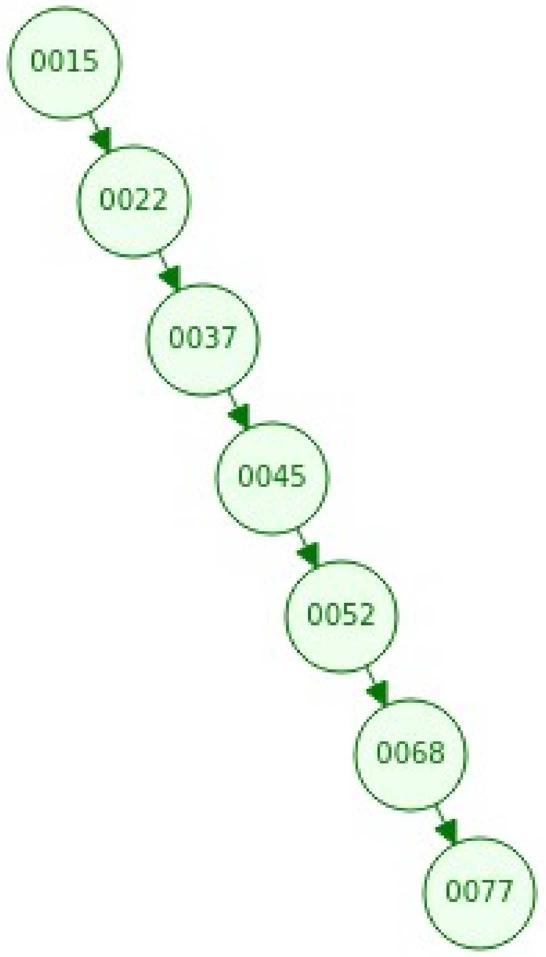
AVL Tree

Prof. Siddharth Shah

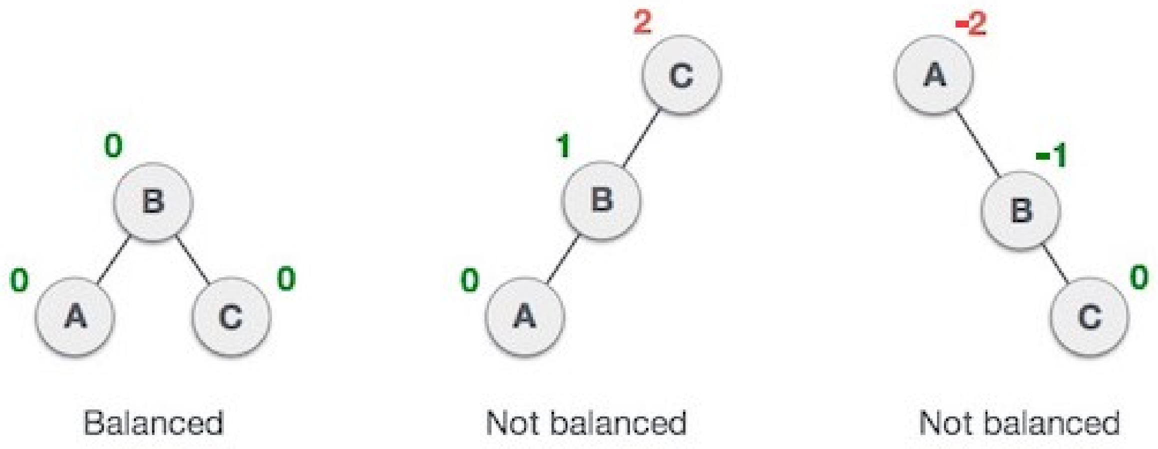
Problem in BST

- What if the input to binary search tree comes in a sorted (ascending or descending) manner?
- It is observed that BST's worst-case performance is closest to linear search algorithms, that is O(n).
- In real-time data, we cannot predict data pattern and their frequencies.
- So, a need arises to balance out the existing BST.



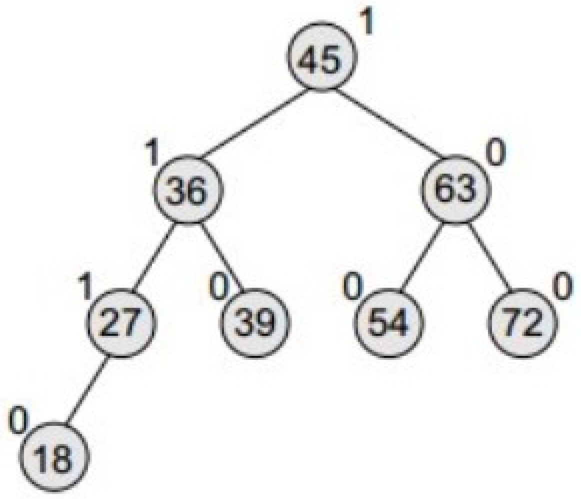
AVL Tree

- Named after their inventor Georgy Adelson-Velsky & Evgenii Landis, AVL tree is self-balancing binary search tree.
- AVL tree checks the height of the left and the right sub-trees and assures that the difference is not more than 1.
- This difference is called the Balance Factor.
- In the below figure, the first tree is balanced and the next two trees are not balanced –

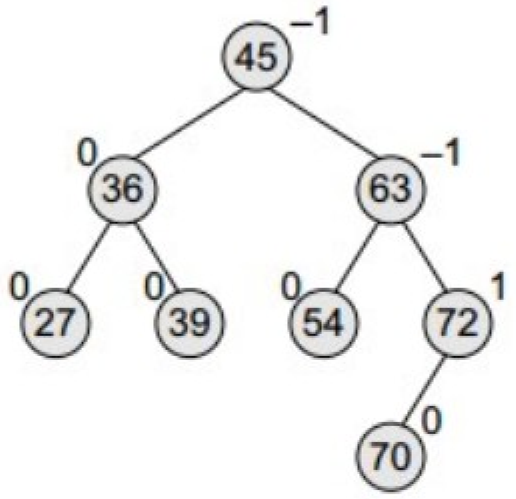


BalanceFactor = height (left-subtree) – height (right-subtree)

