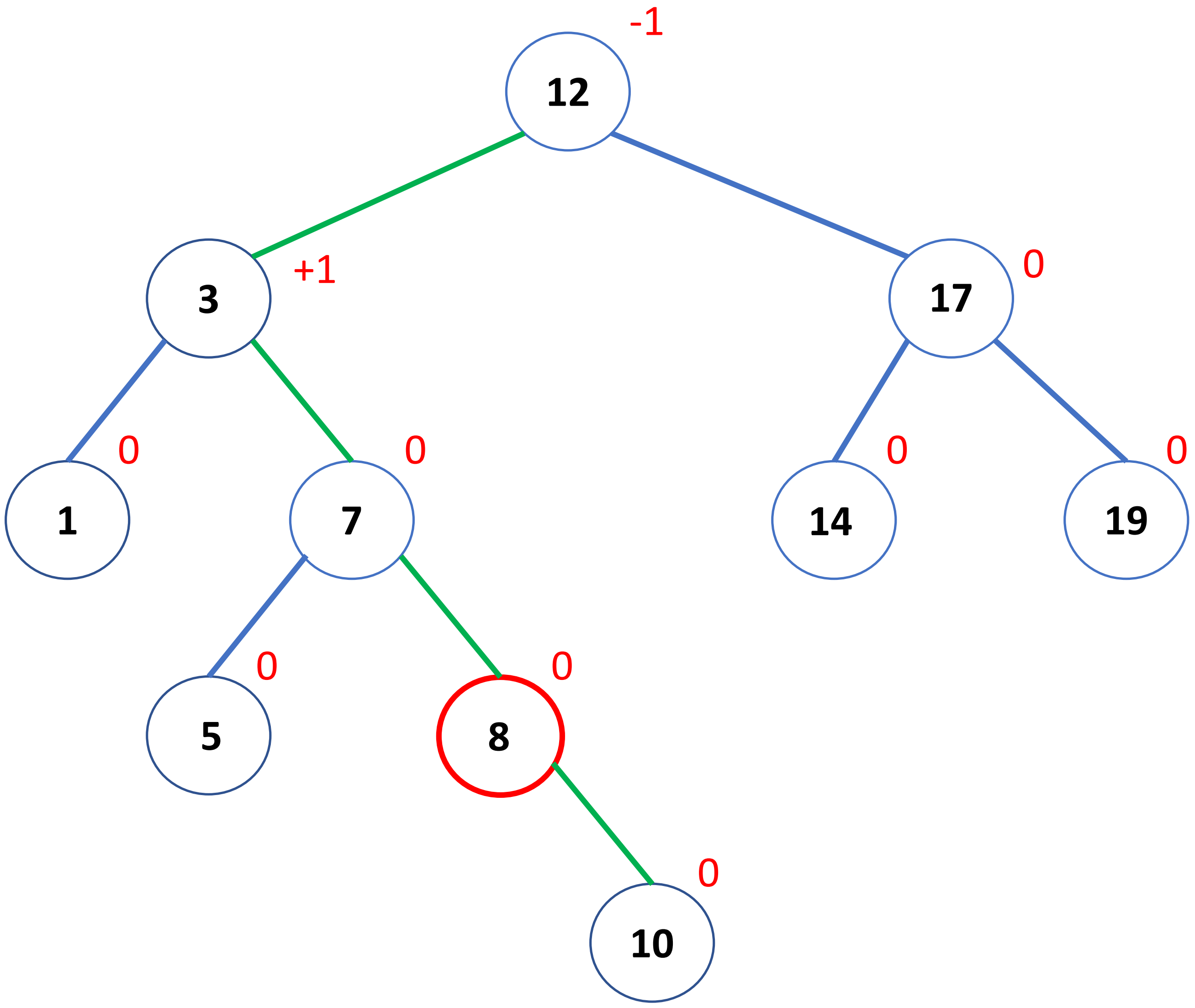
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 10)



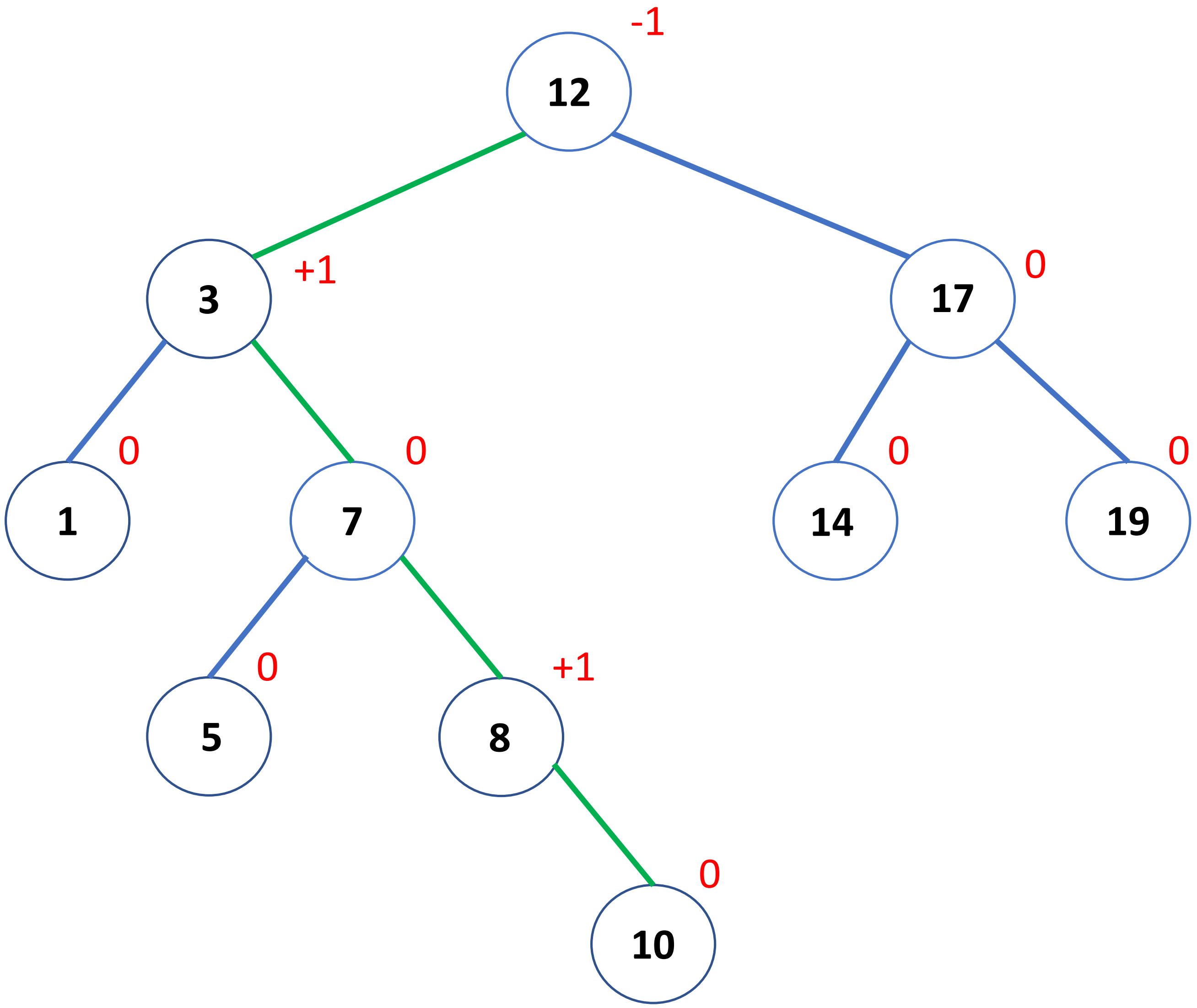
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 10)



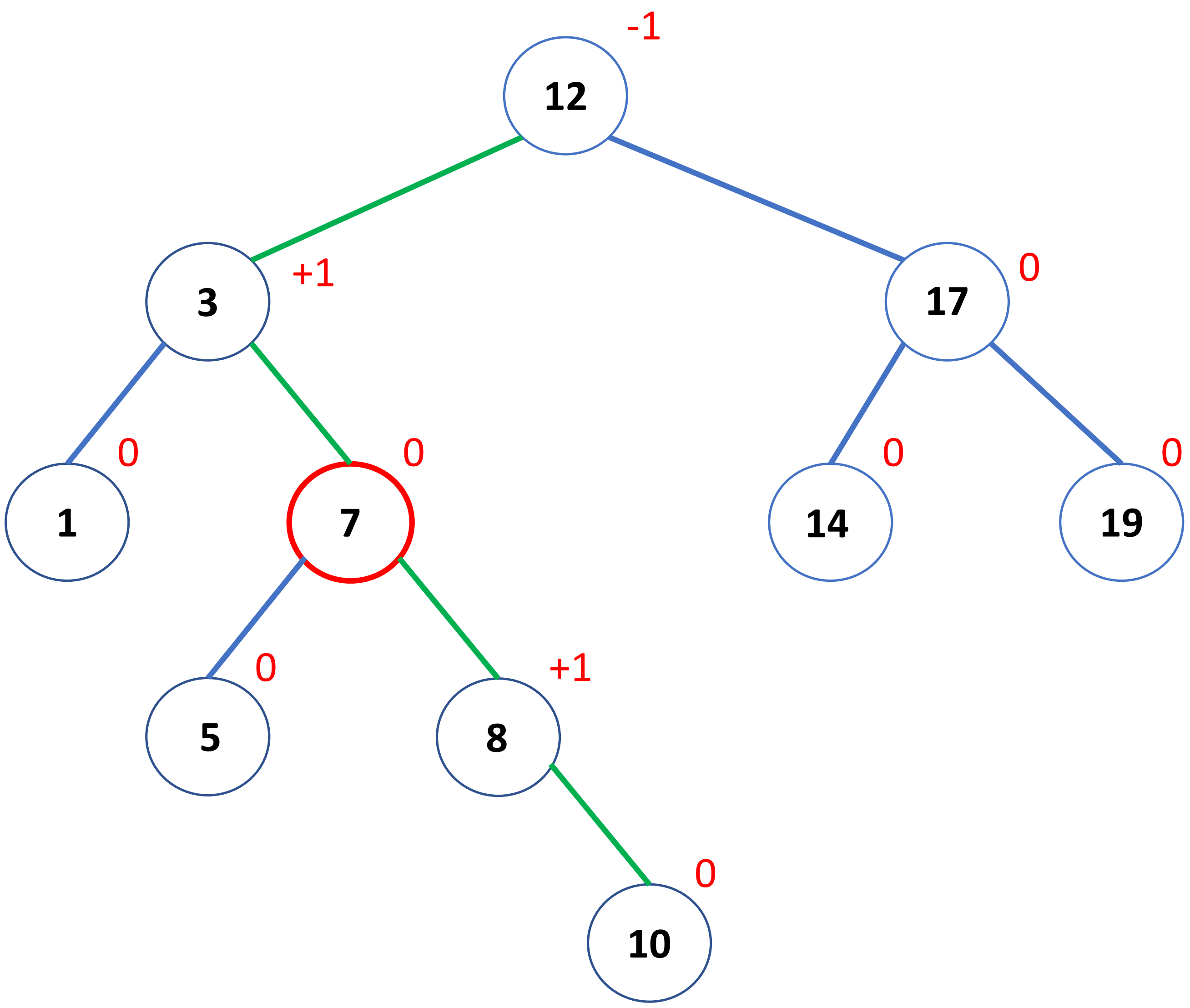
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 10)



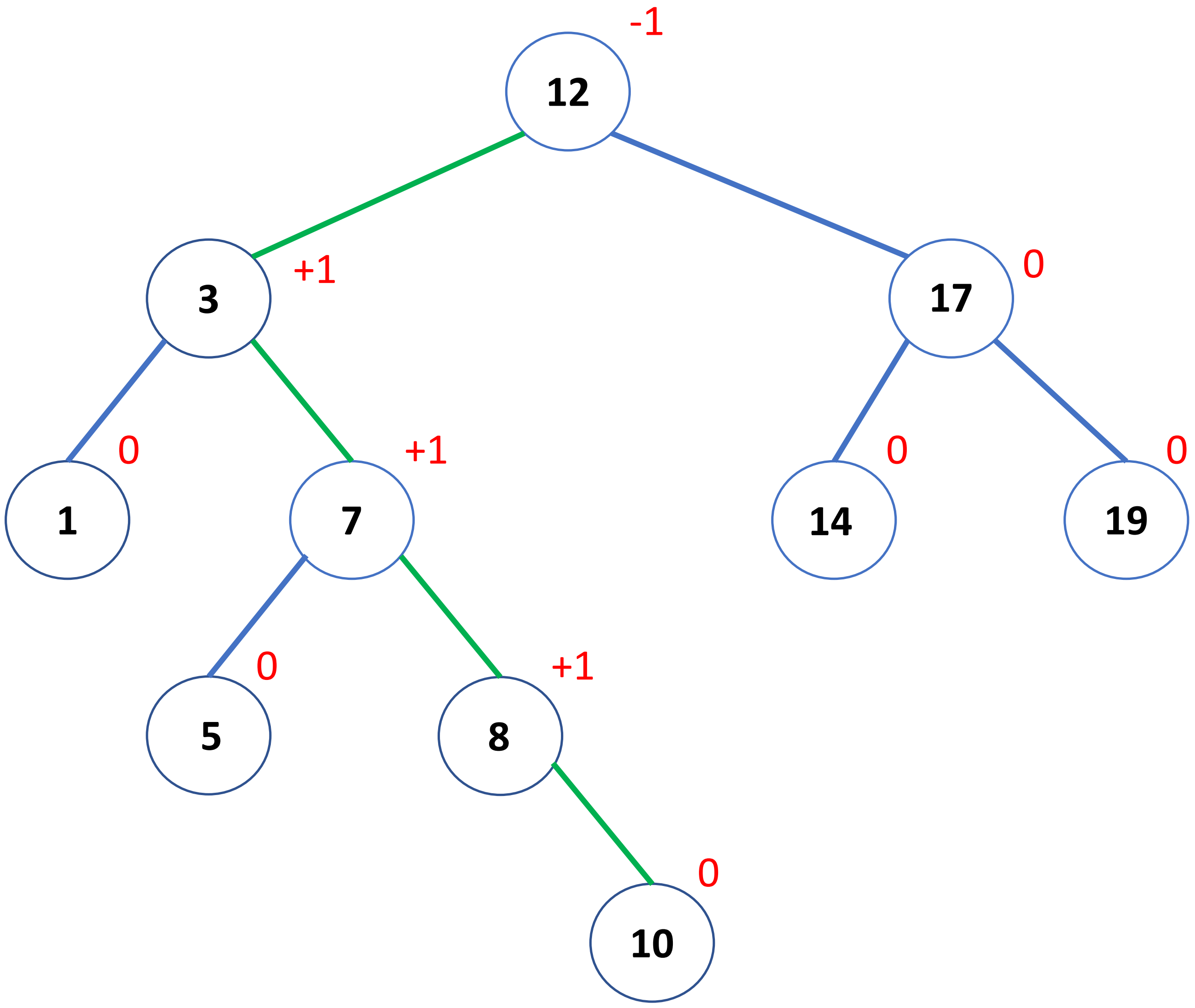
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 10)



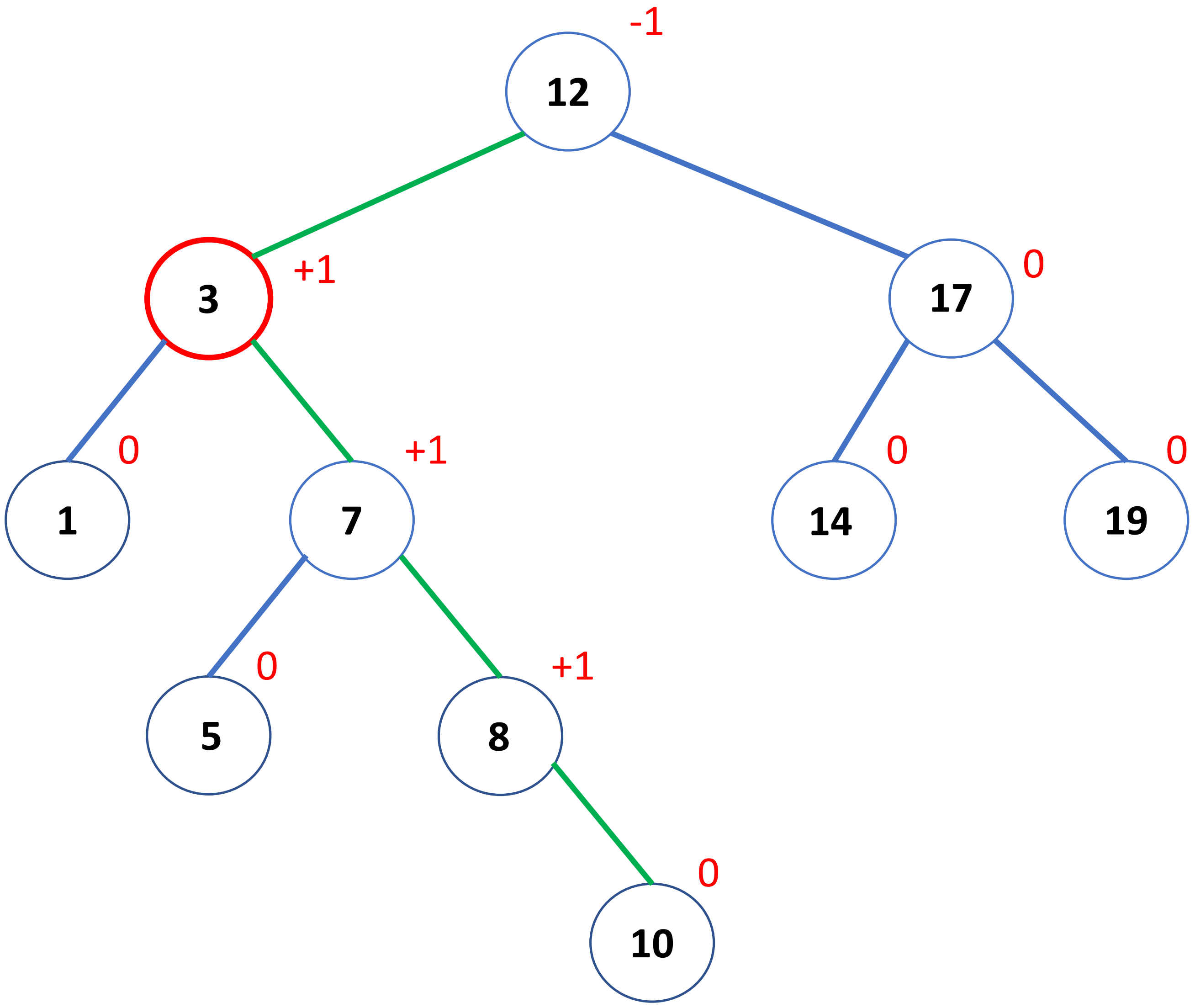
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 10)



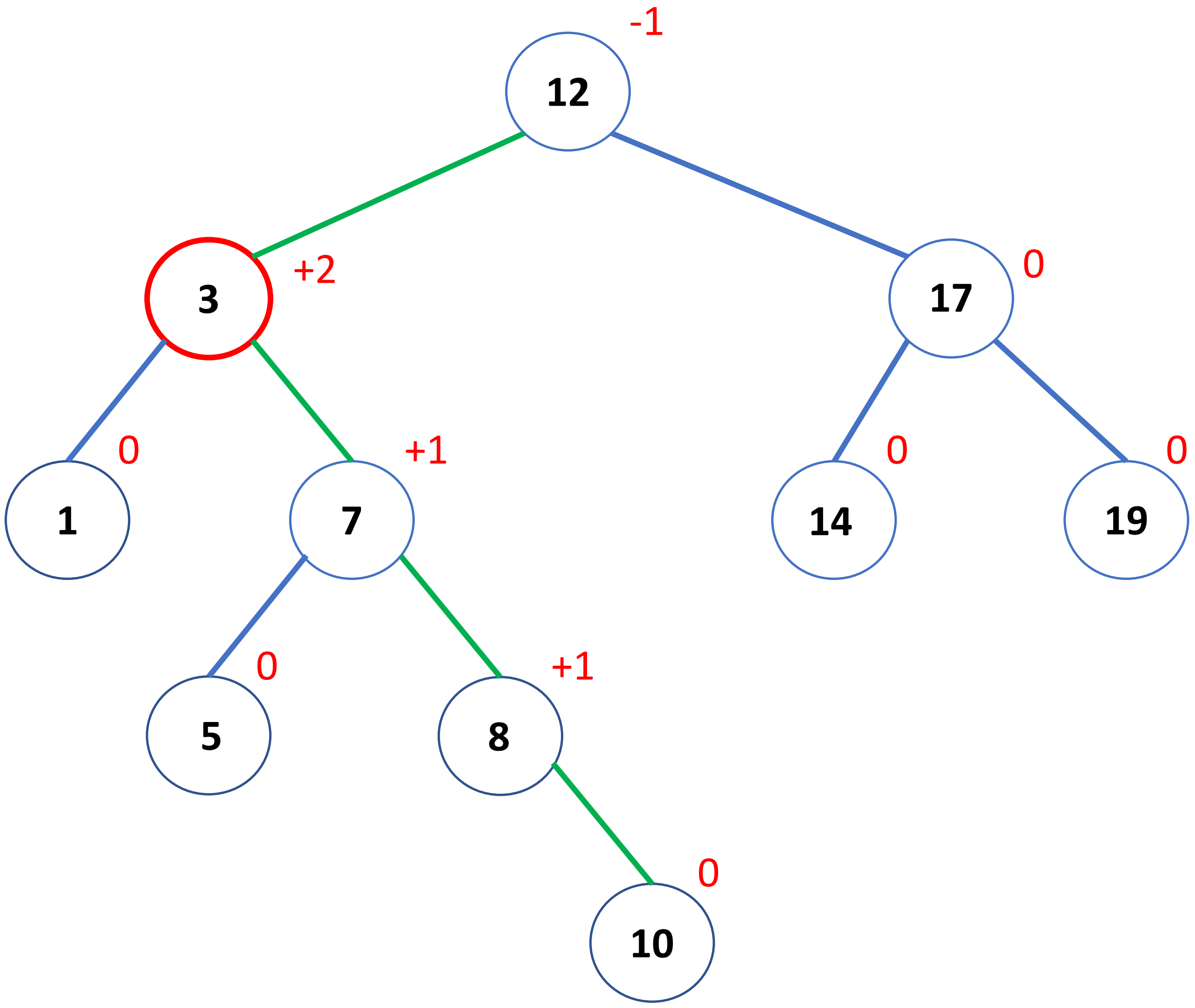
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 10)



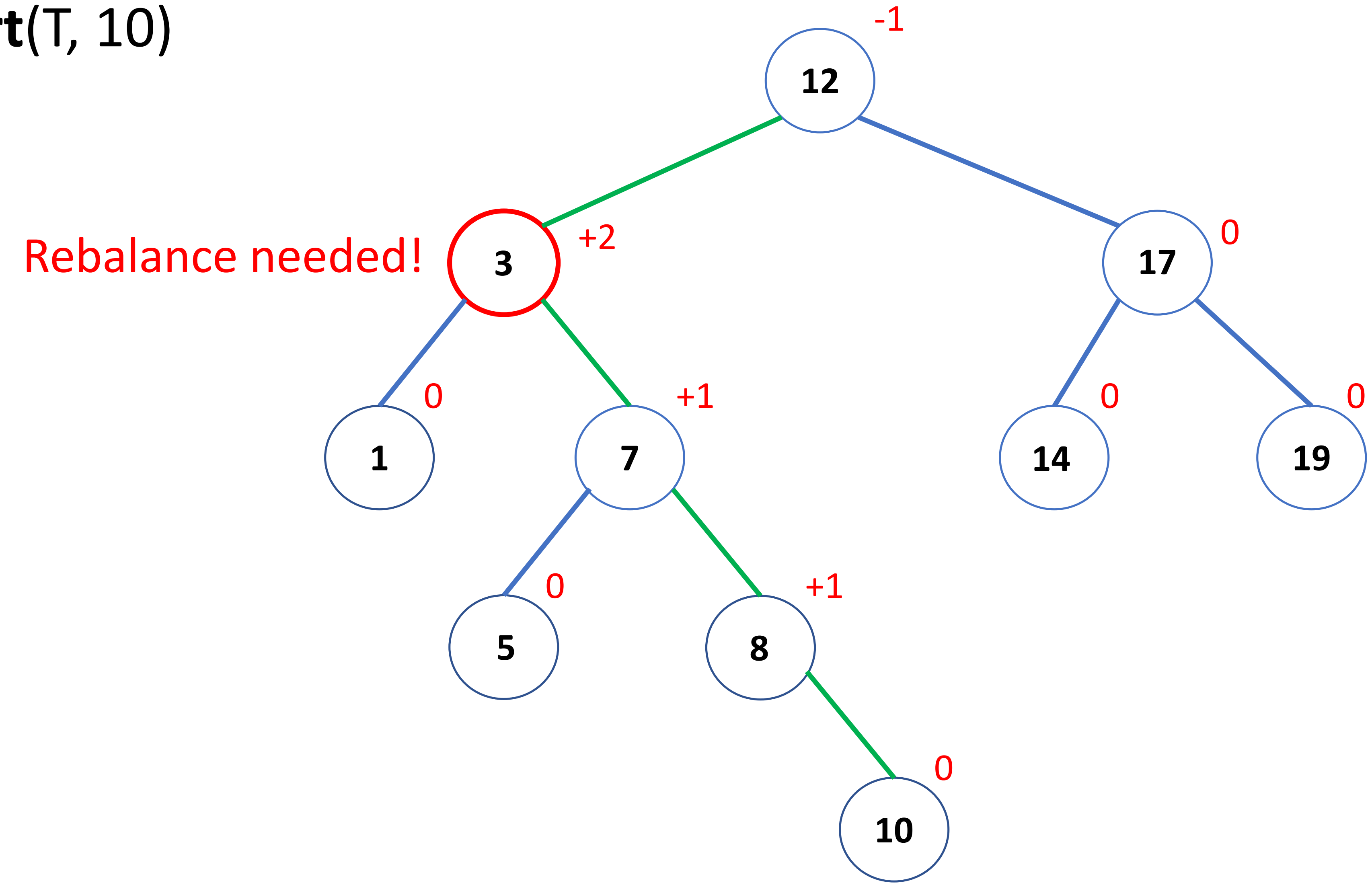
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 10)

