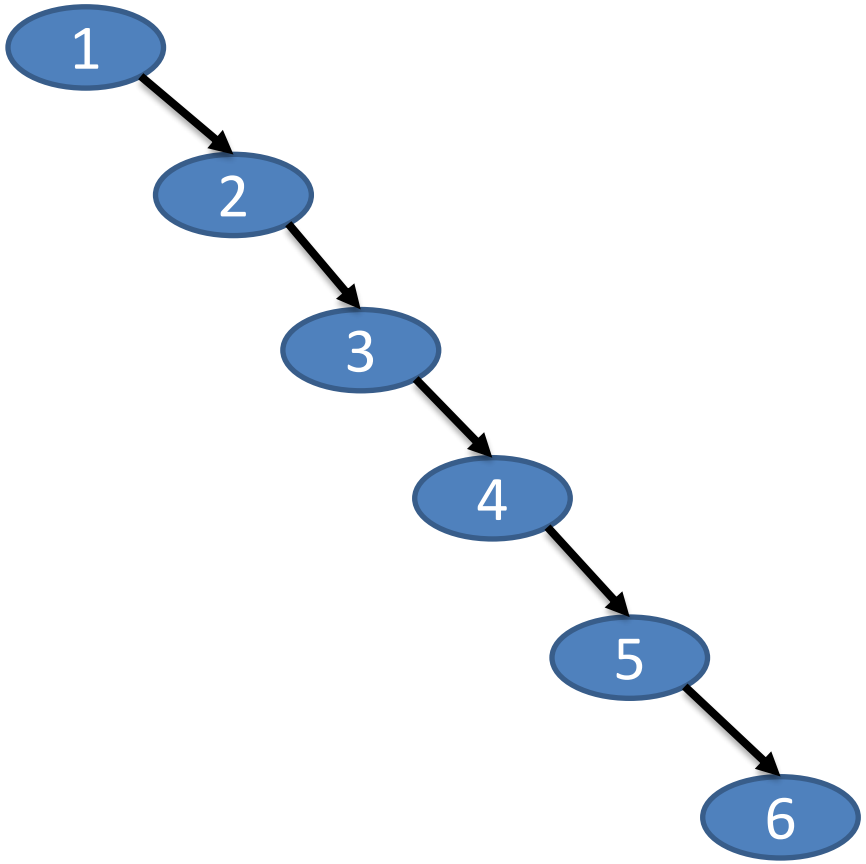
AVL Trees

CS223: Data Structures

Imbalanced trees are similar to linked lists

Build a BST for the this sequence of values 1,2,3,4,5,6

Trees need to be balanced

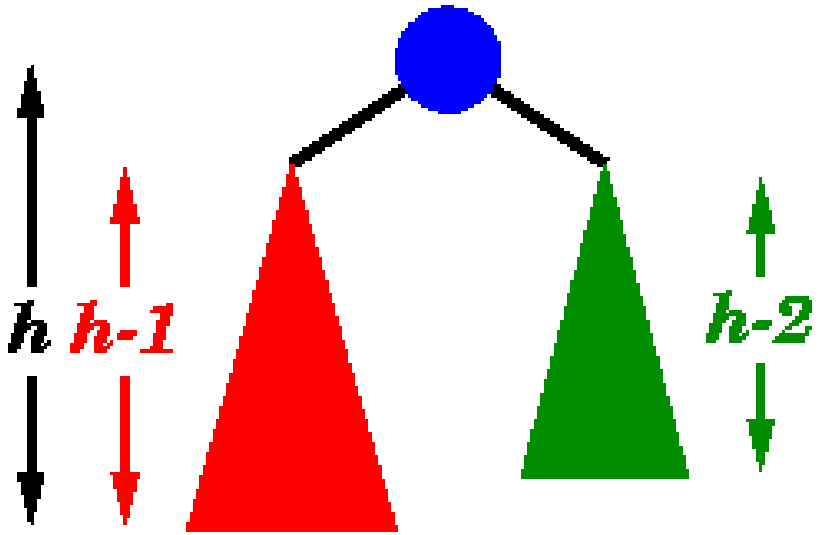


AVL trees

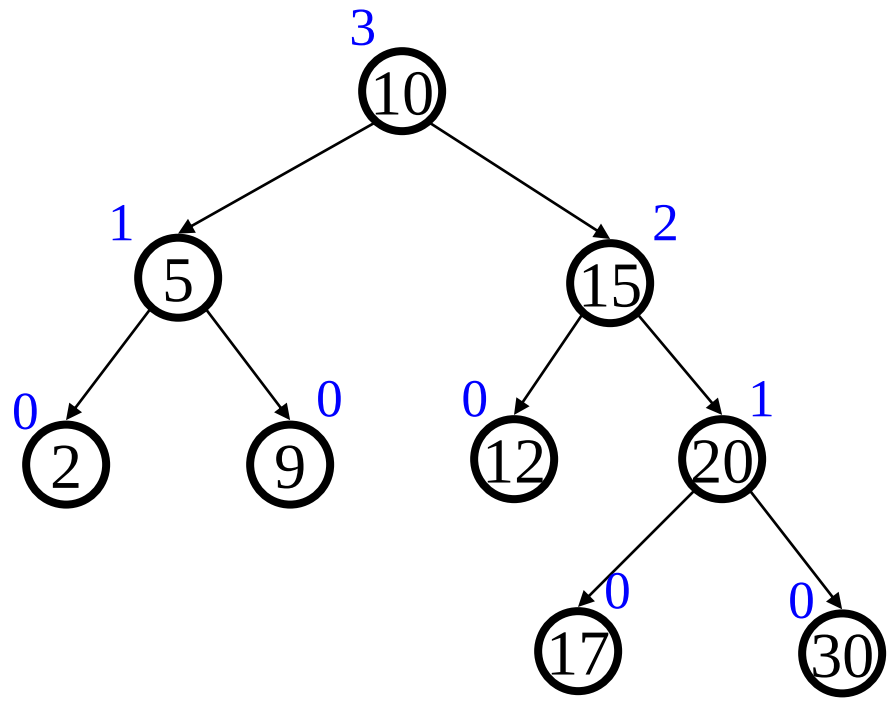
- Named after the inventors **A**delson-**V**elskii and **L**andis.
- The tree is dynamically balanced.
- Binary search tree with **balance condition** in which the sub-trees of each node can differ by at most 1 in their height.
  - for every internal node v , the *heights of the children of v can differ by at most 1.*
- AVL tree remains **balanced** by applying “rotation” operations.

Properties of AVL

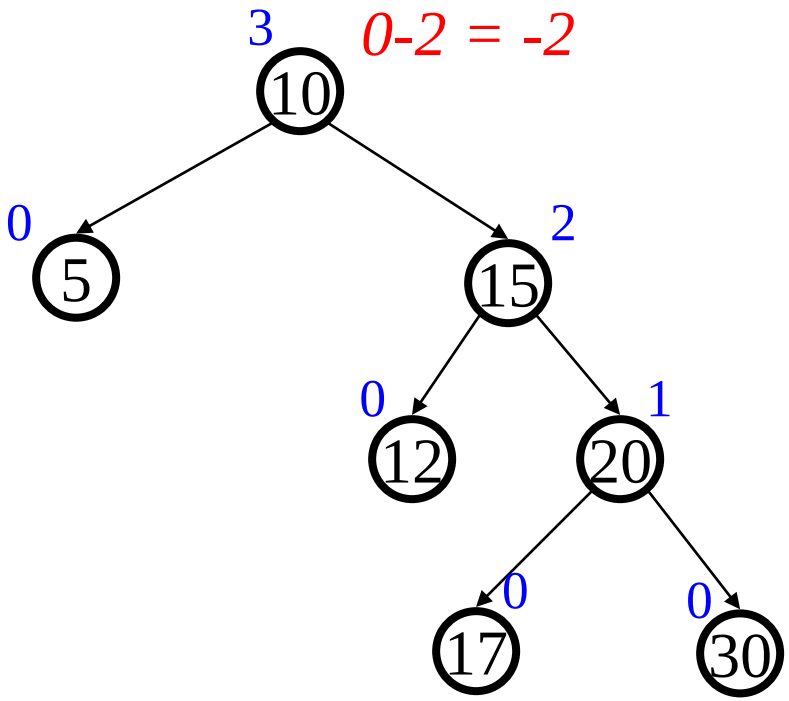
1. Sub-trees of each node can differ by at most 1 in their height
2. Every sub-trees is an AVL tree

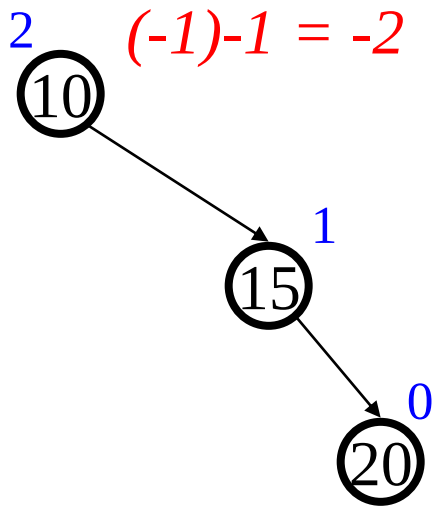


An AVL Tree



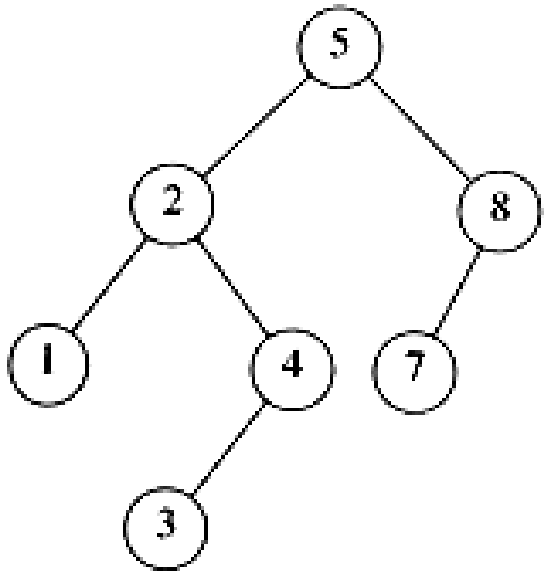
Not AVL Trees





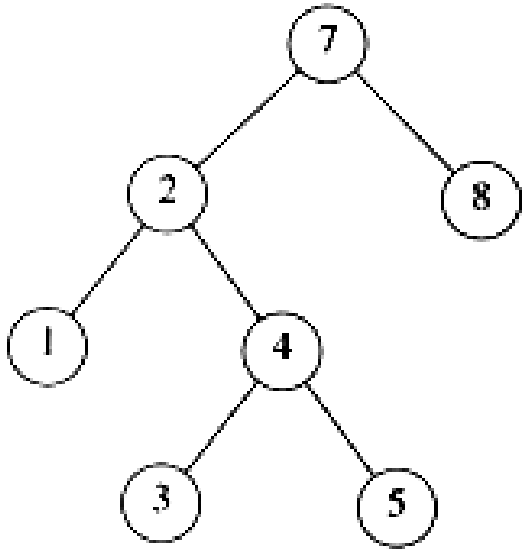
Note: height(empty tree) == -1

Which of these can be considered AVL?



YES

Each left sub-tree has height 1 greater than each right sub-tree



NO

Left sub-tree has height 2, but right sub-tree has height 0

Insertions and deletions

- It is performed as in binary search trees.
- If the balance is destroyed, rotation(s) is performed to re-balance the tree.
- Perform suitable rotations starting from the leafs and up toward the root.

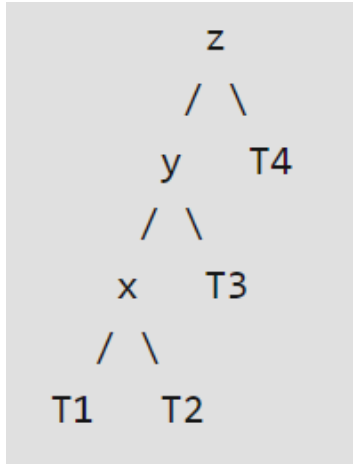
Insertion

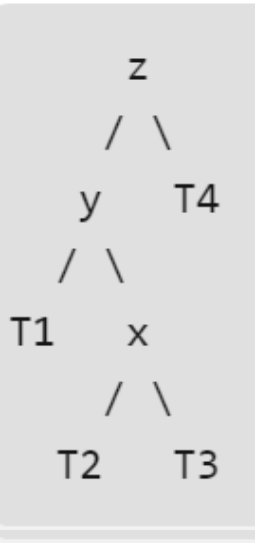
1) Perform standard BST insert for node *w*.
2) Starting from *w*, travel up and find the first unbalanced node. Assume that:

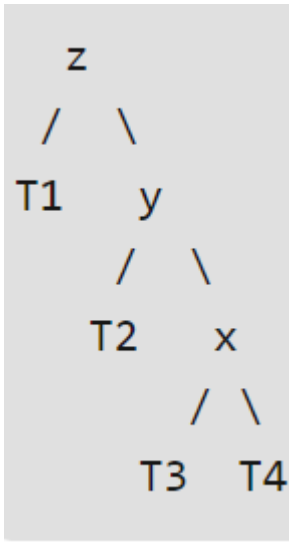
- z be the first unbalanced node,
- y be the child of z that comes on the path from w to z
- x be the grandchild of z that comes on the path from w to z.

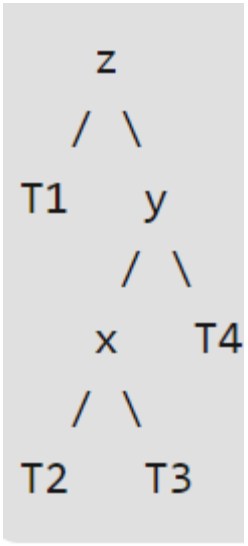
3) Re-balance the tree by performing appropriate rotations on the subtree rooted with z.

- There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways.









Insertion

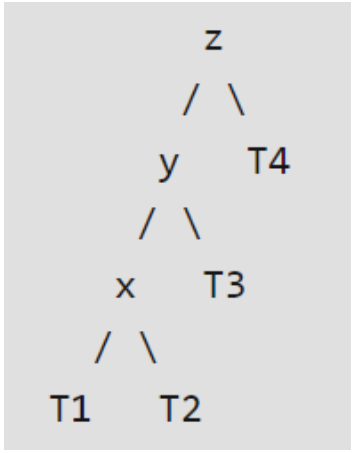
The possible 4 arrangements are:

a) y is left child of z and x is left child of y (Left Left Case)

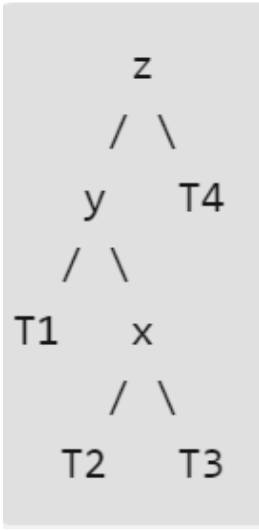
b) y is left child of z and x is right child of y (Left Right Case)

c) y is right child of z and x is right child of y (Right Right Case)

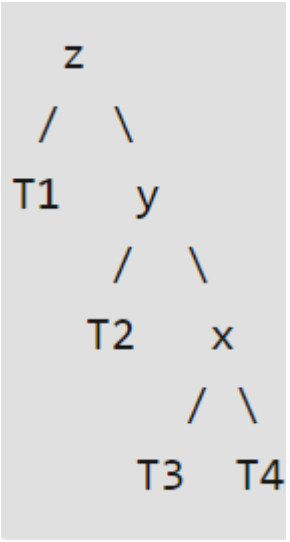
d) y is right child of z and x is left child of y (Right Left Case)



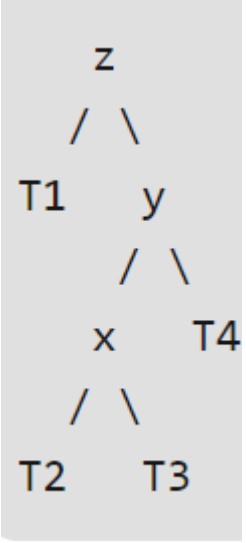
Left Left Case



Left Right Case



Right Right Case

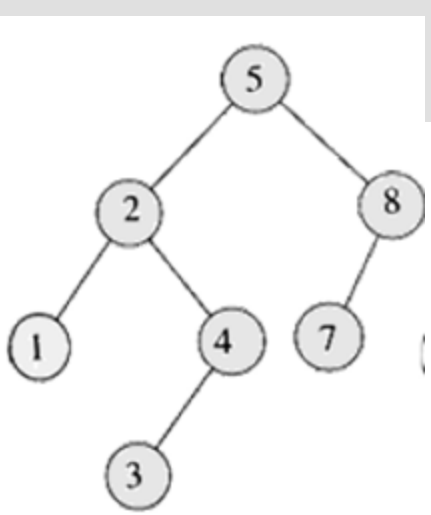


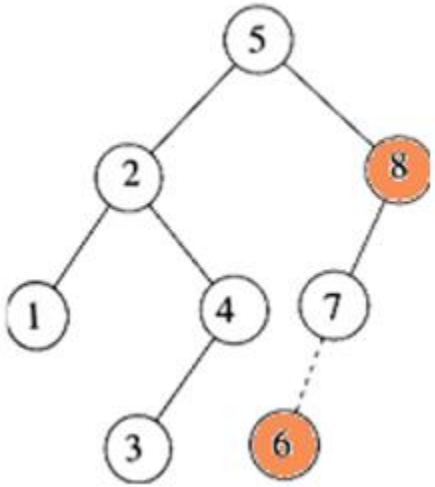
Right Left Case

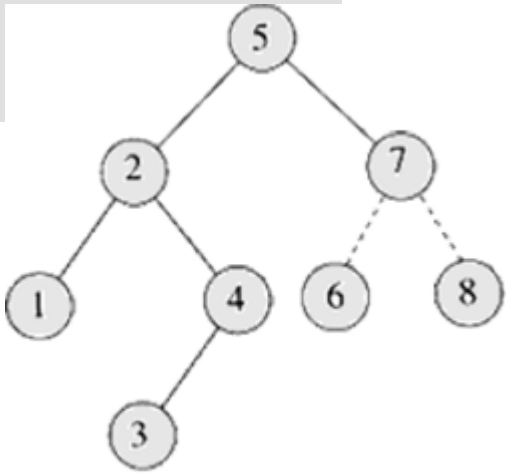
Left Left Case

```
T1, T2, T3 and T4 are subtrees.
          z                                      y
         / \                                   /   \
        y   T4      Right Rotate (z)          x      z
       / \          - - - - - - - - ->       / \    / \
      x   T3                                T1  T2 T3  T4
     / \
   T1   T2
```

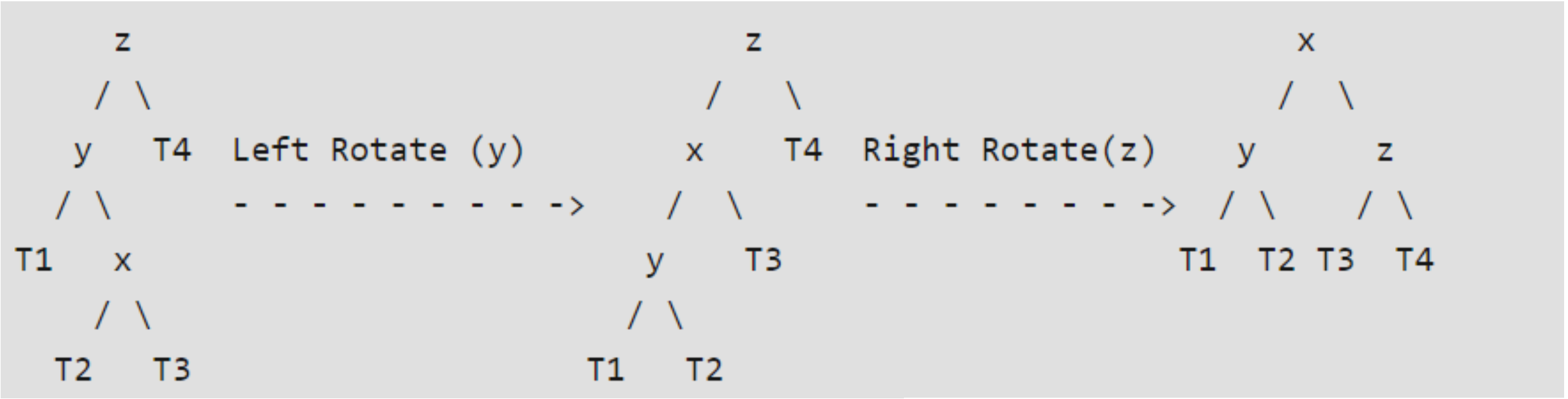
Right rotate 8, it becomes right child of 7

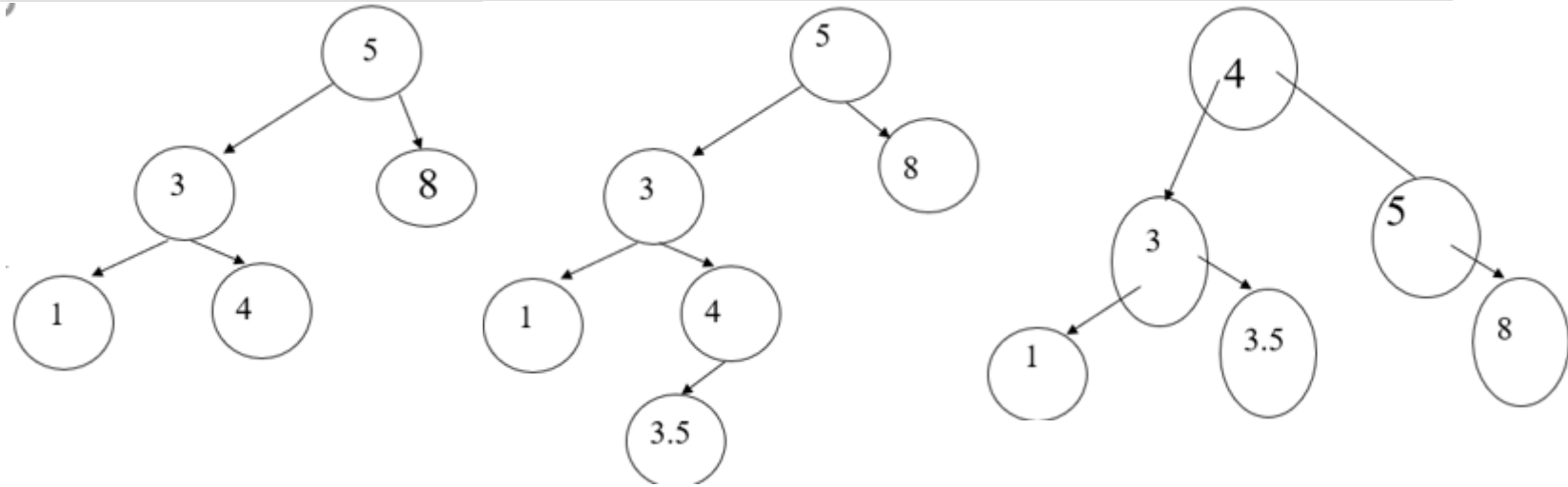






Left Right Case

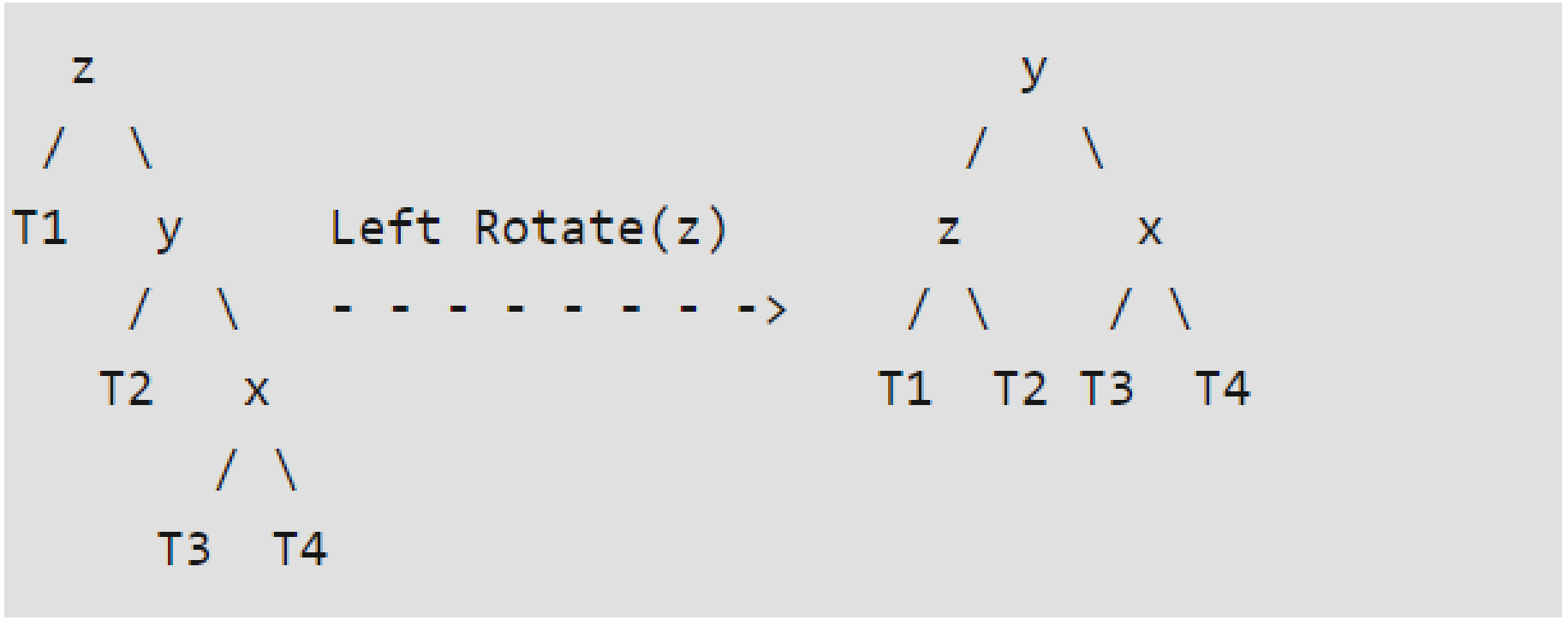




First, left rotate 3, it becomes left child of 4

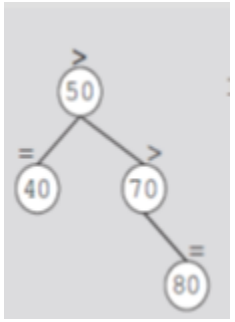
Second, right rotate 5, it becomes right child of 4