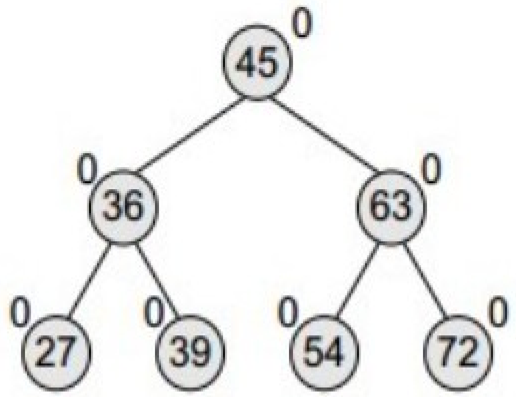
Types of AVL Tree



Left-Heavy AVL Tree



Right-Heavy AVL Tree



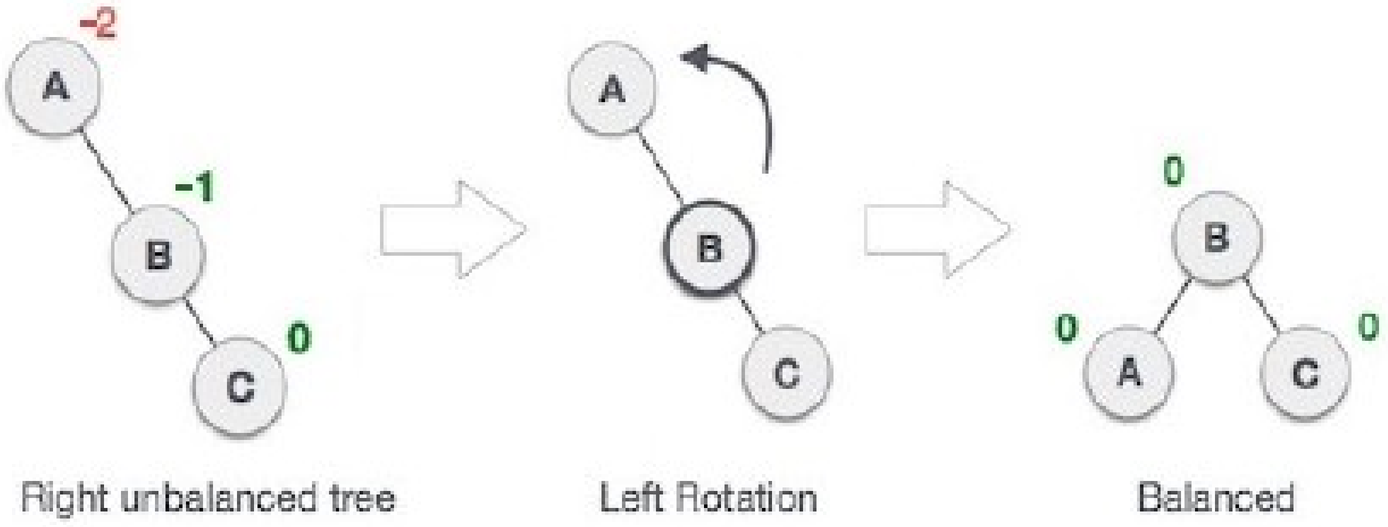
Balanced AVL Tree

AVL Rotations

- Insertions and deletions from an AVL tree may disturb the balance factor of the nodes and, thus, rebalancing of the tree may have to be done
- The tree is rebalanced by performing rotation at the critical node (a node where balance factor is other than -1, 0 and 1).
- To balance itself, an AVL tree may perform the following four kinds of rotations
  1. **Left rotation (Right-Right Case)**
  2. **Right rotation (Left-Left Case)**
  3. **Left-Right rotation (Left-Right Case)**
  4. **Right-Left rotation (Right-Left Case)**
- The first two rotations are single rotations and the next two rotations are double rotations.

Left Rotation

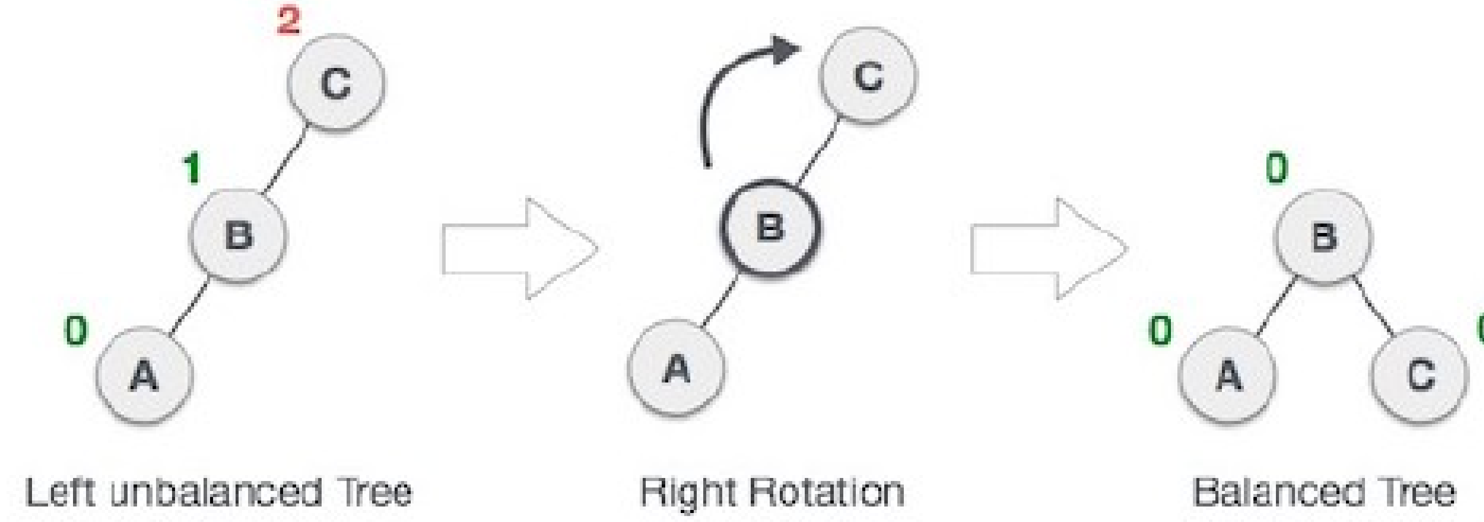
- If a tree becomes imbalanced, when a node is inserted into the right subtree of the right subtree, then a single left rotation is performed to balance it.