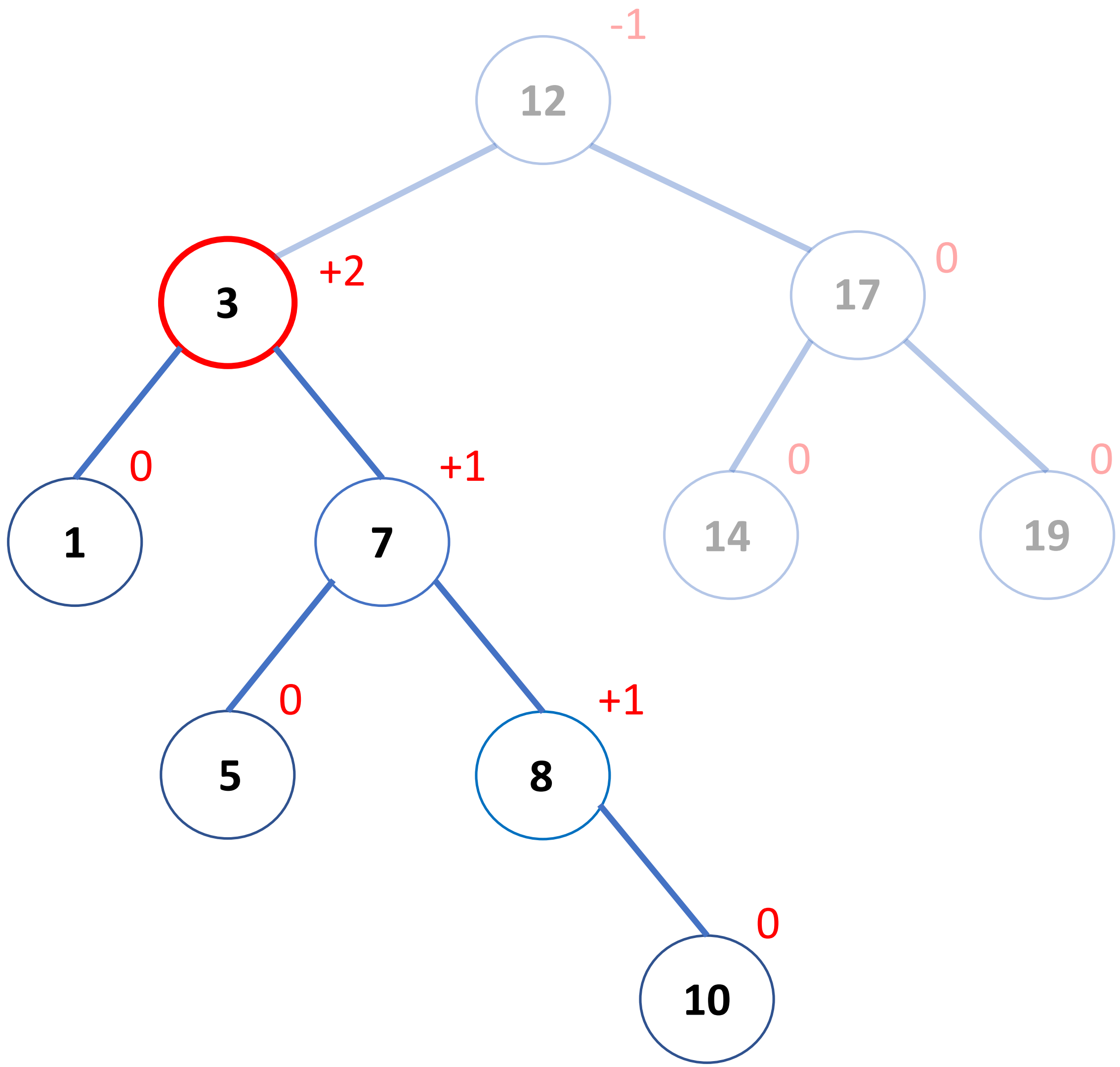


Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

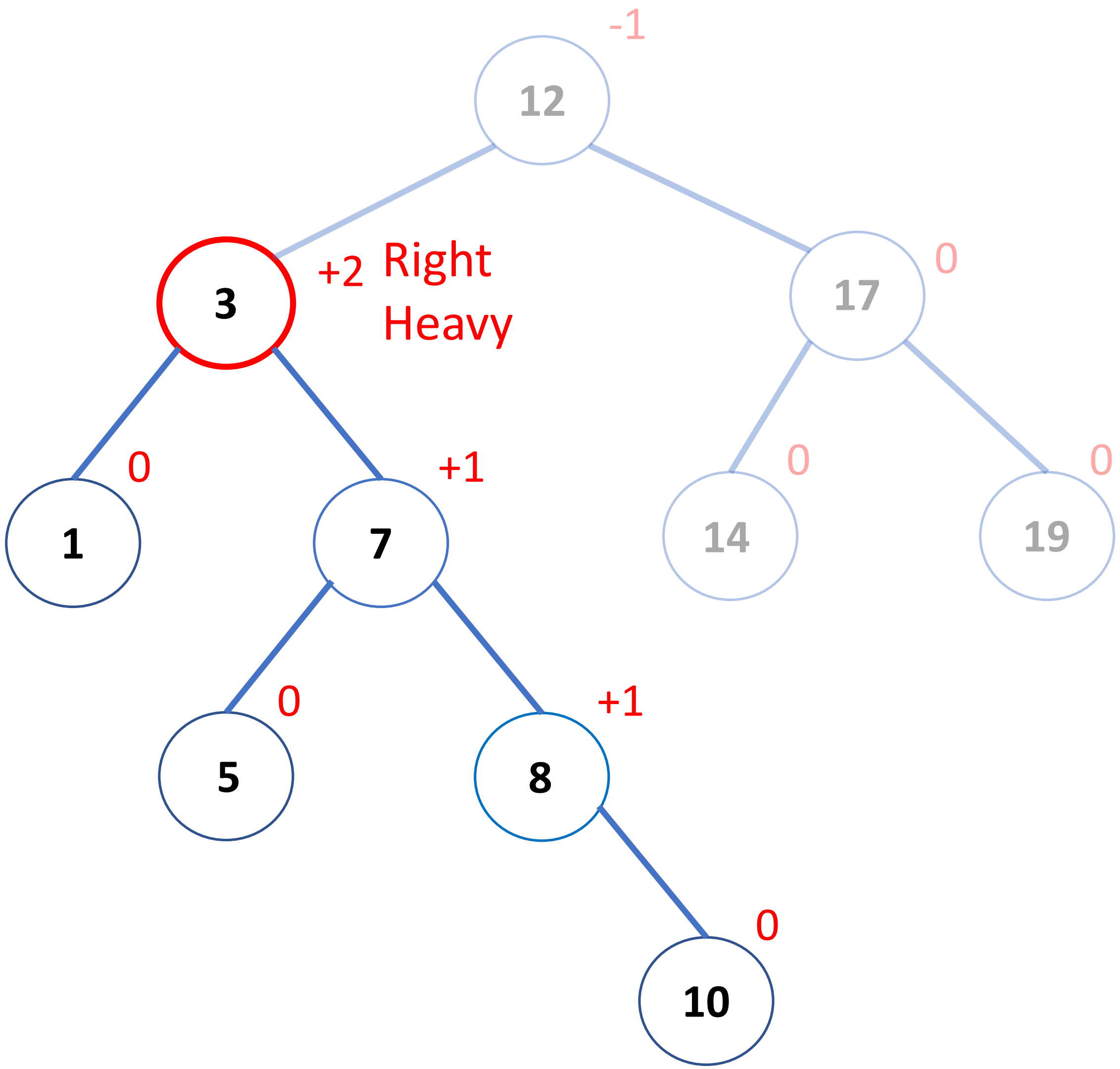
Insert(T, 10)



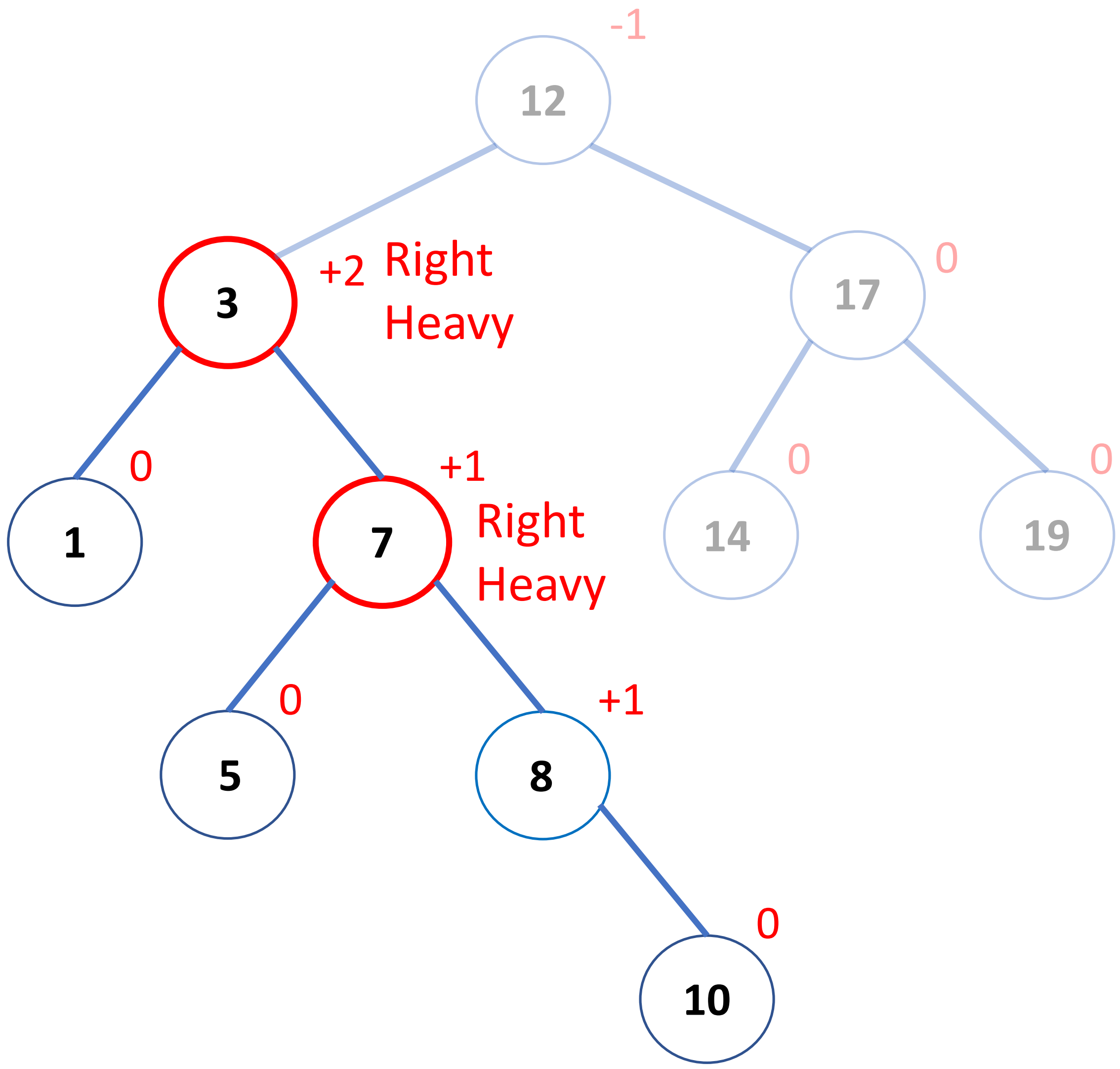
Rebalancing left subtree of 12:



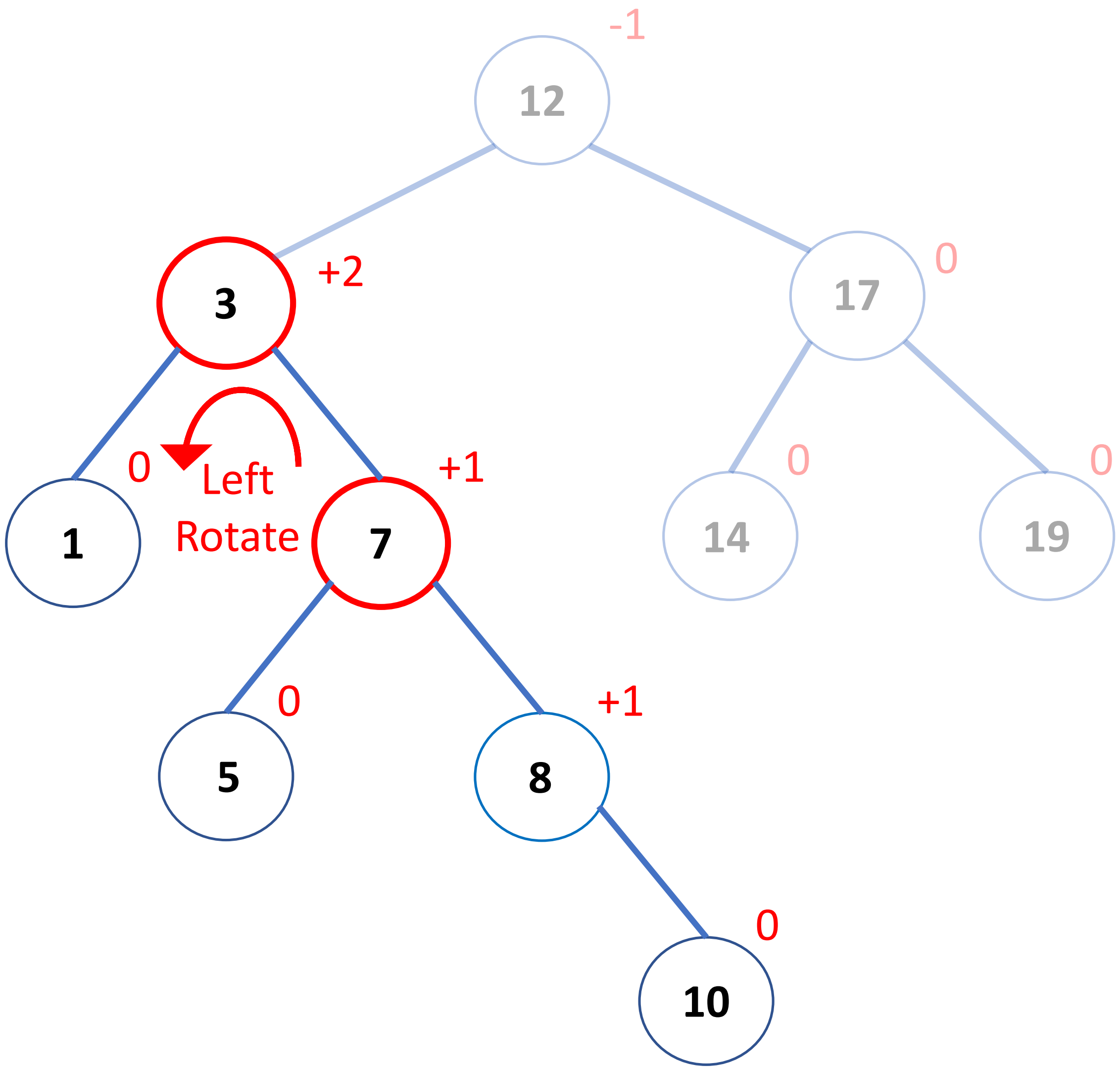
Rebalancing left subtree of 12:



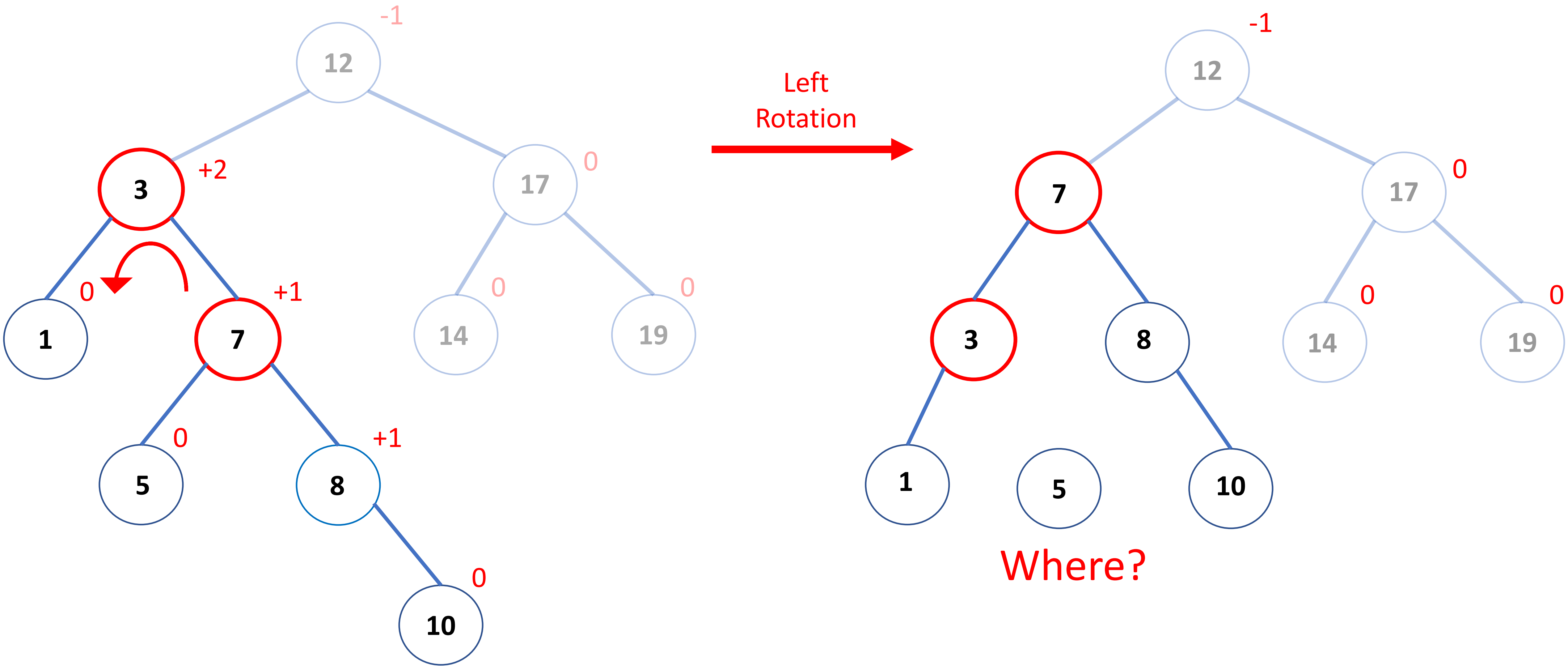
Rebalancing left subtree of 12:



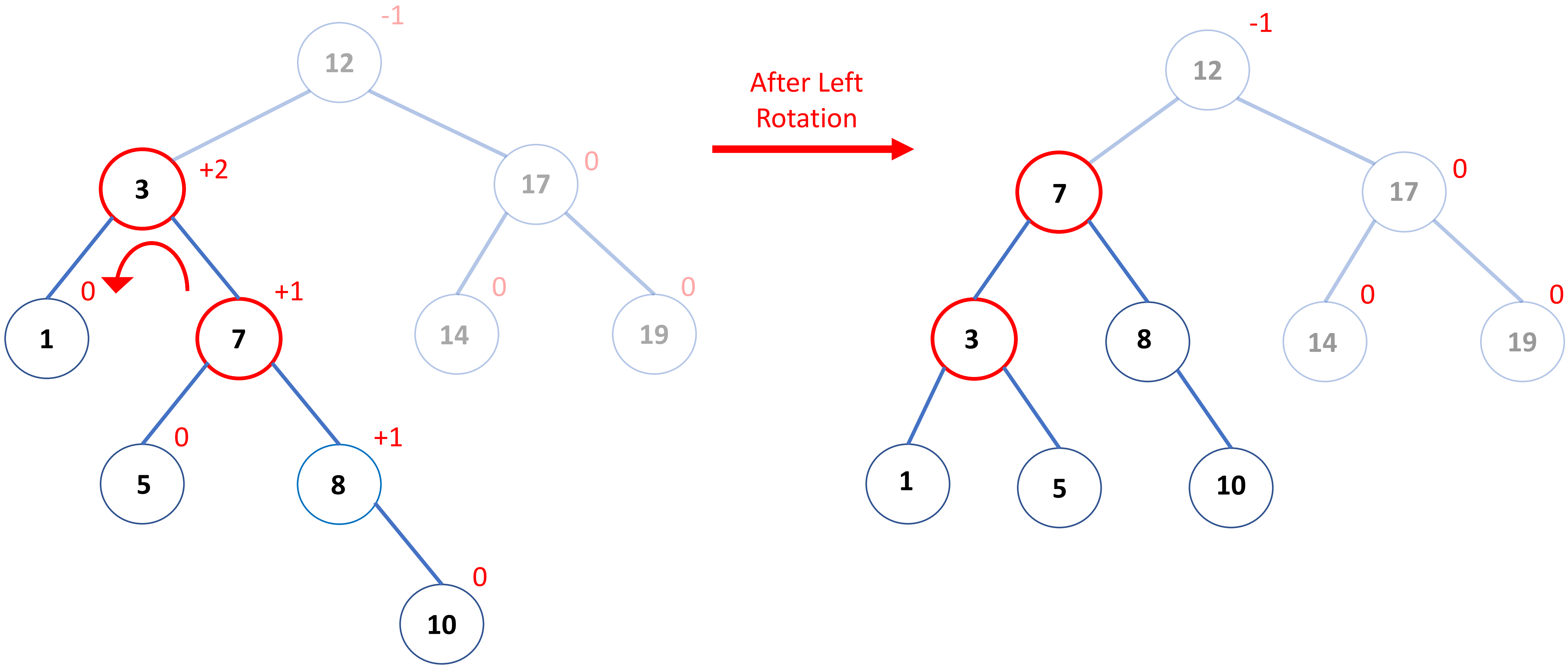
Rebalancing left subtree of 12:



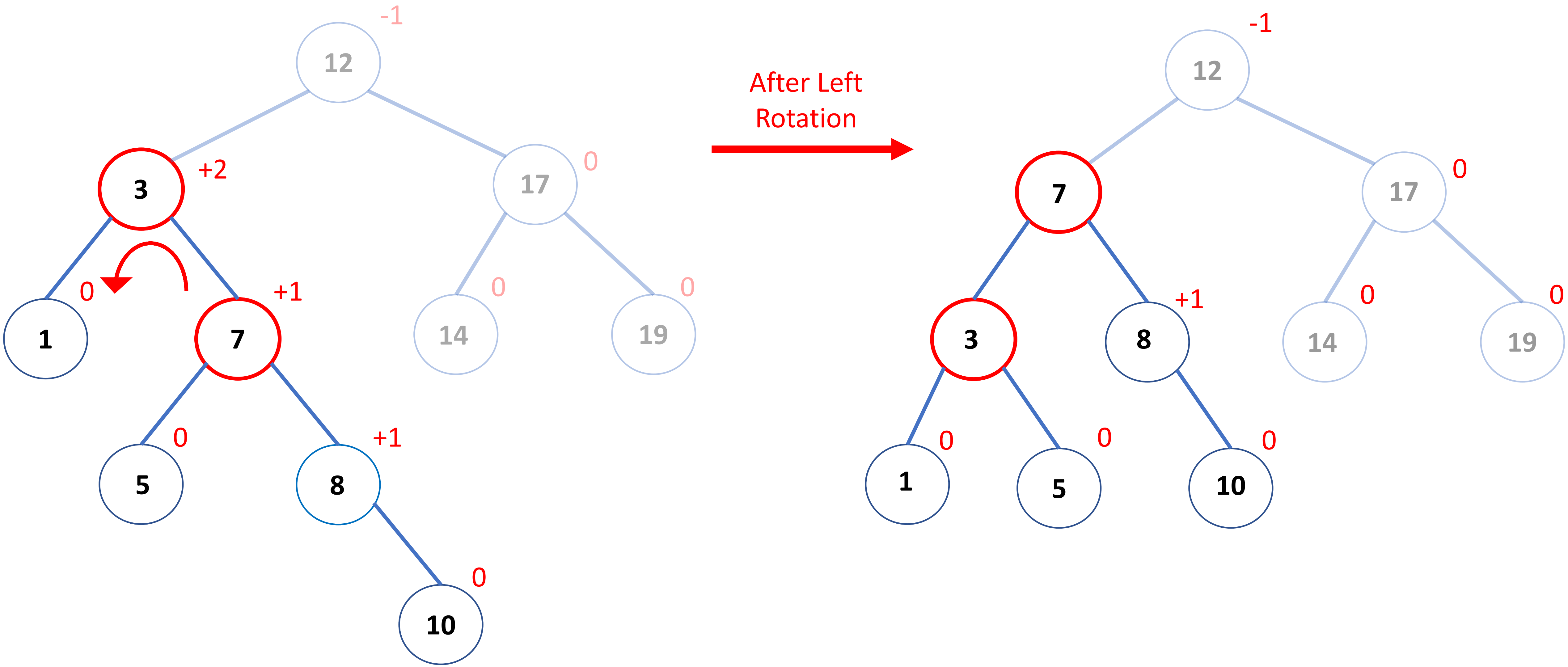
Rebalancing left subtree of 12:



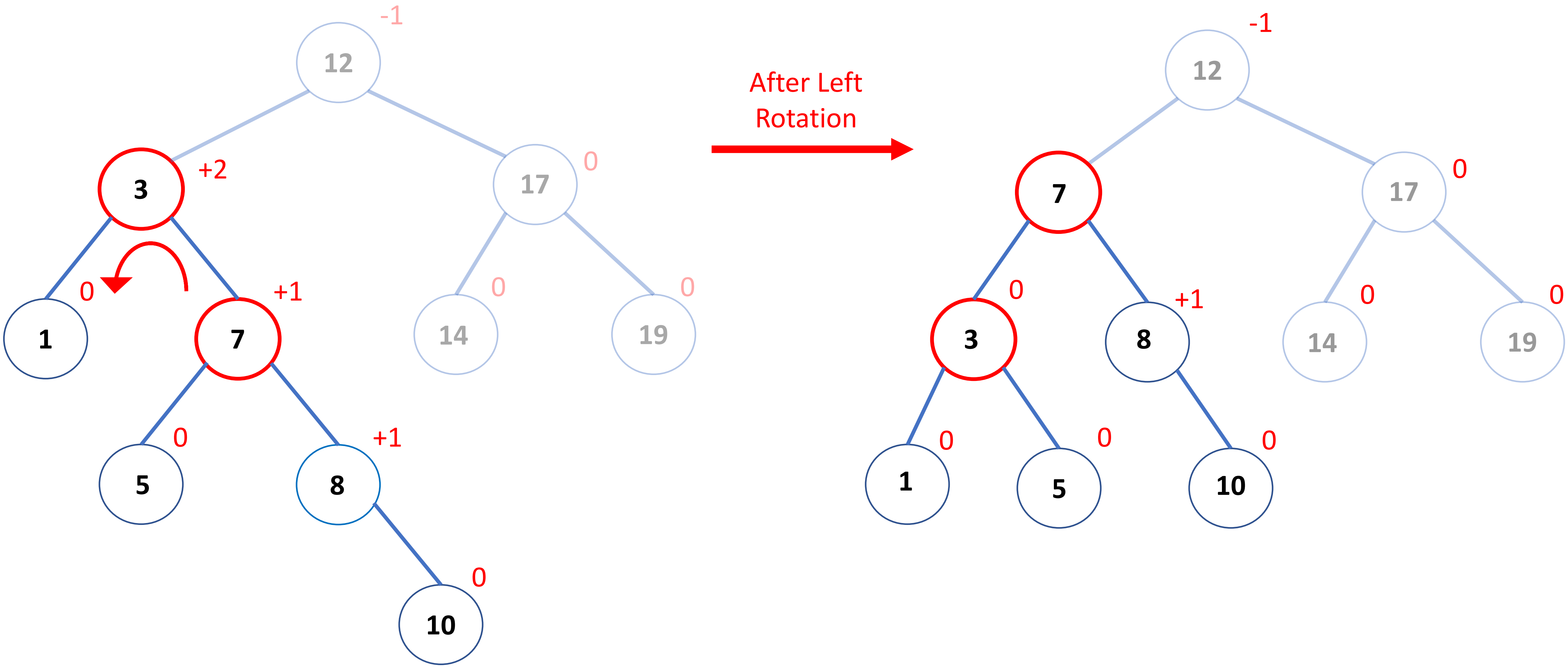
Rebalancing left subtree of 12:



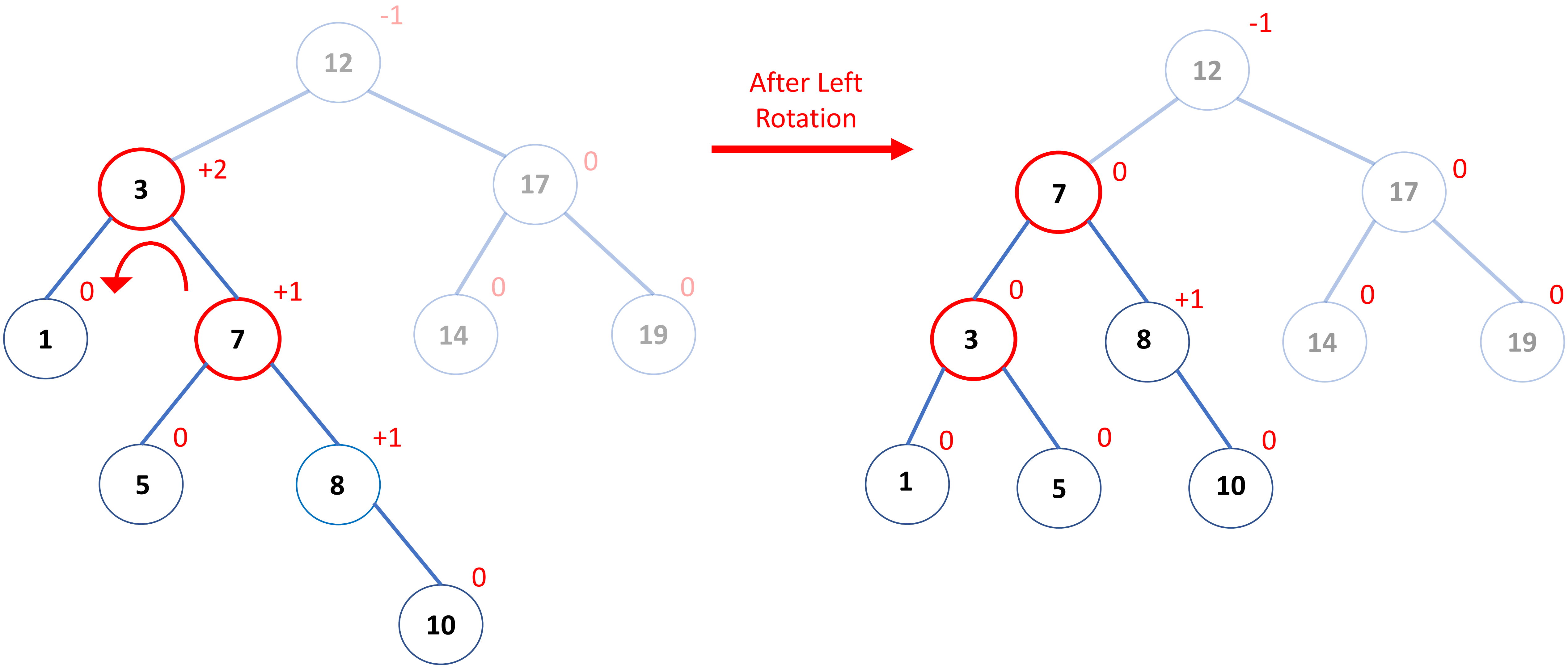
Rebalancing left subtree of 12:



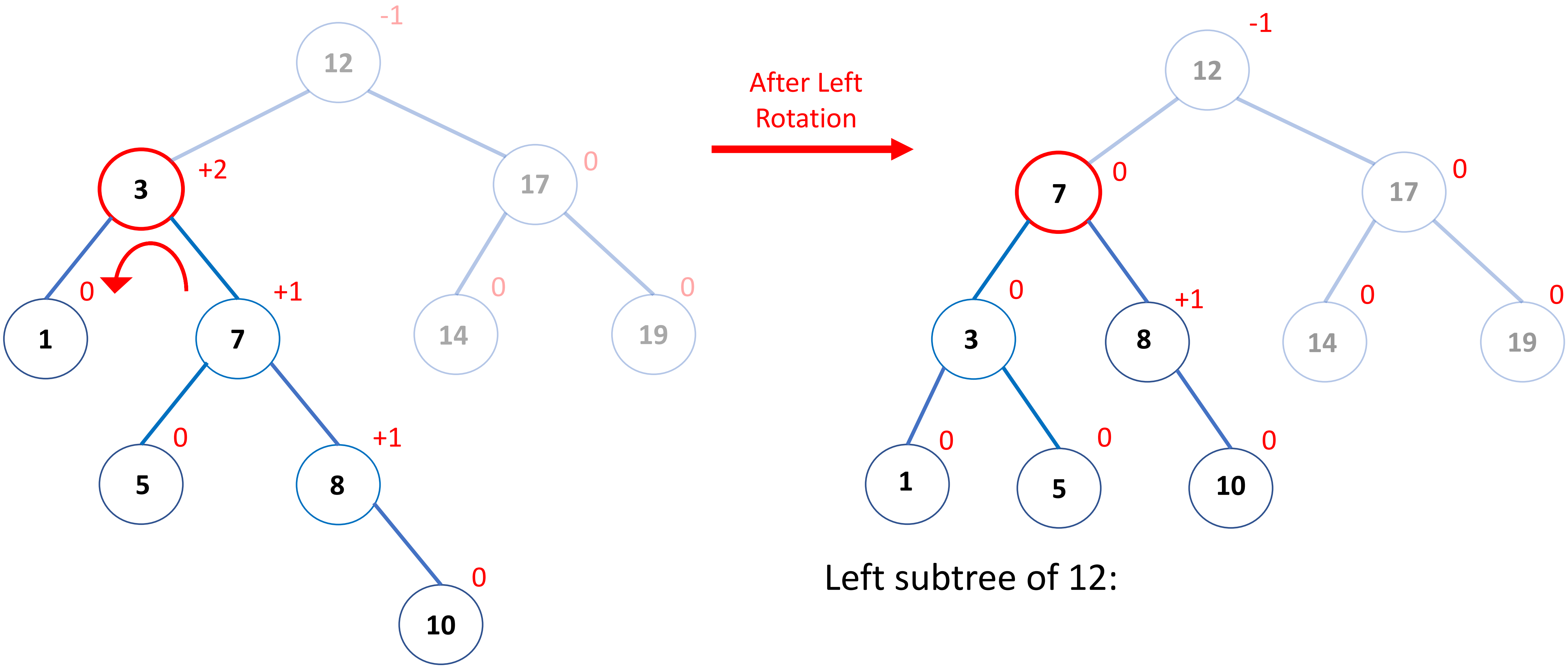
Rebalancing left subtree of 12:



Rebalancing left subtree of 12:

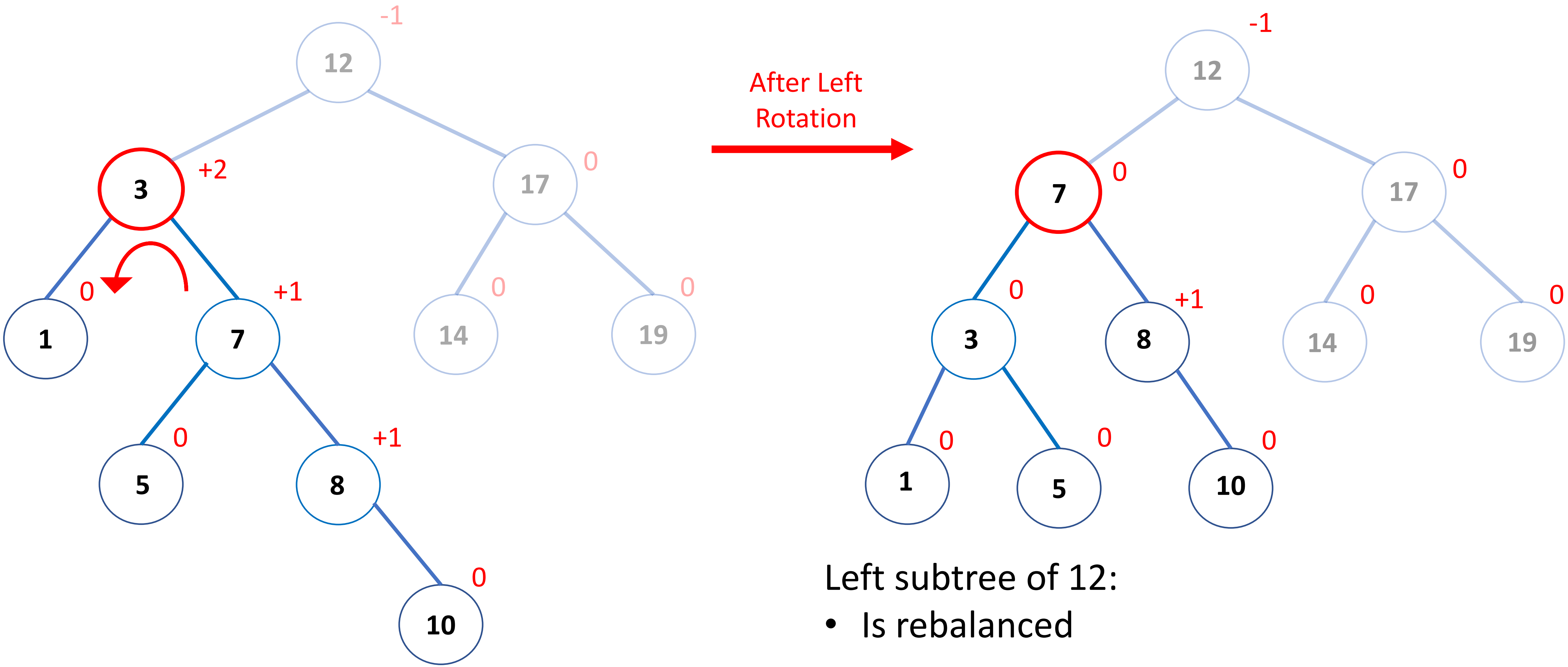


Rebalancing left subtree of 12:



Left subtree of 12:

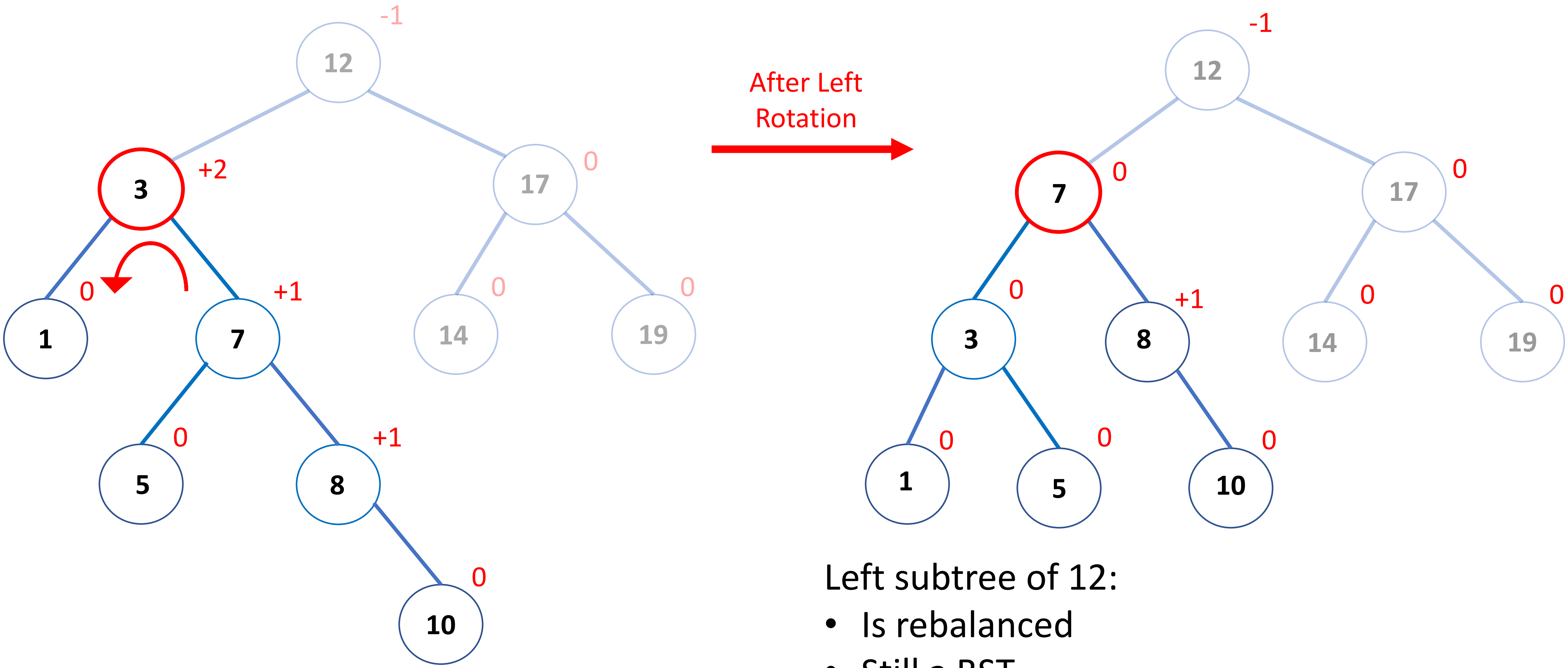
Rebalancing left subtree of 12:



Left subtree of 12:

- Is rebalanced

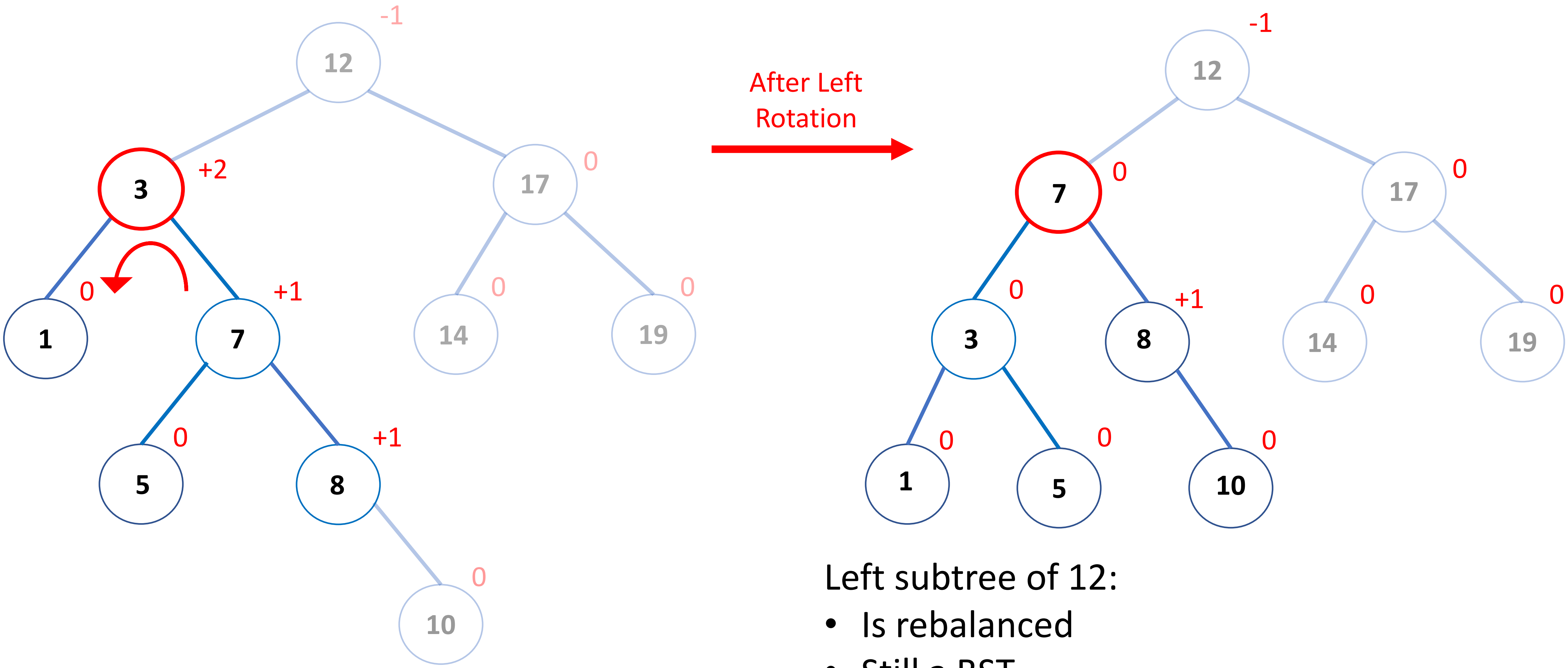
Rebalancing left subtree of 12:



Left subtree of 12:

- Is rebalanced
- Still a BST

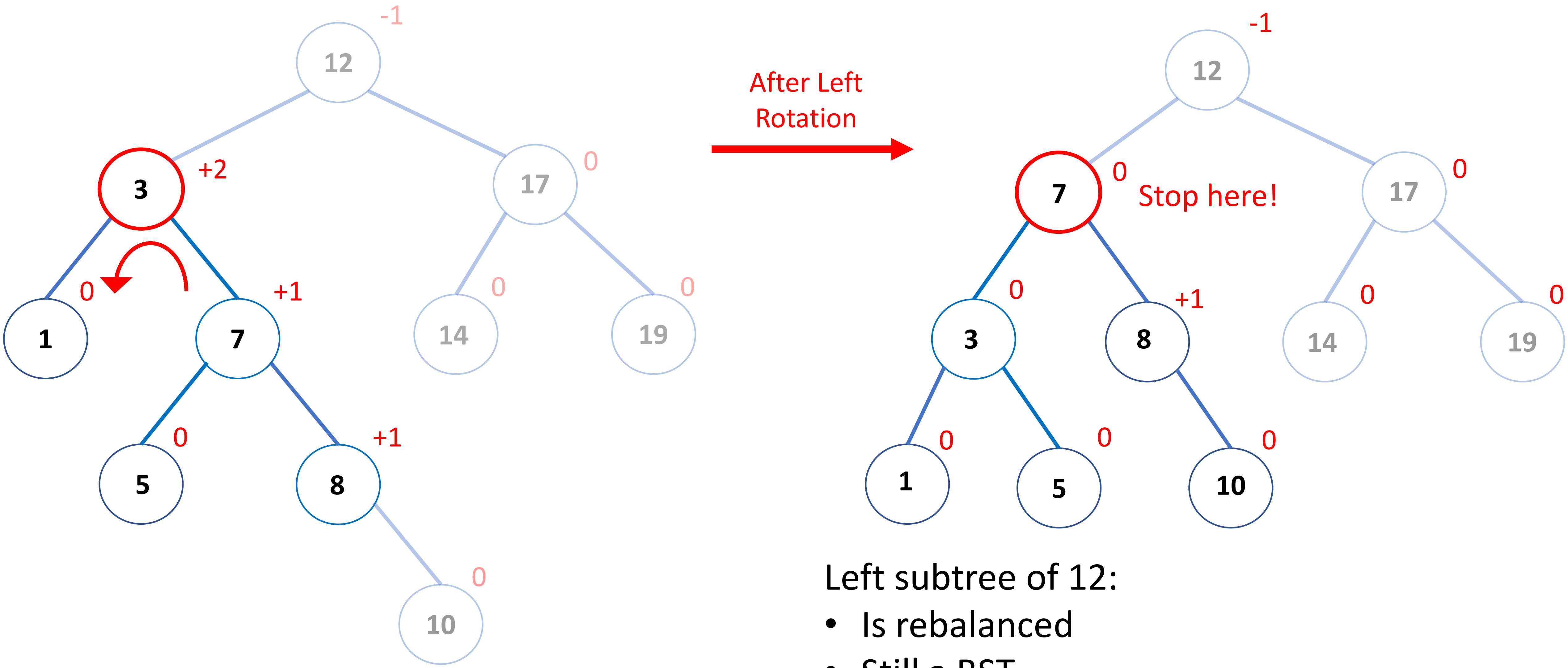
Rebalancing left subtree of 12:



Left subtree of 12:

- Is rebalanced
- Still a BST
- **Bonus**: height same as before inserting 10

Rebalancing left subtree of 12:

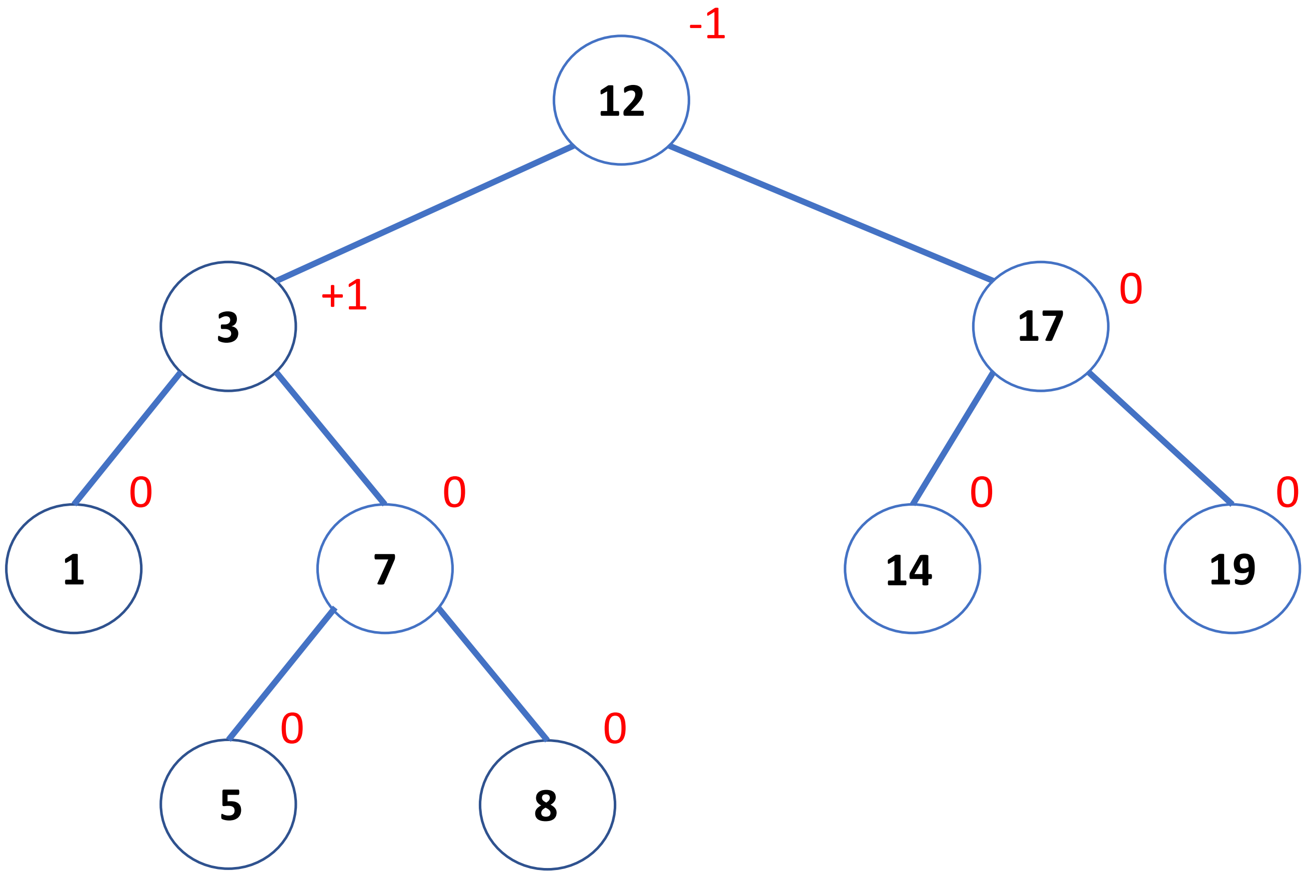


Left subtree of 12:

- Is rebalanced
- Still a BST
- **Bonus**: height same as before inserting 10

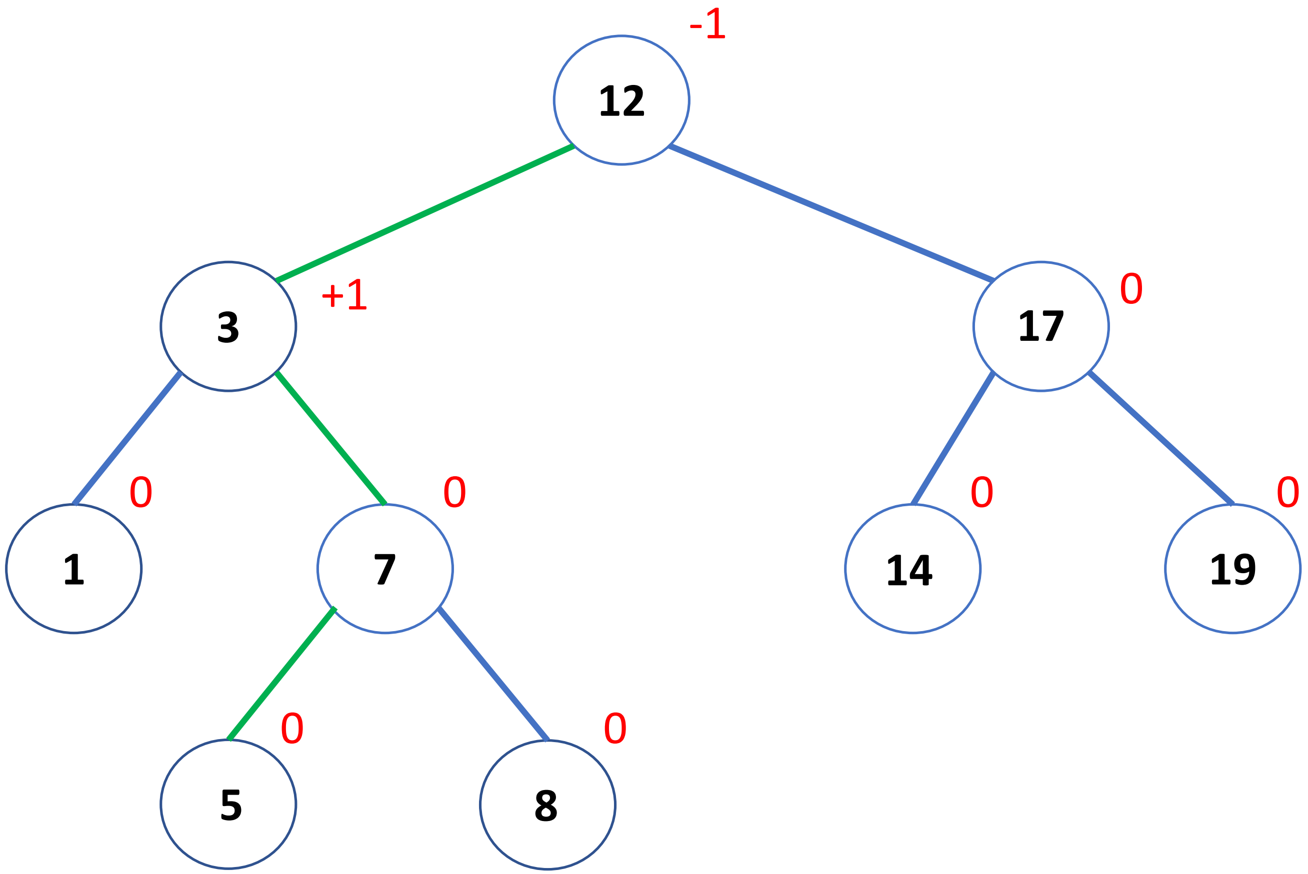
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 6)



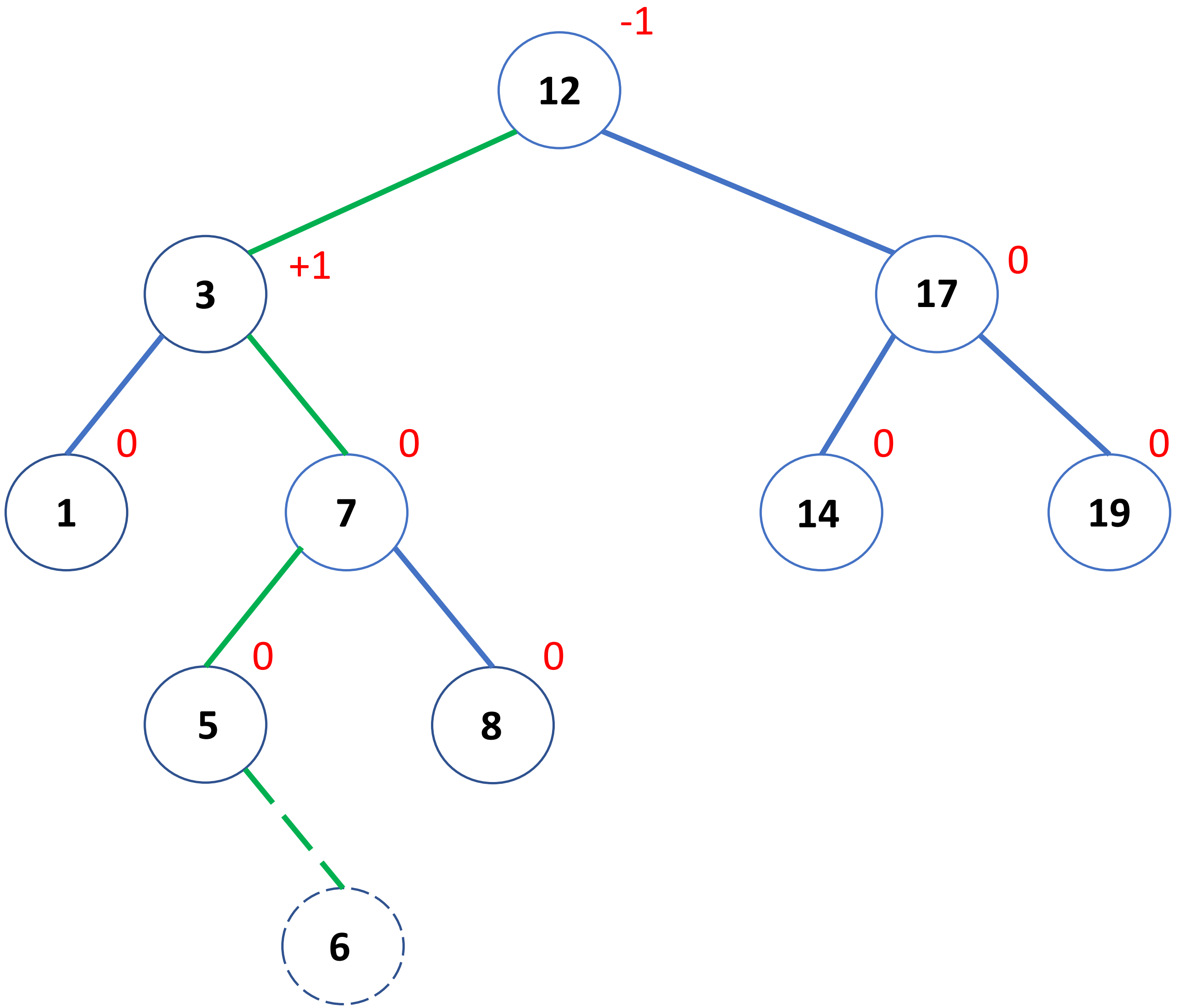
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 6)



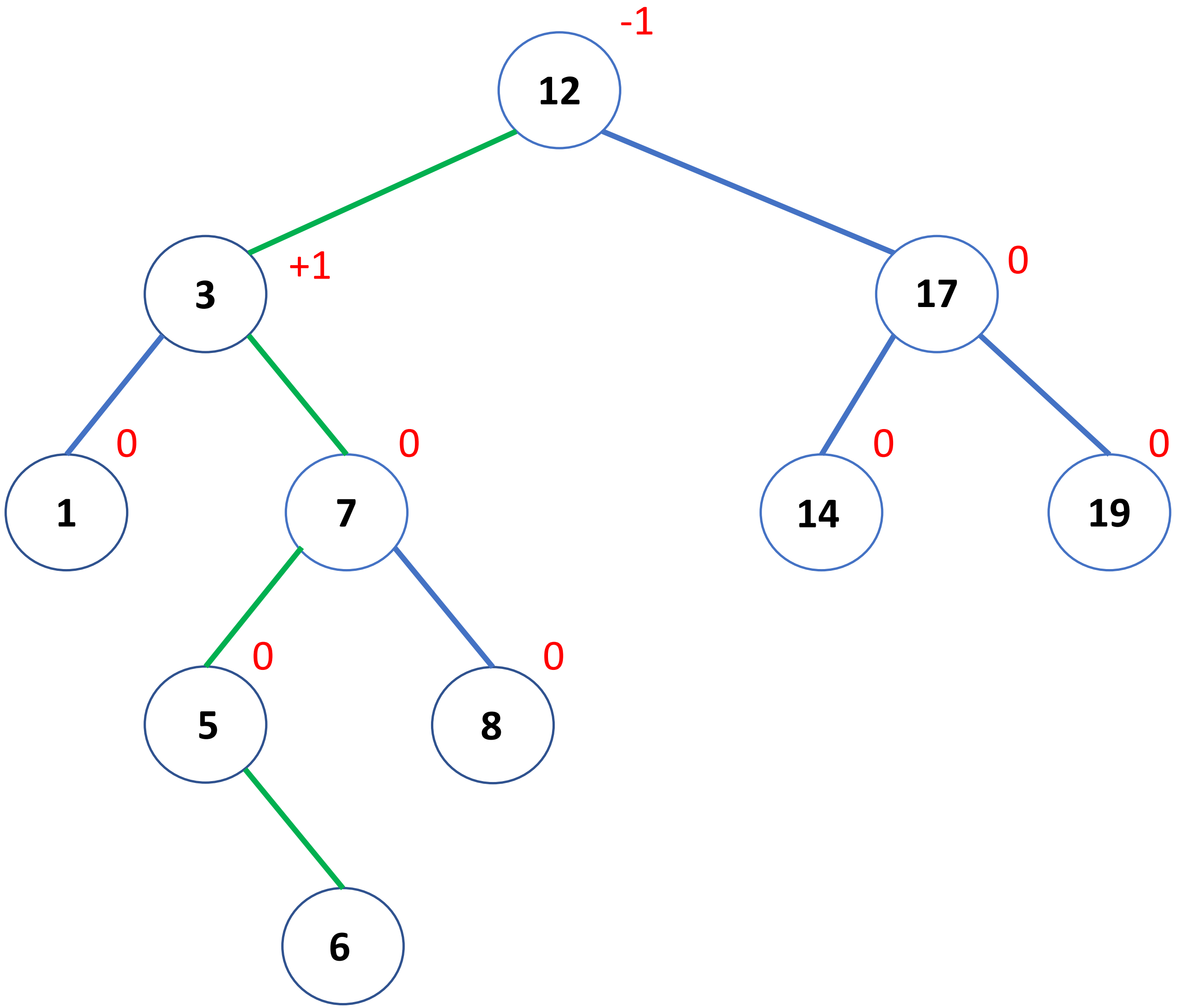
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 6)



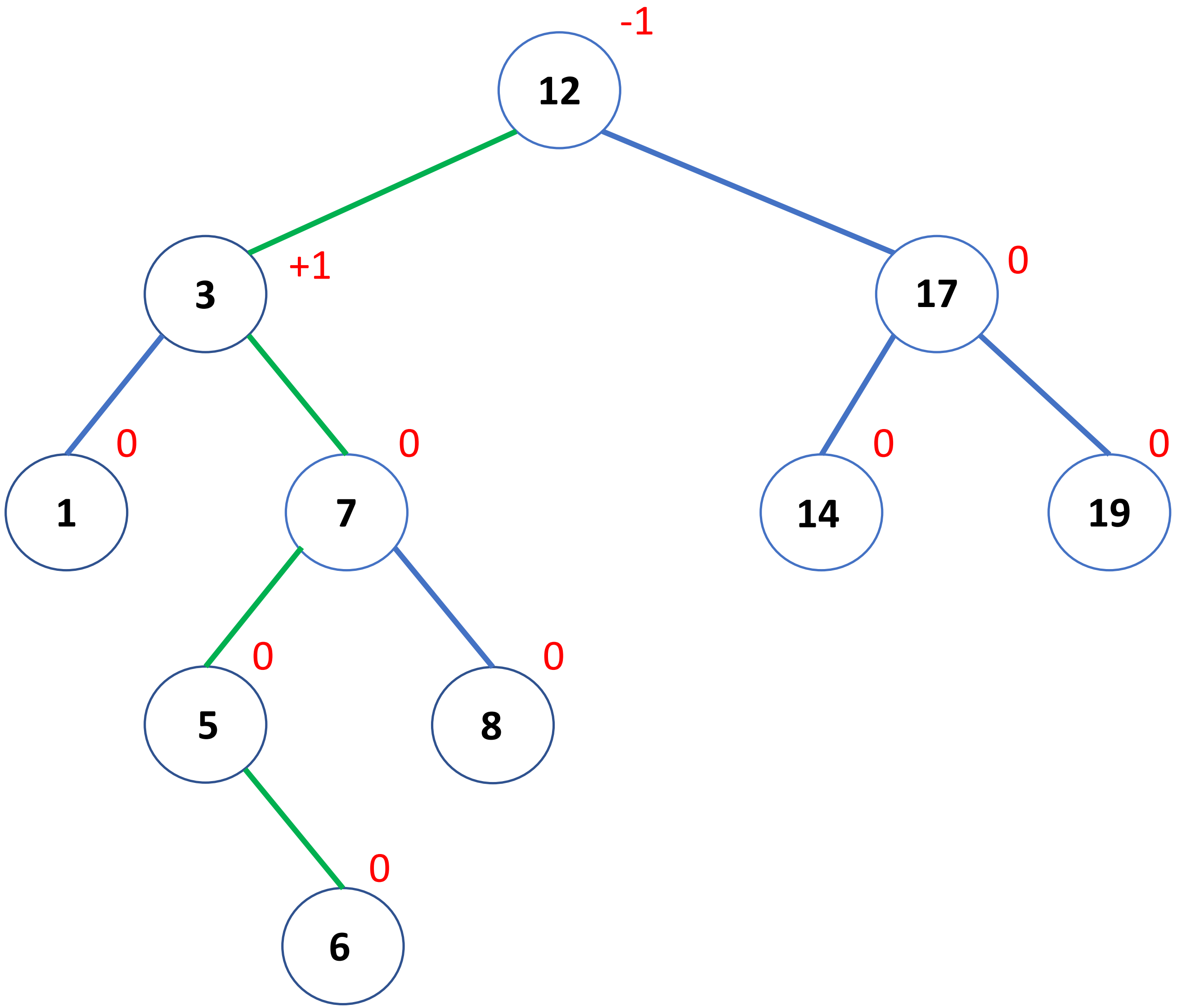
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 6)



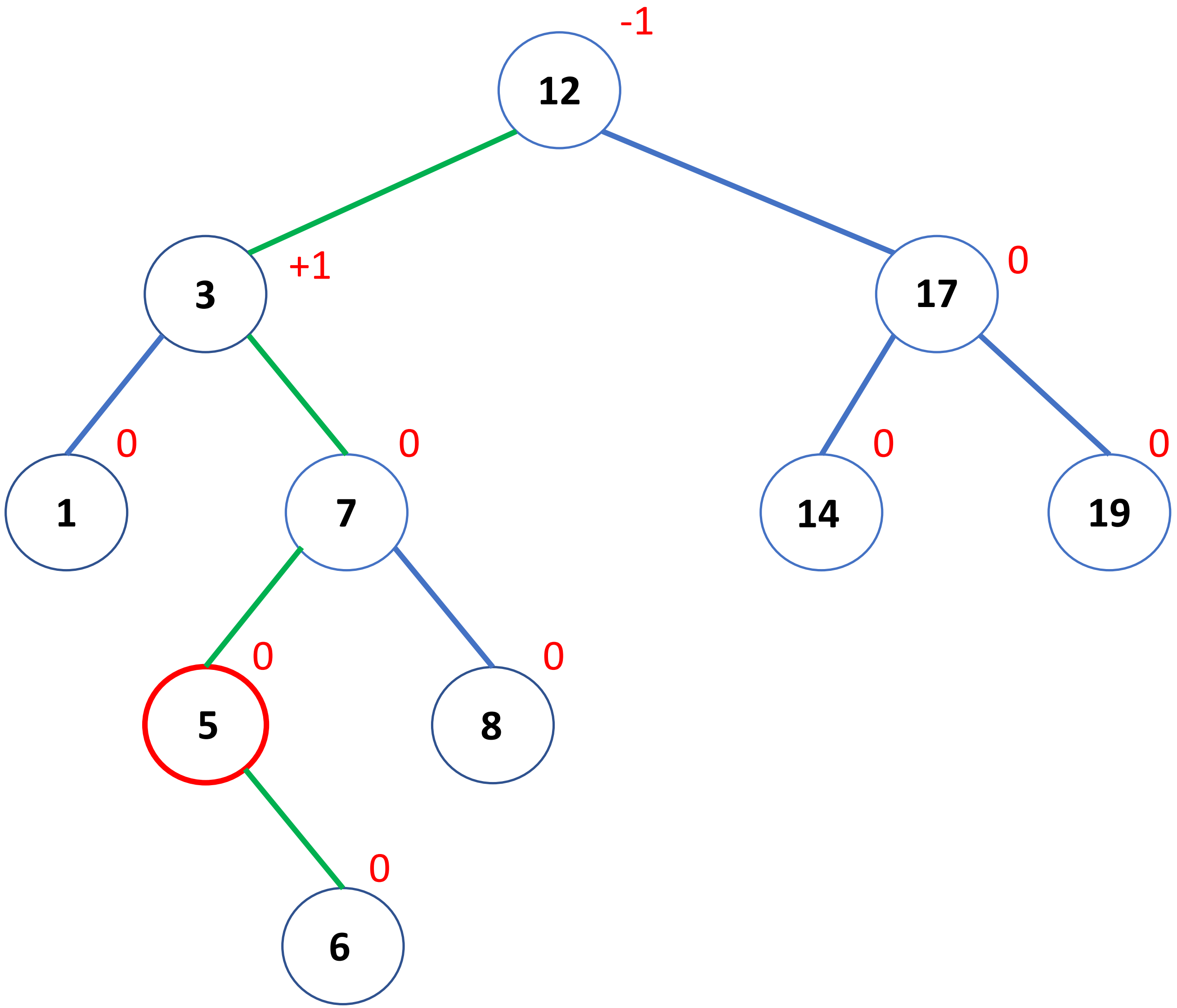
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 6)



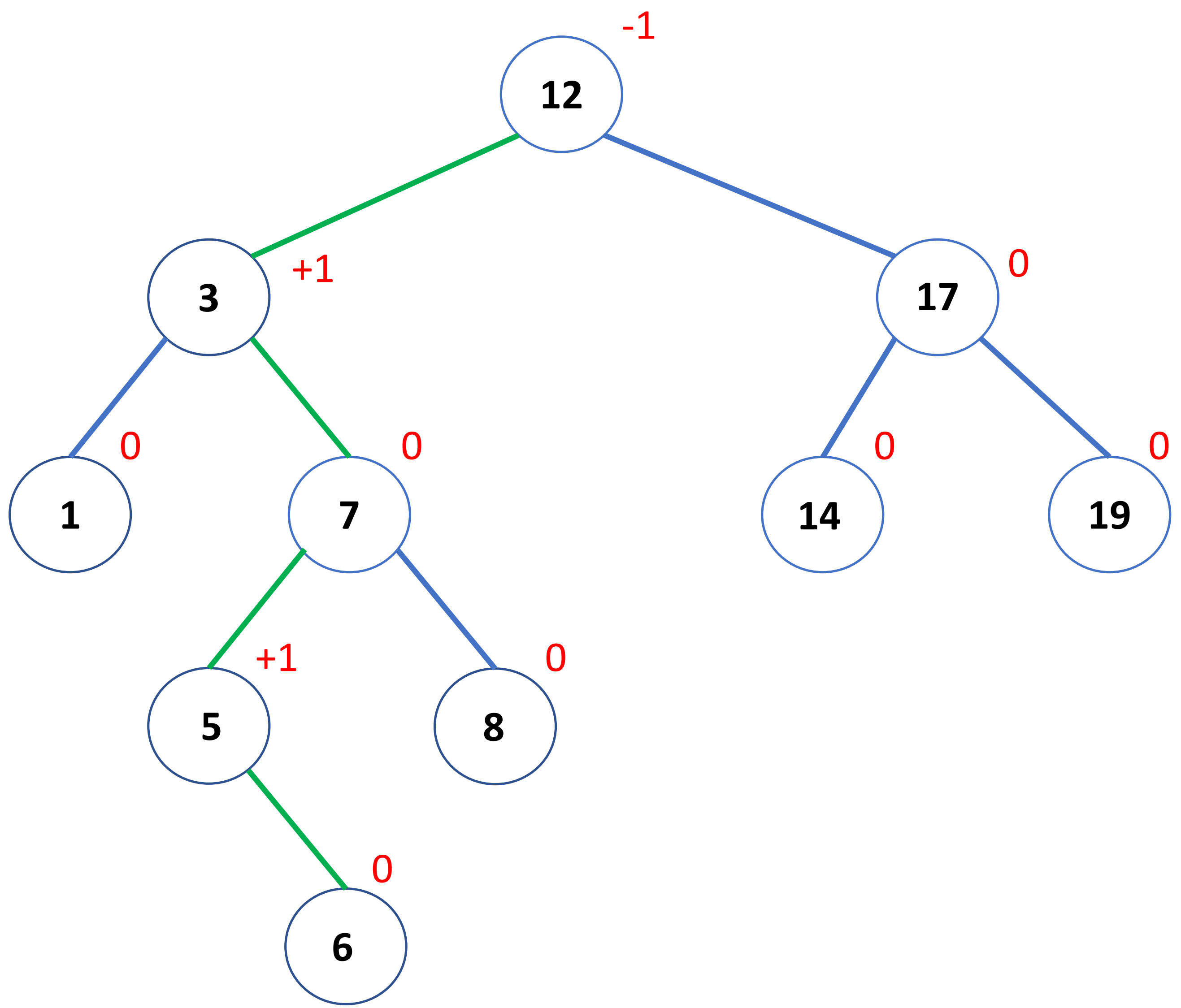
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 6)