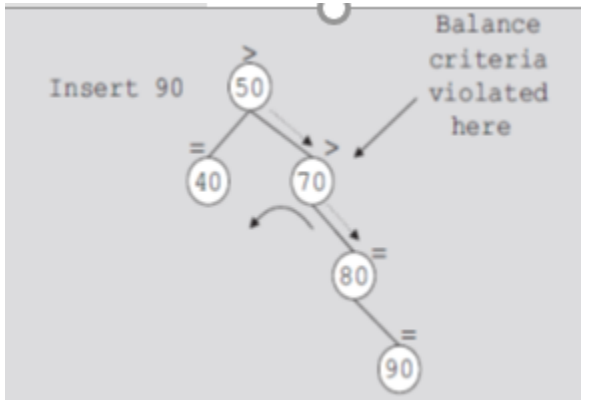
Right Right Case

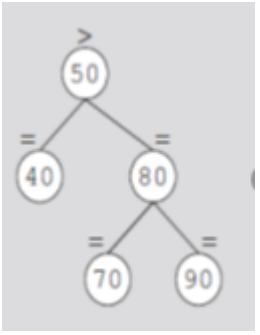


Example 1

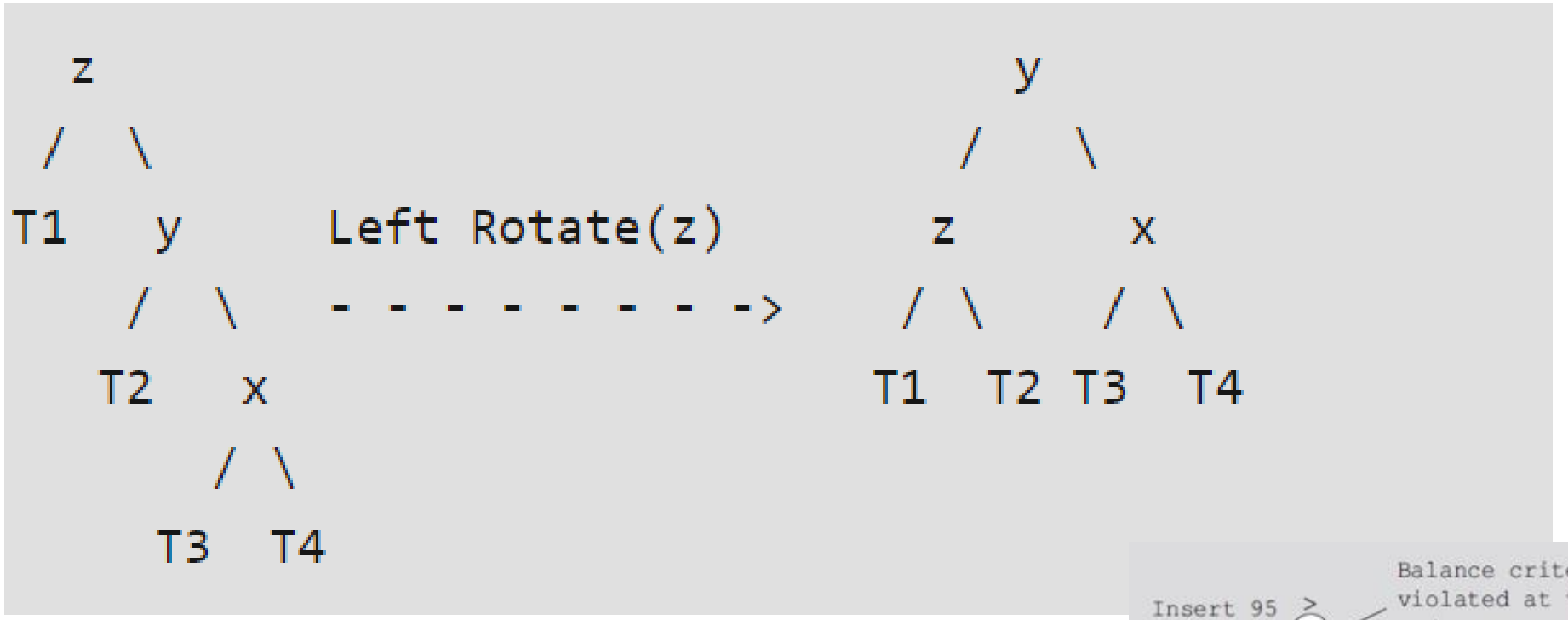
Left rotate 70, it becomes left child of 80



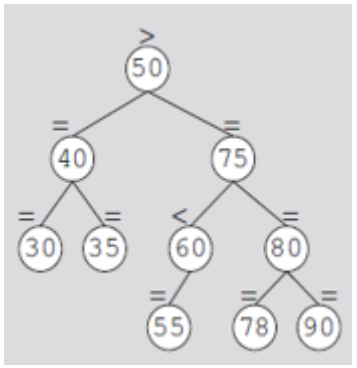


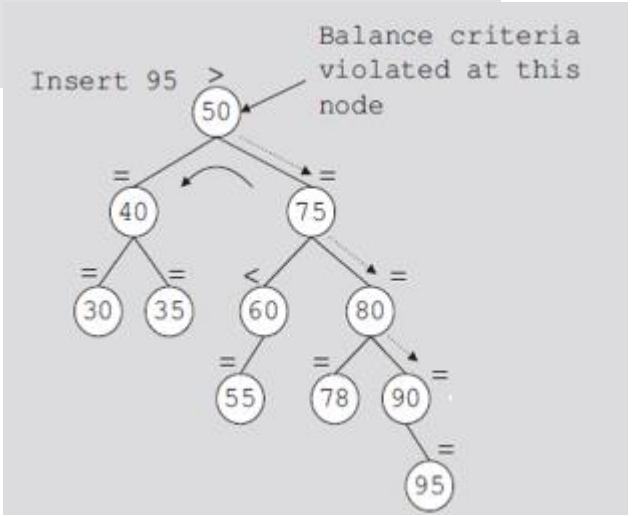


Right Right Case

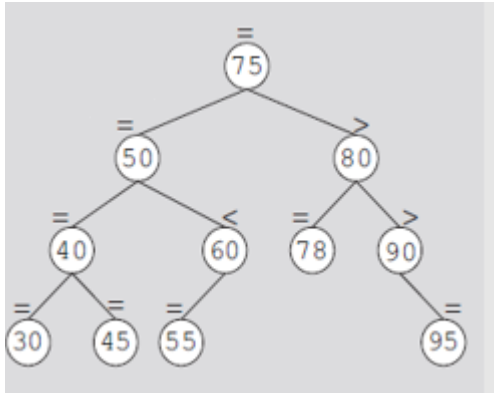


Left rotate 50, it becomes left child of 75

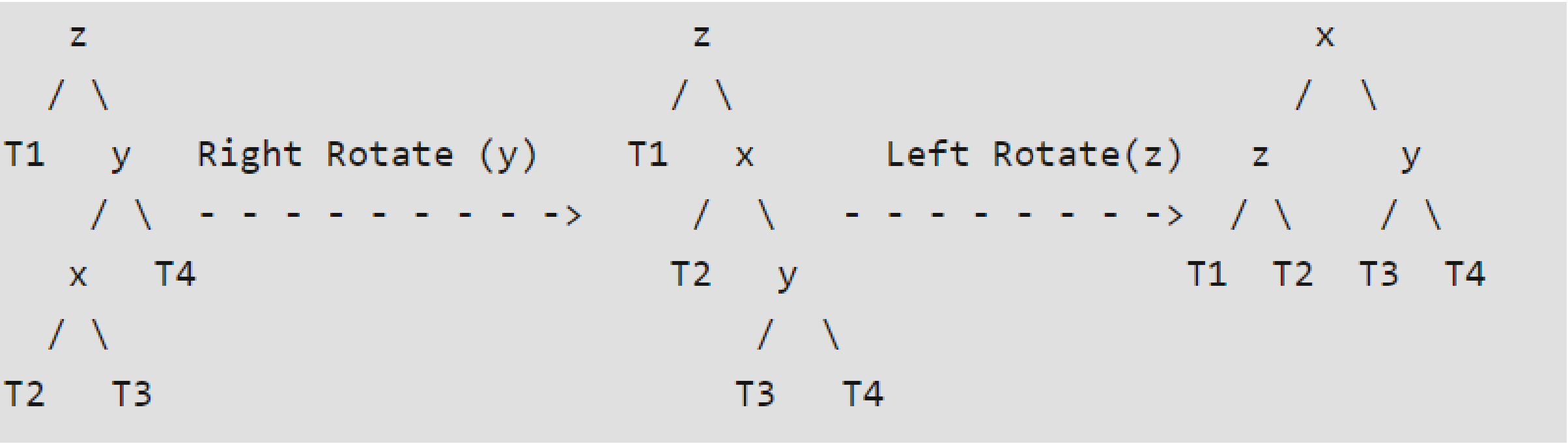




Example 2

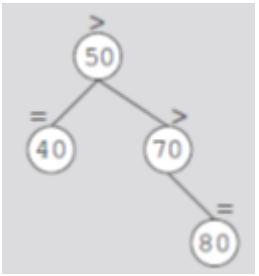


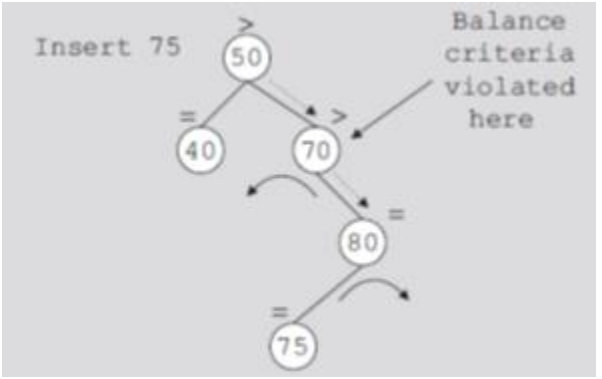
Right Left Case

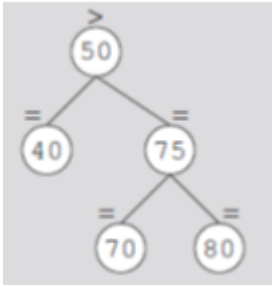


First, right rotate 80, it becomes right child of 75 and 75 become right child of 70

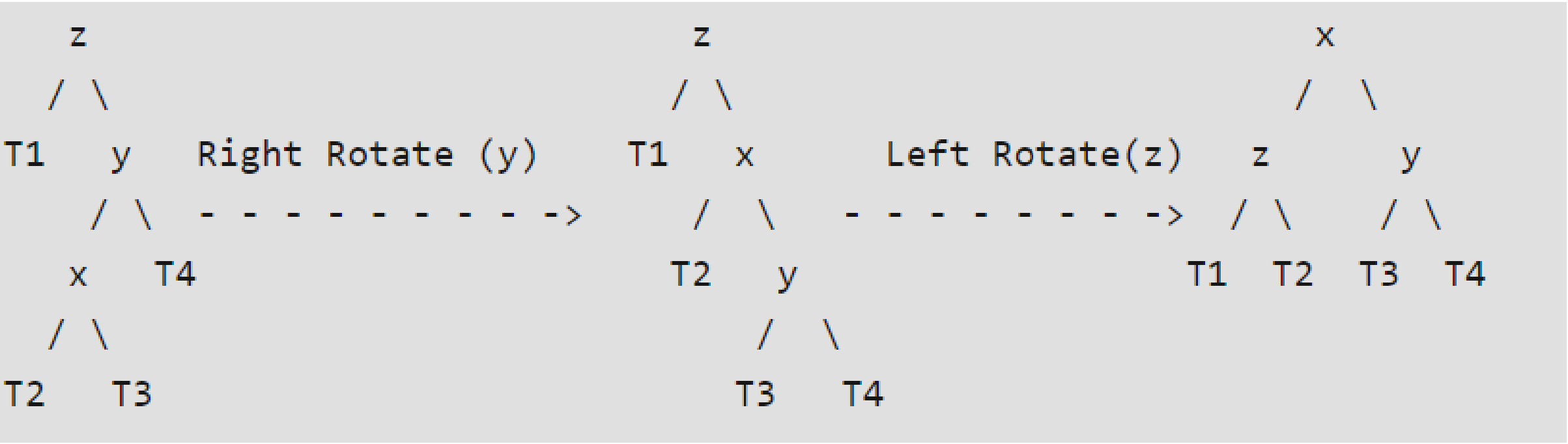
Second, left rotate 70, it becomes left child of 75





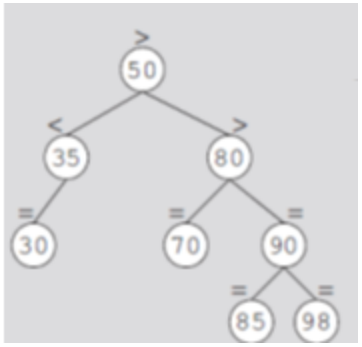


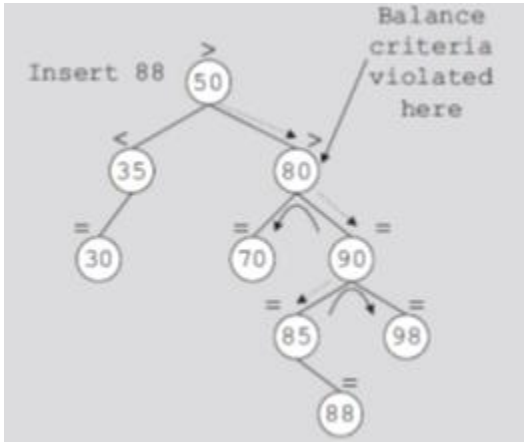
Right Left Case

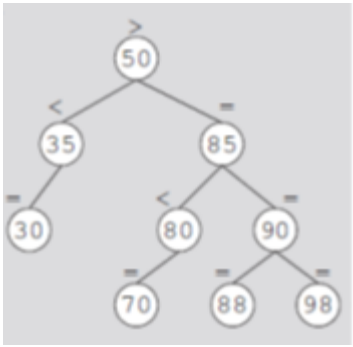


First, right rotate 90, it becomes right child of 85 and 85 become right child of 80

Second, left rotate 80, it becomes left child of 85

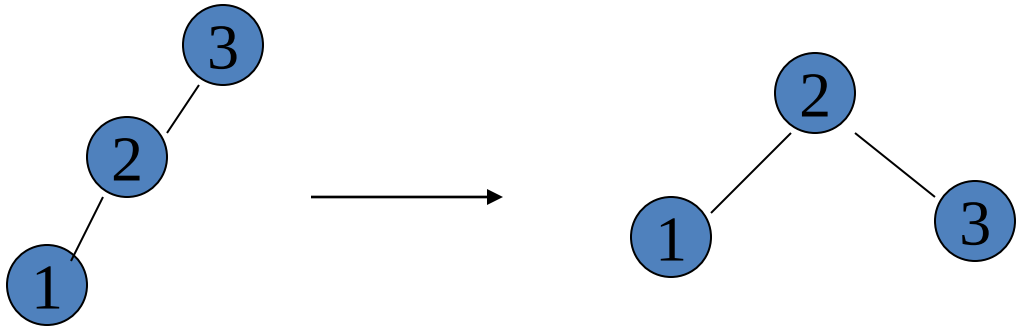






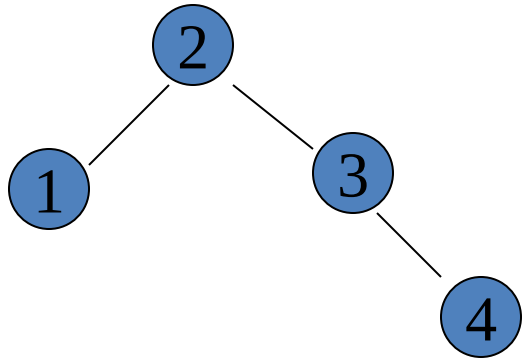
Example

Inserting 3, 2, 1, and then 4 to 7 sequentially into empty AVL tree

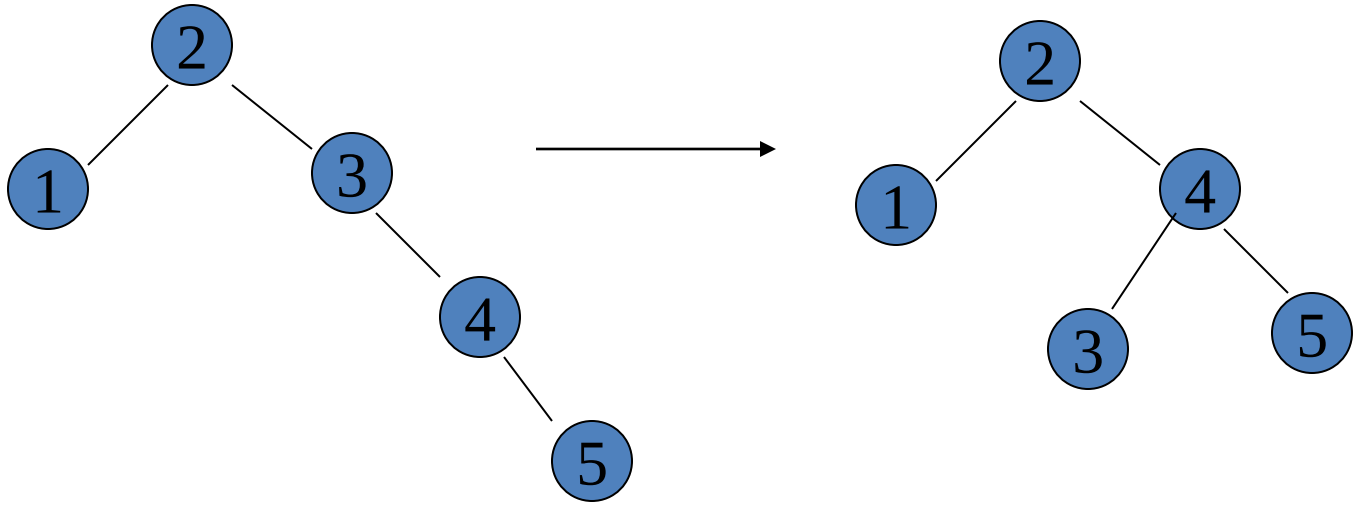


Example (Cont'd)

Inserting 4

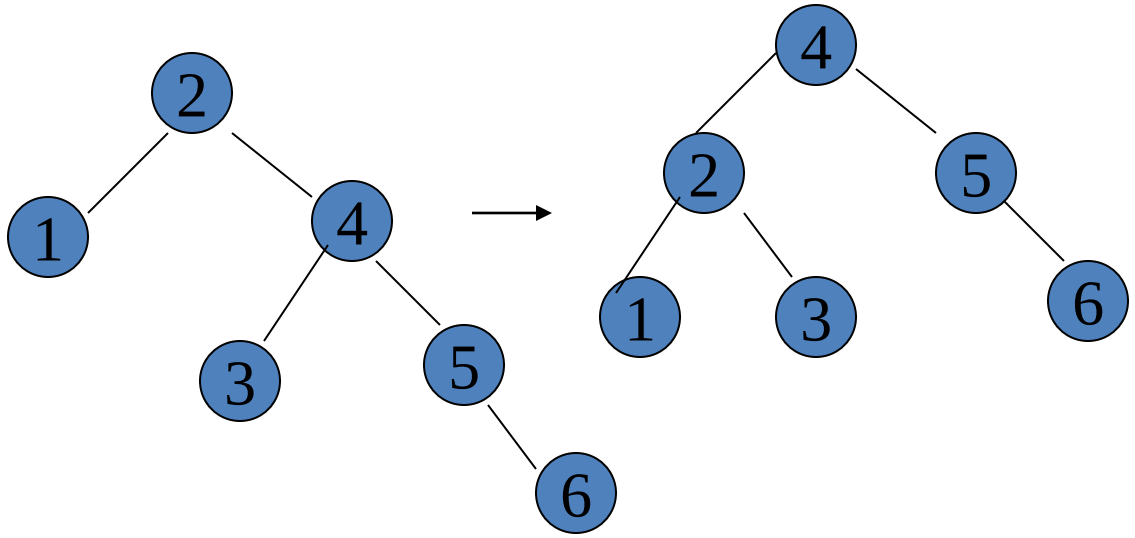


Inserting 5

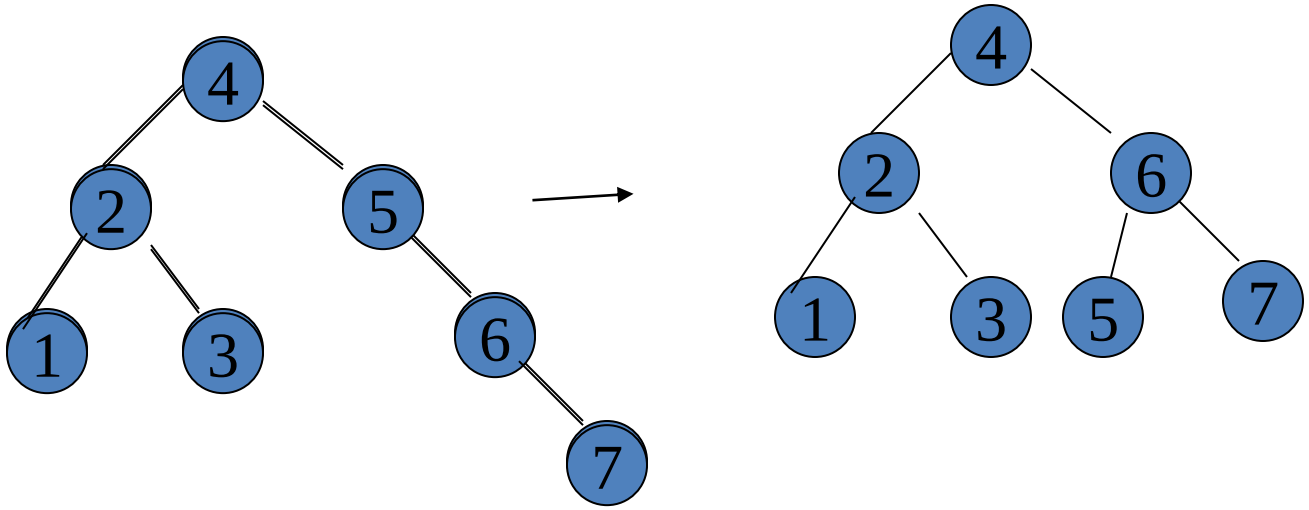


Example (Cont'd)