- In this example, node A has become imbalanced as a node C is inserted in the right subtree of A's right subtree (B).
- The left rotation is performed by making A the left-subtree of B.

Right Rotation

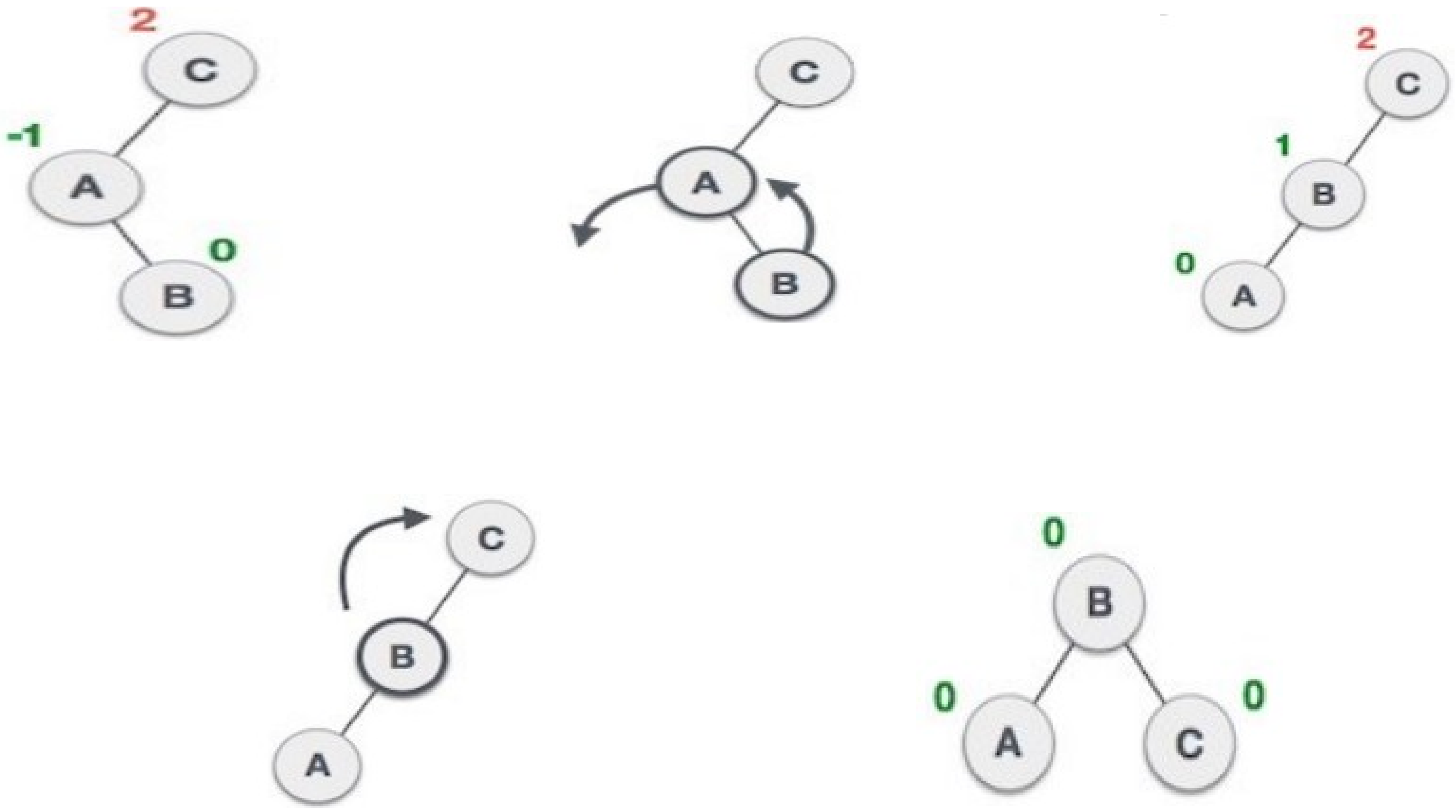
- AVL tree may become imbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.



- As depicted, the imbalanced node becomes the right child of its left child by performing a right rotation.