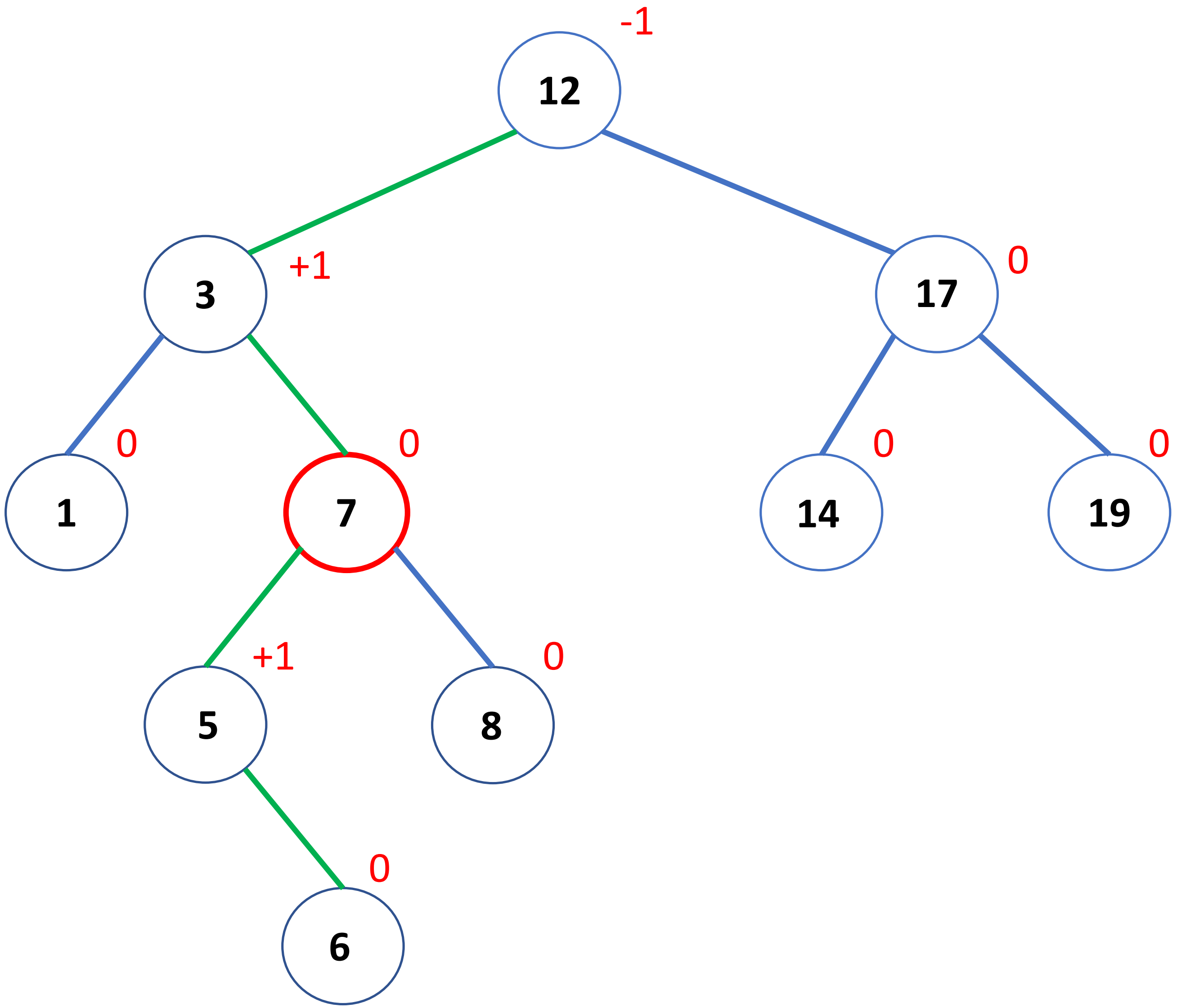
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 6)



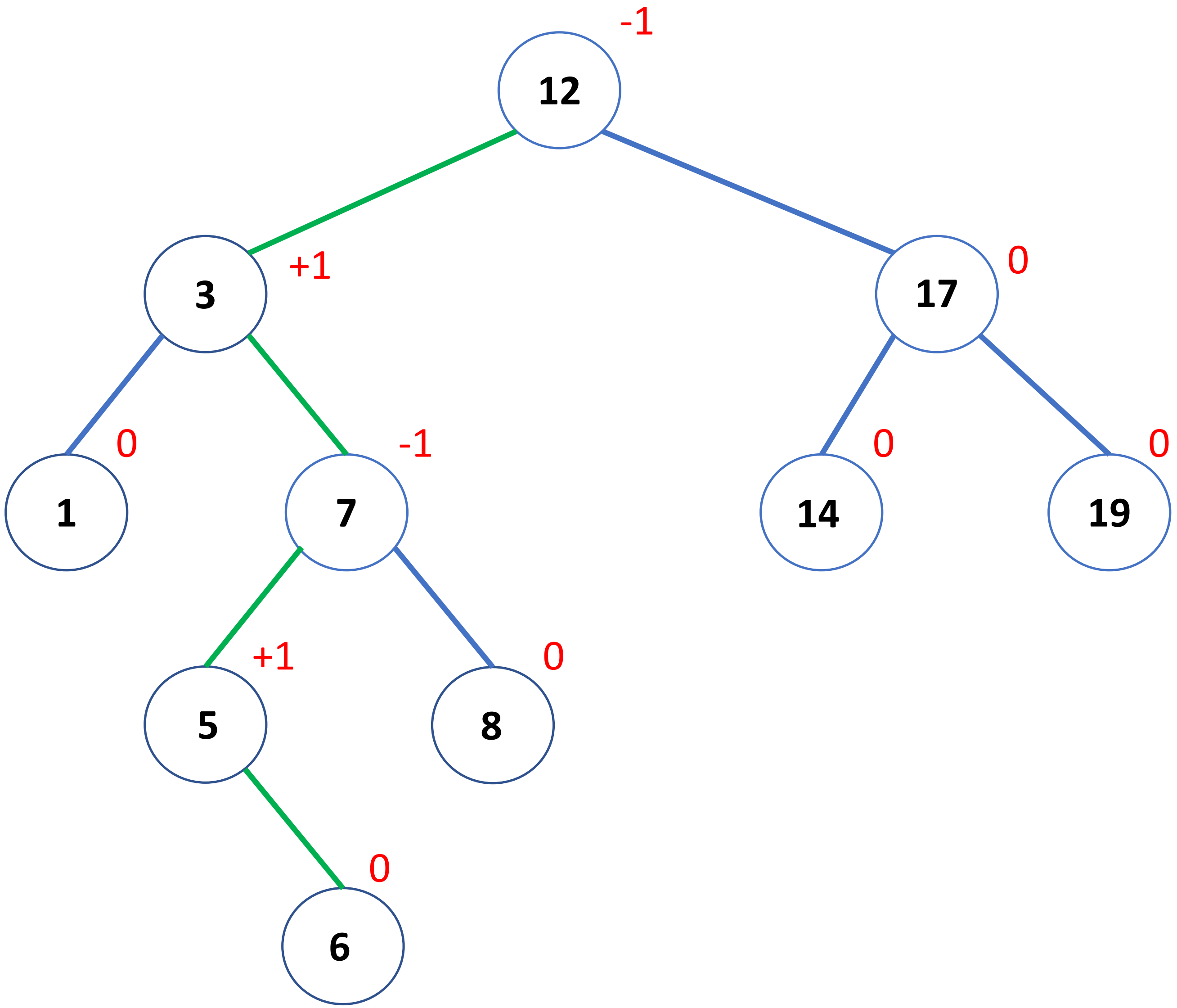
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 6)



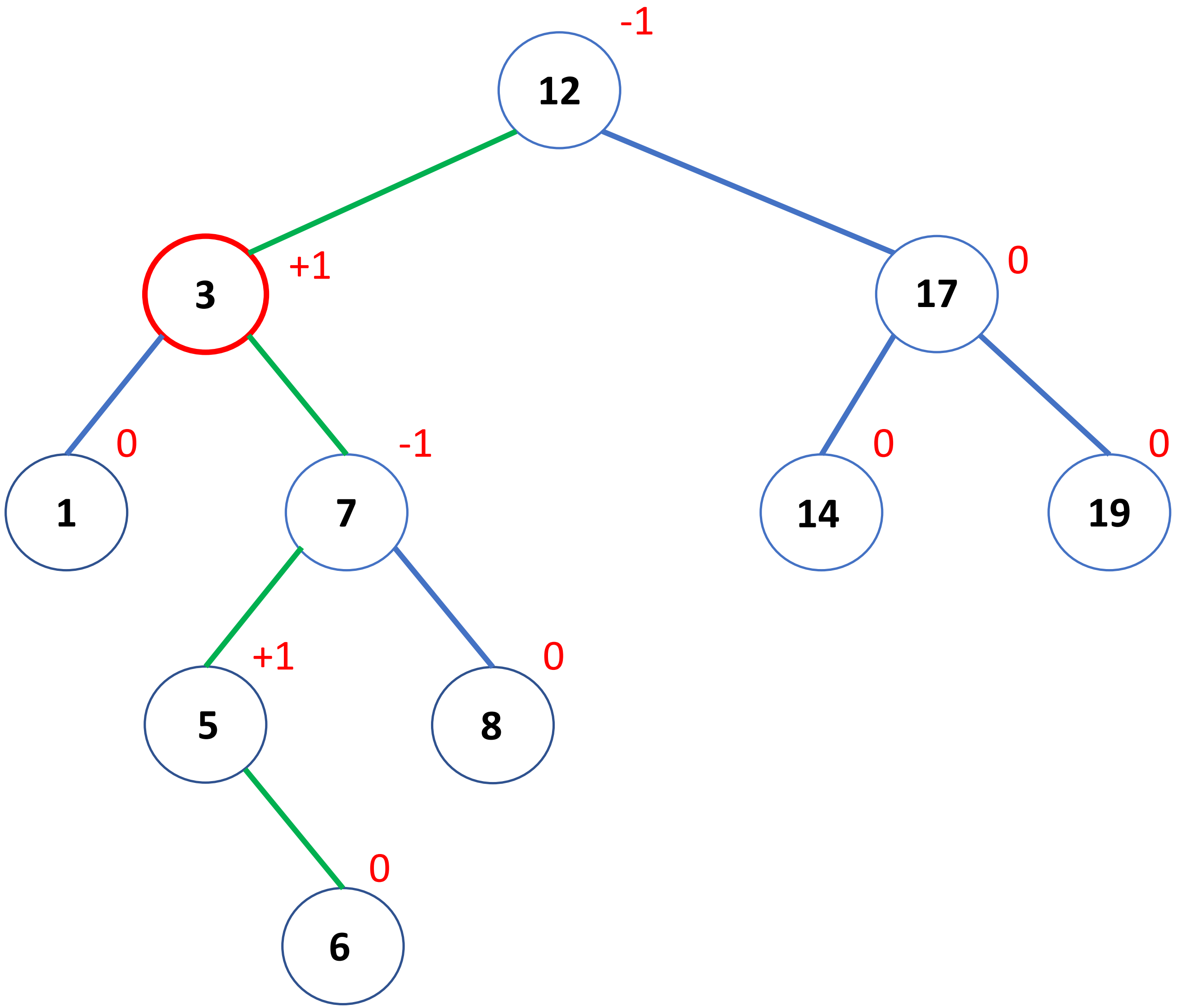
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 6)



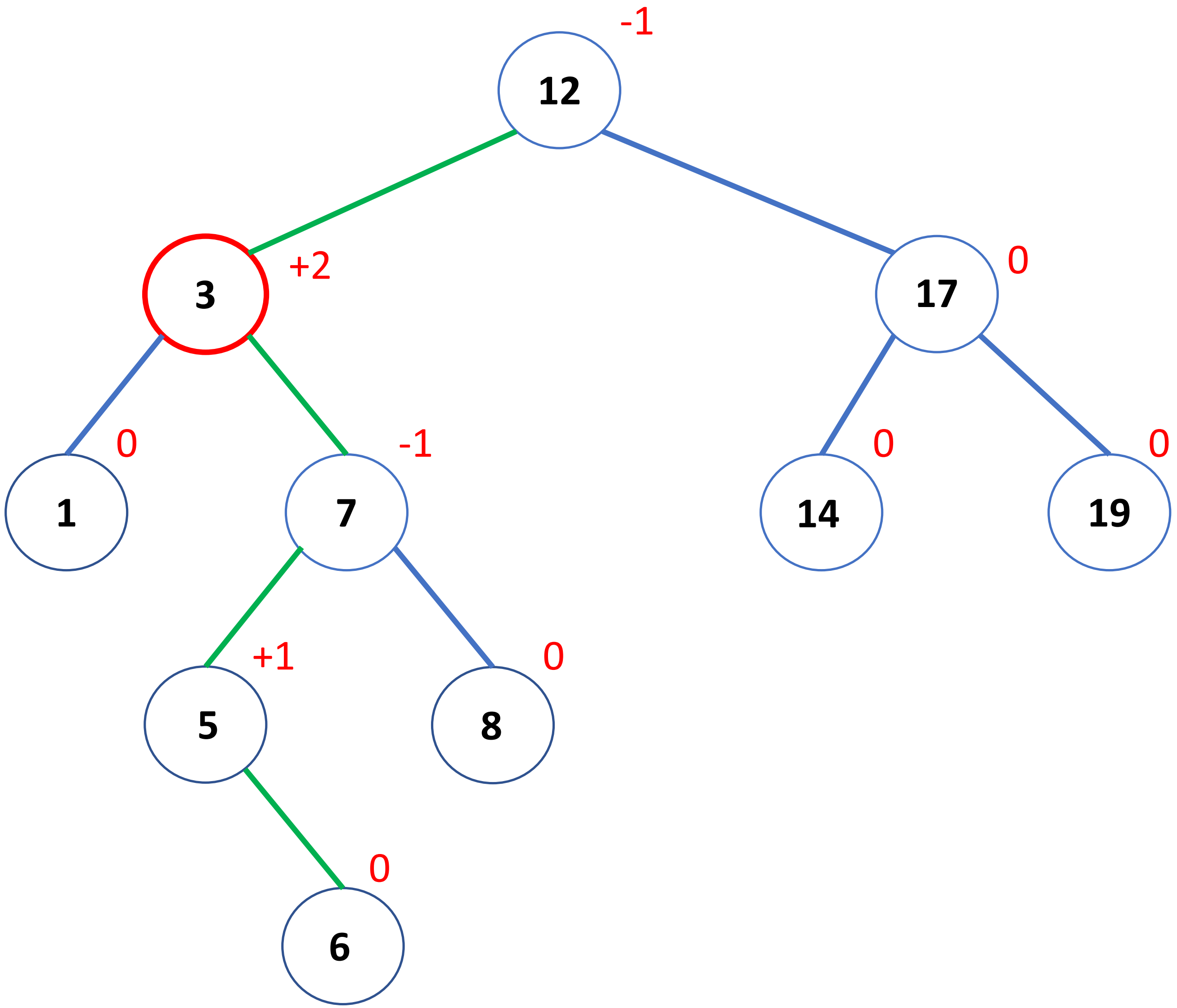
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 6)



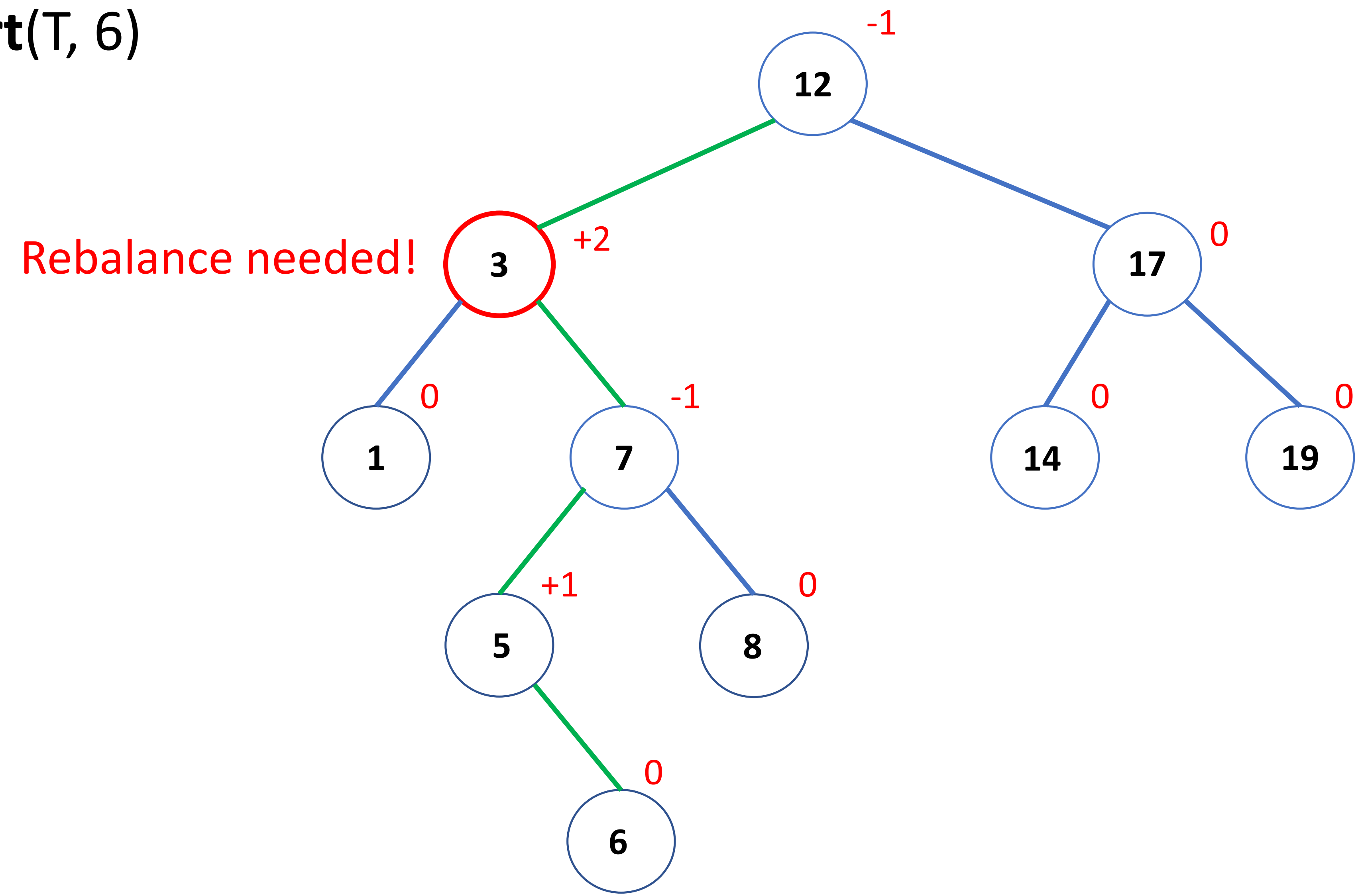
Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

Insert(T, 6)

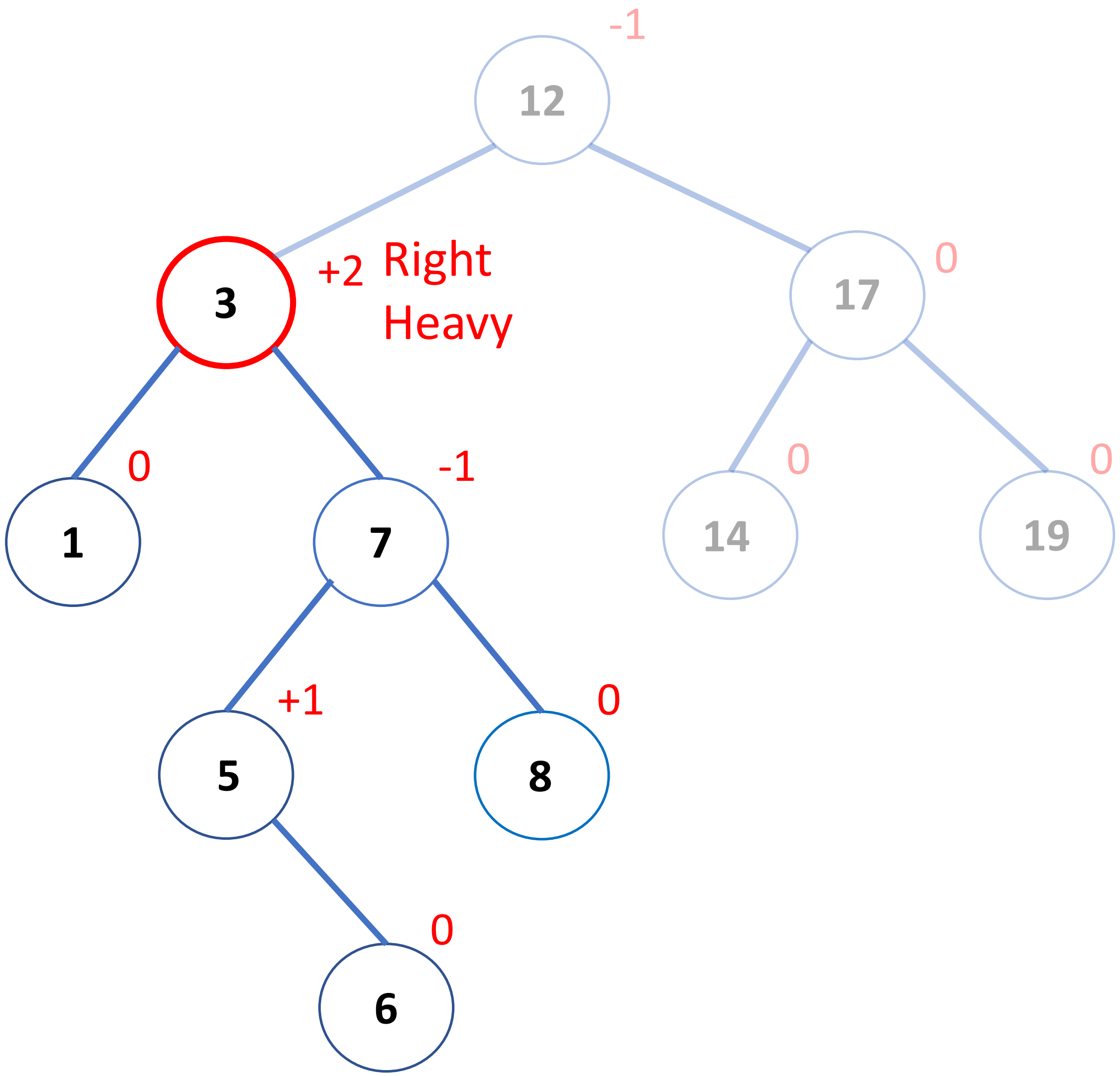


Example: AVL of {1, 3, 5, 7, 8, 12, 14, 17, 19}

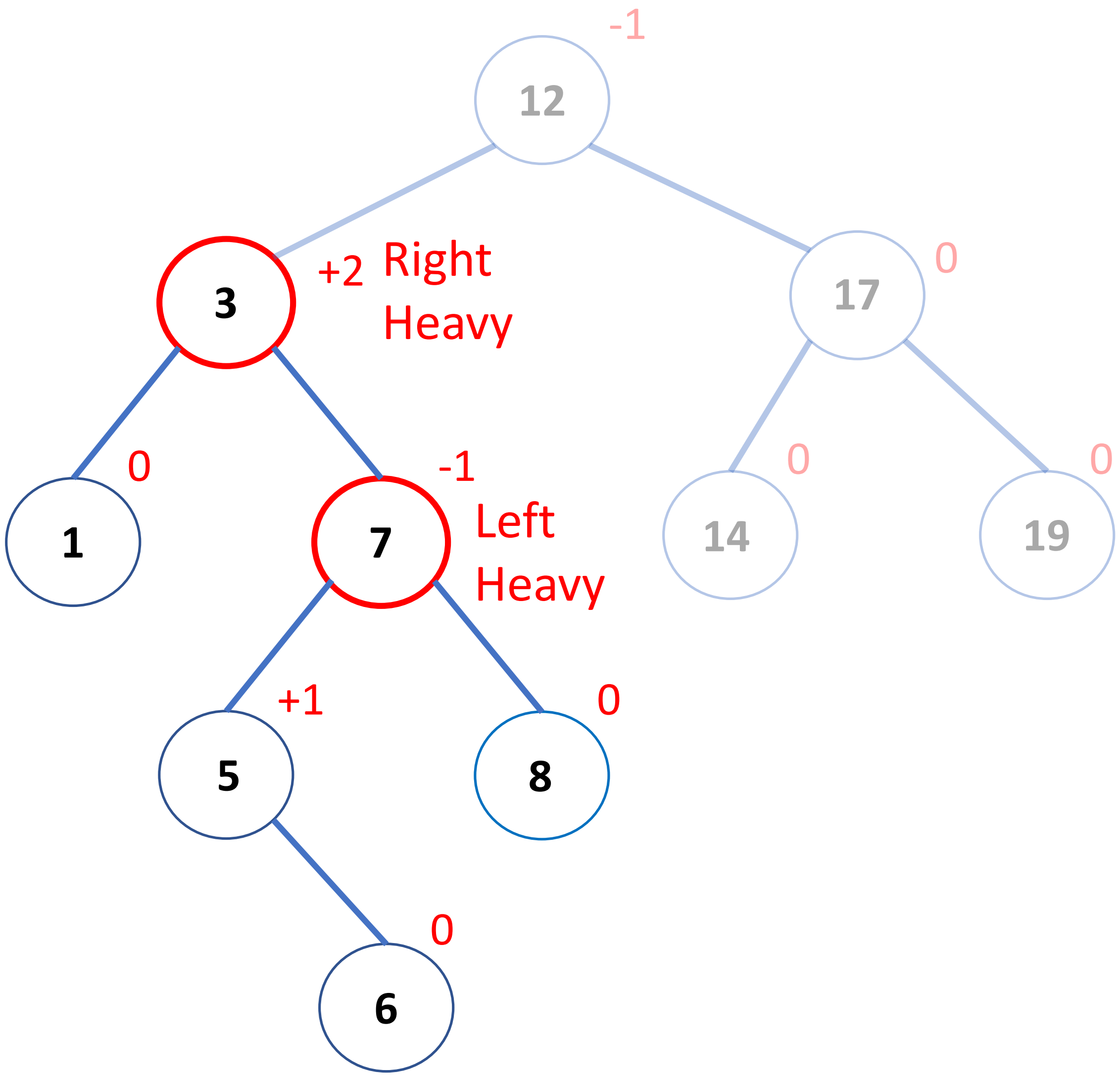
Insert(T, 6)