Inserting 6



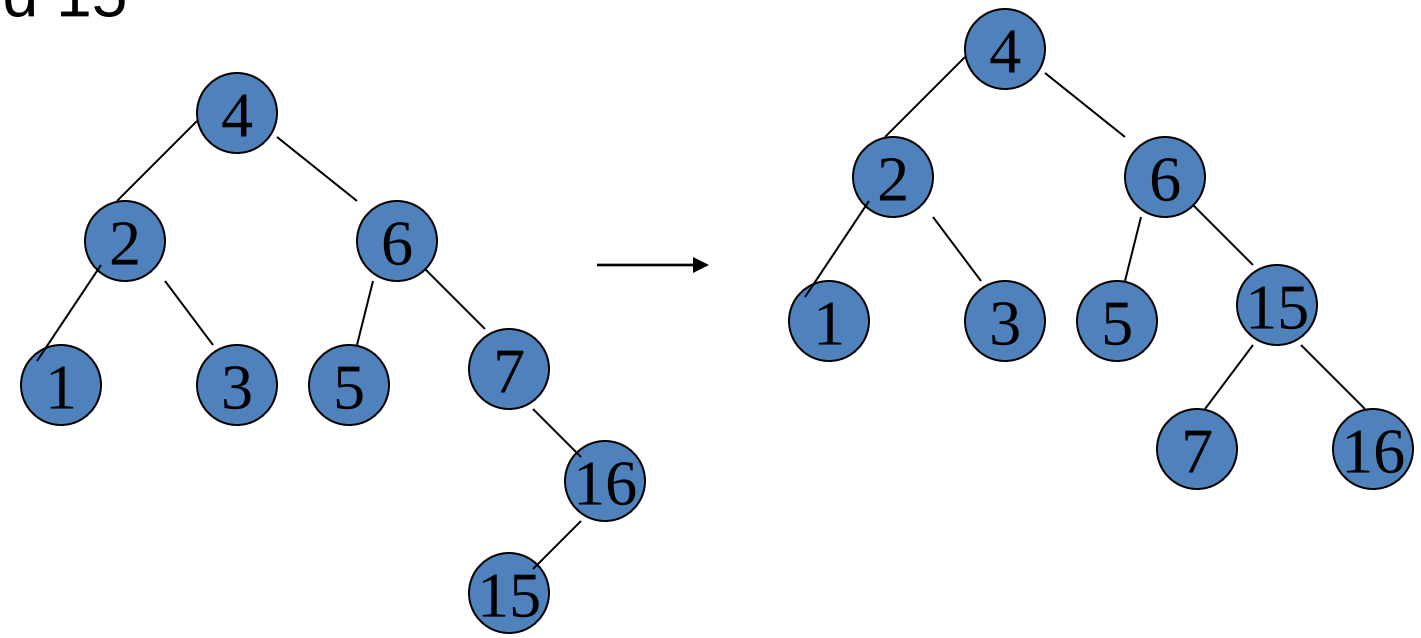
Inserting 7



Example

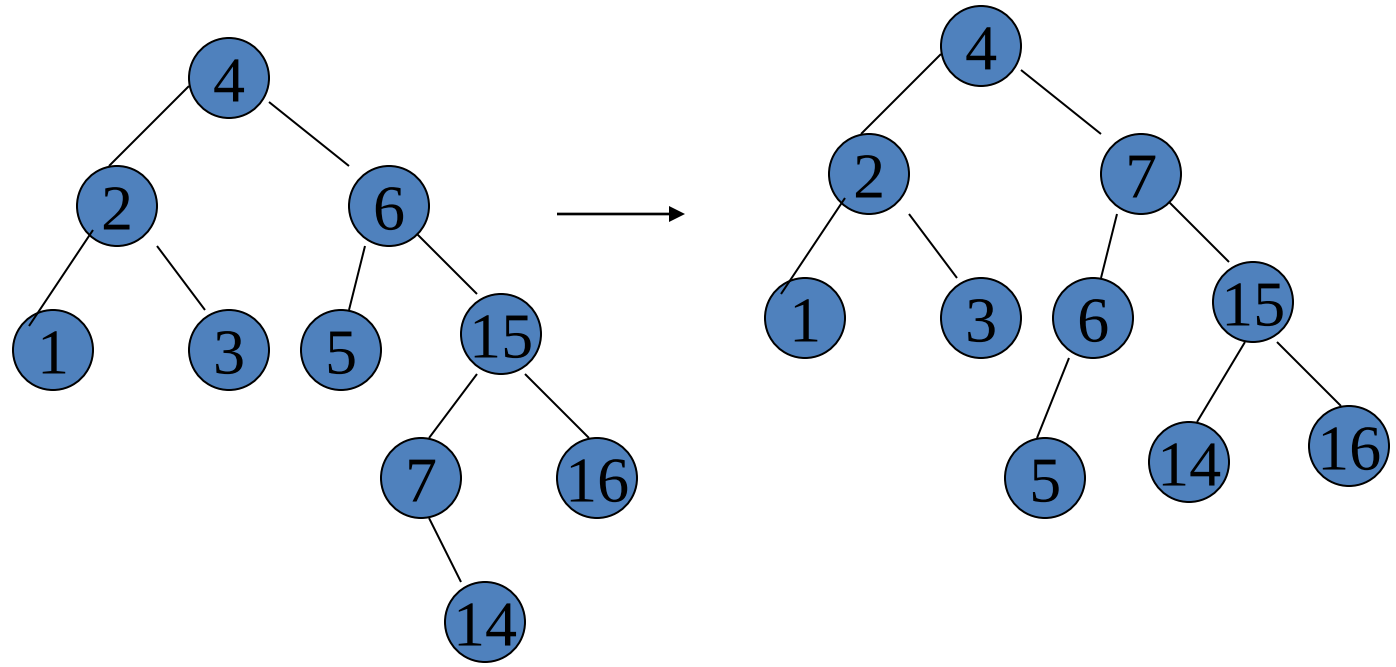
Continuing the previous example by inserting
16, 15, and 14

Inserting 16 and 15

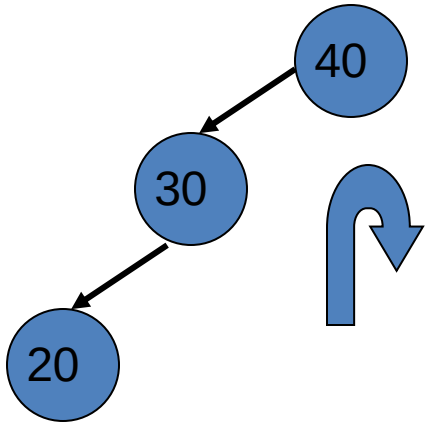


Example (Cont'd)

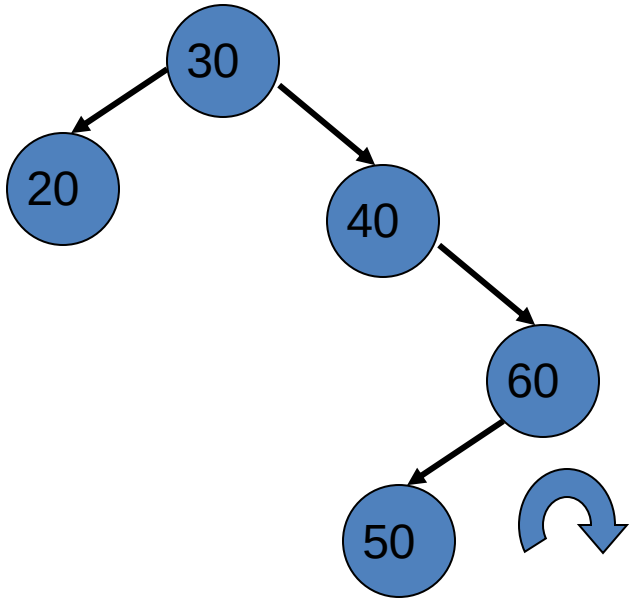
Inserting 14



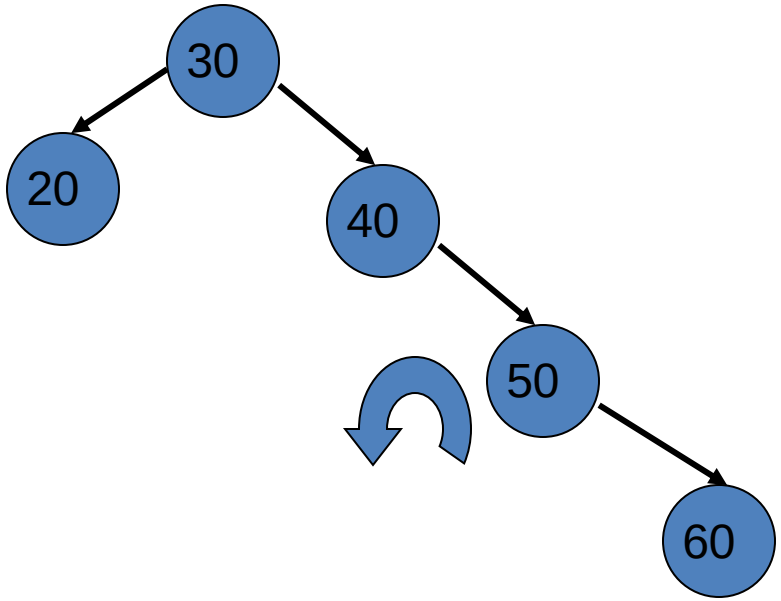
40 30 20 60 50 80 15 28 25



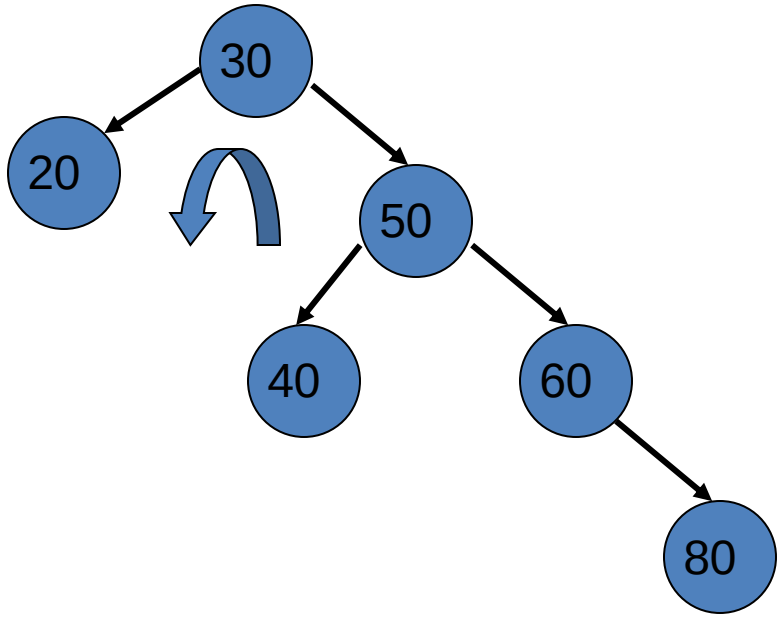
40 30 20 60 50 80 15 28 25



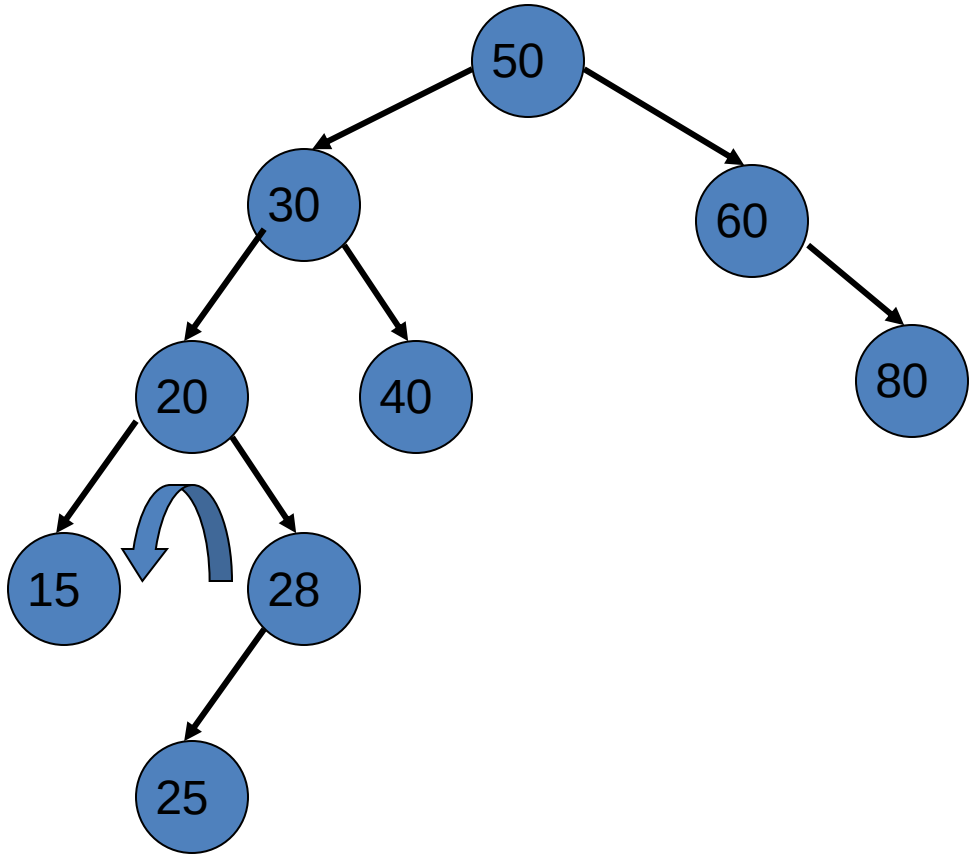
40 30 20 60 50 80 15 28 25



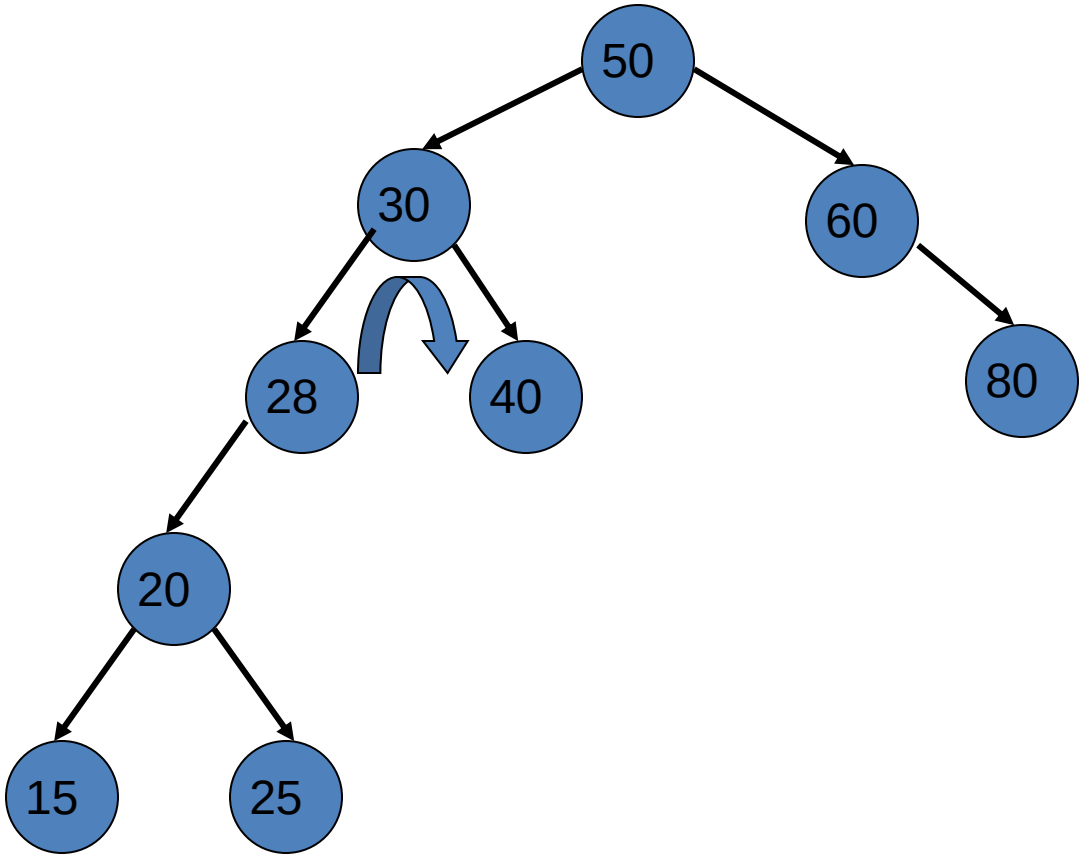
40 30 20 60 50 80 15 28 25



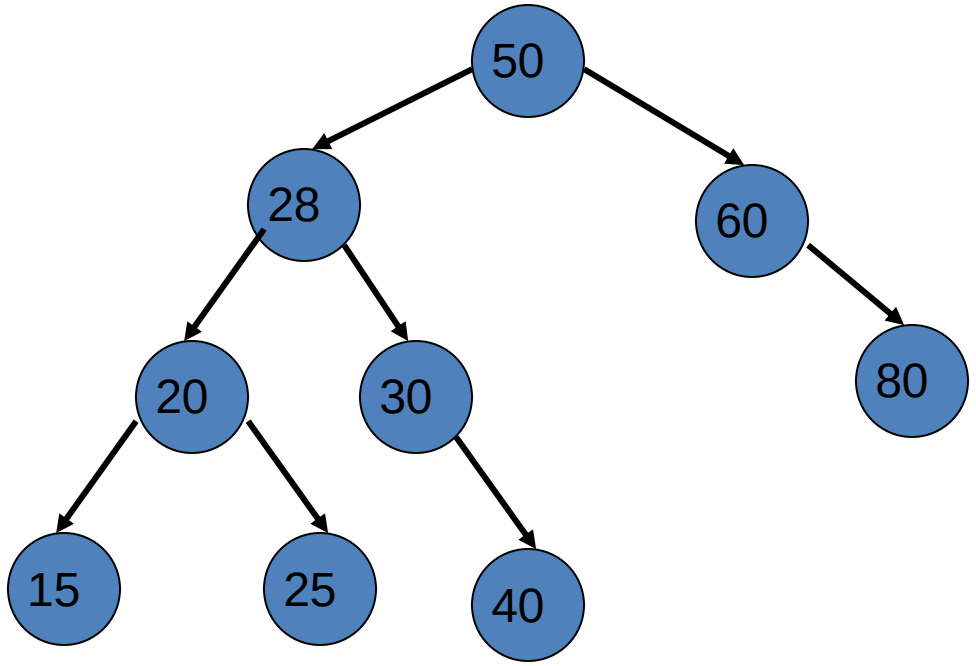
40 30 20 60 50 80 15 28 25



40 30 20 60 50 80 15 28 25



40 30 20 60 50 80 15 28 25



Deletion

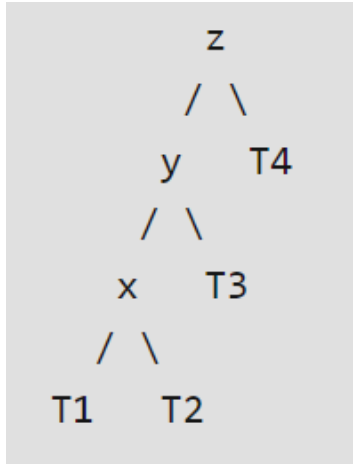
1) Perform standard BST delete for *w*.

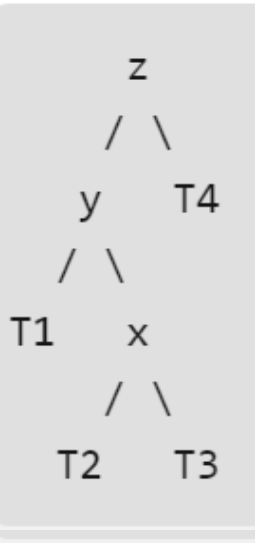
2) Starting from *w*, travel up and find the first unbalanced node. Assume that:

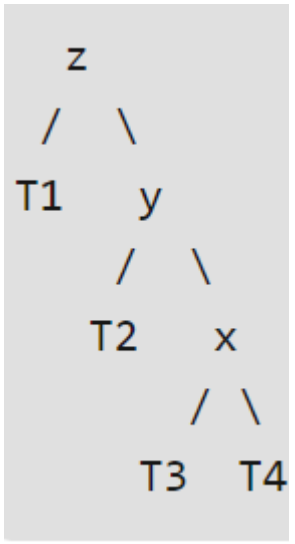
- z be the first unbalanced node,
- y be the larger height child of z
- x be the larger height child of y.

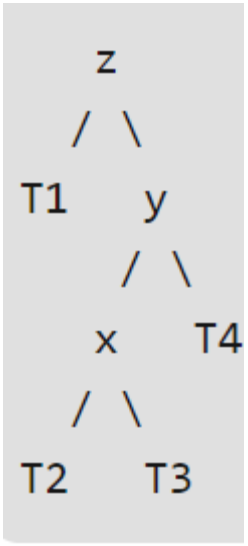
3) Re-balance the tree by performing appropriate rotations on the subtree rooted with z.

- There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways.





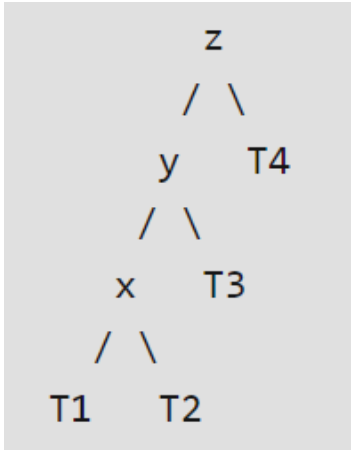




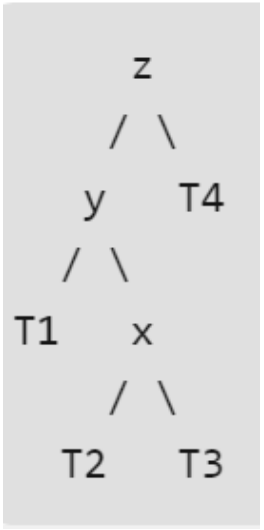
Deletion

The possible 4 arrangements are:

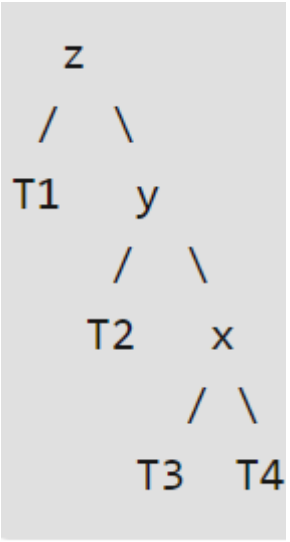
a) y is left child of z and x is left child of y (Left Left Case)
b) y is left child of z and x is right child of y (Left Right Case)
c) y is right child of z and x is right child of y (Right Right Case)
d) y is right child of z and x is left child of y (Right Left Case)



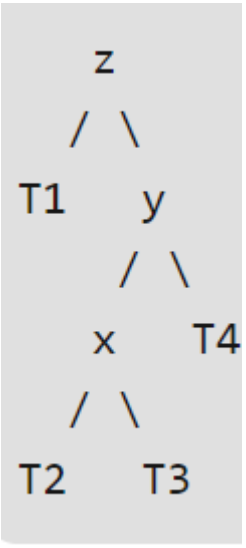
Left Left Case



Left Right Case

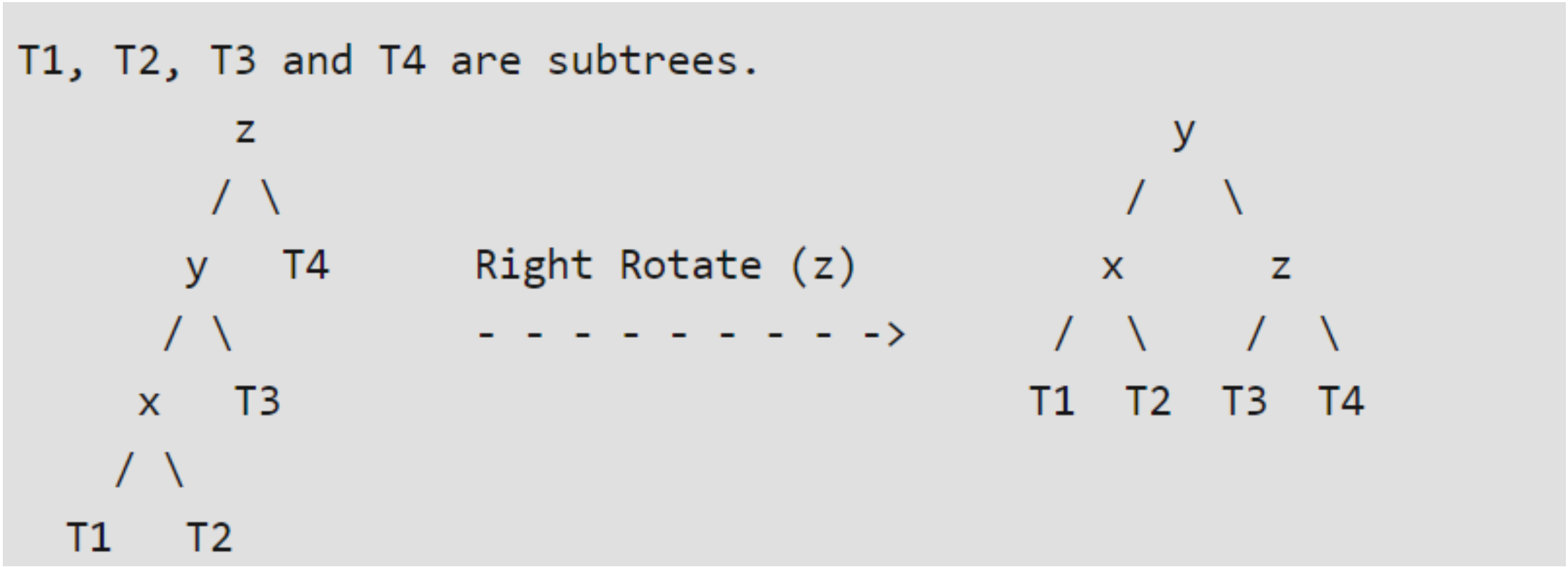


Right Right Case



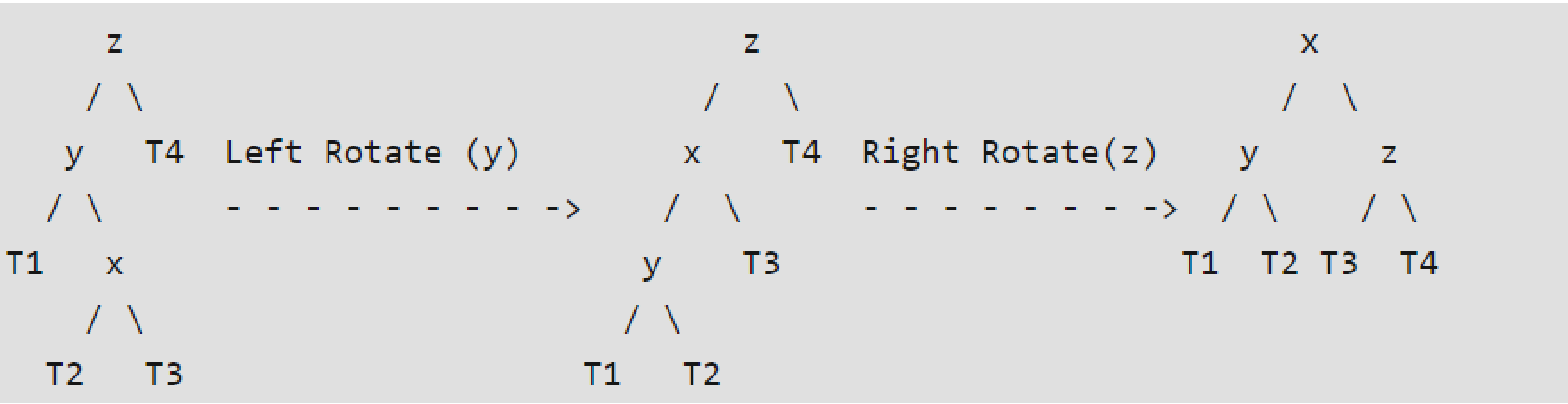
Right Left Case

Left Left Case



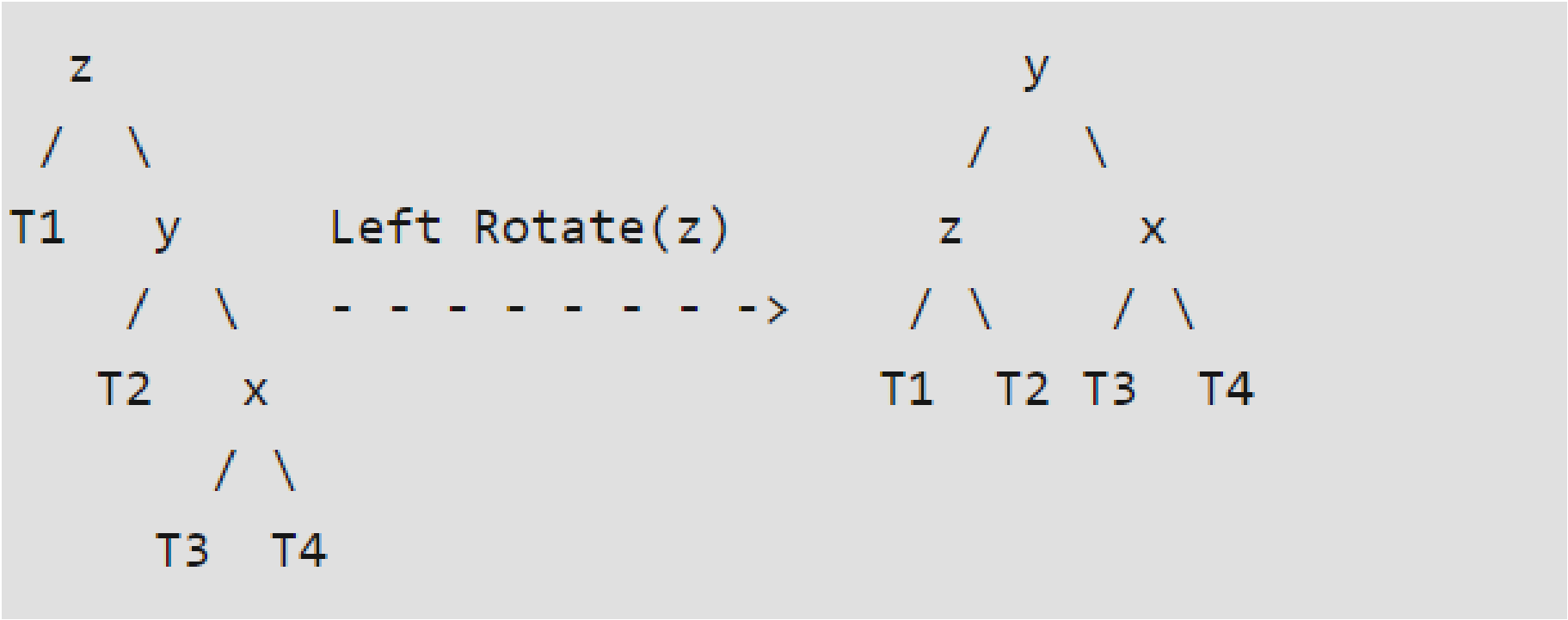
Unlike insertion, in **deletion**, after we perform a rotation at z, we may have to perform a rotation at **ancestors of z**. Thus, we must continue to **trace the path until we reach the root**.