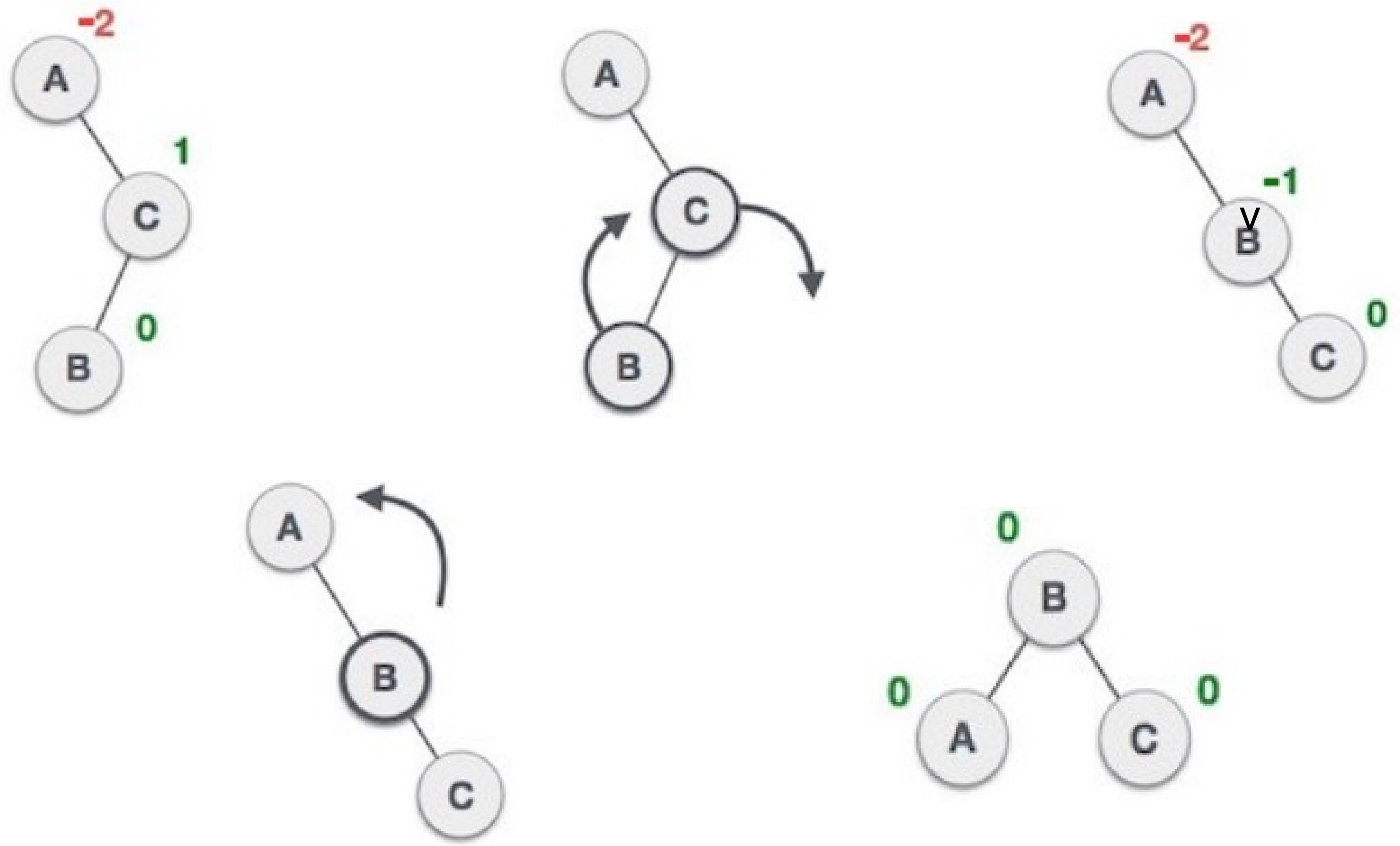
Left - Right Rotation

- A left-right rotation is a combination of left rotation followed by right rotation.



Right - Left Rotation

- A right-left rotation is a combination of right rotation followed by left rotation.



Operations on AVL Tree

- The following operations are performed on an AVL tree...
    1. Search
    2. Insertion
    3. Deletion

Search AVL Tree

- In an AVL tree, the search operation is performed with O(log n) time complexity.
- The search operation is performed similar to Binary search tree search operation.
- The following steps are used to search an element in AVL tree...

  1. Read the search element from the user
  2. Compare, the search element with the value of root node in the tree.
  3. If both are matching, then display "Given node found!!!" and terminate the function
  4. If both are not matching, then check whether search element is smaller or larger than that node value.

Search AVL Tree (Cont..)

6. If search element is smaller, then continue the search process in left subtree.
7. If search element is larger, then continue the search process in right subtree.
8. Repeat the same until we found exact element or we completed with a leaf node
9. If we reach to the node with search value, then display "Element is found" and terminate the function.
10. If we reach to a leaf node and it is also not matching, then display "Element not found" and terminate the function.

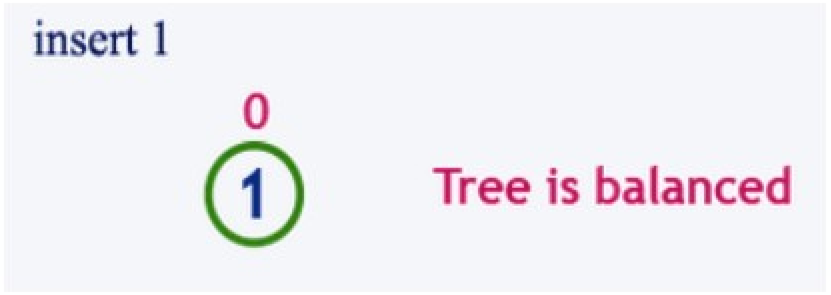
Insertion in AVL Tree

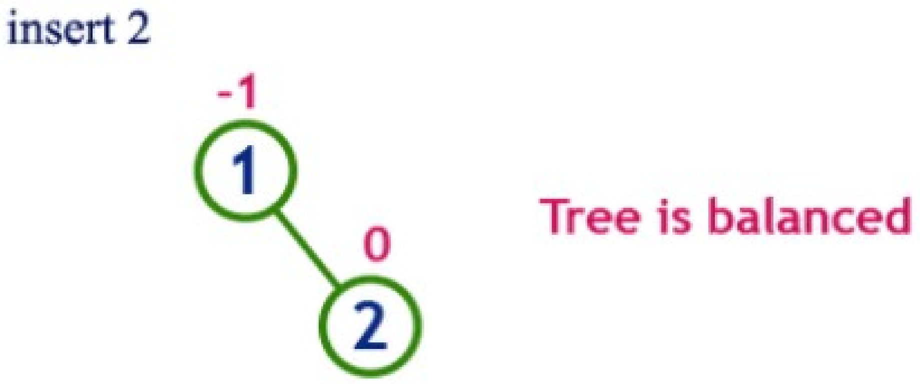
- In an AVL tree, the insertion operation is performed with O(log n) time complexity. In AVL Tree, new
- node is always inserted as a leaf node. The insertion operation is performed as follows...