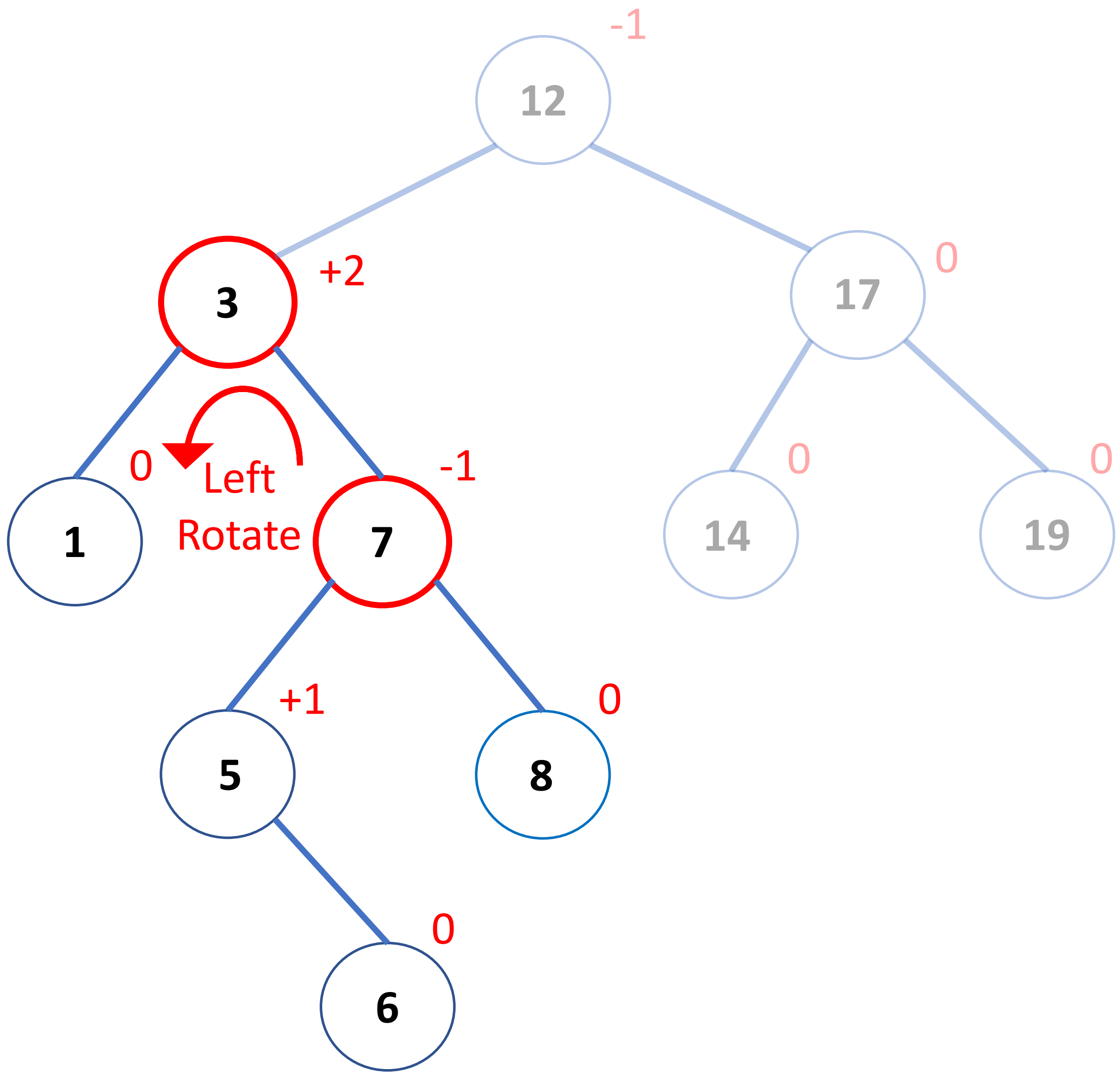
Rebalancing left subtree of 12:



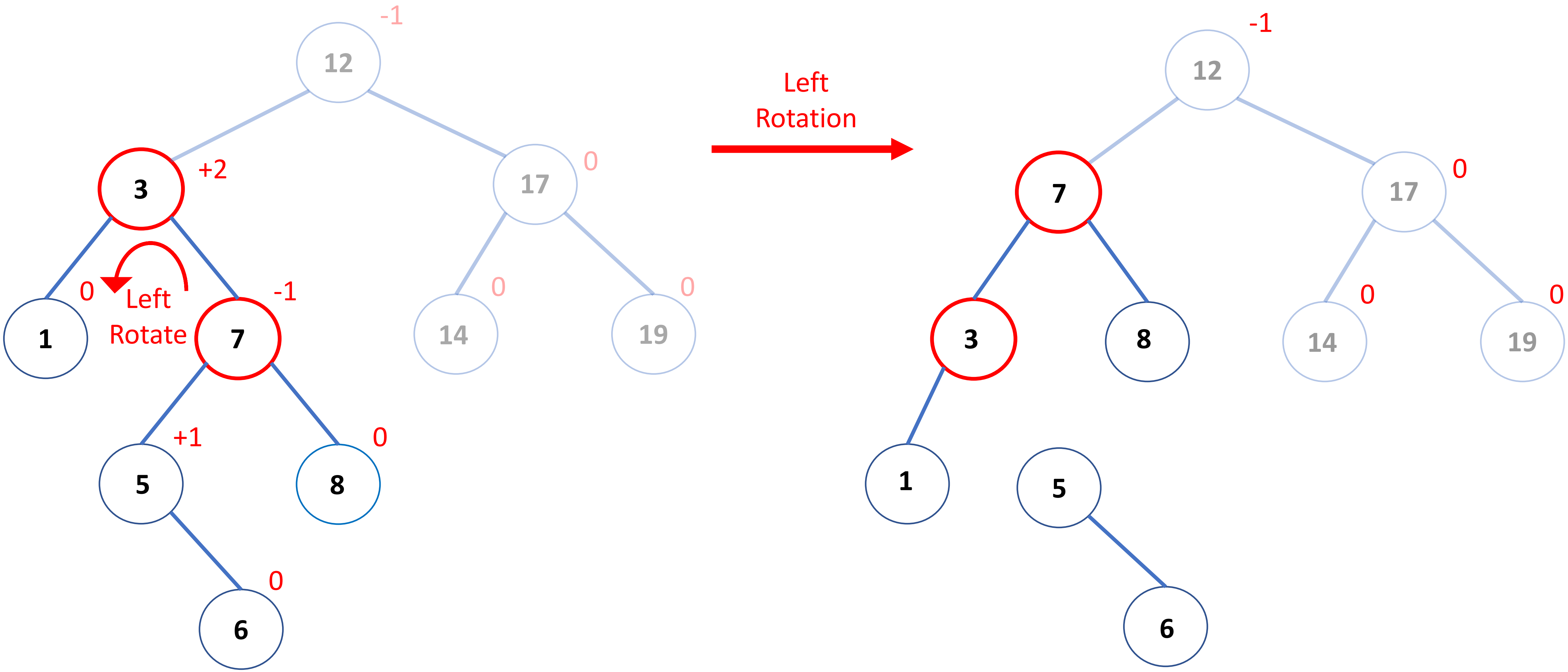
Rebalancing left subtree of 12:



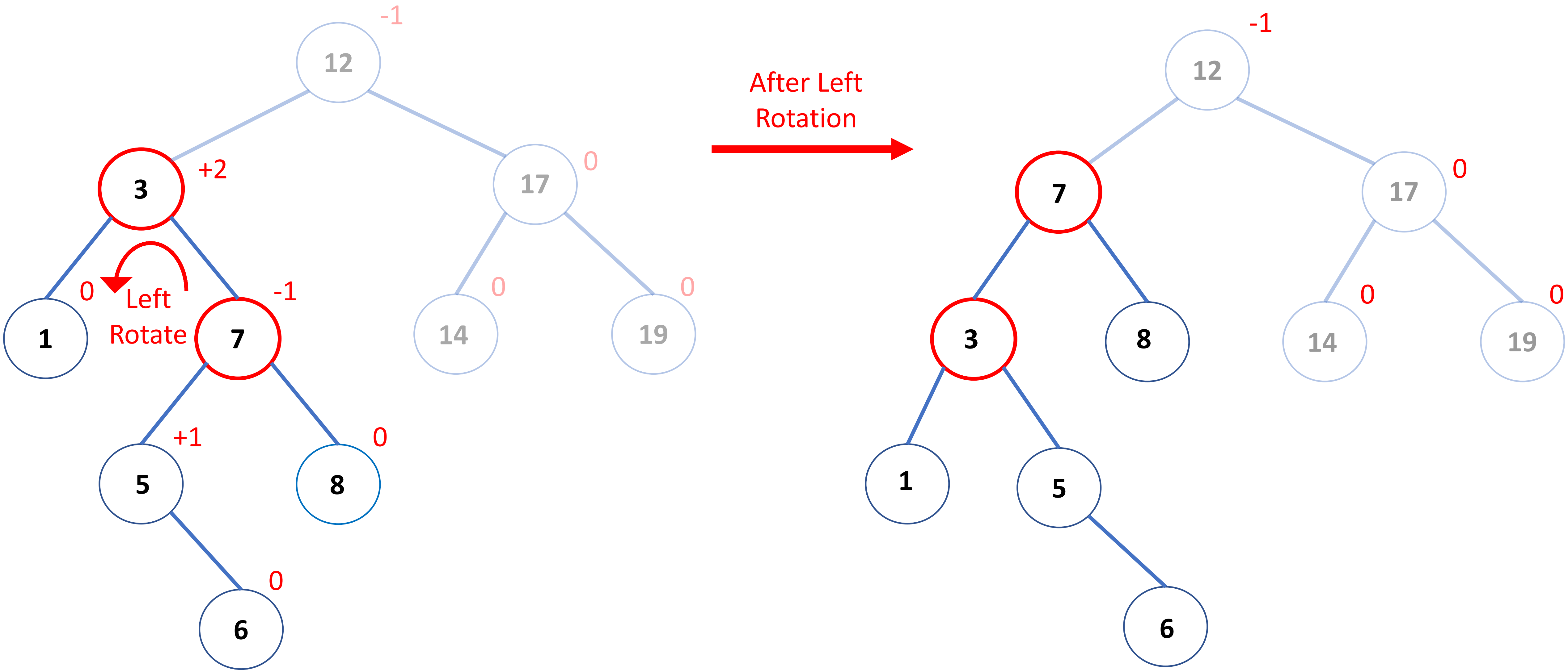
Rebalancing left subtree of 12: Try Left Rotation !



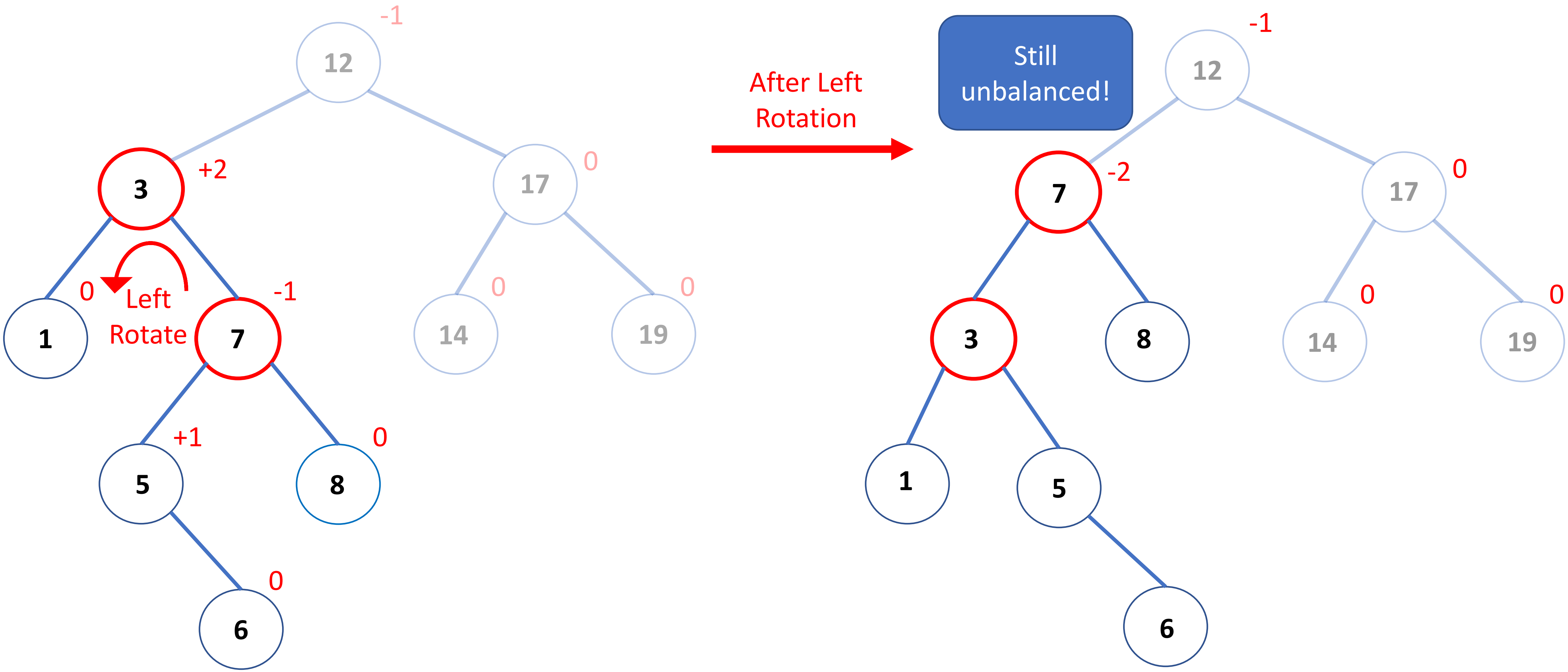
Rebalancing left subtree of 12: Try Left Rotation !



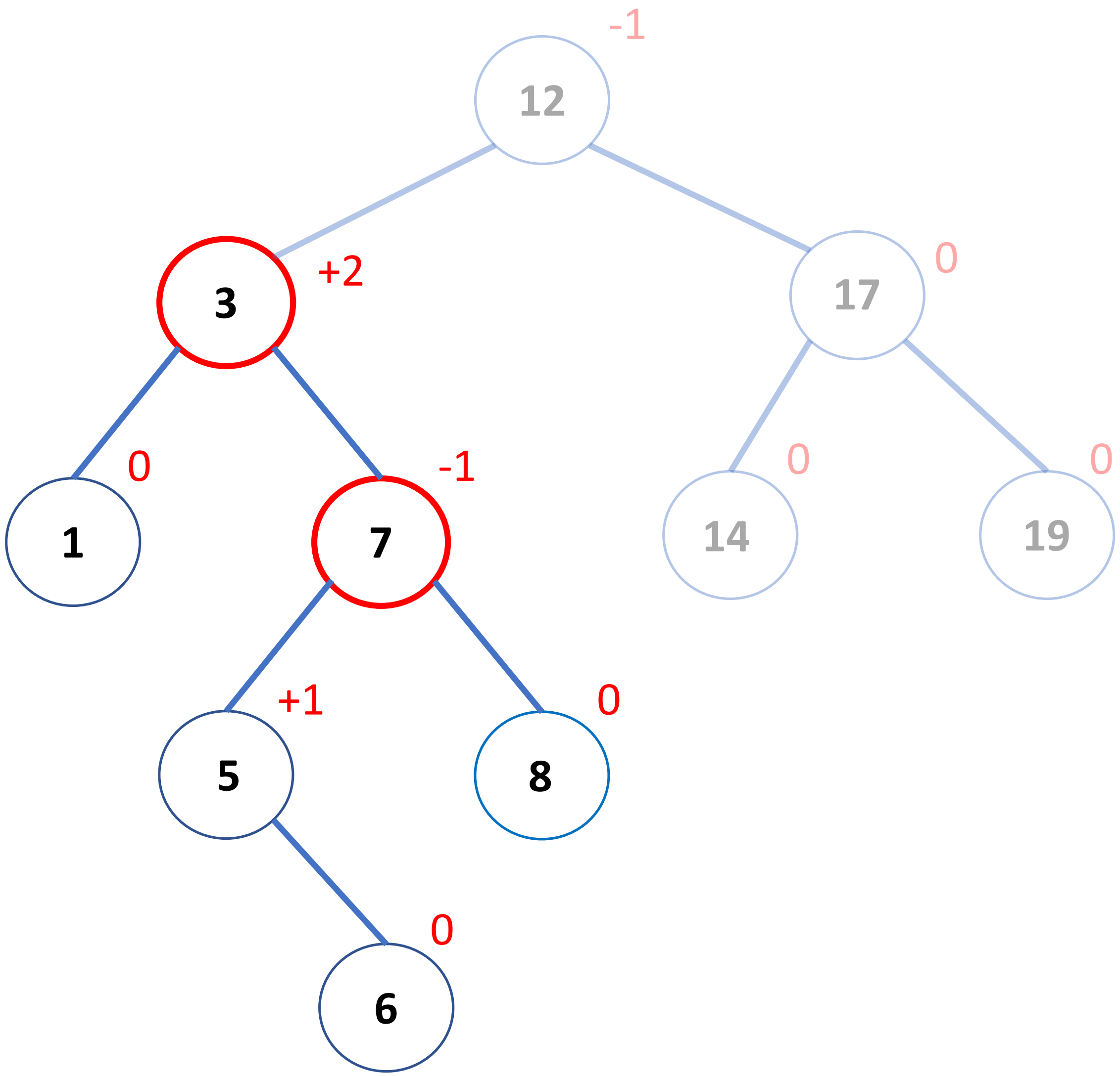
Rebalancing left subtree of 12: Try Left Rotation !



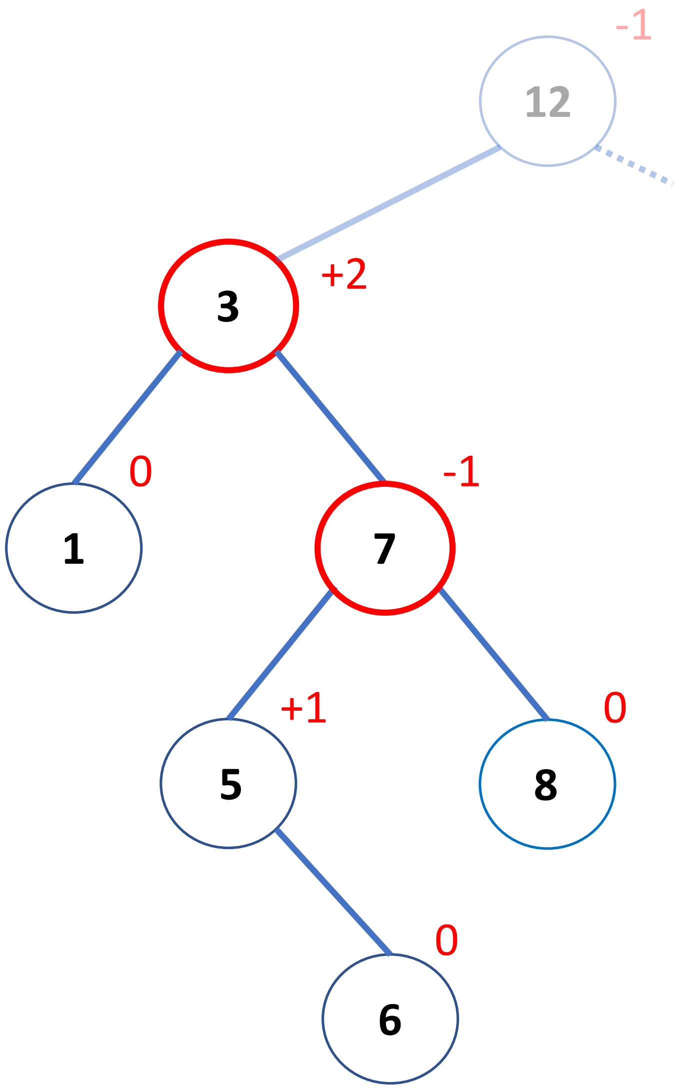
Rebalancing left subtree of 12: Try Left Rotation !



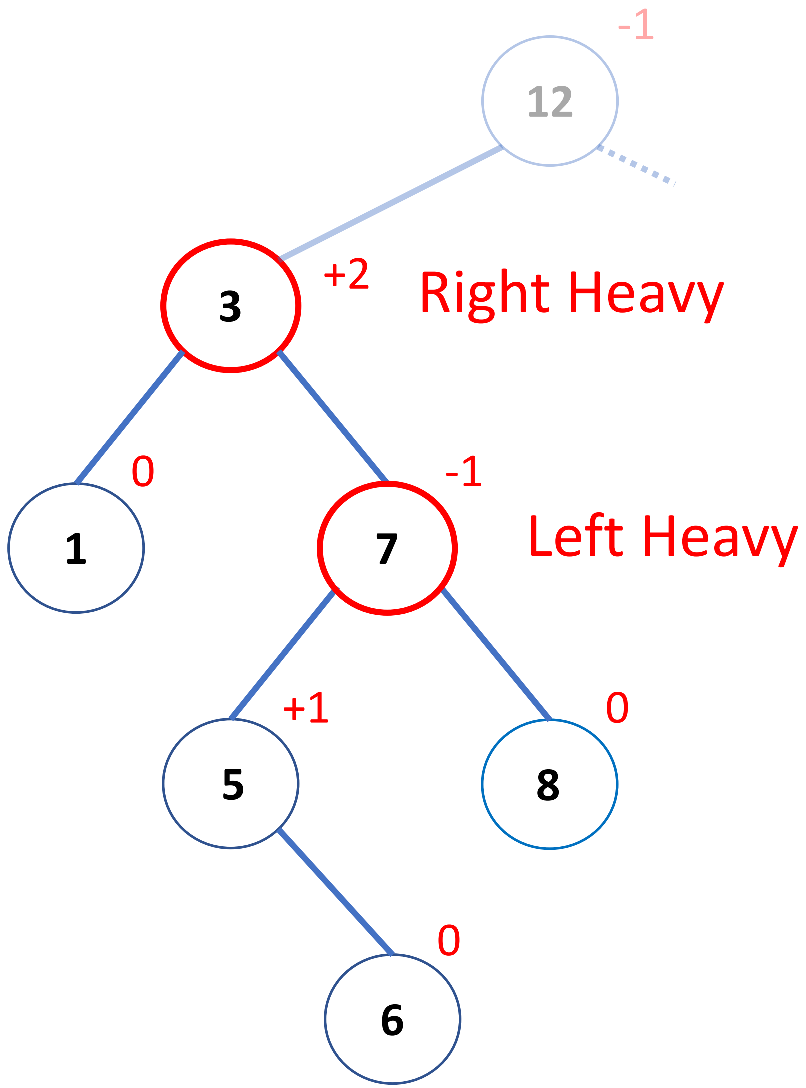
Rebalancing left subtree of 12: Attempt two!