Left Right Case



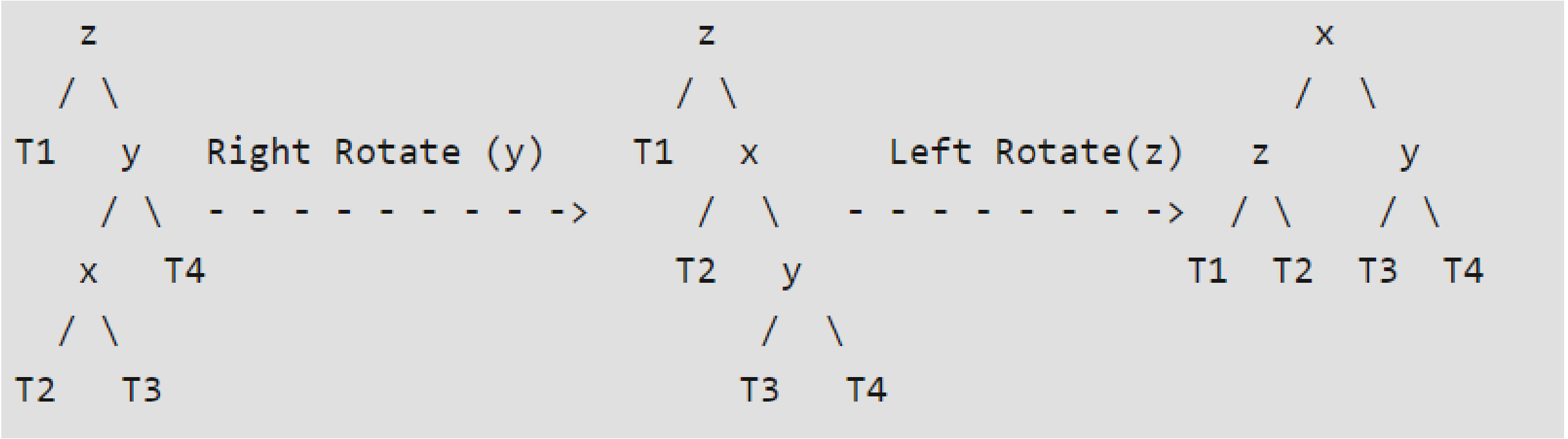
Unlike insertion, in **deletion**, after we perform a rotation at z, we may have to perform a rotation at **ancestors of z**. Thus, we must continue to **trace the path until we reach the root**.

Right Right Case



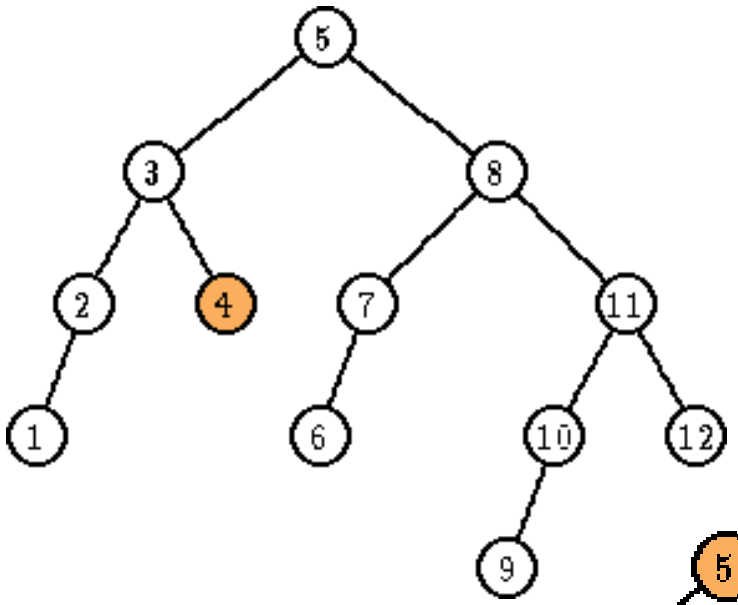
Unlike insertion, in **deletion**, after we perform a rotation at z, we may have to perform a rotation at **ancestors of z**. Thus, we must continue to **trace the path until we reach the root**.

Right Left Case

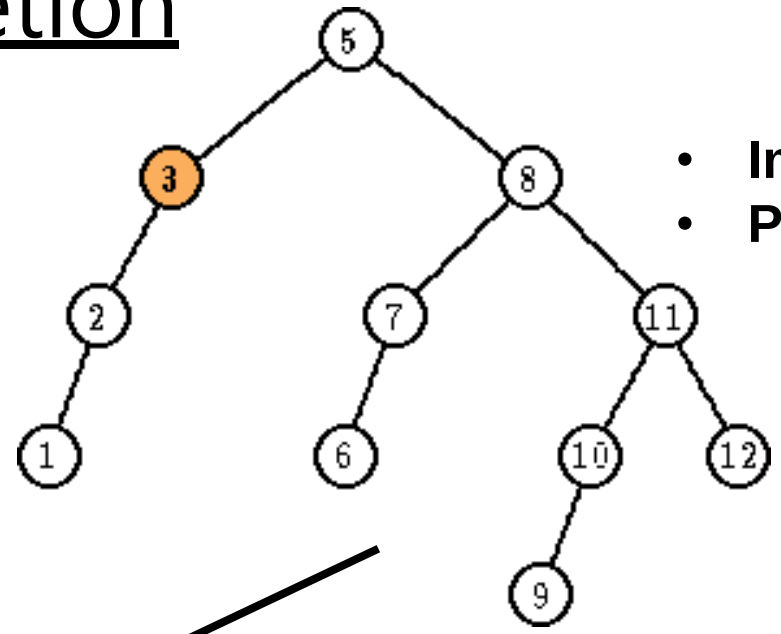


Unlike insertion, in **deletion**, after we perform a rotation at z, we may have to perform a rotation at **ancestors of z**. Thus, we must continue to **trace the path until we reach the root**.

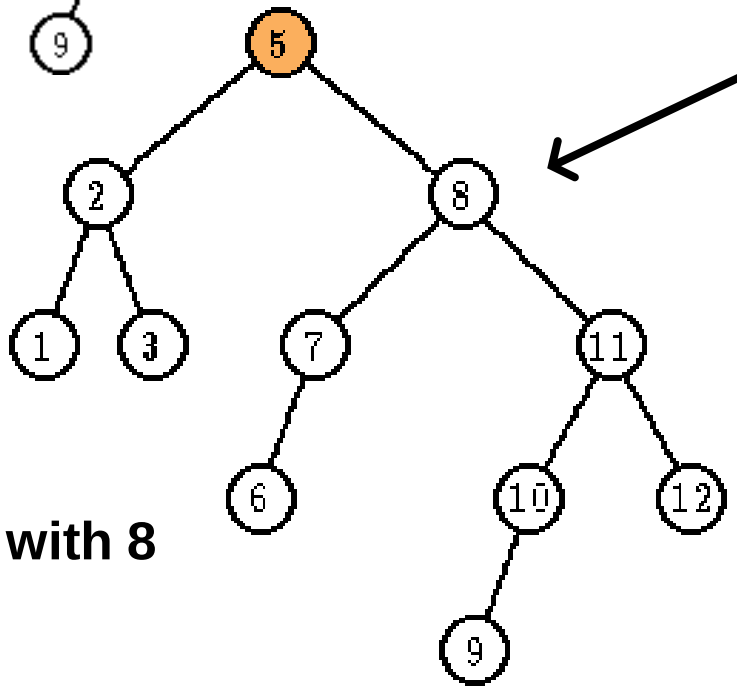
Deletion



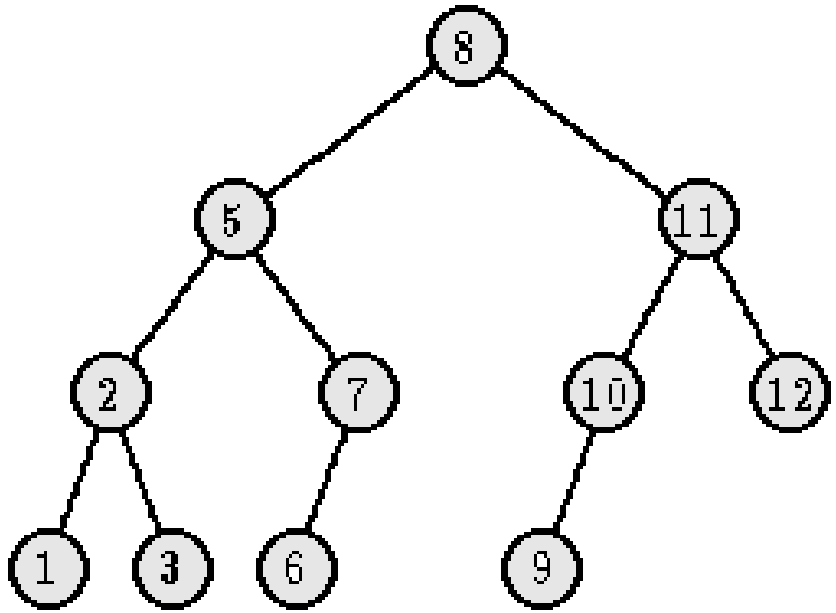
Delete 4



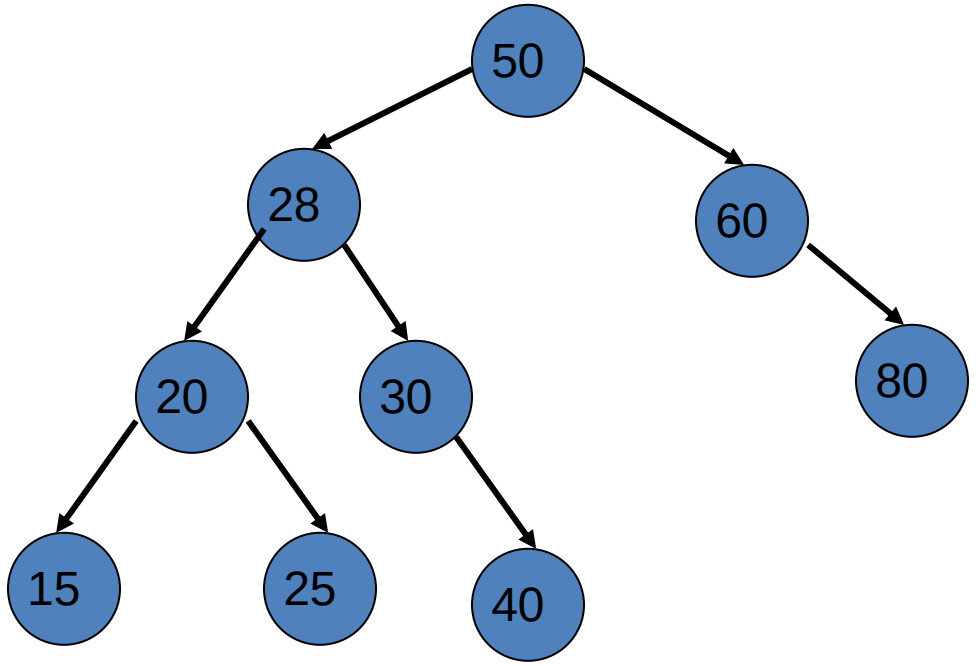
- Imbalance at 3
- Perform rotation with 2