    1. Insert the new element into the tree using Binary Search Tree insertion logic.
    2. After insertion, check the Balance Factor of every node.
    3. If the Balance Factor of every node is 0 or 1 or -1 then complete the operation.
    4. If the Balance Factor of any node is other than 0 or 1 or -1 then tree is said to be imbalanced. Then perform the suitable Rotation to make it balanced.

Example

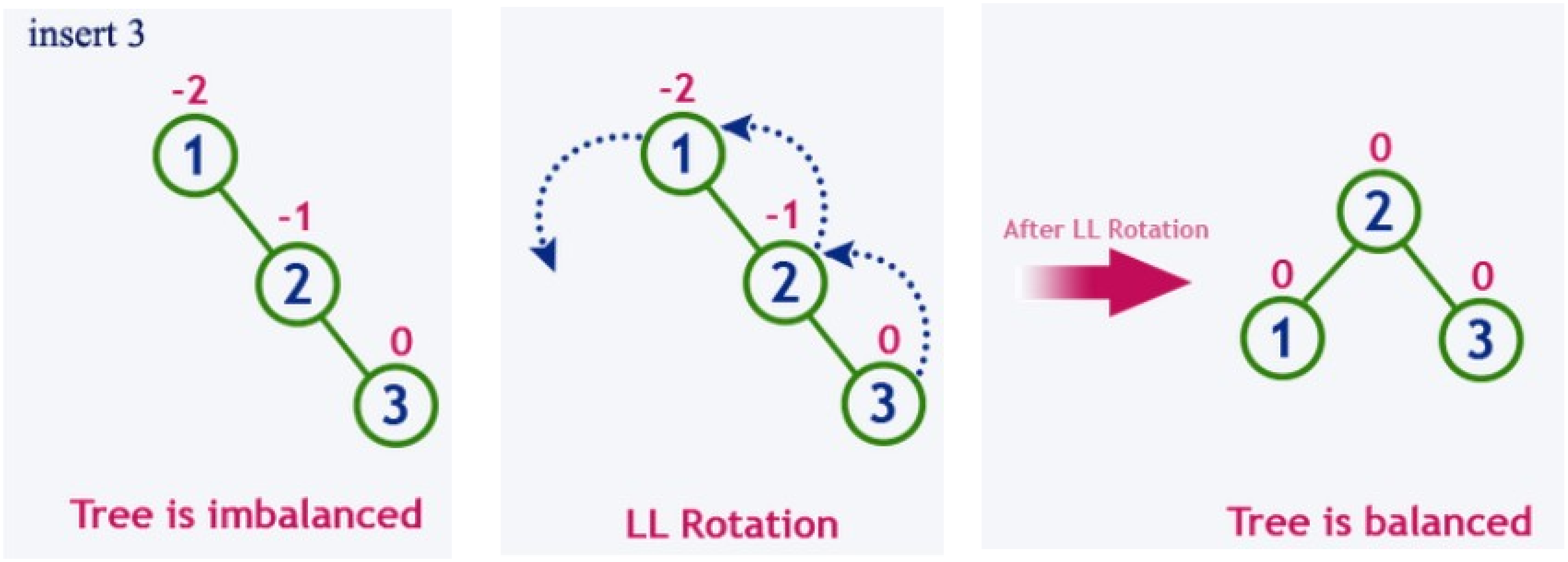
Construct an AVL Tree by inserting numbers from 1 to 8.





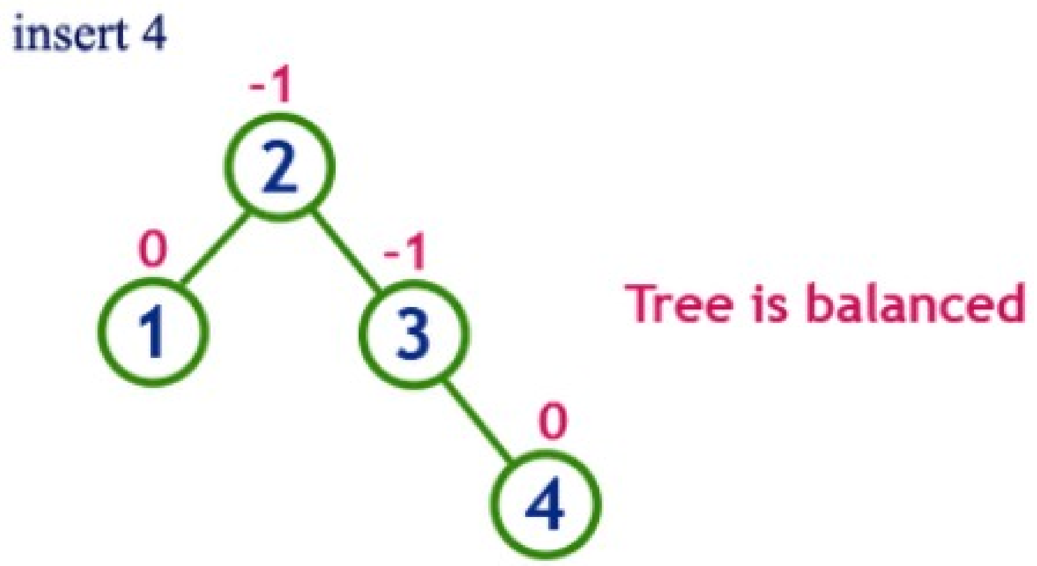
Example

Construct an AVL Tree by inserting numbers from 1 to 8.