Rebalancing left subtree of 12: Attempt two!

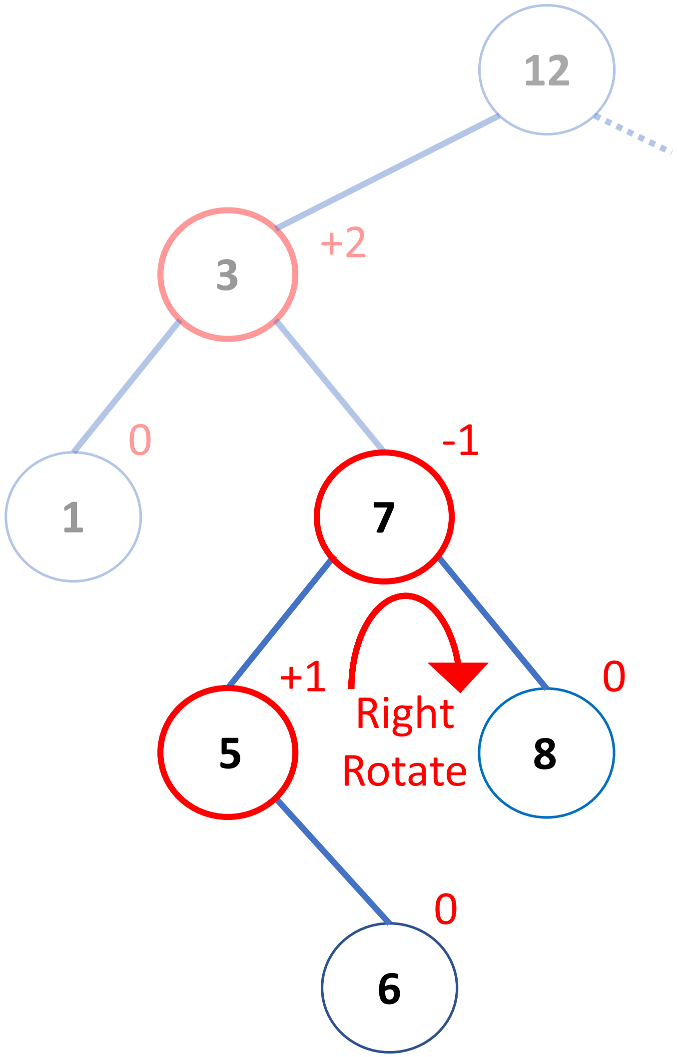


Rebalancing left subtree of 12: Attempt two!



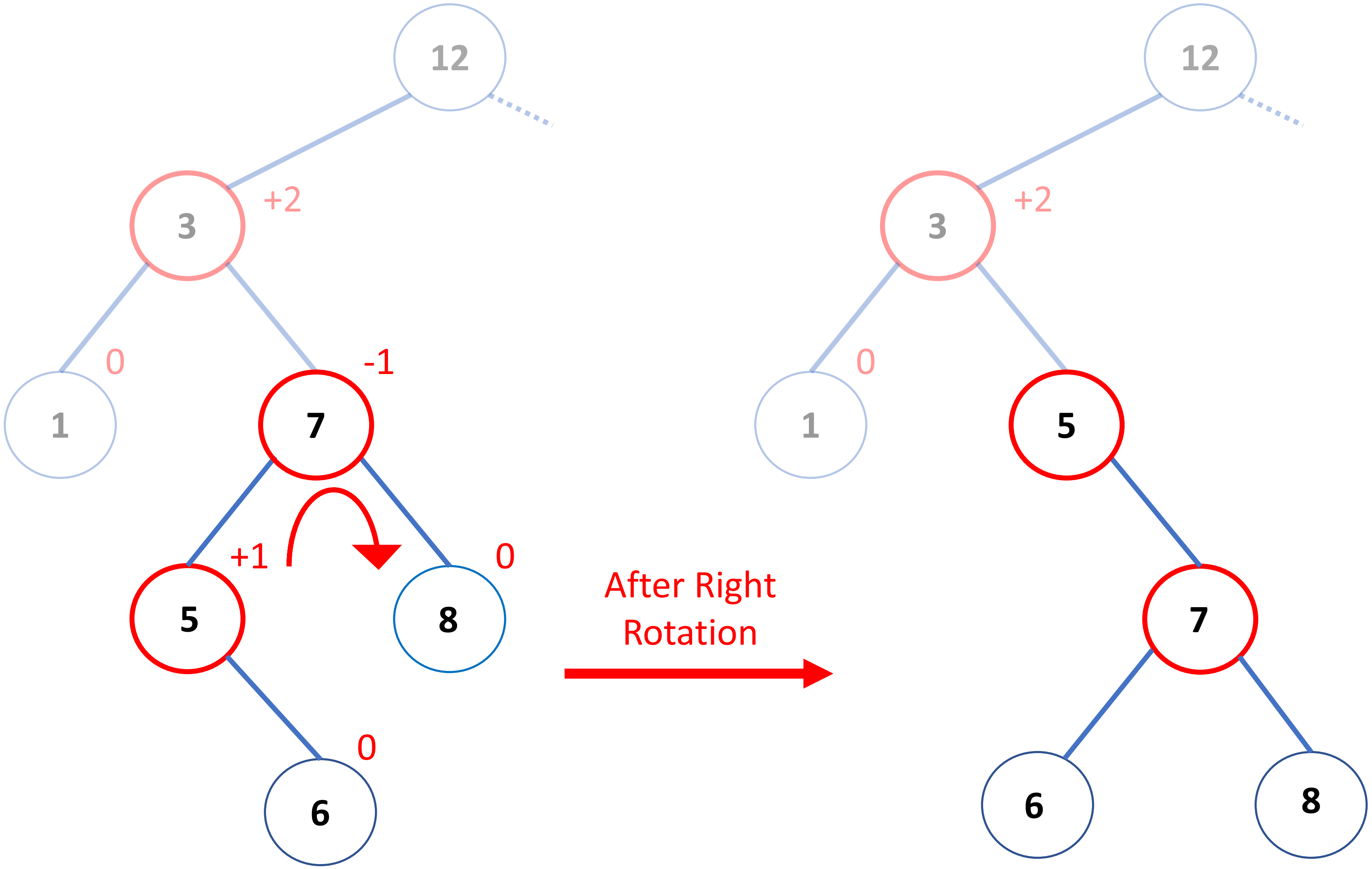
Rebalancing left subtree of 12: Attempt two!

Step 1. Right rotate right subtree of 3



Rebalancing left subtree of 12: Attempt two!

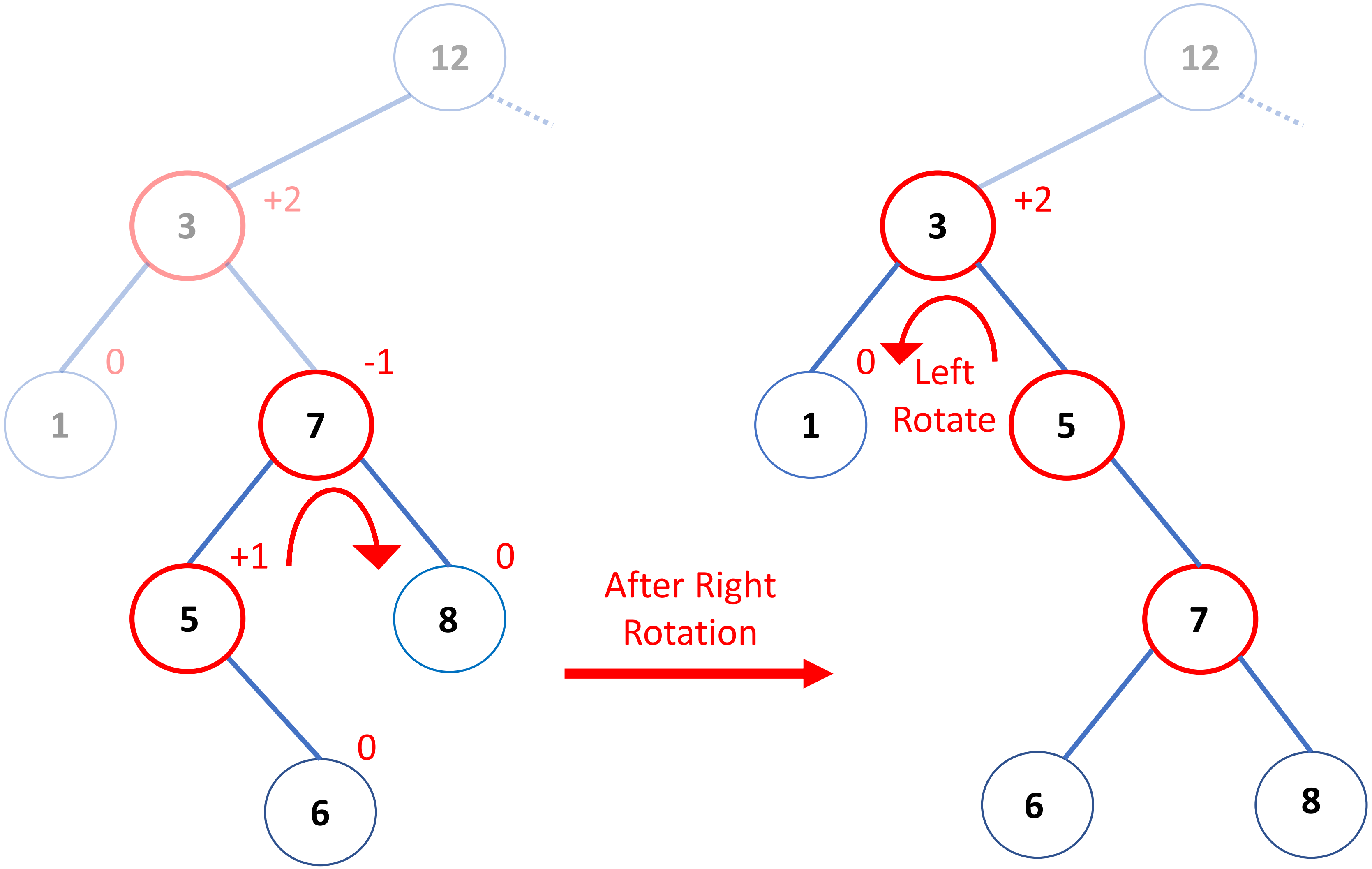
Step 1. Right rotate right subtree of 3



Rebalancing left subtree of 12: Attempt two!

Step 1. Right rotate right subtree of 3

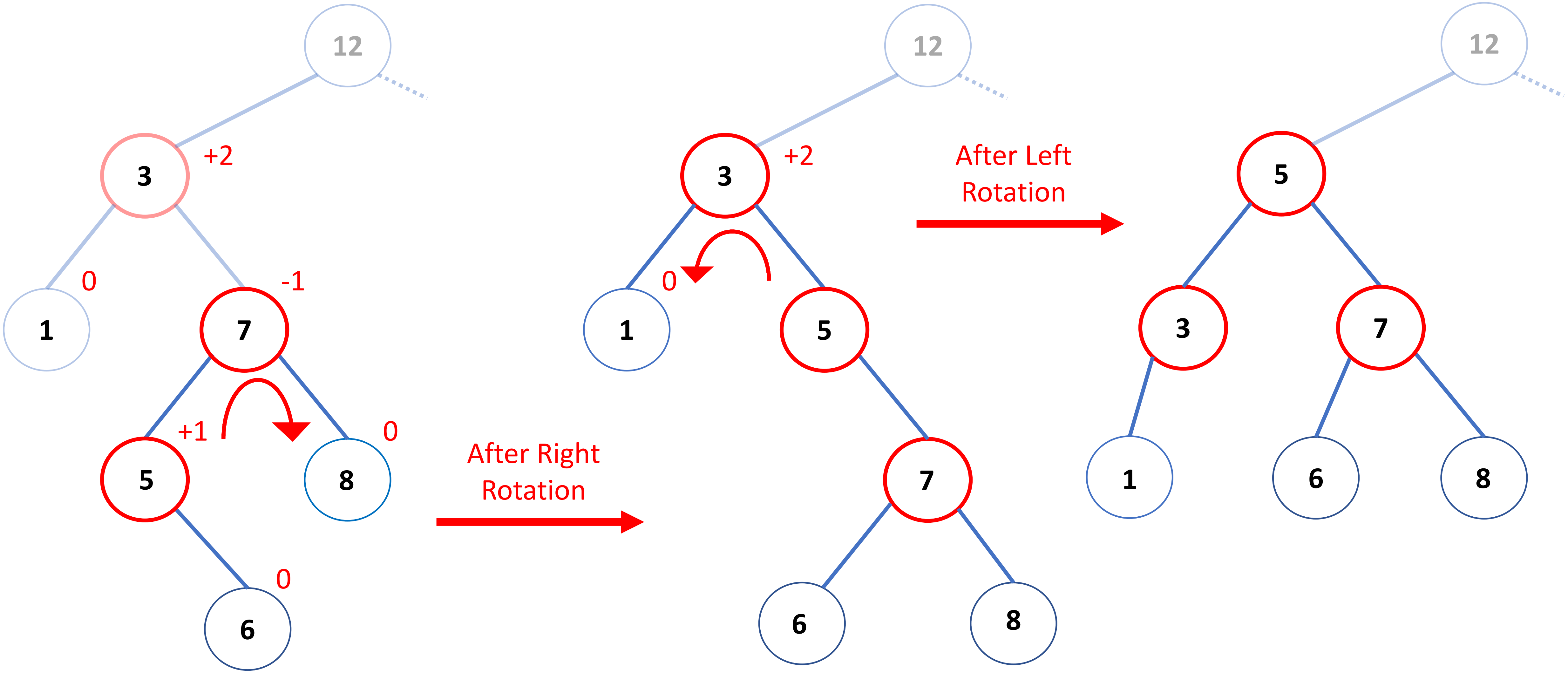
Step 2. Left rotate left subtree of 12



Rebalancing left subtree of 12: Attempt two!

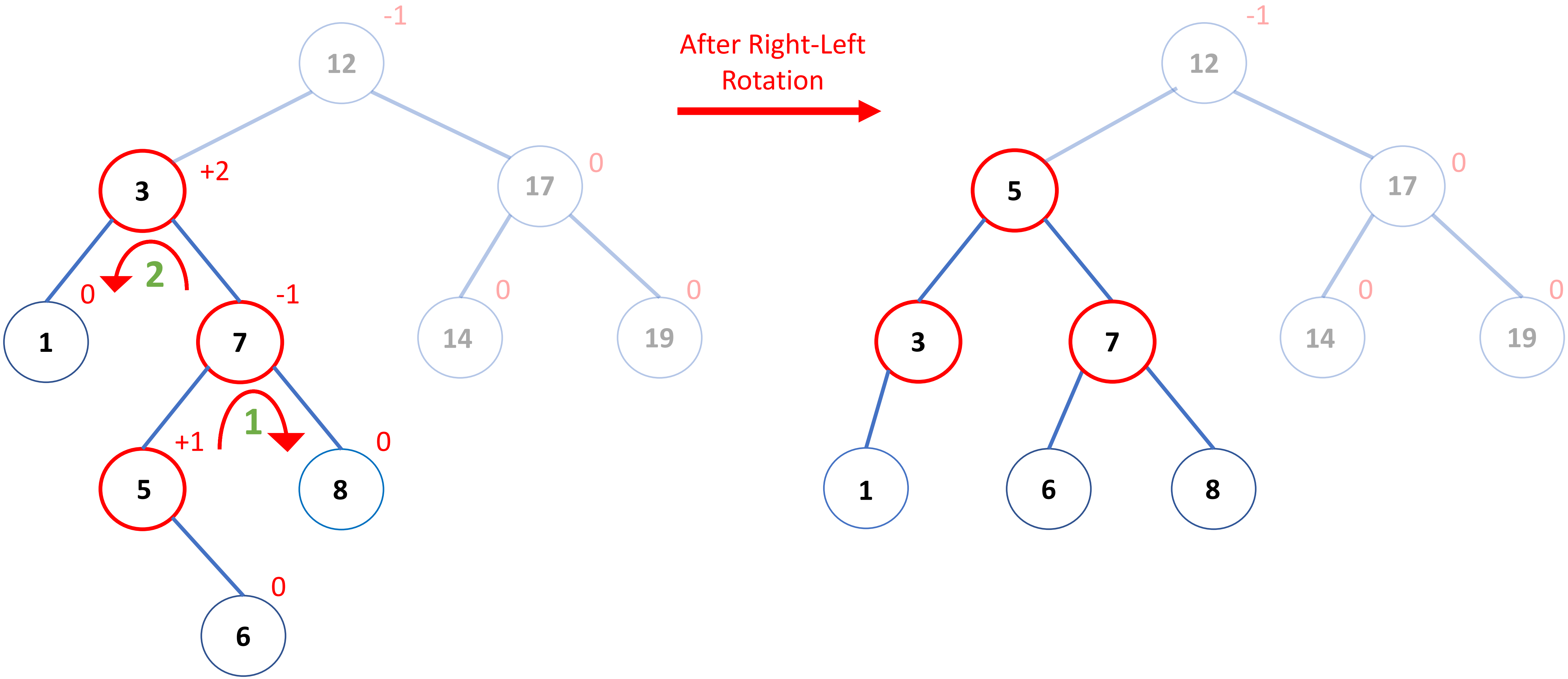
Step 1. Right rotate right subtree of 3

Step 2. Left rotate left subtree of 12

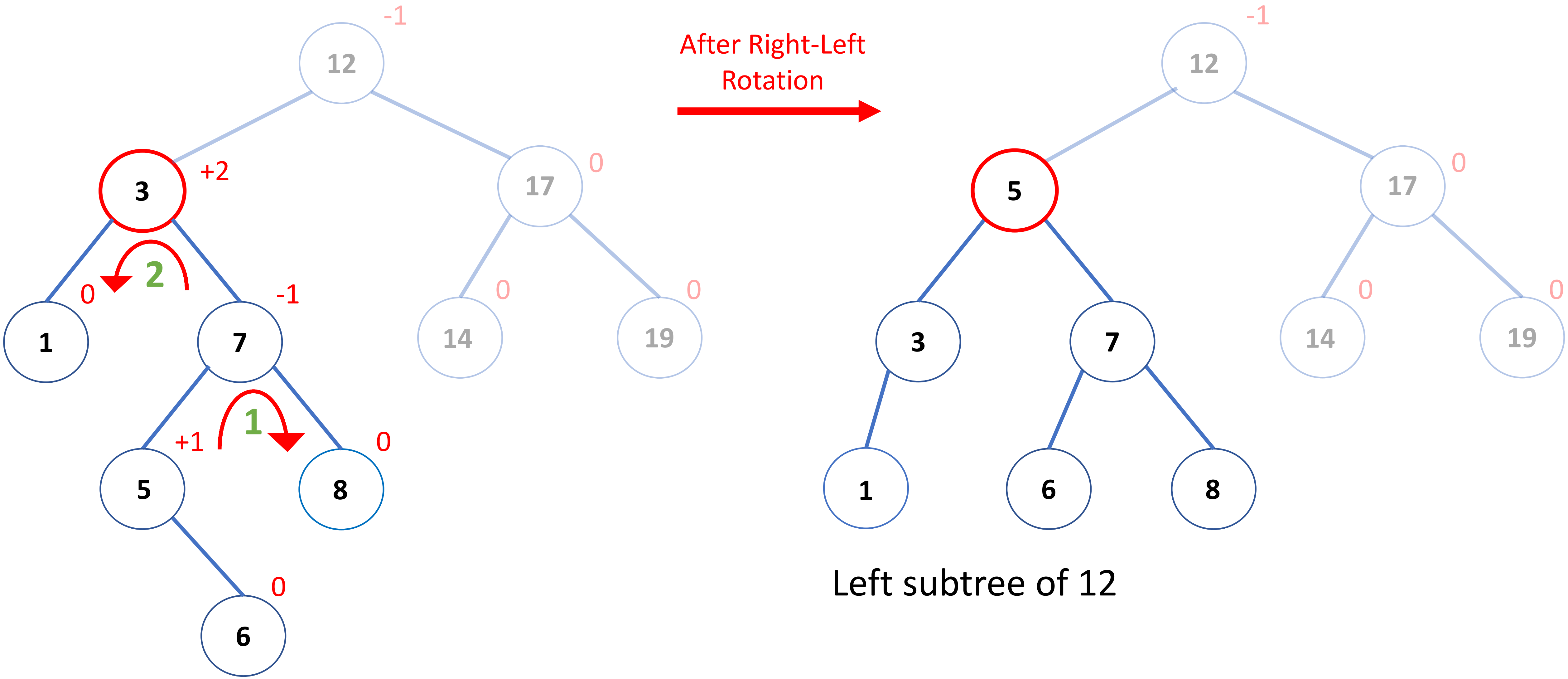


Rebalancing left subtree of 12: Double Right-Left Rotation

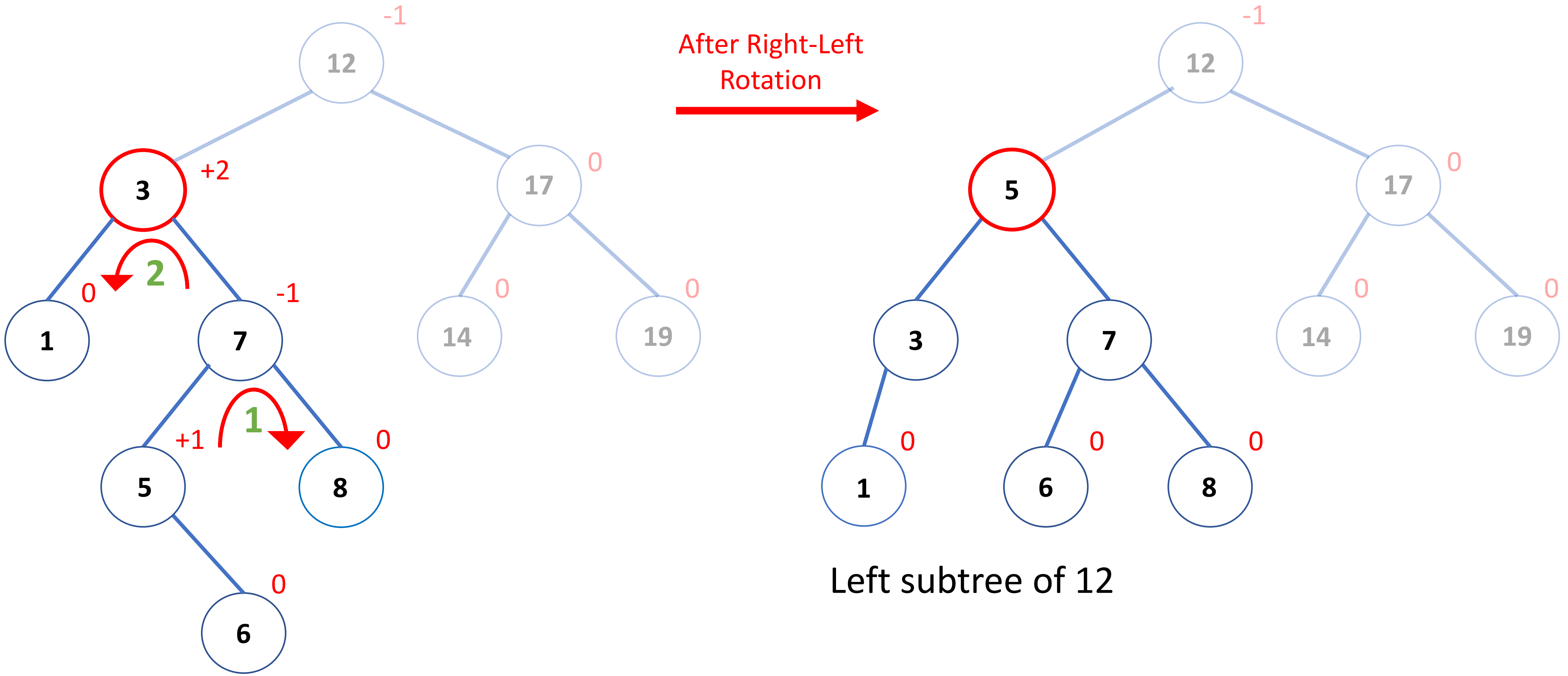
Rebalancing left subtree of 12: Double Right-Left Rotation