- Imbalance at 5
- Perform rotation with 8

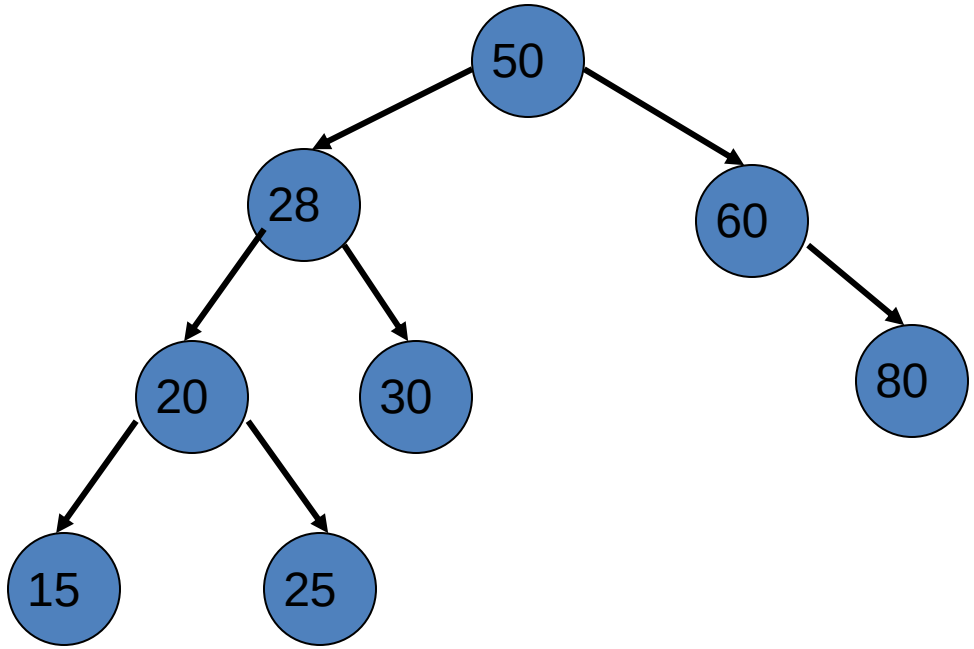


Delete 40

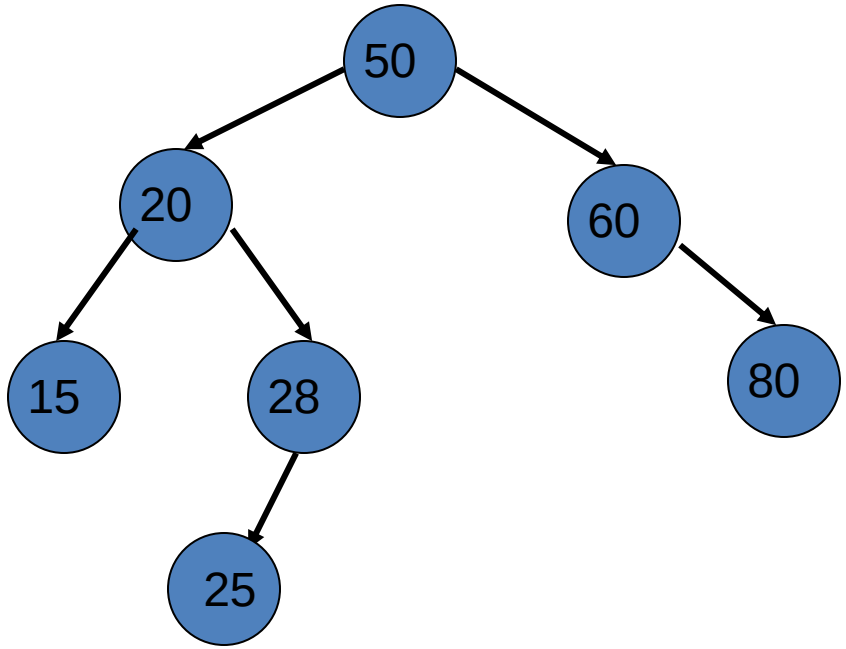


Still AVL

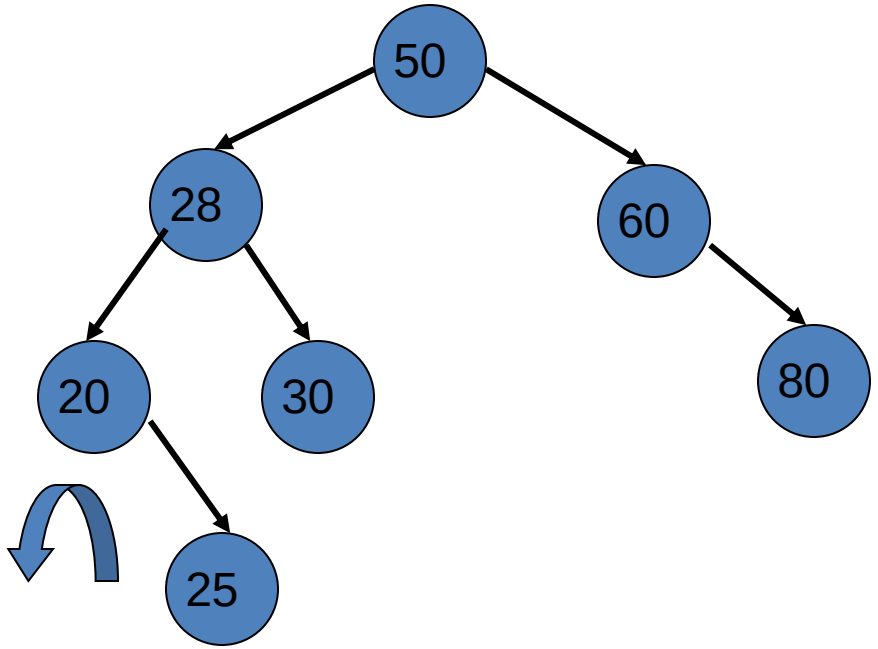
Delete 30



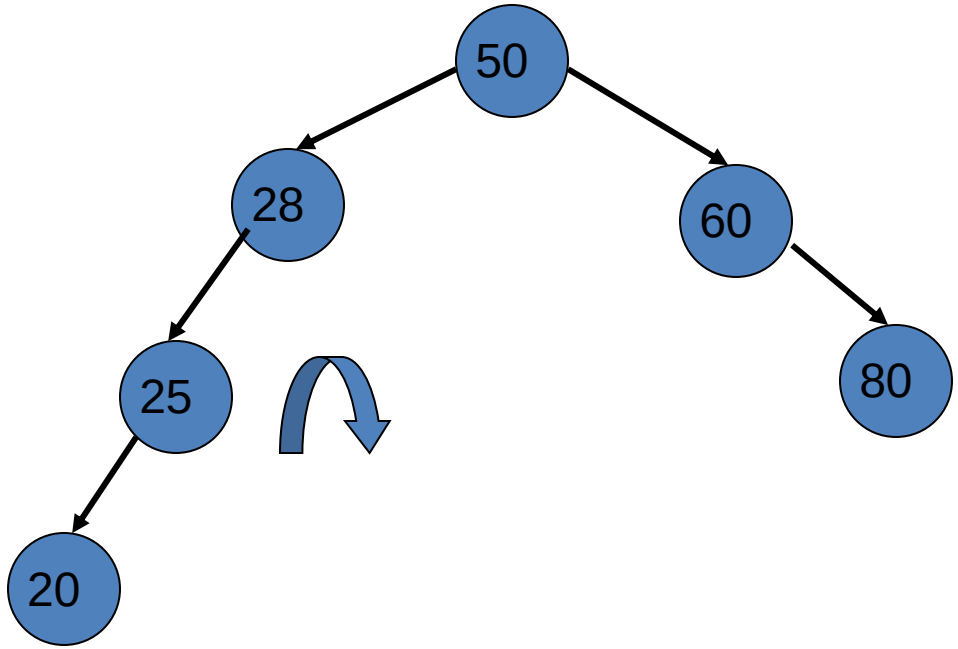
Delete 30



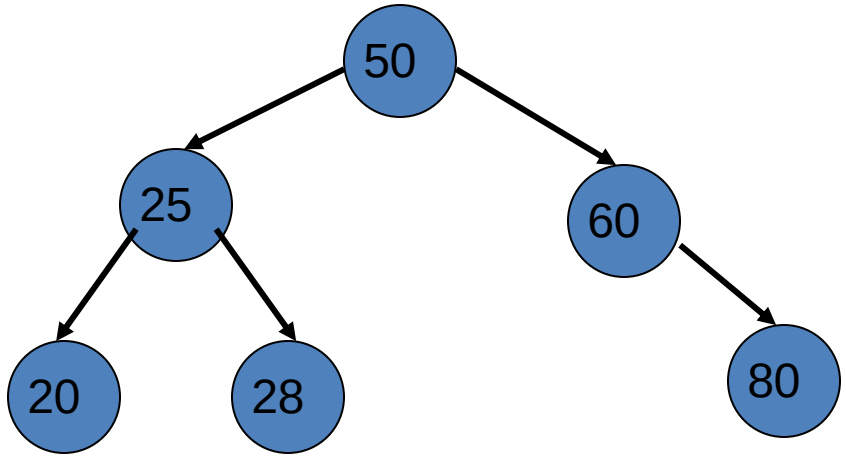
Another example Delete 30



Another example Delete 30

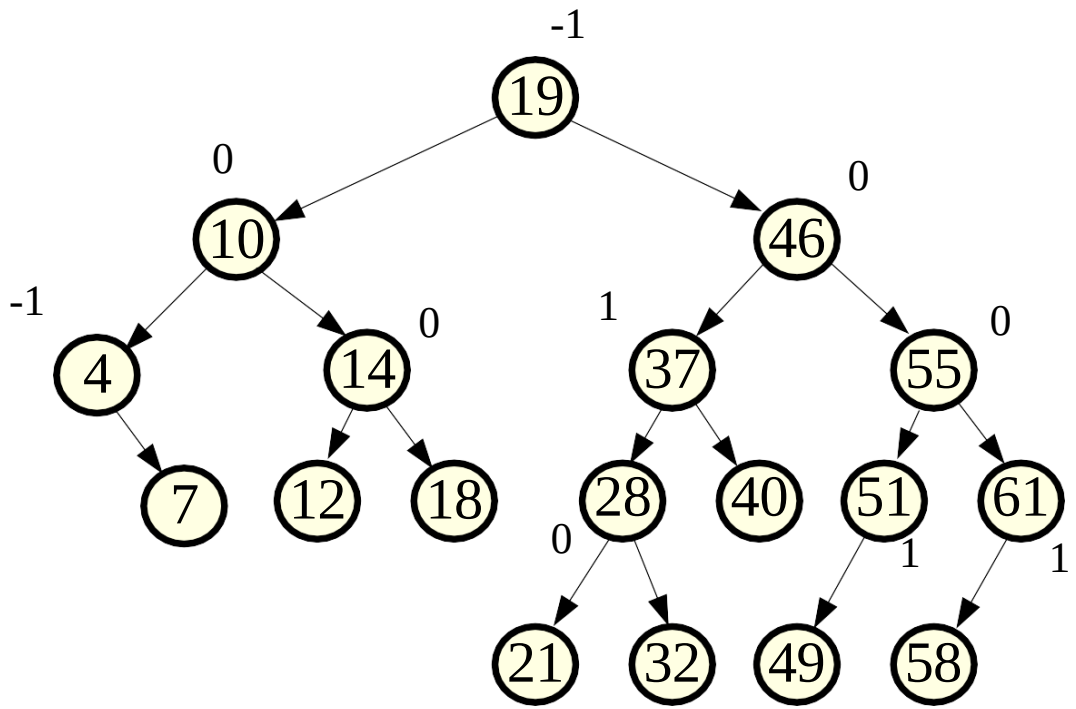


Another example Delete 30

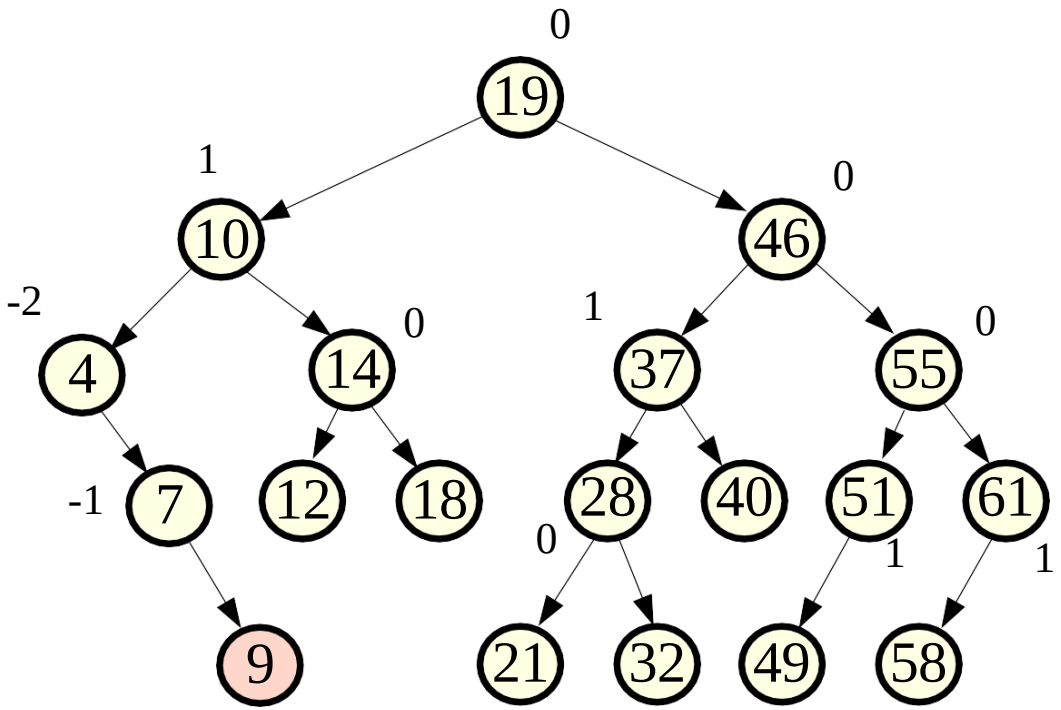


Additional Examples

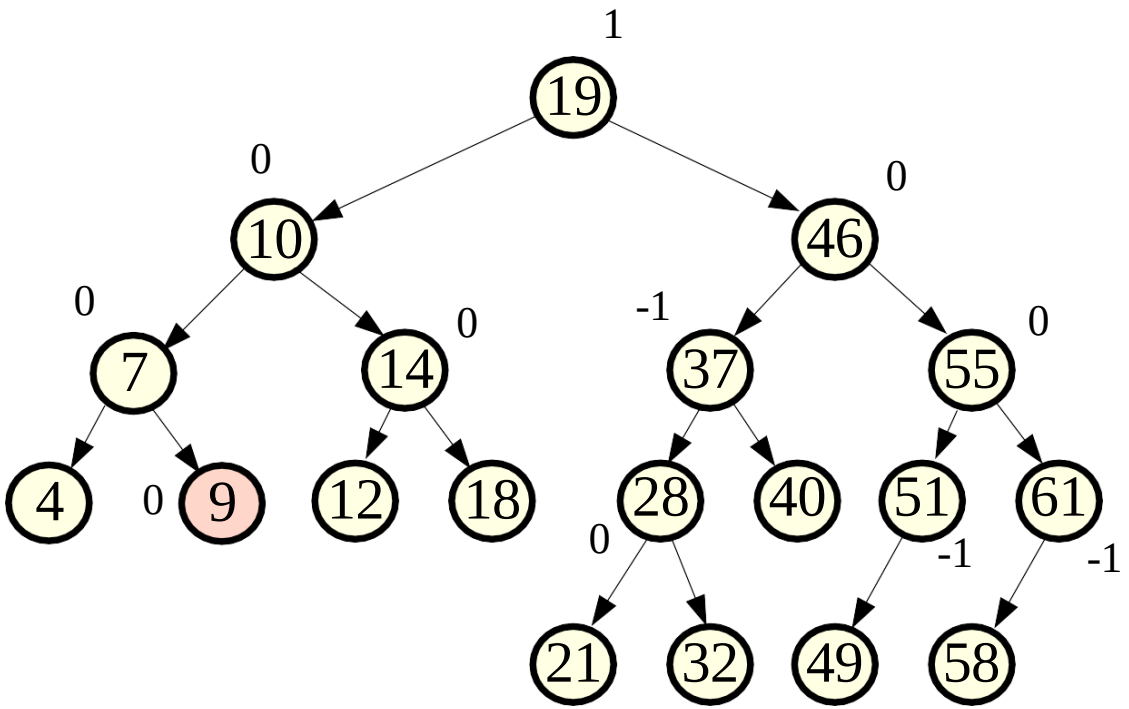
Example:



Example: Insert 9

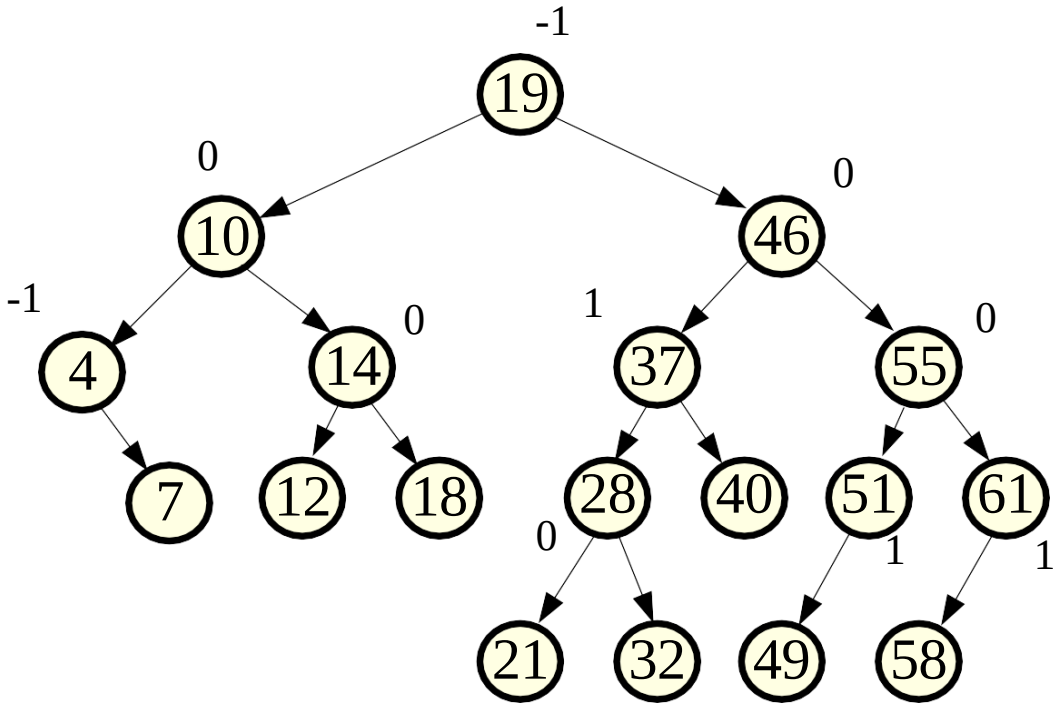


Example: Insert 9

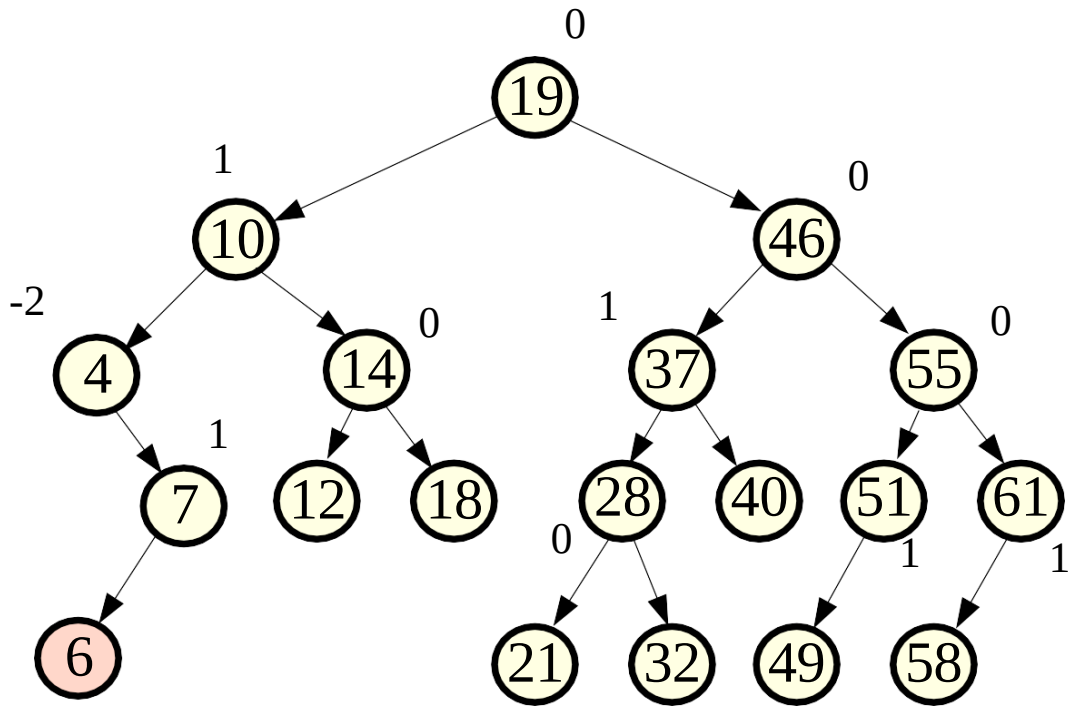


Rotation around 7

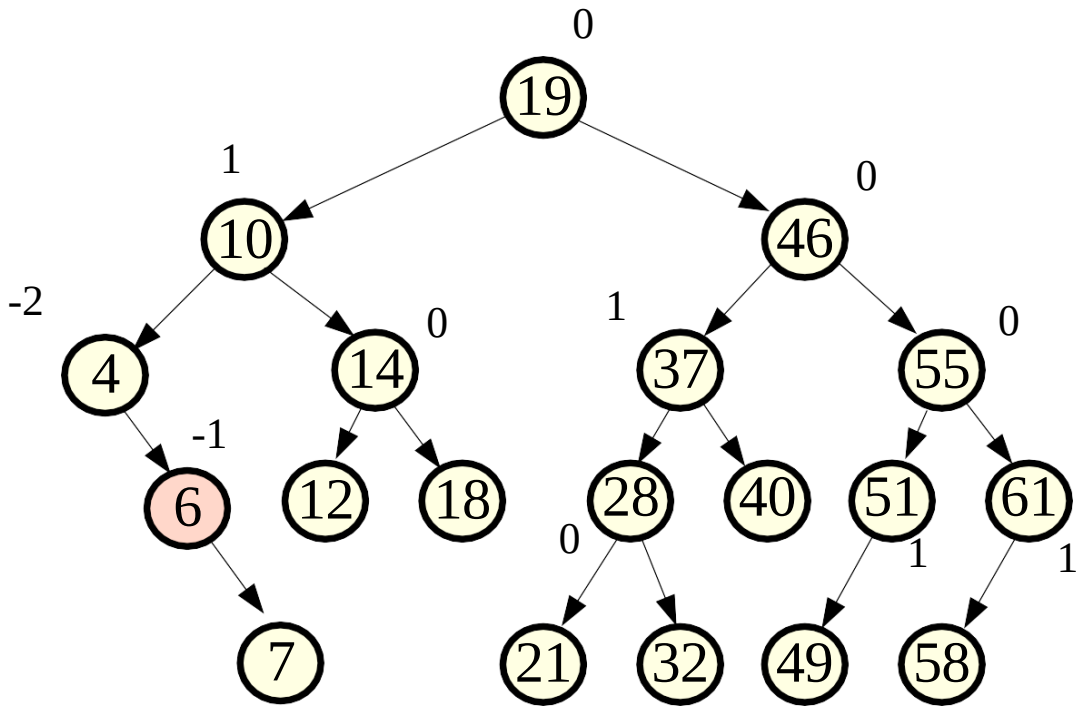
Example:



Example: Insert 6

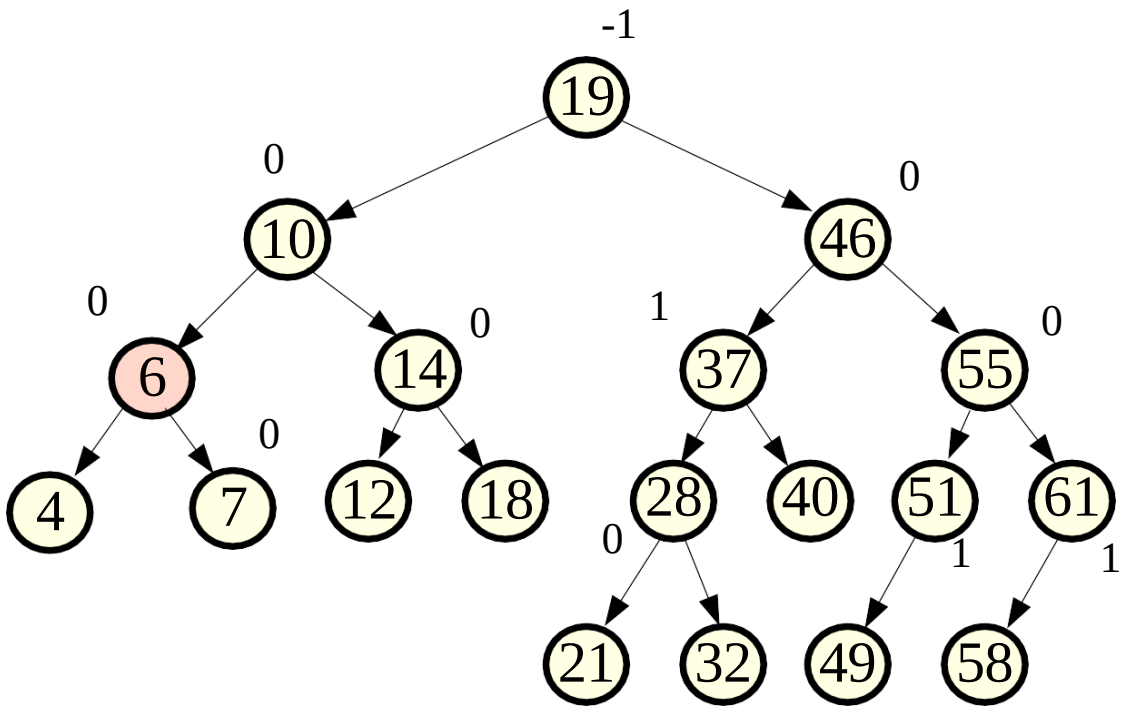


Example: Insert 6



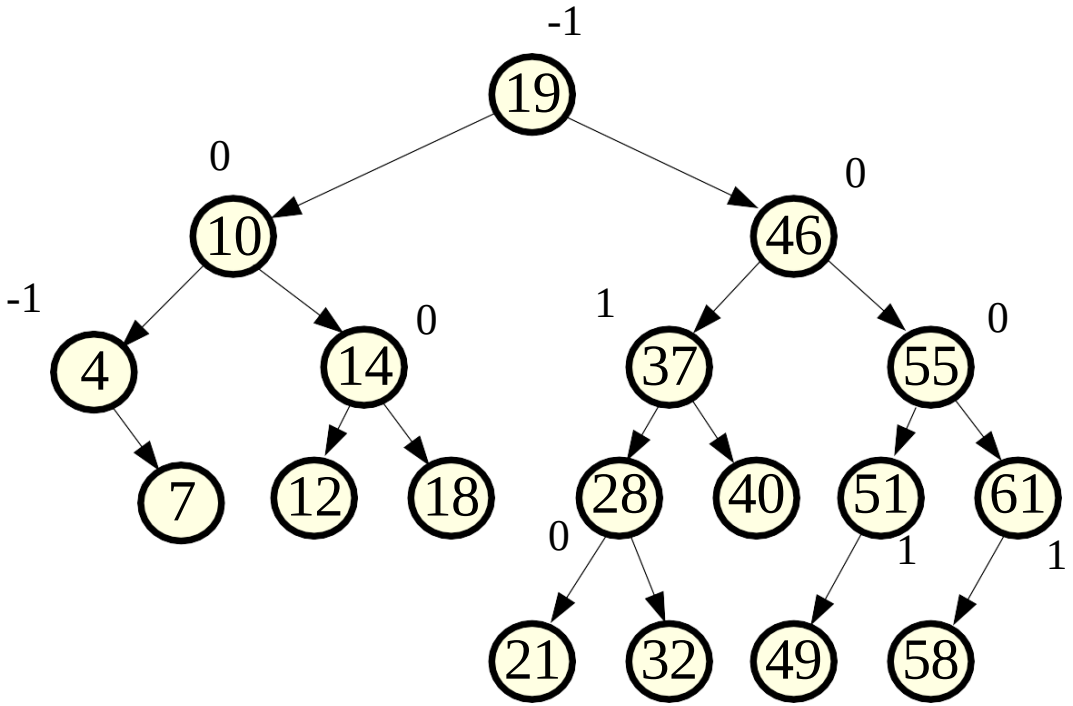
Double rotation

Example: Insert 6

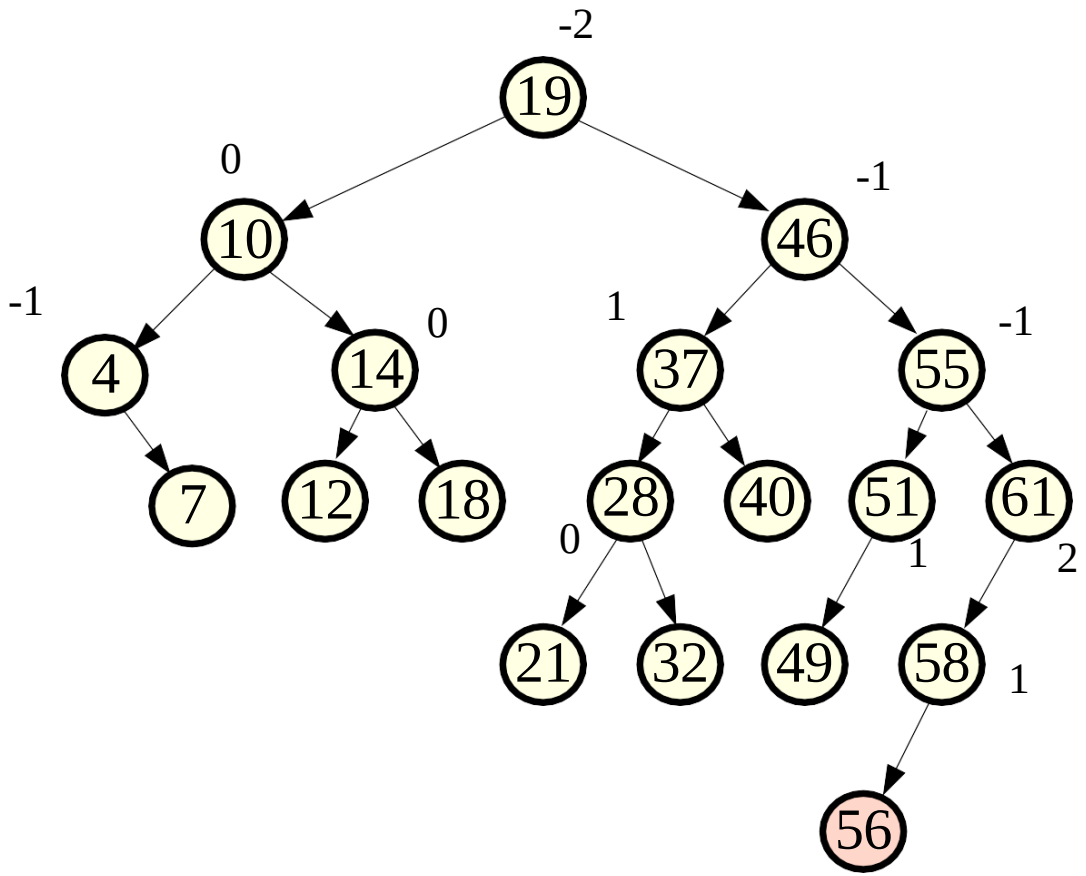


Double rotation

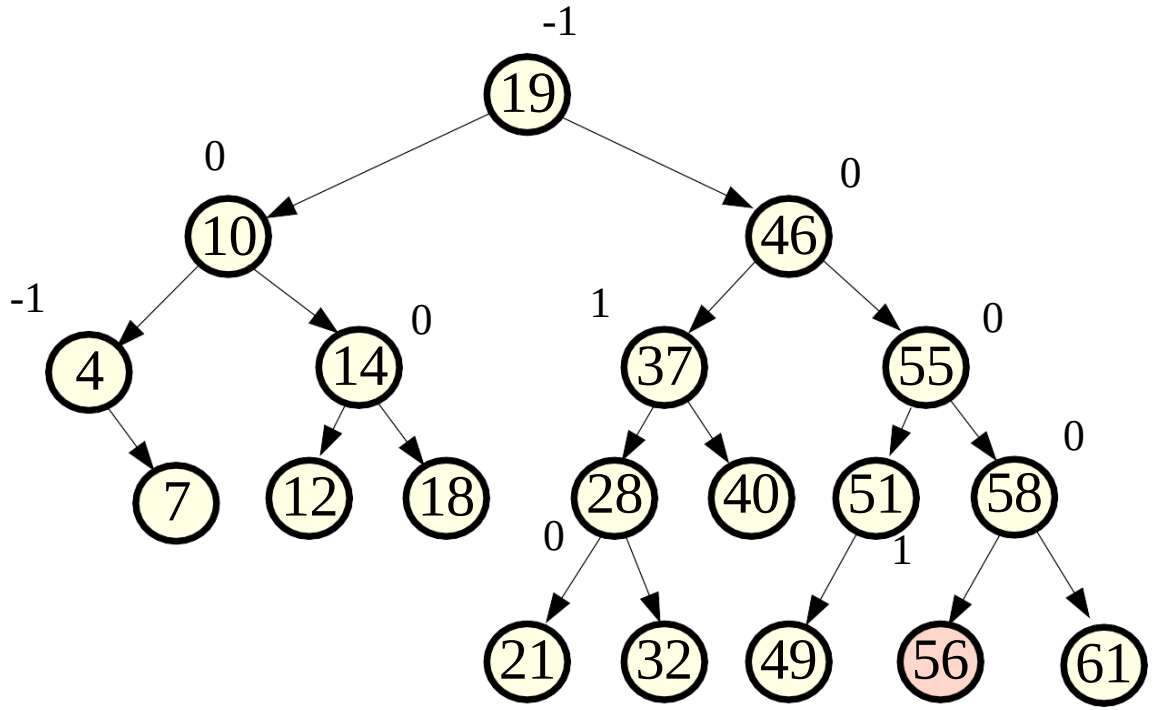
Example:



Example: Insert 56

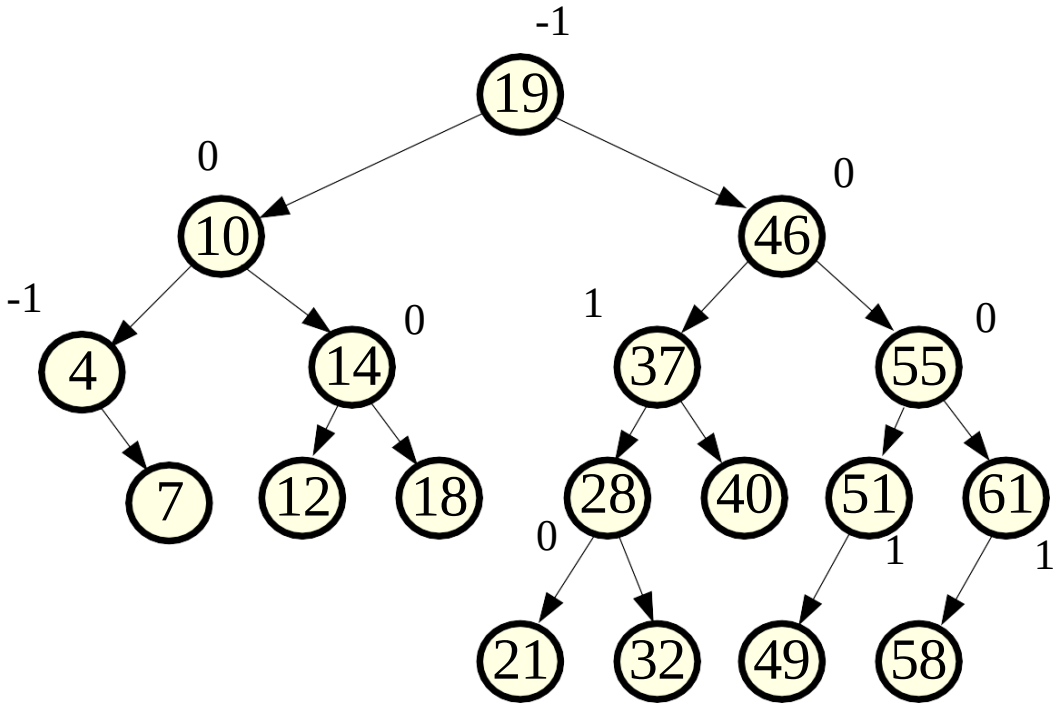


Example: Insert 56

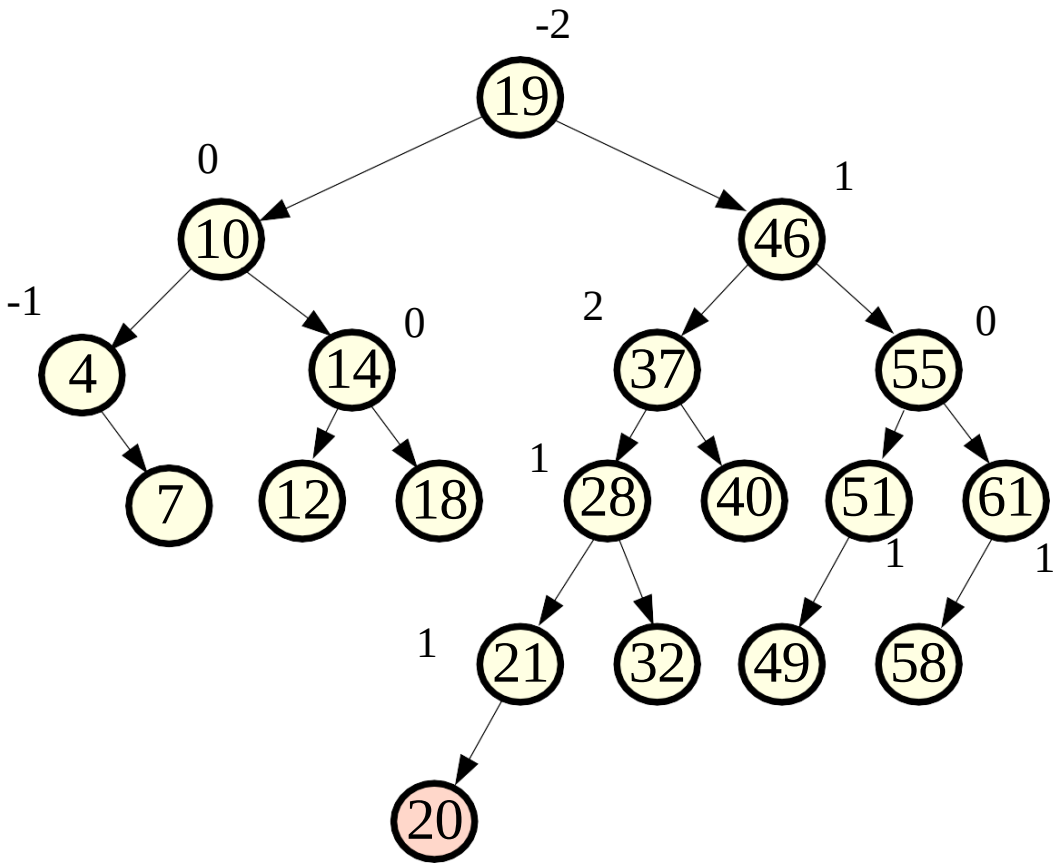


Single rotation around 58

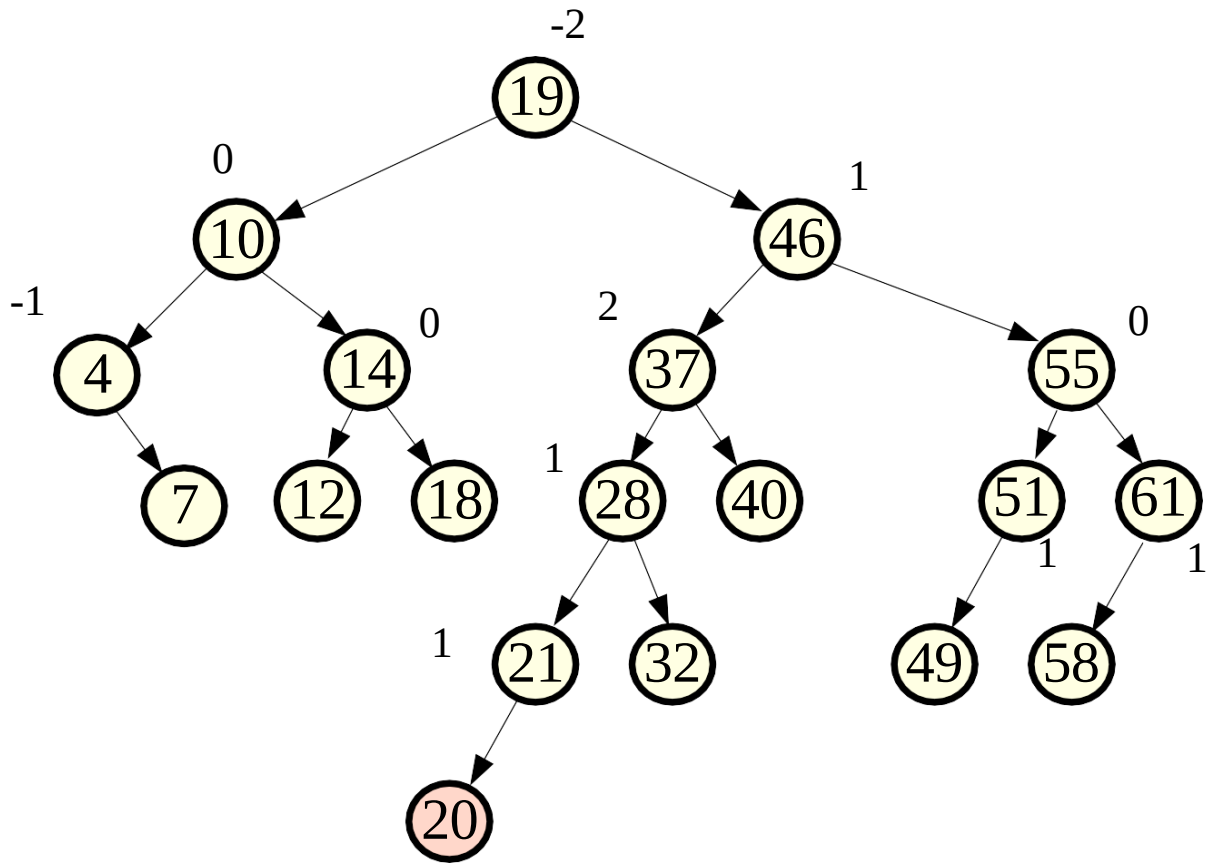
Example:



Example: Insert 20

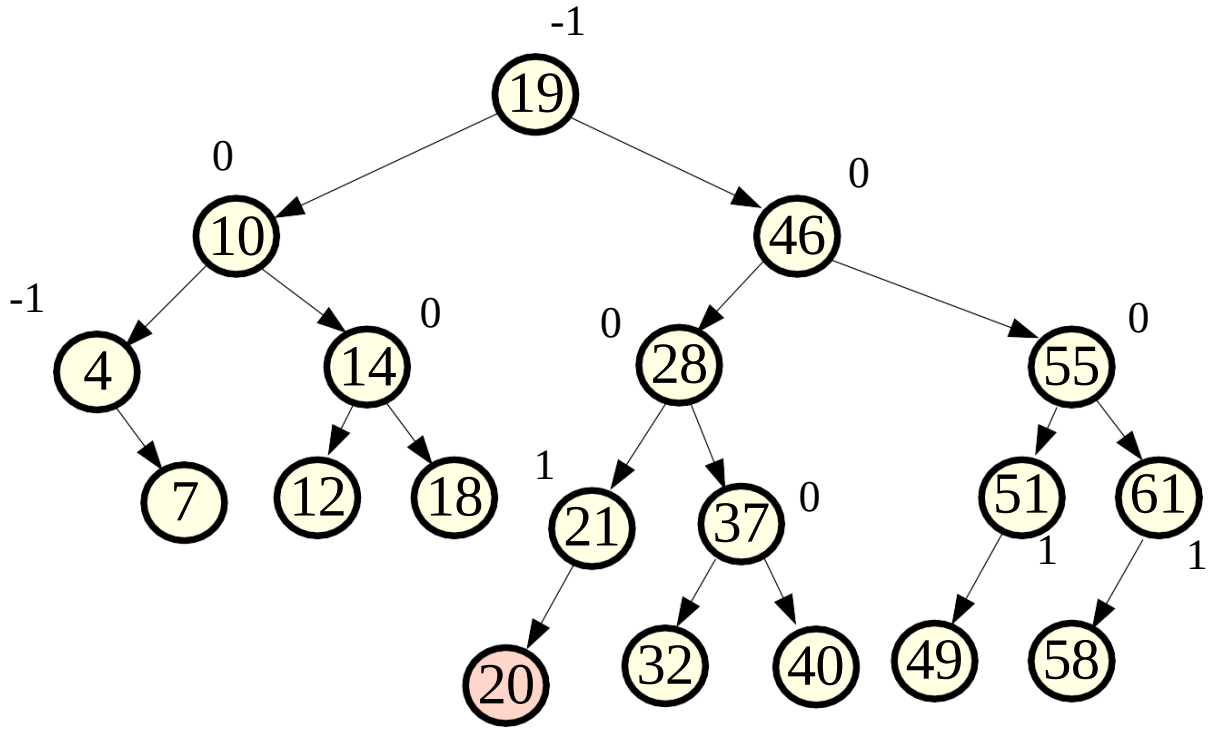


Example: Insert 20



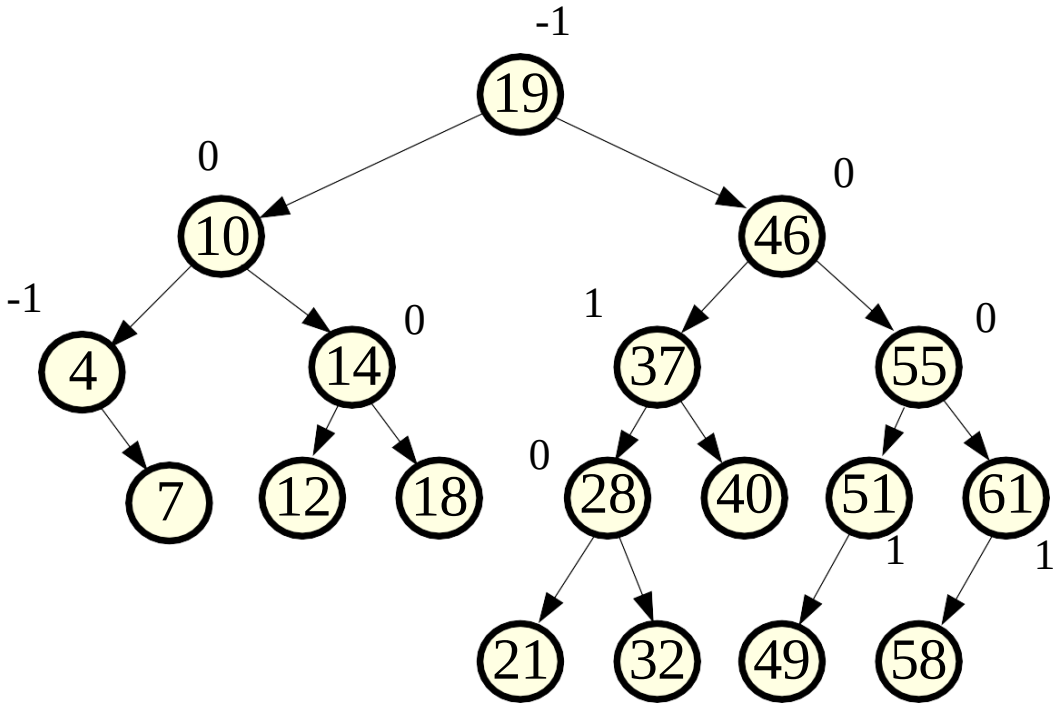
Rotate around 28

Example: Insert 20

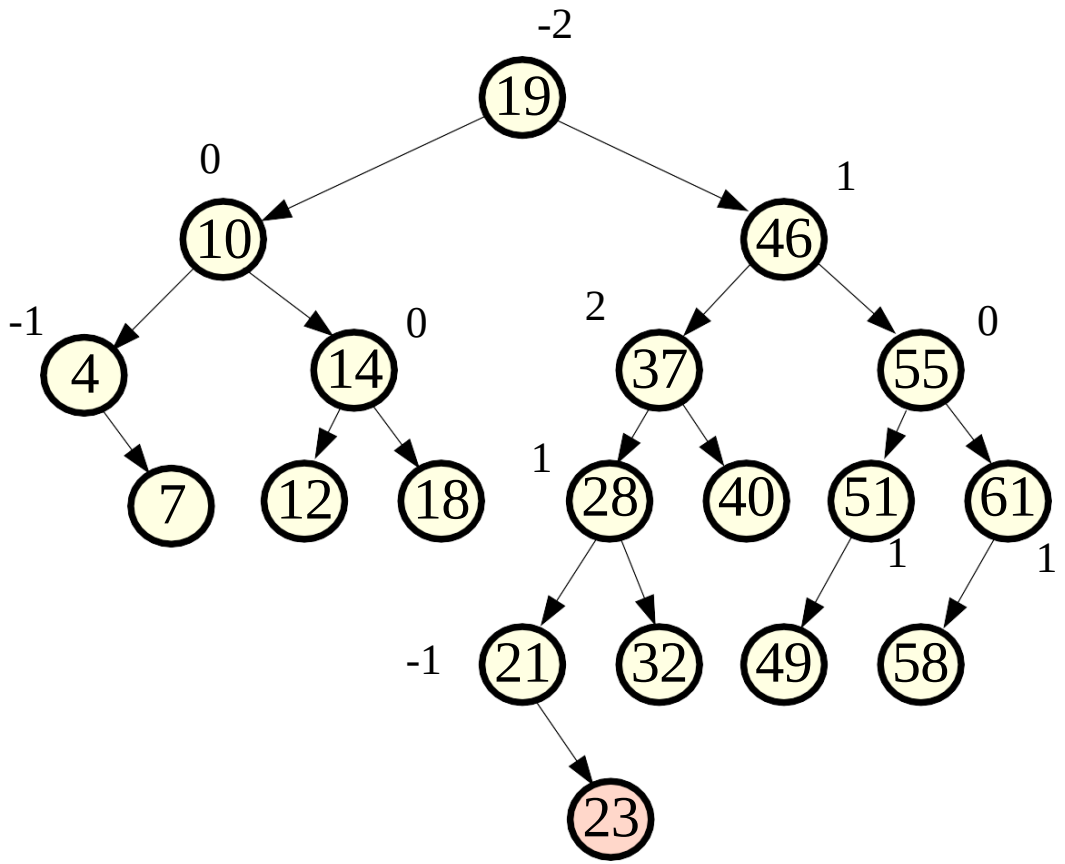


Rotate around 28

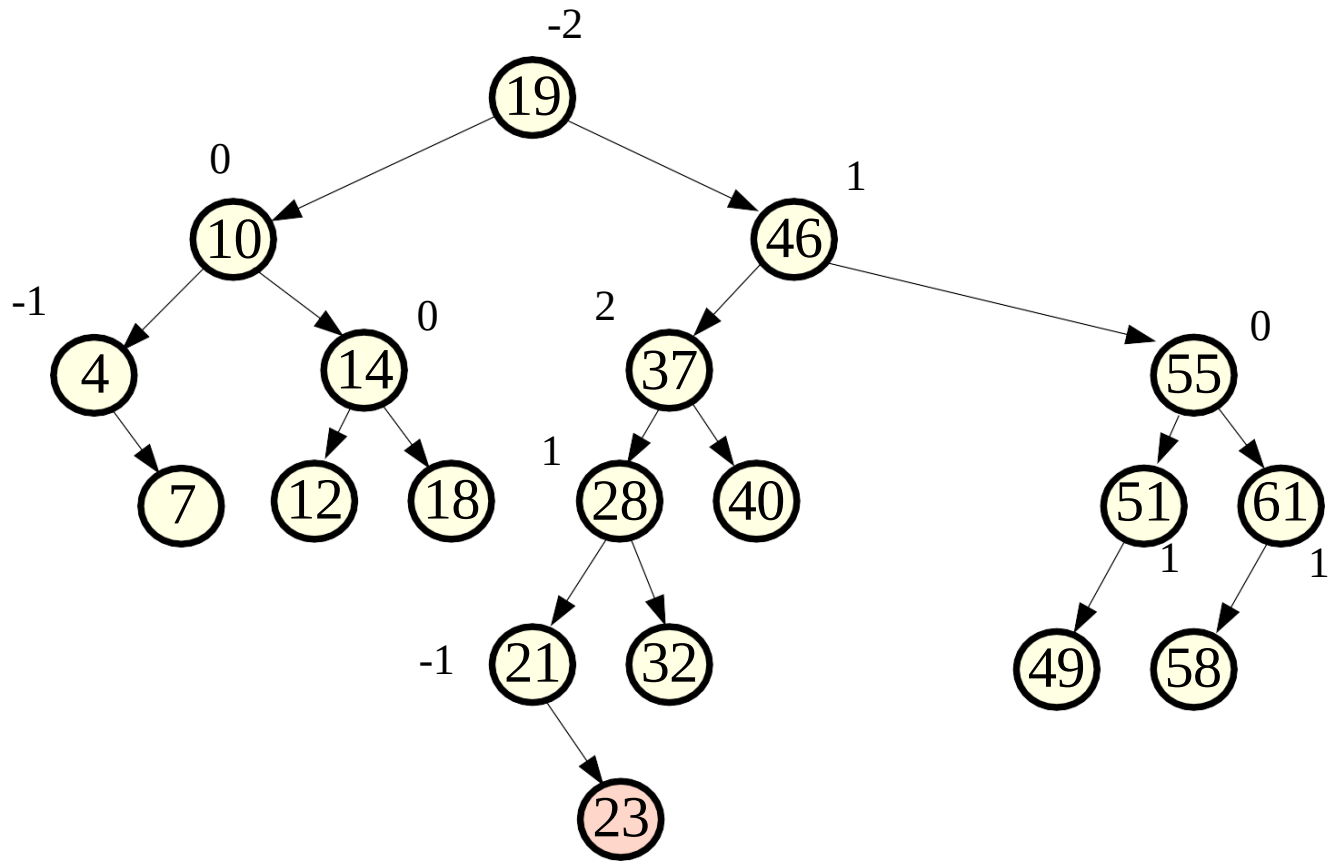
Example:



Example: Insert 23

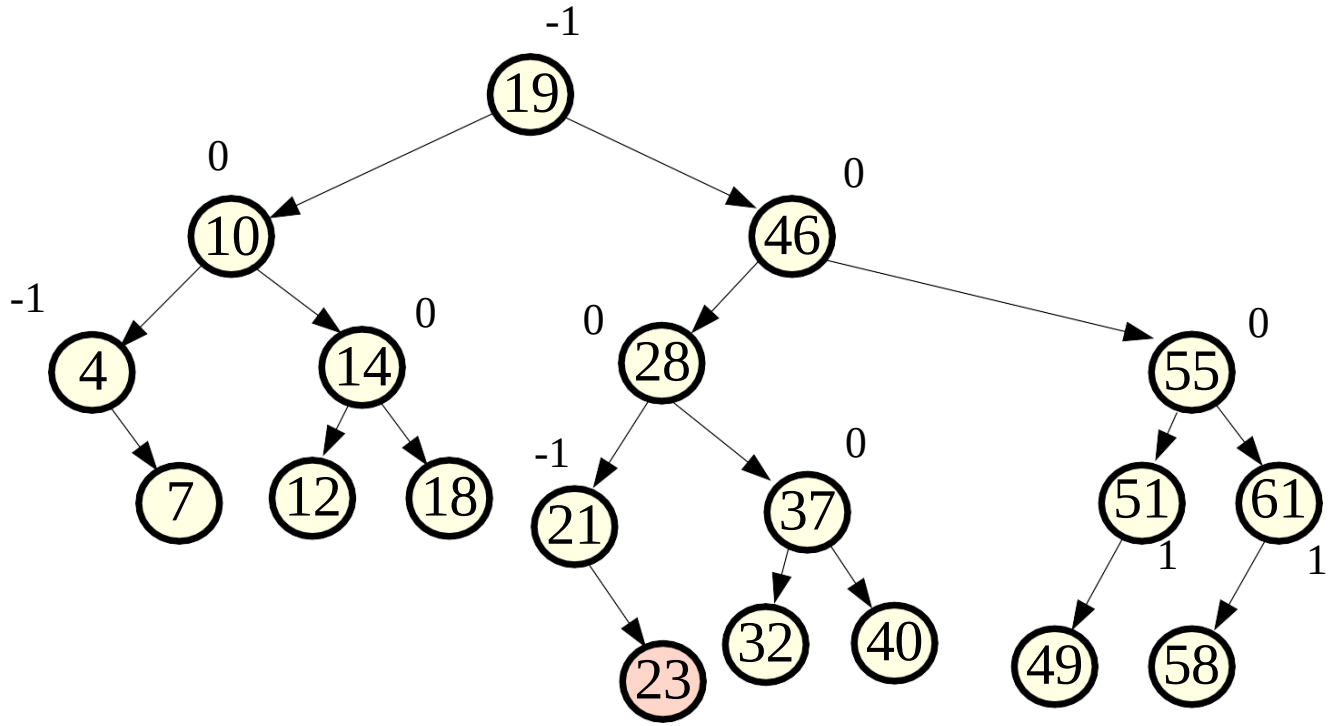


Example: Insert 23



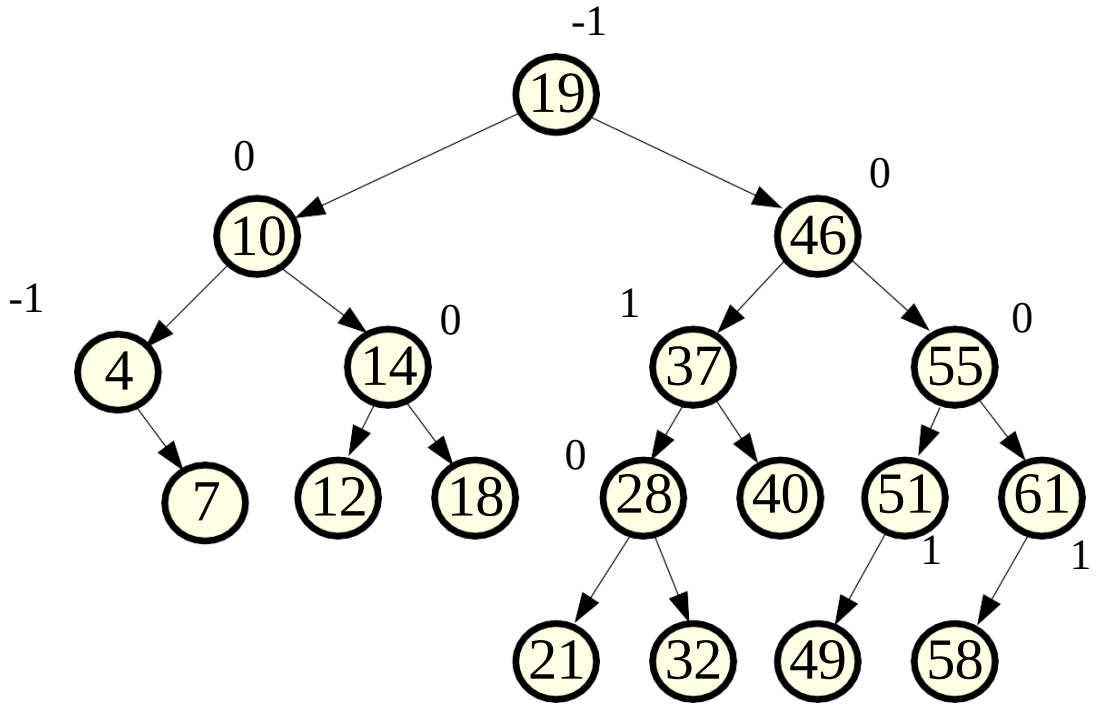
Rotation around 28

Example: Insert 23

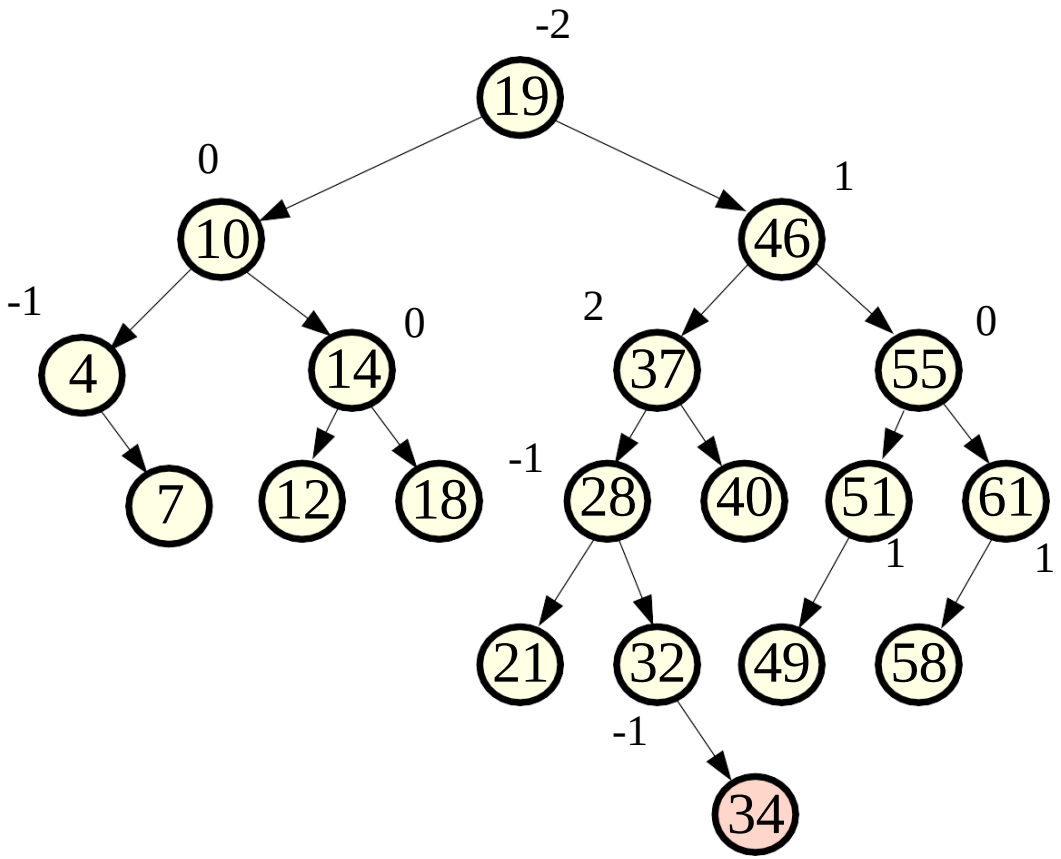


Rotation around 28

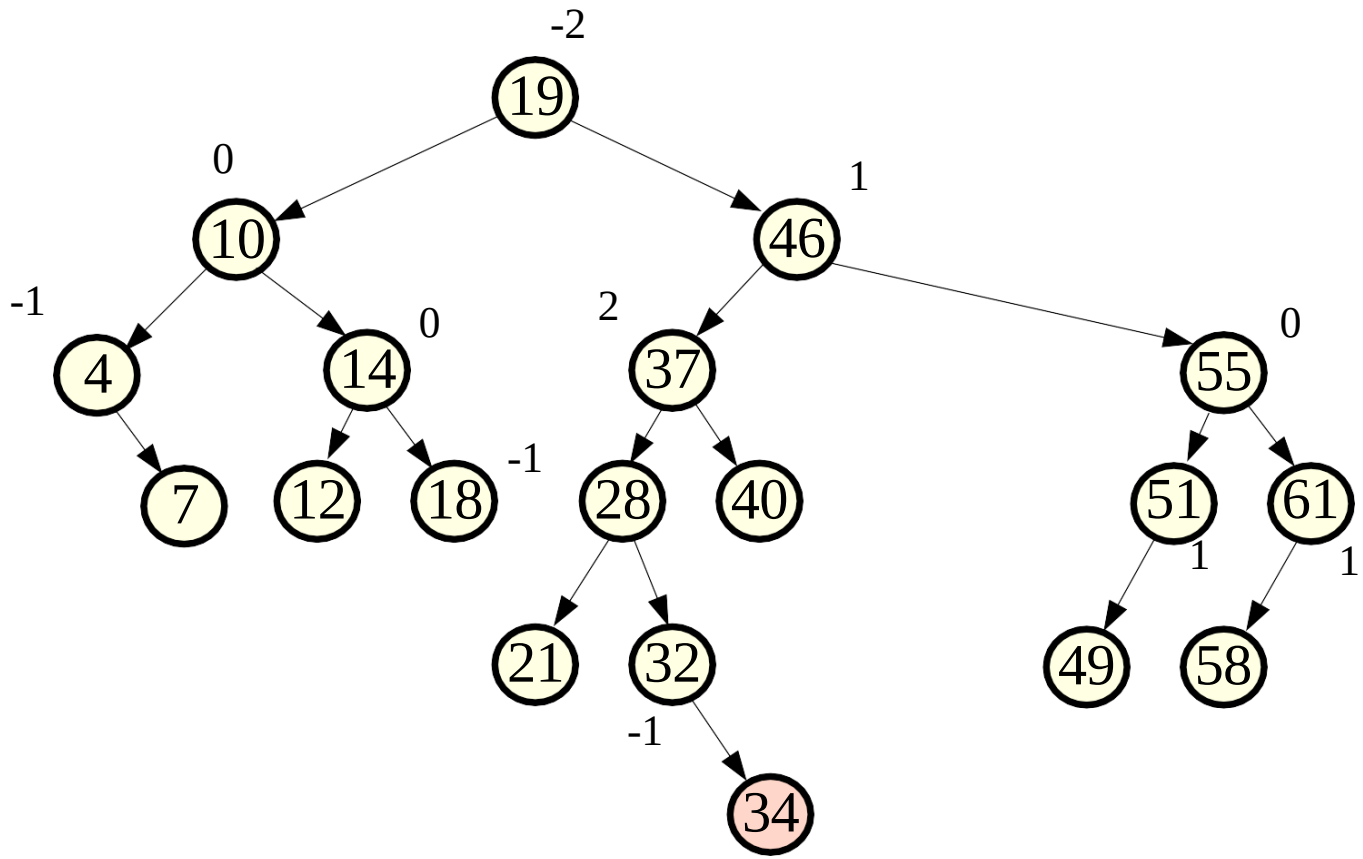
Example:



Example: Insert 34

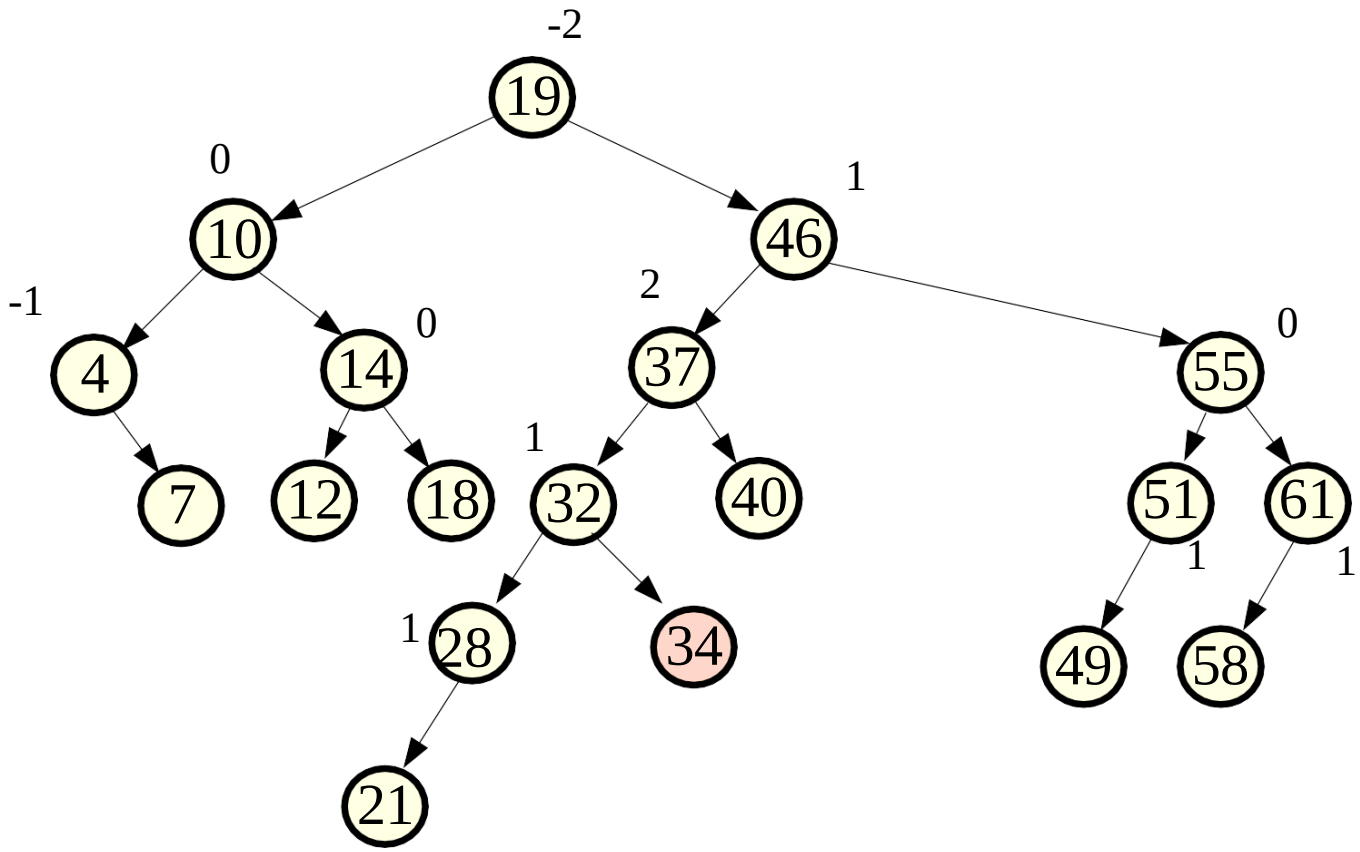


Example: Insert 34



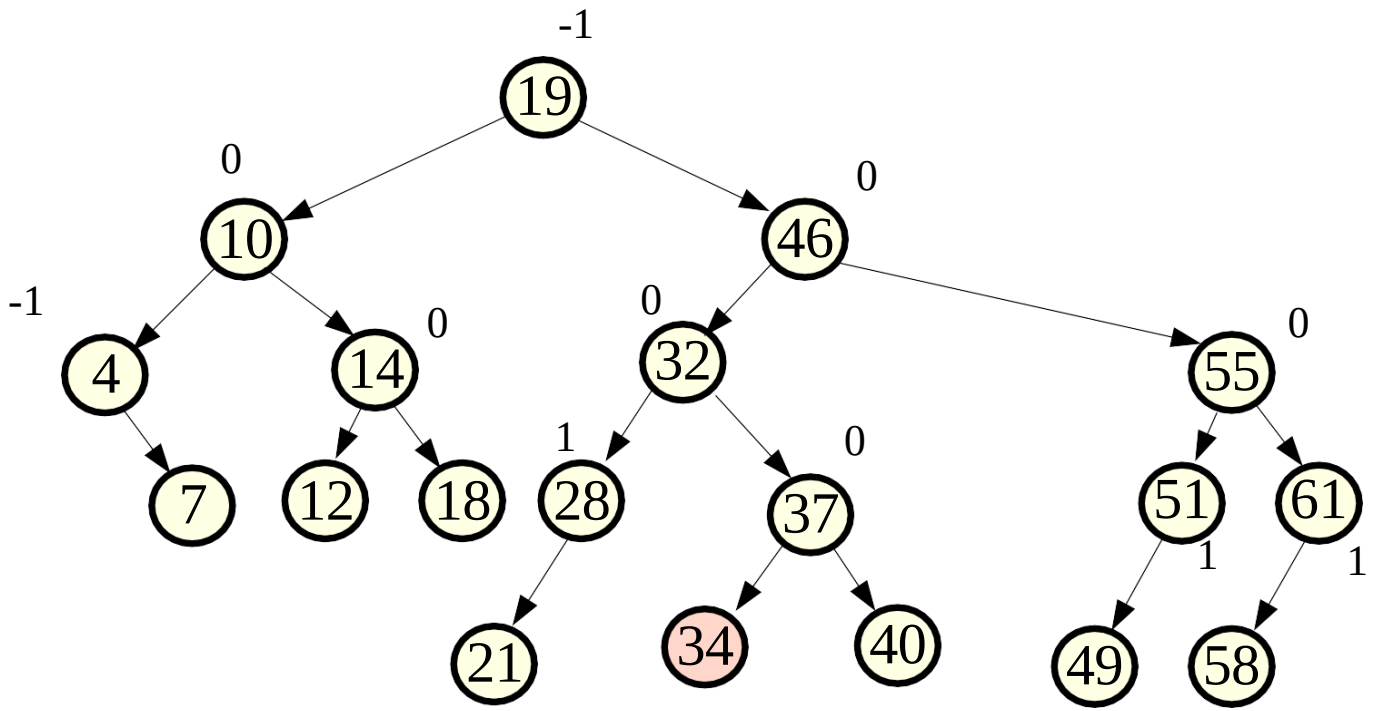
Double rotation around 32

Example: Insert 34



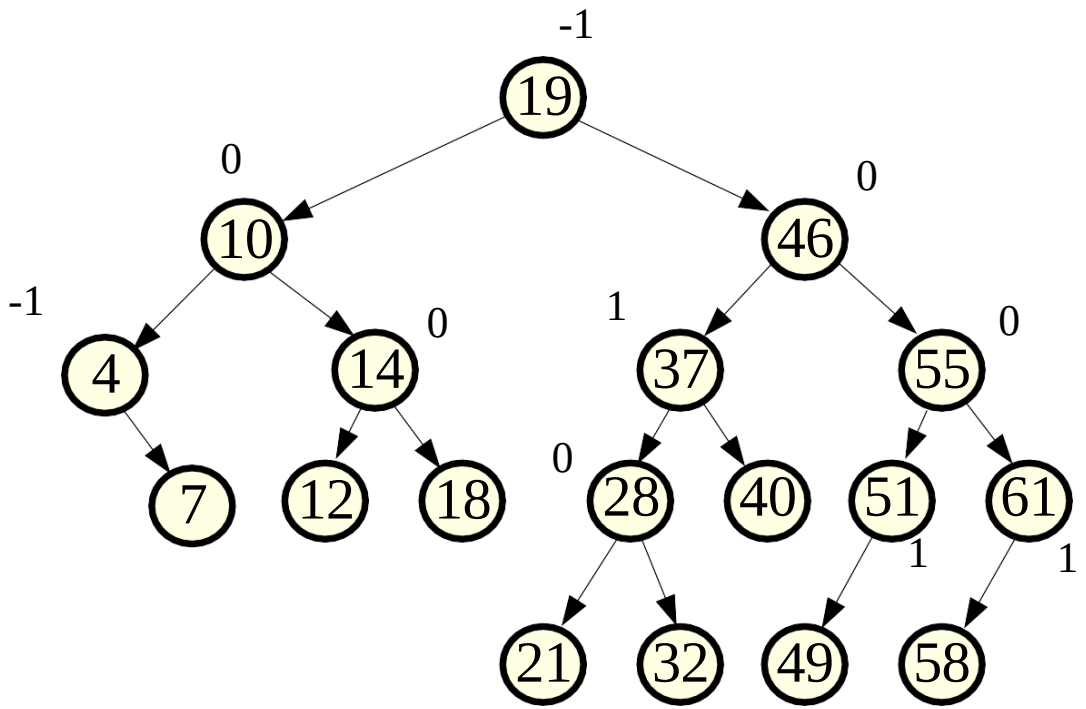
Double rotation around 32

Example: Insert 34

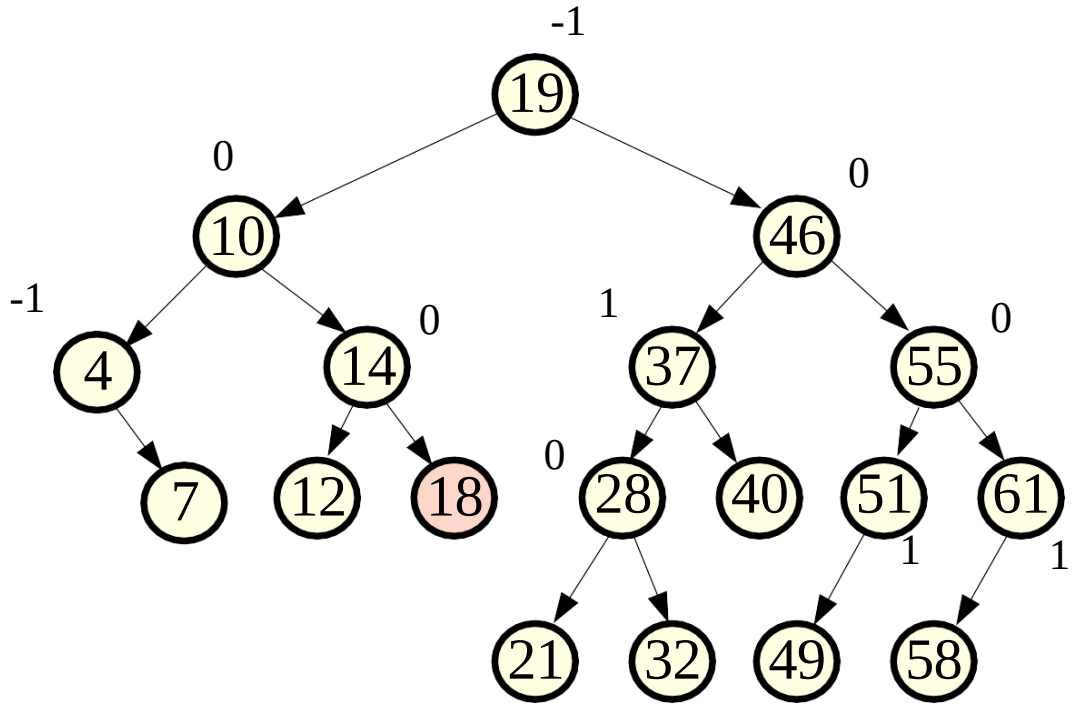


Double rotation around 32

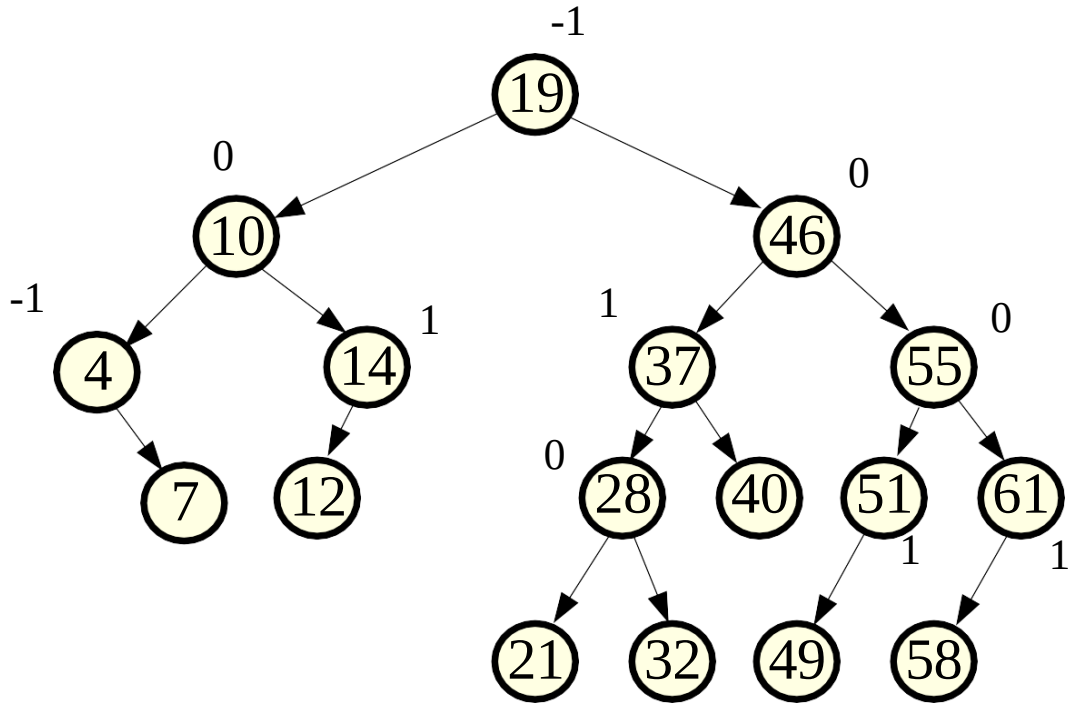
Example:



Example: Delete 18

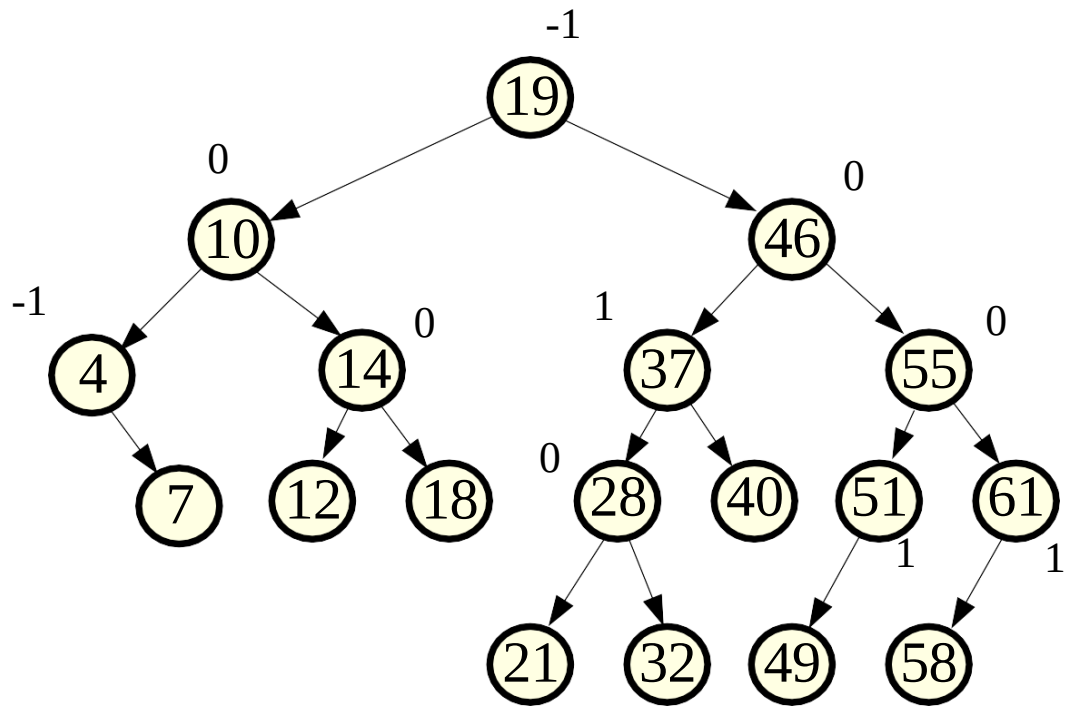


Example: Delete 18

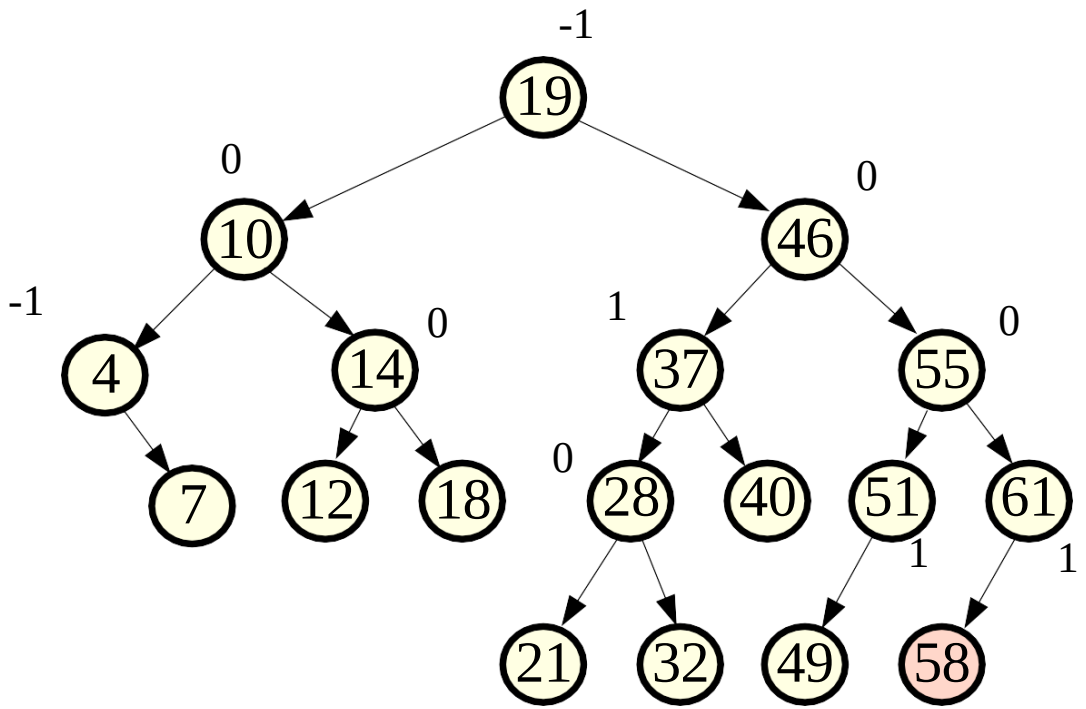


No change

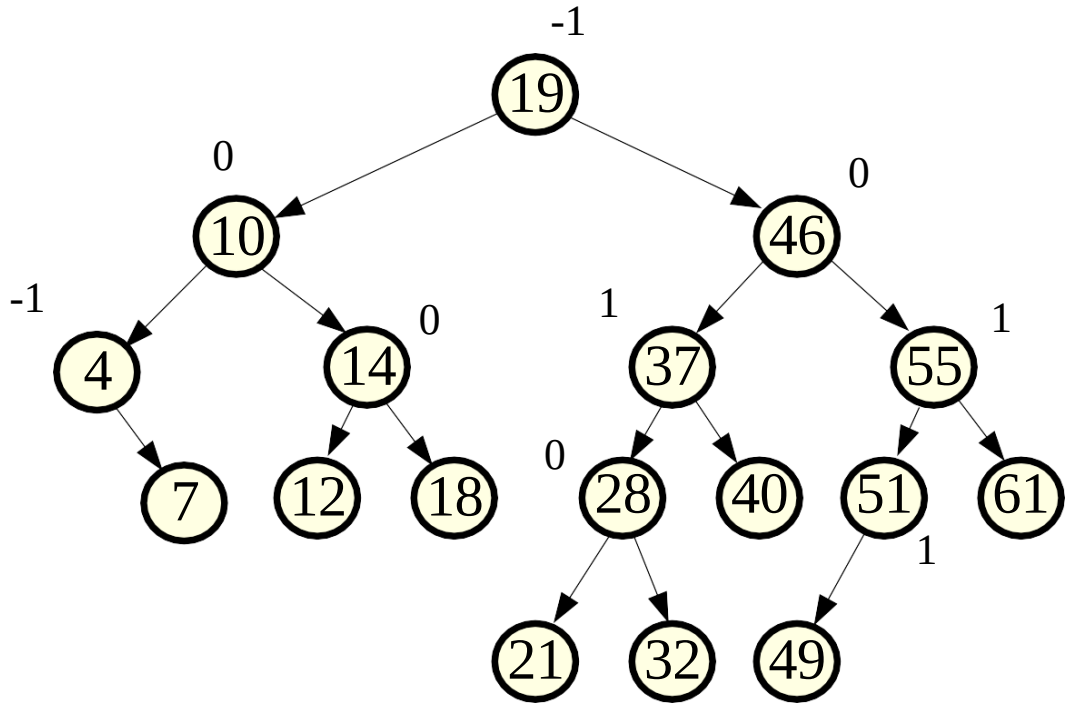
Example:



Example: Delete 58

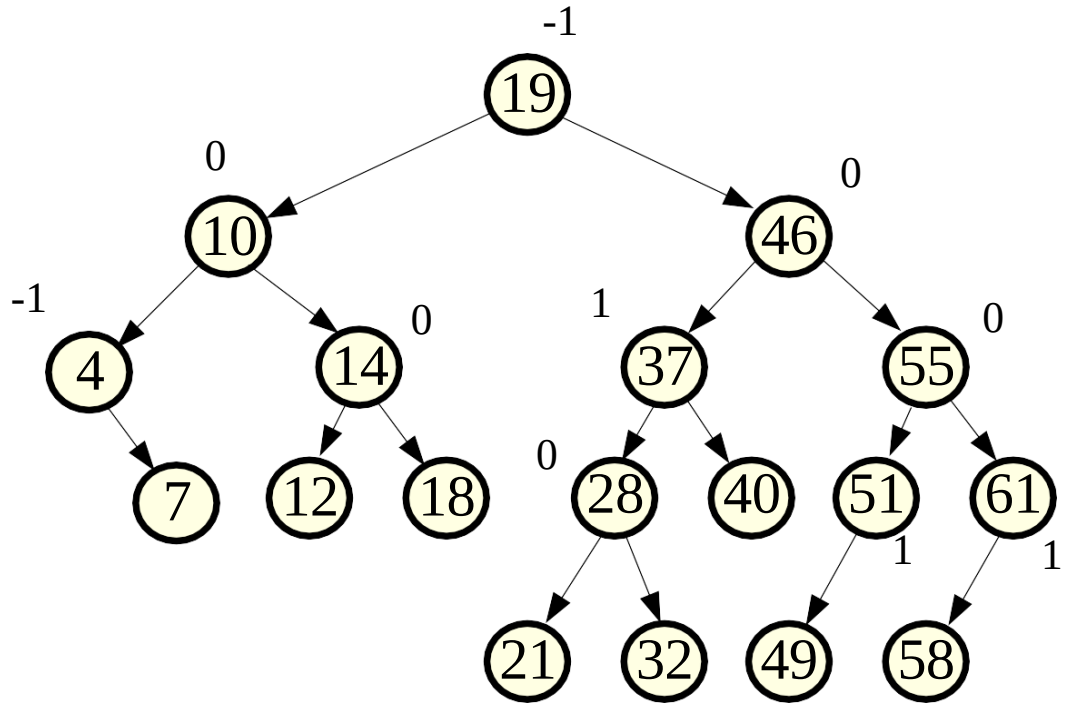


Example: Delete 58

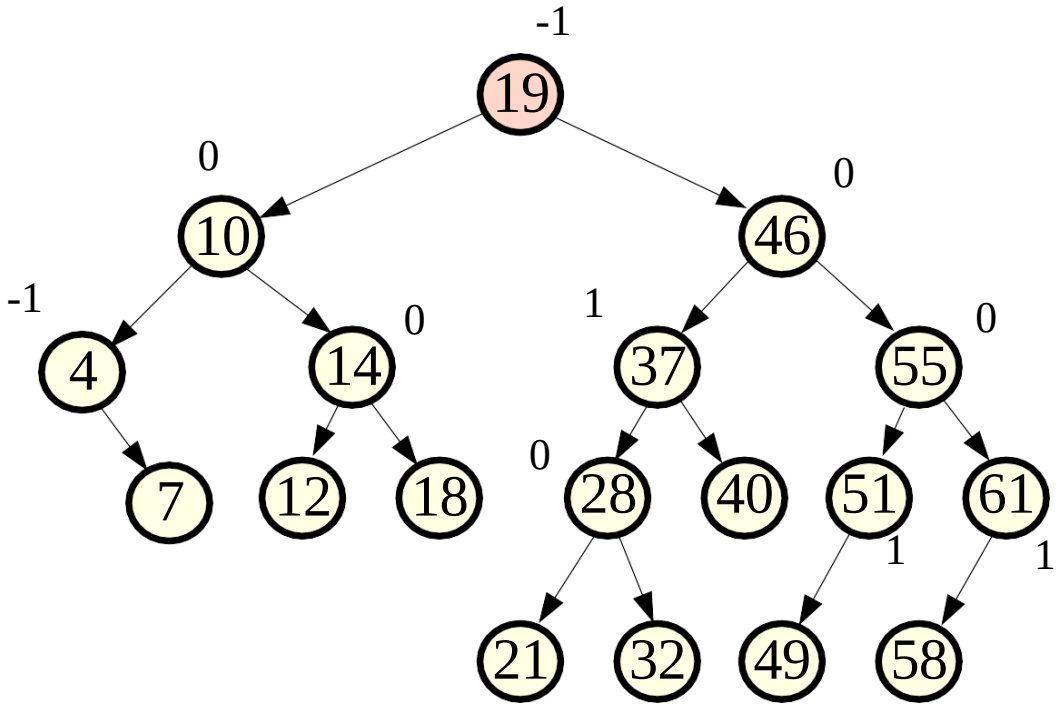


No change

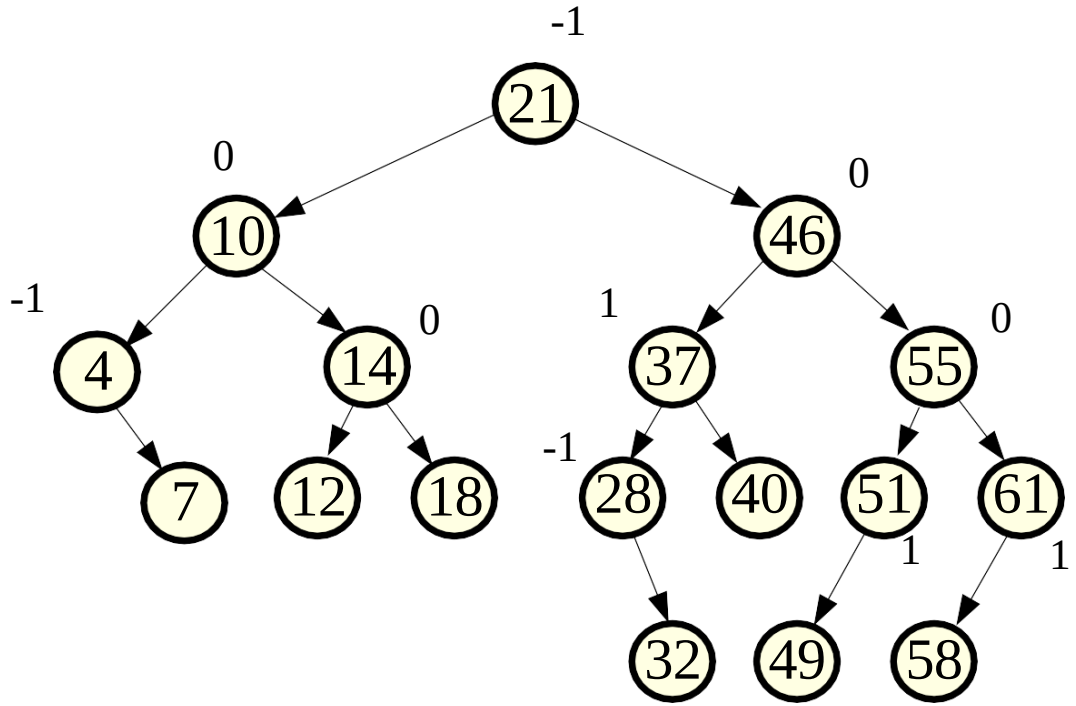
Example:



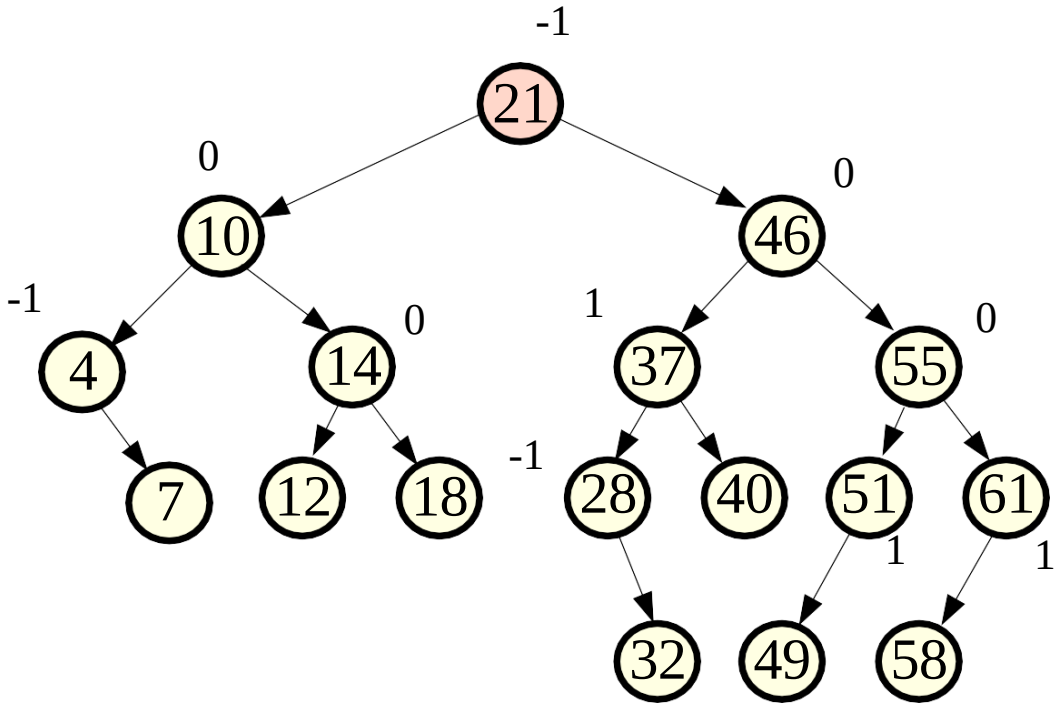
Example: Delete 19



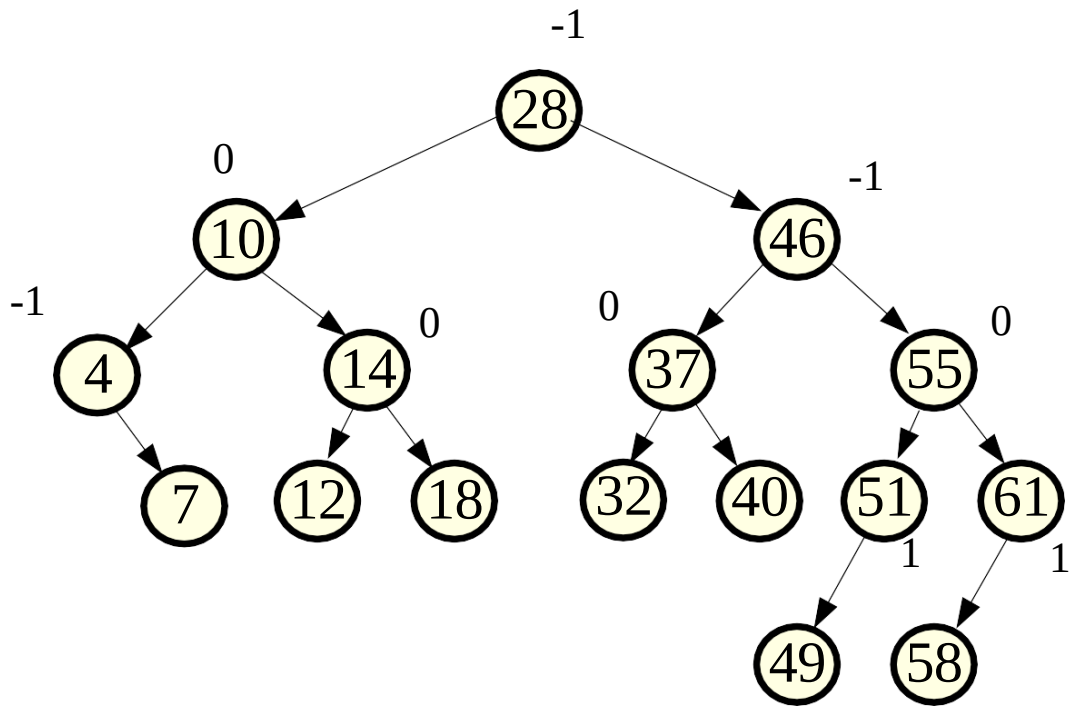
Example: Delete 19



Example: Delete 21

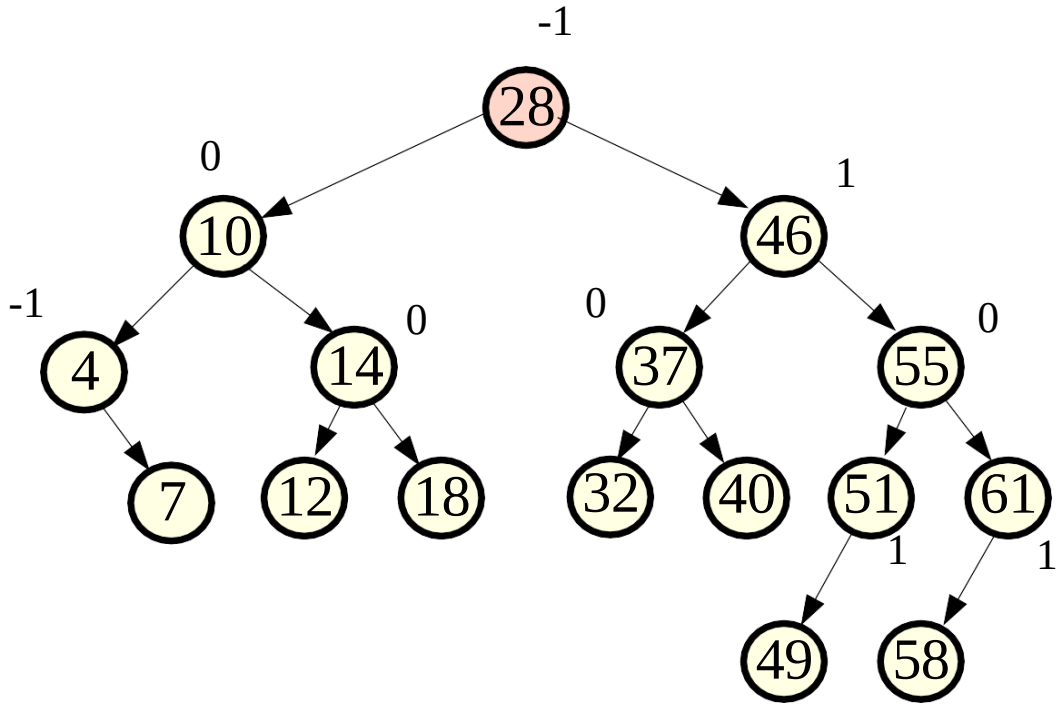


Example: Delete 21

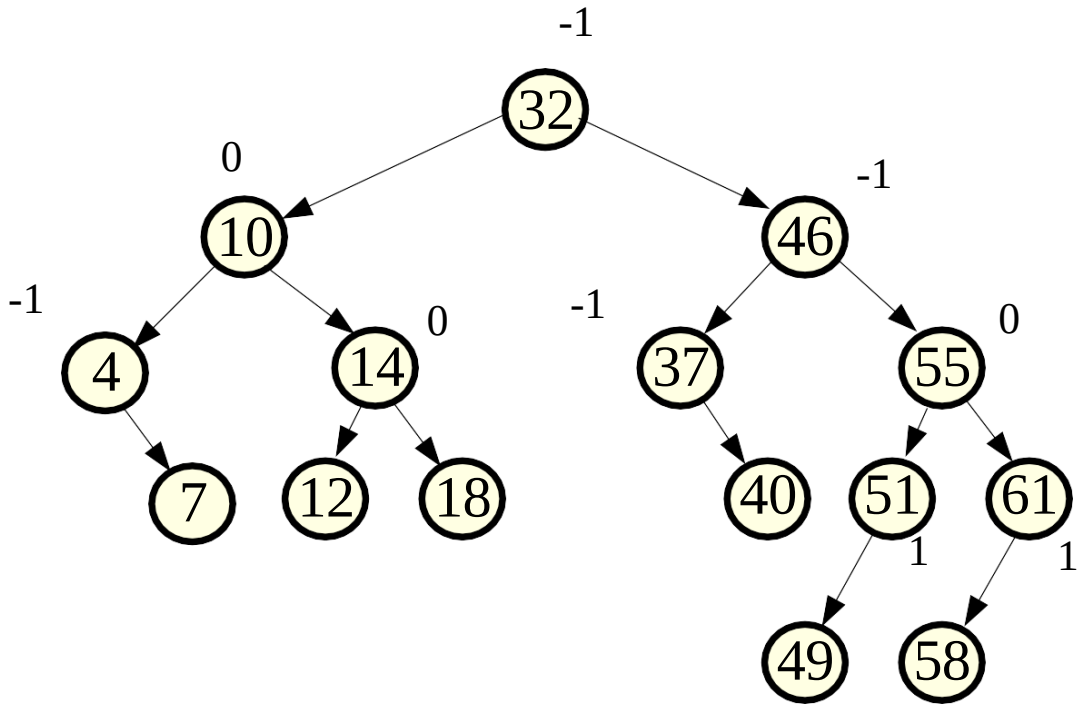


No change

Example: Delete 28

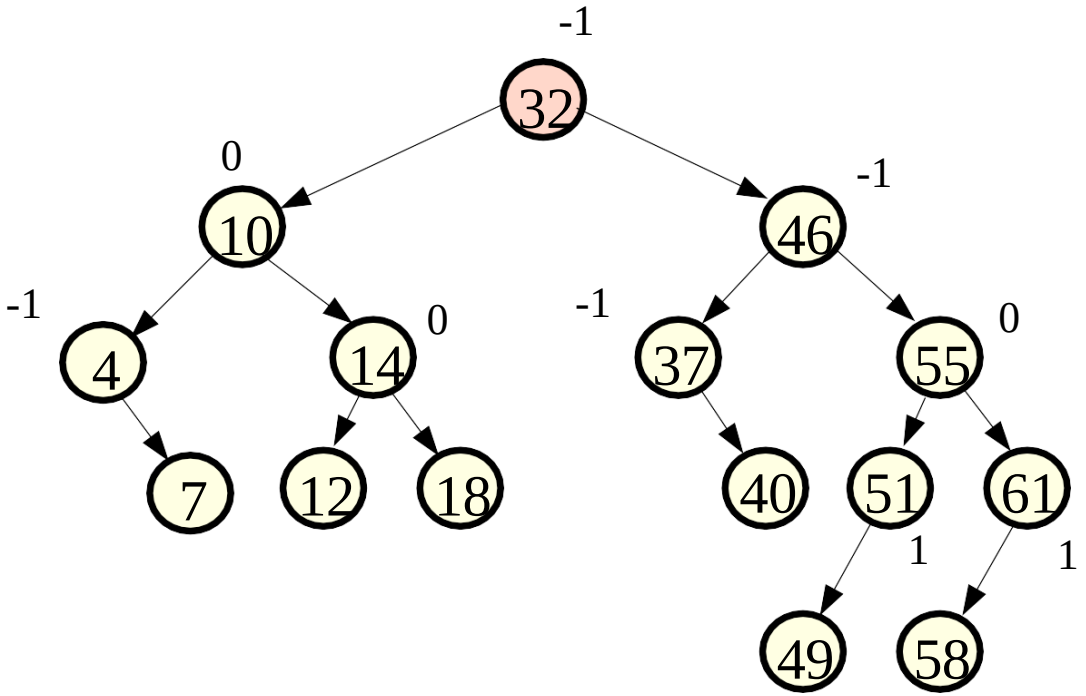


Example: Delete 28

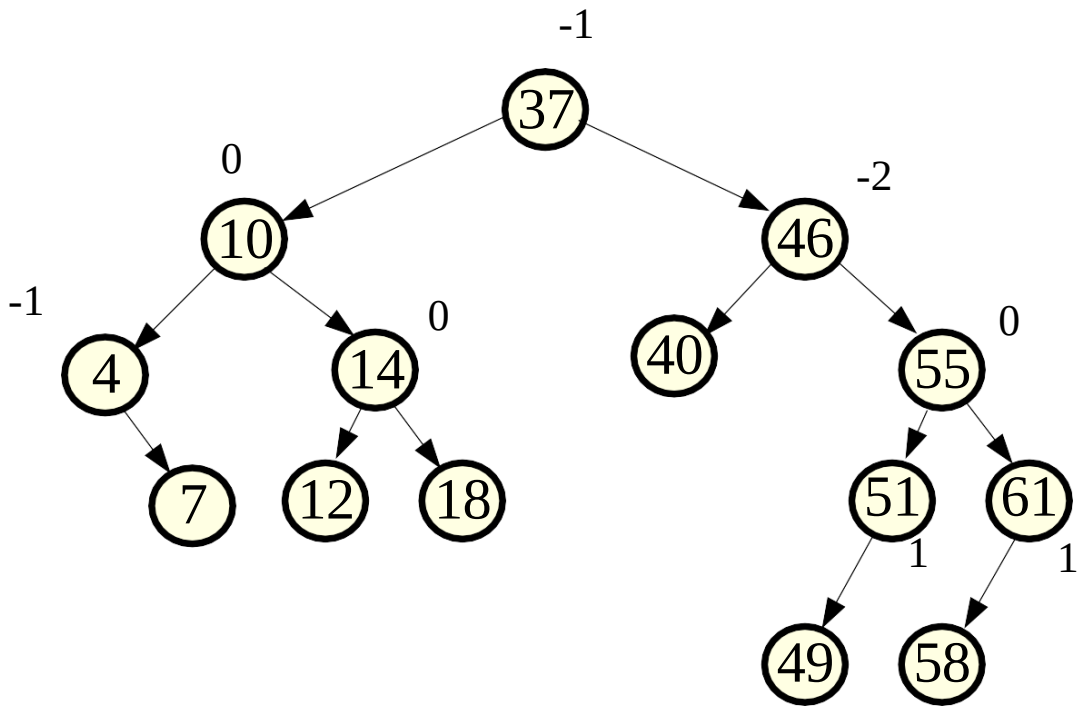


No change

Example: Delete 32

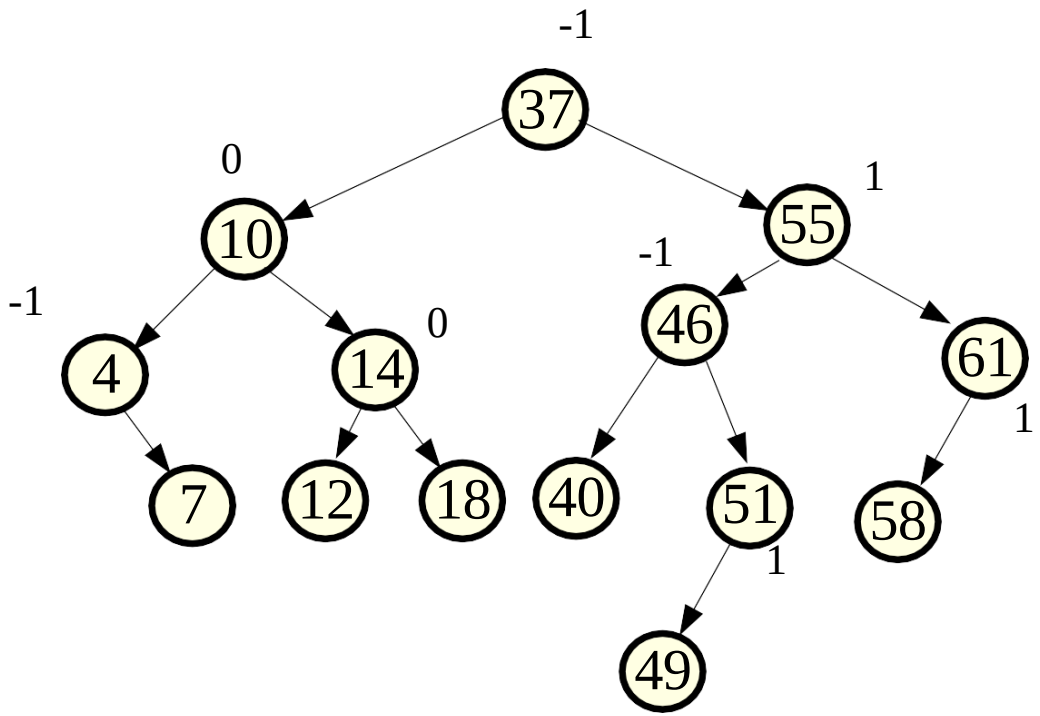


Example: Delete 32



Rotation around 55

Example: Delete 32



Rotation around 55