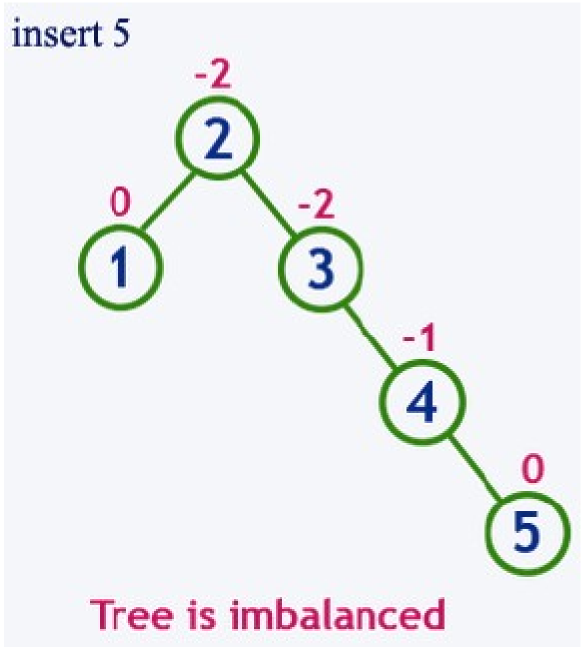
Example

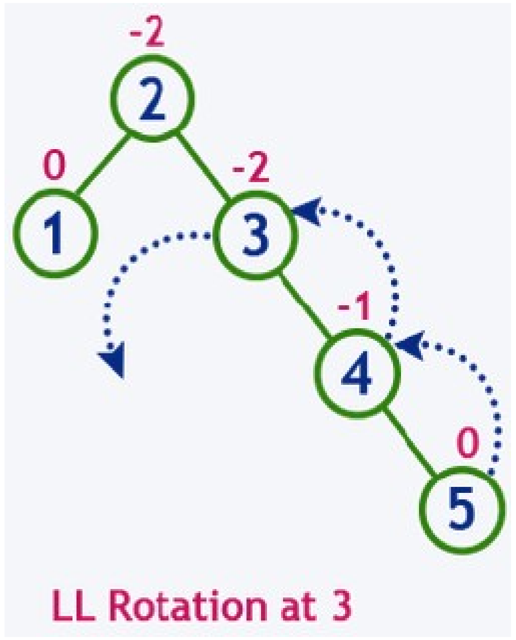
Construct an AVL Tree by inserting numbers from 1 to 8.

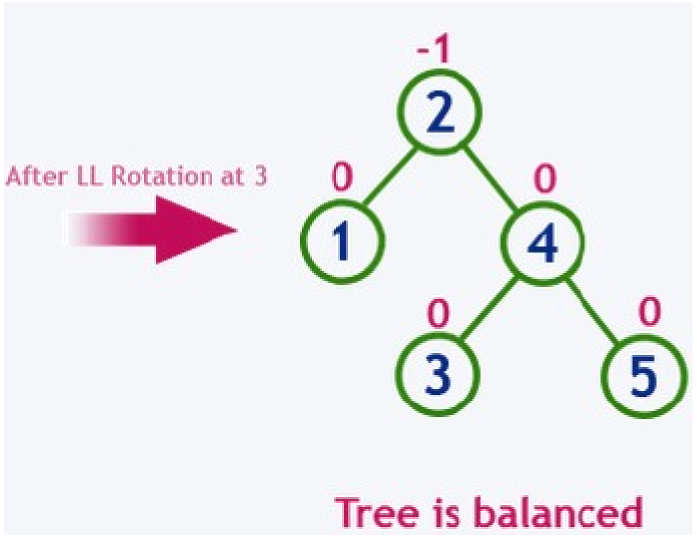


Example

Construct an AVL Tree by inserting numbers from 1 to 8.