Rebalancing left subtree of 12: Double Right-Left Rotation

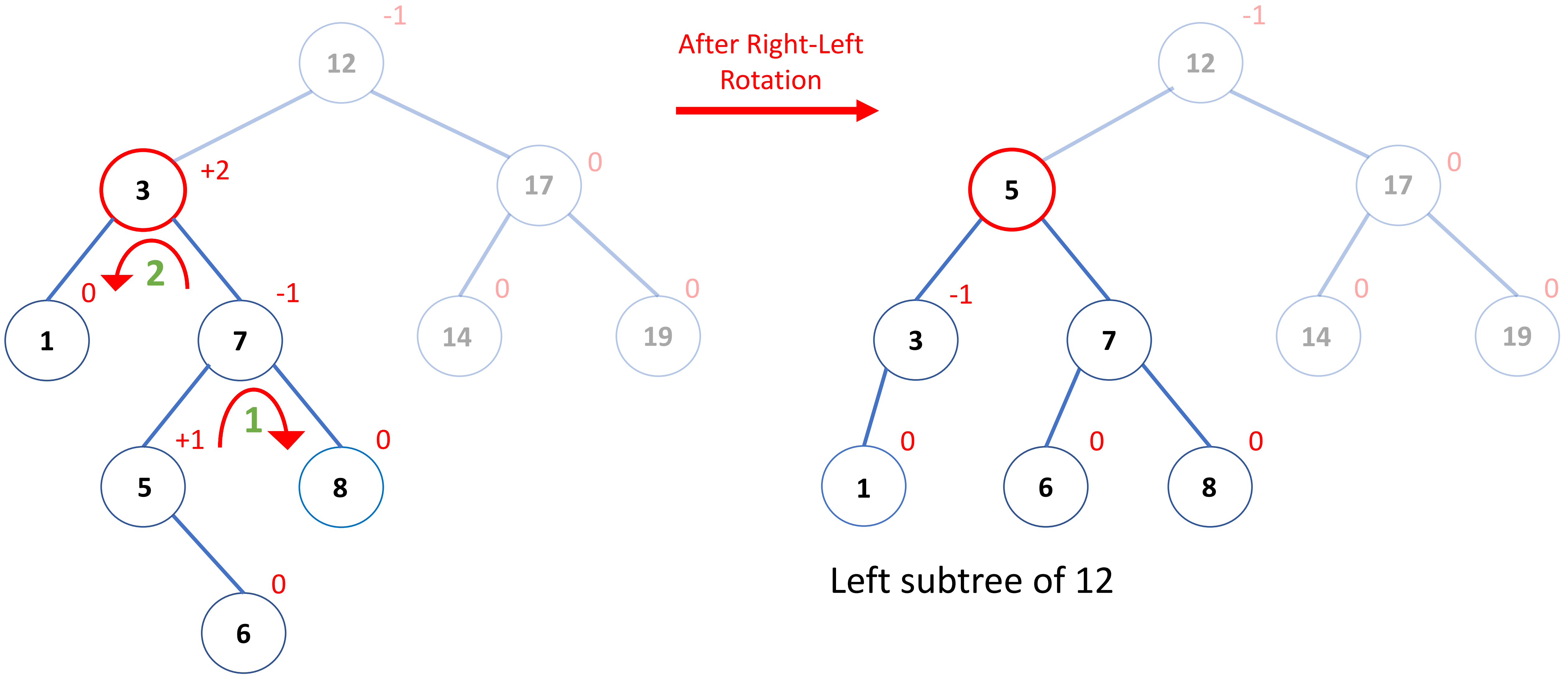


Rebalancing left subtree of 12: Double Right-Left Rotation



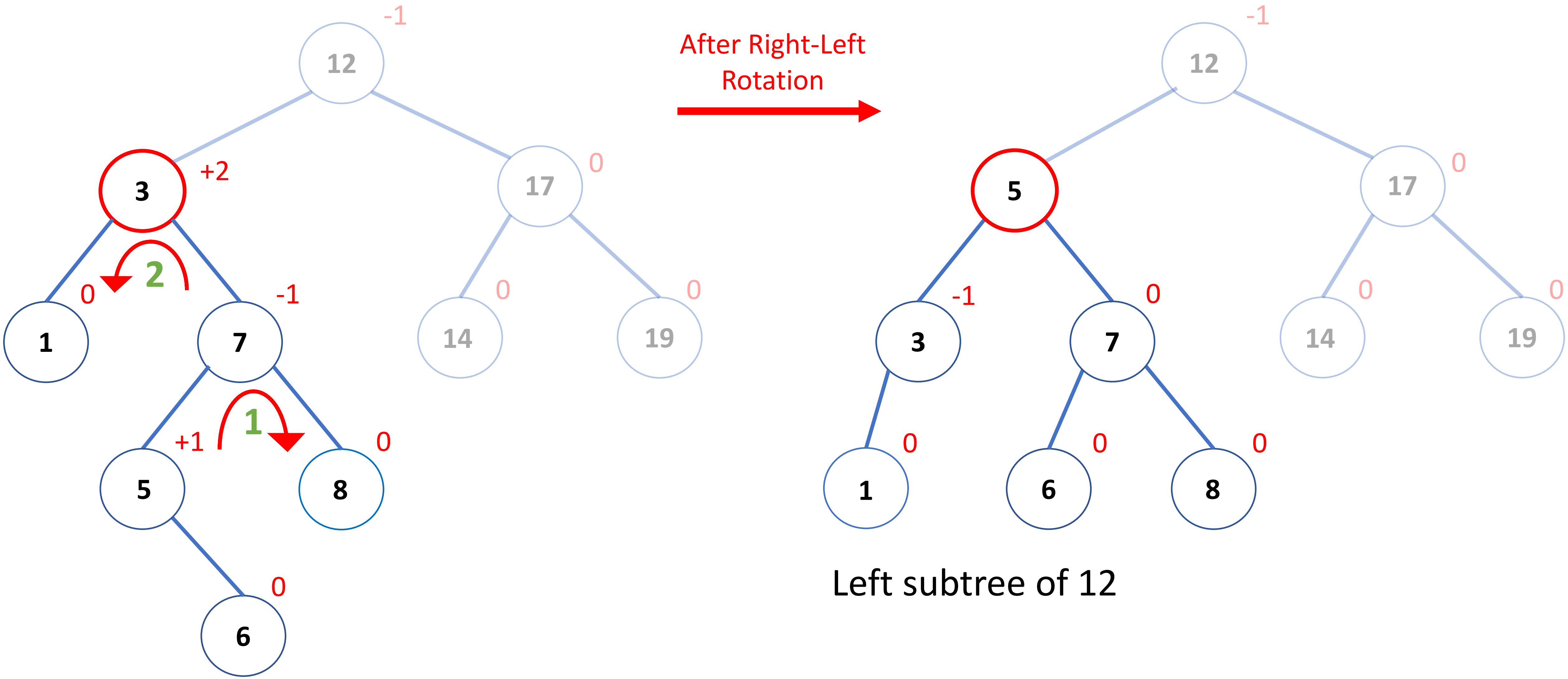
Left subtree of 12

Rebalancing left subtree of 12: Double Right-Left Rotation



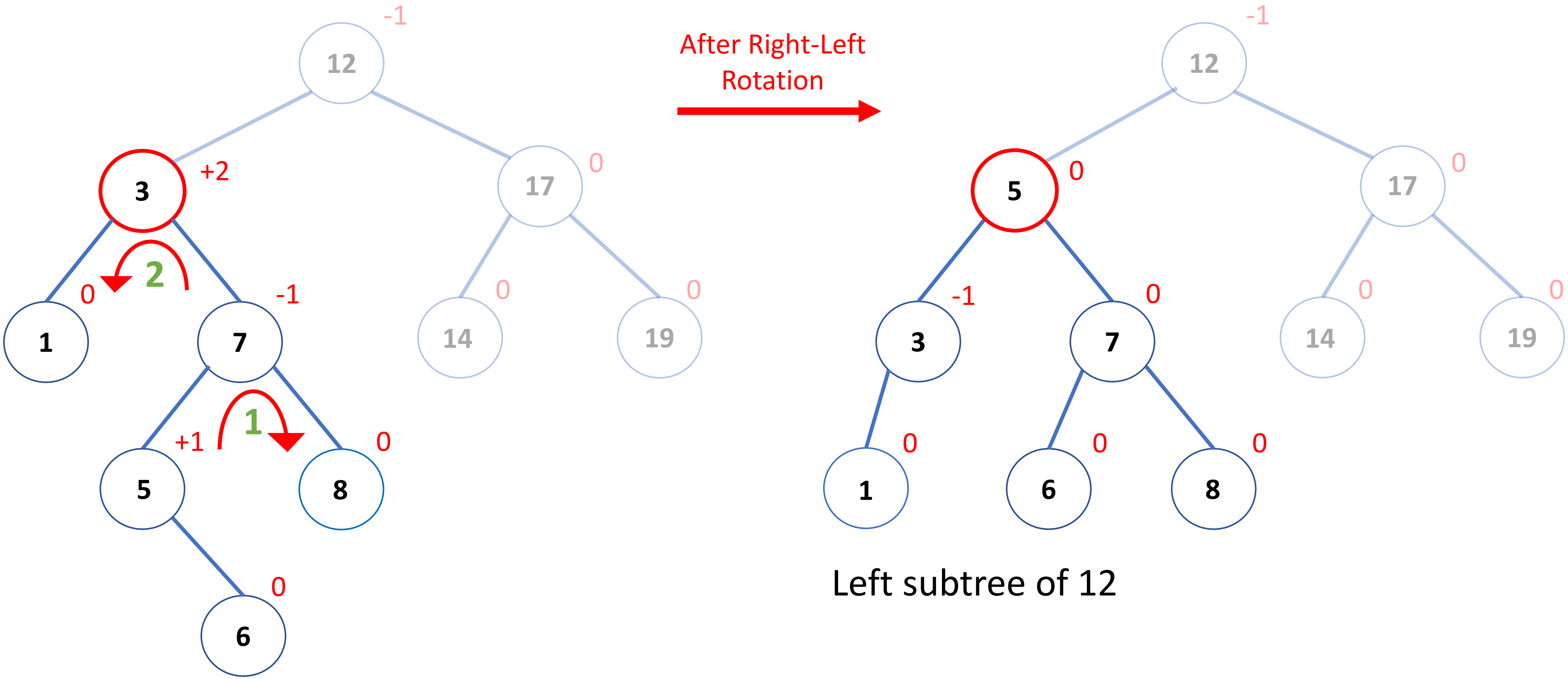
Left subtree of 12

Rebalancing left subtree of 12: Double Right-Left Rotation



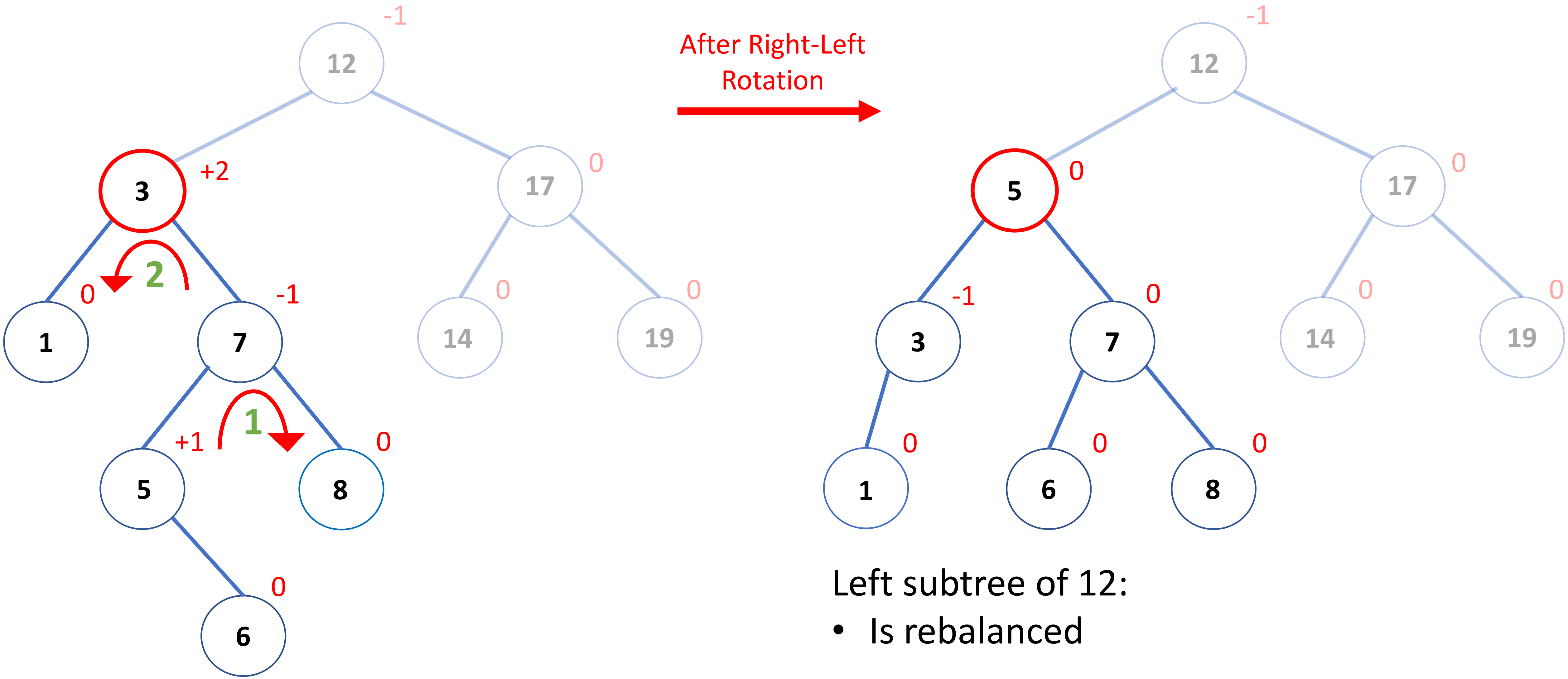
Left subtree of 12

Rebalancing left subtree of 12: Double Right-Left Rotation



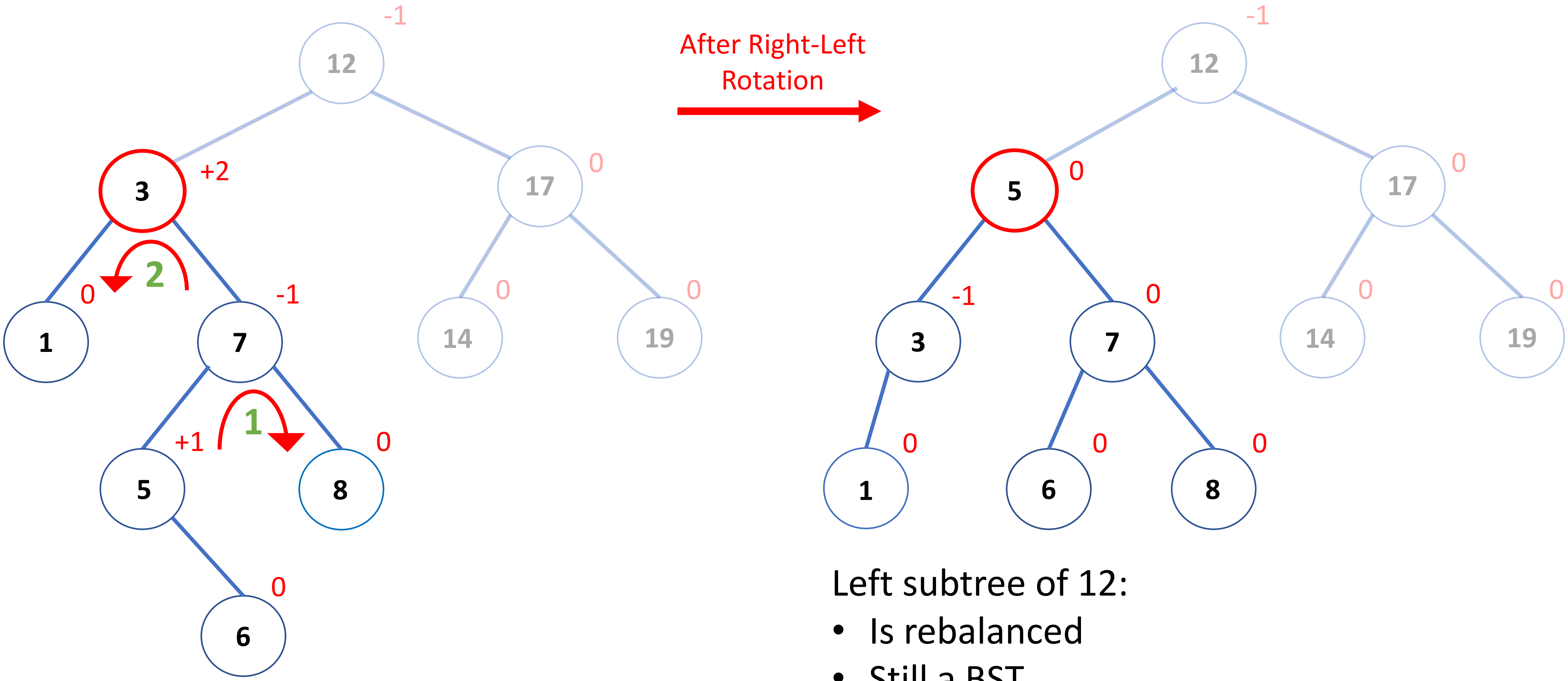
Left subtree of 12

Rebalancing left subtree of 12: Double Right-Left Rotation



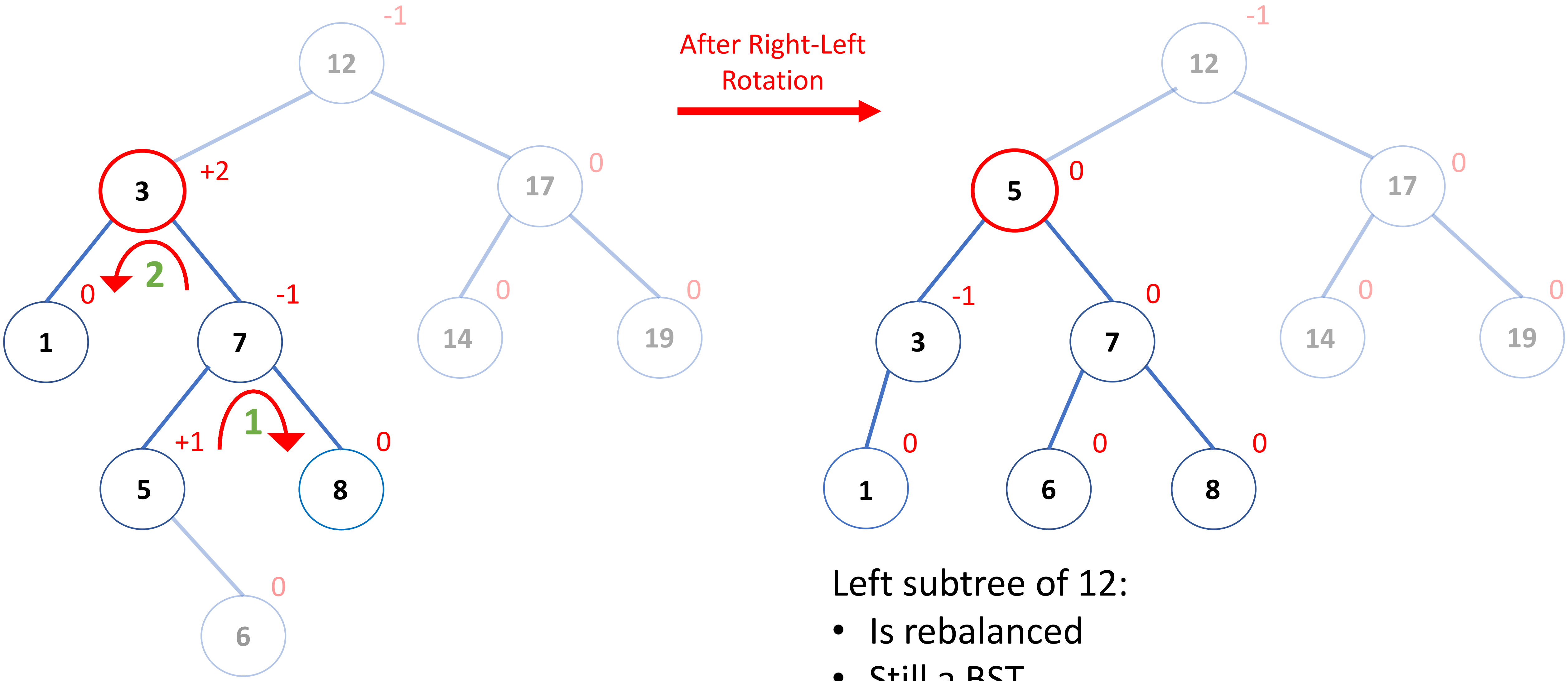
Left subtree of 12:
- Is rebalanced

Rebalancing left subtree of 12: Double Right-Left Rotation



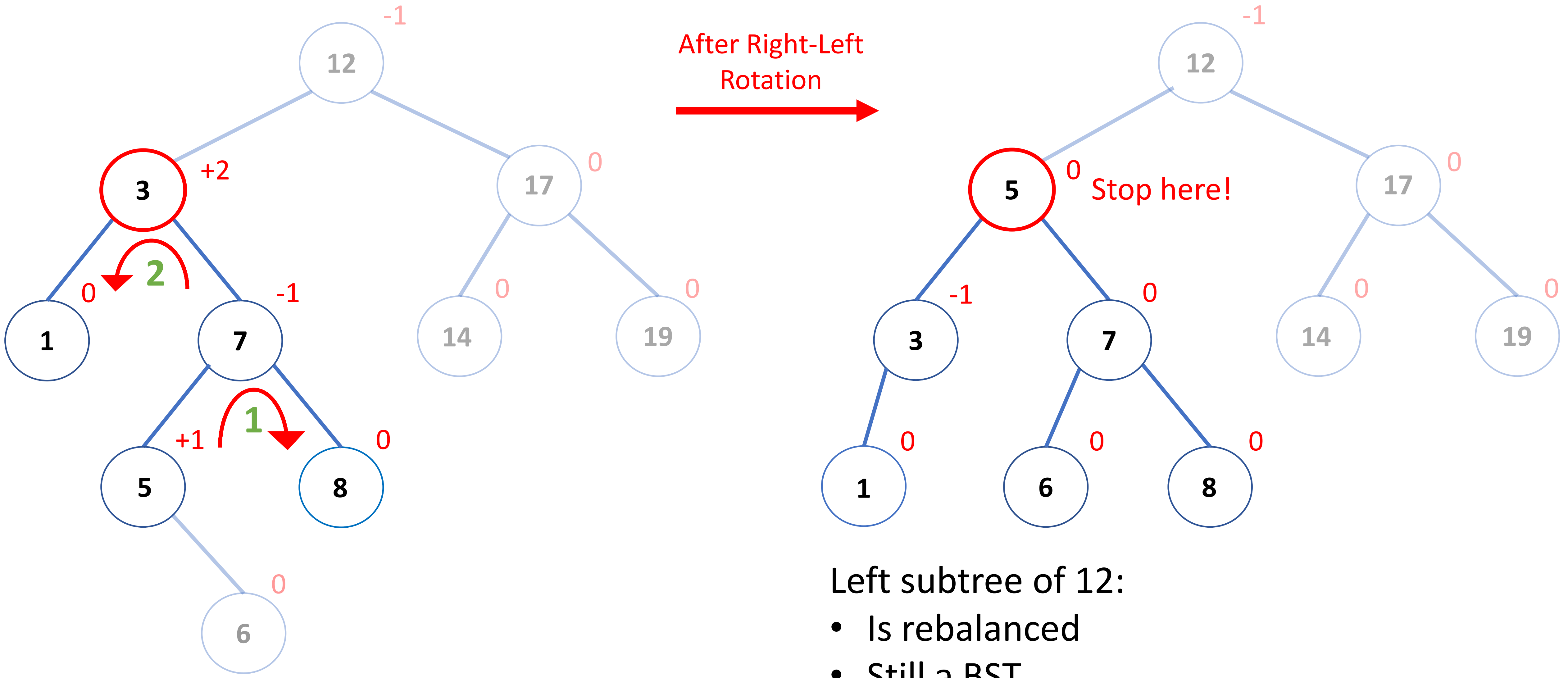
Left subtree of 12:
- Is rebalanced
- Still a BST

Rebalancing left subtree of 12: Double Right-Left Rotation



Left subtree of 12:
- Is rebalanced
- Still a BST
- **Bonus**: height same as before inserting 6

Rebalancing left subtree of 12: Double Right-Left Rotation



Left subtree of 12:

- Is rebalanced
- Still a BST
- **Bonus**: height same as before inserting 6