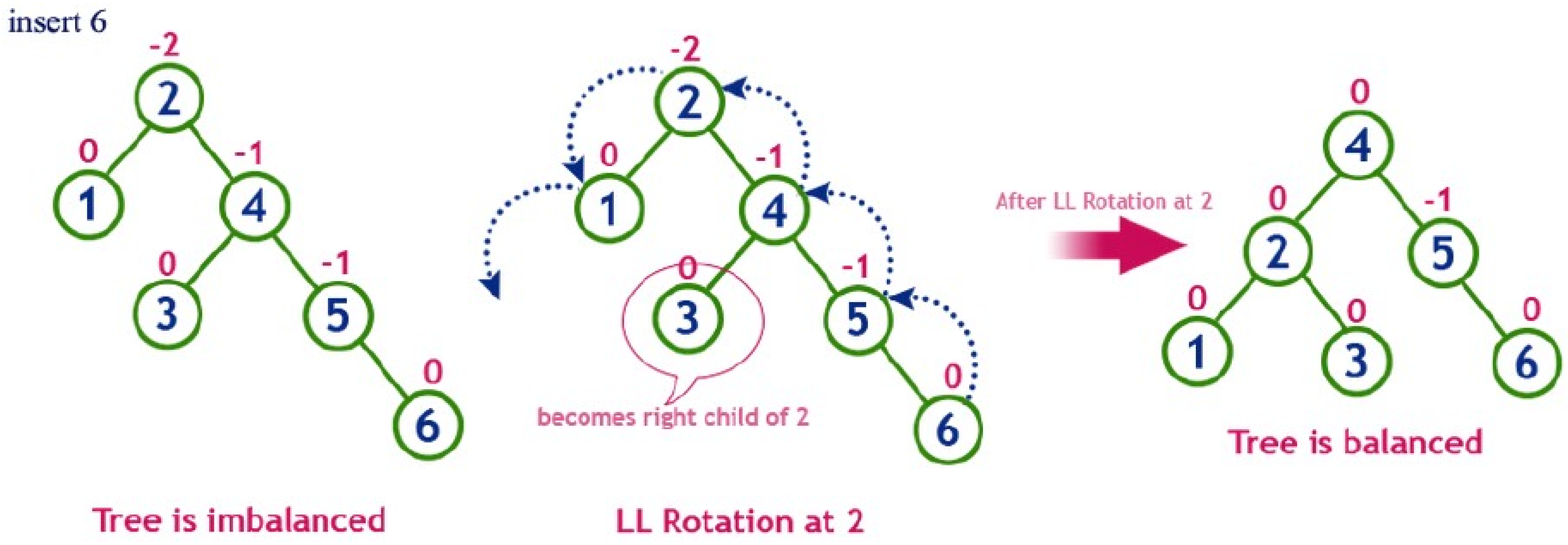






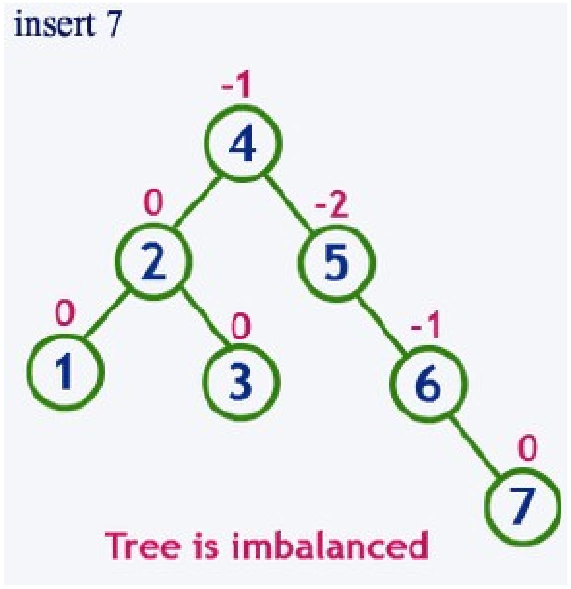
Example

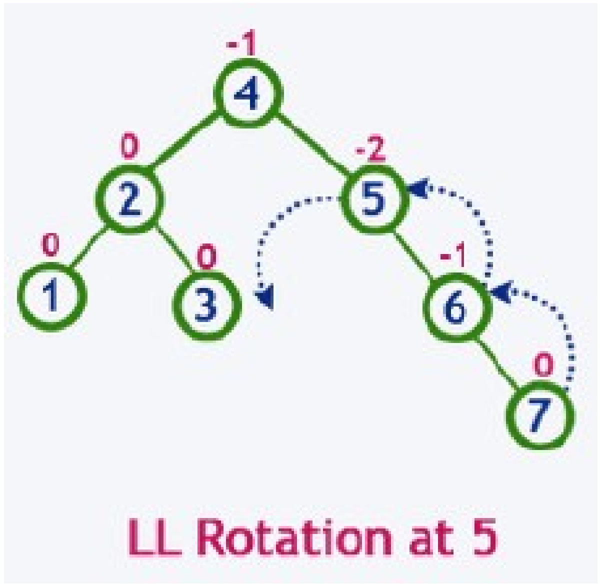
Construct an AVL Tree by inserting numbers from 1 to 8.

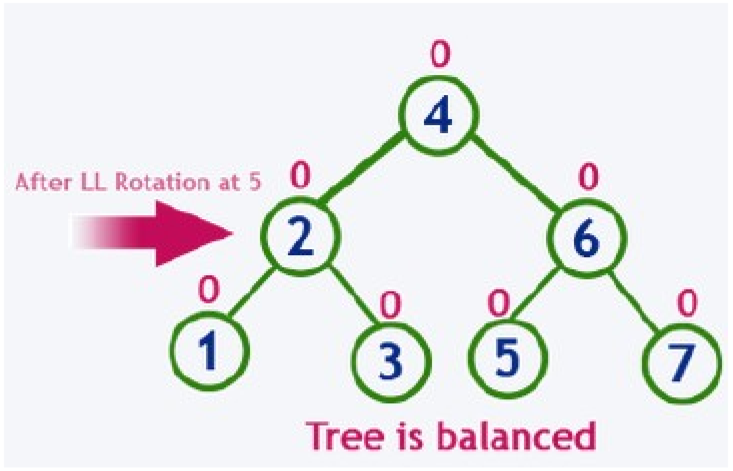


Example

Construct an AVL Tree by inserting numbers from 1 to 8.

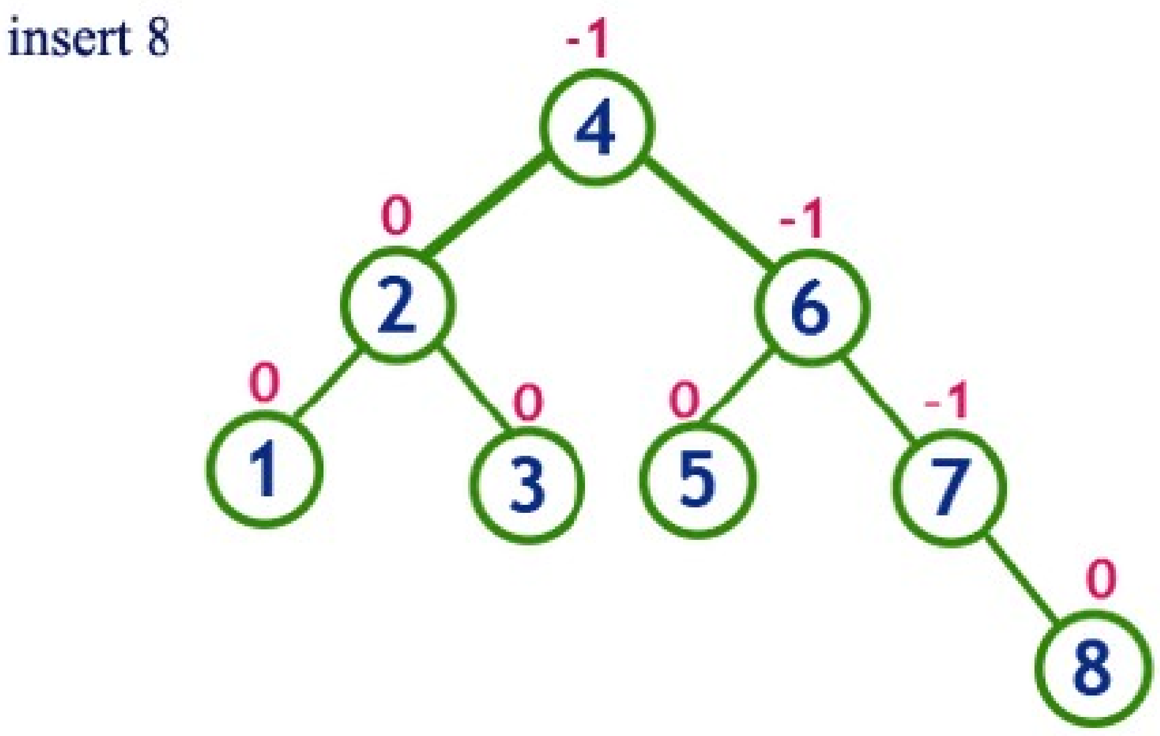






Example

Construct an AVL Tree by inserting numbers from 1 to 8.



Tree is balanced

Assignment

1. Construct an AVL Tree by inserting numbers 14, 17, 11, 7, 53, 4, 13 into an empty AVL tree

2. Construct an AVL Tree by inserting numbers 15, 20, 24, 10, 13, 7, 30, 36, 25 into an empty AVL tree

Deletion in AVL Tree

- Deletion of a node in an AVL tree is similar to that of binary search trees. But it goes one step ahead.
- Deletion may disturb the balance of the tree, so to rebalance the AVL tree; rotation(s) needs to be performed.
- There are two classes of rotations that can be performed on an AVL tree after deleting a given node.
- These rotations are **R** rotation and **L** rotation.
- On deletion of node X from the AVL tree, if node A becomes the critical node, then the type of rotation depends on whether X is in the left sub-tree of A or in its right sub-tree.

Deletion in AVL Tree

- Let A be the node where balance must be restored.
- If the deleted node X was in A's **right subtree**, then let B be the root of A's left subtree. Then:

B's BF	Rotation
+1	Right Rotation (RR)
-1	Left-Right Rotation (LR)
0	Either RR or LR

- If the deleted node X was in A's **left subtree**, then let B be the root of A's right subtree. Then:

B's BF	Rotation
+1	Right-Left Rotation (RL)
-1	Left Rotation (LL)
0	Either RL or LL

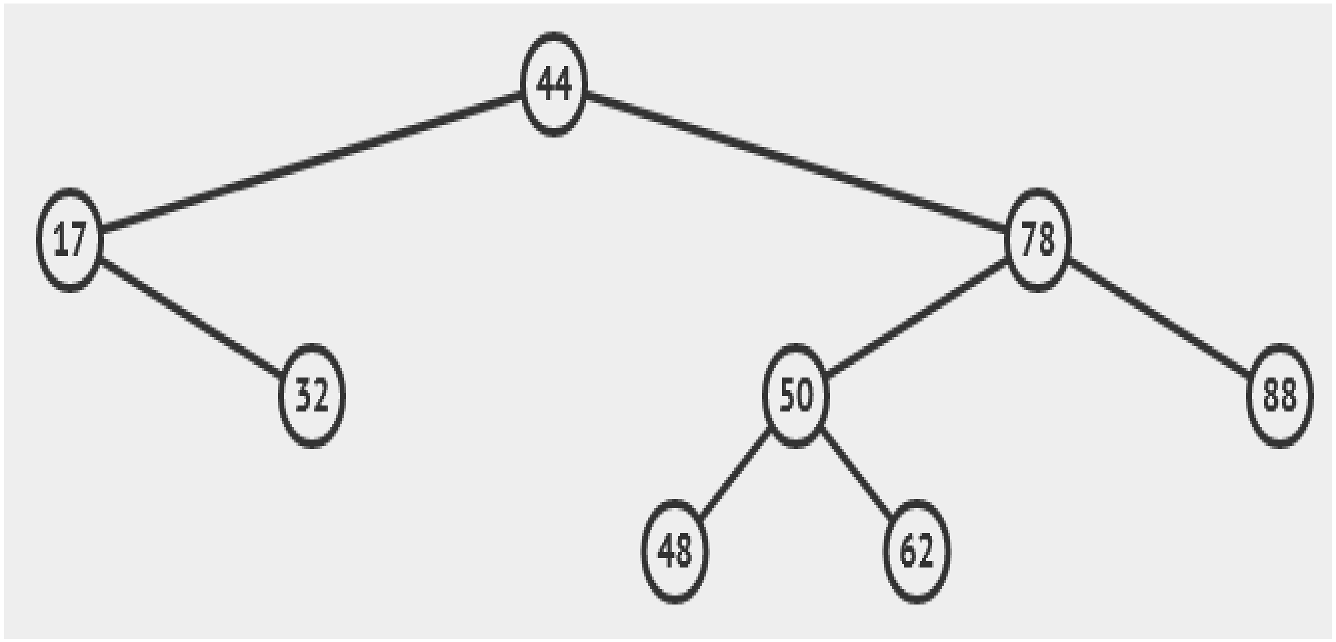
Deletion in AVL Tree

- Following are the steps to perform to delete a node from AVL Tree.

1. Delete the element from the tree using Binary Search Tree deletion logic.
2. After deletion, check the Balance Factor of every node.
3. If the Balance Factor of every node is 0 or 1 or -1 then complete the operation.
4. If the Balance Factor of any node is other than 0 or 1 or -1 then tree is said to be imbalanced. Then perform the suitable Rotation to make it balanced.

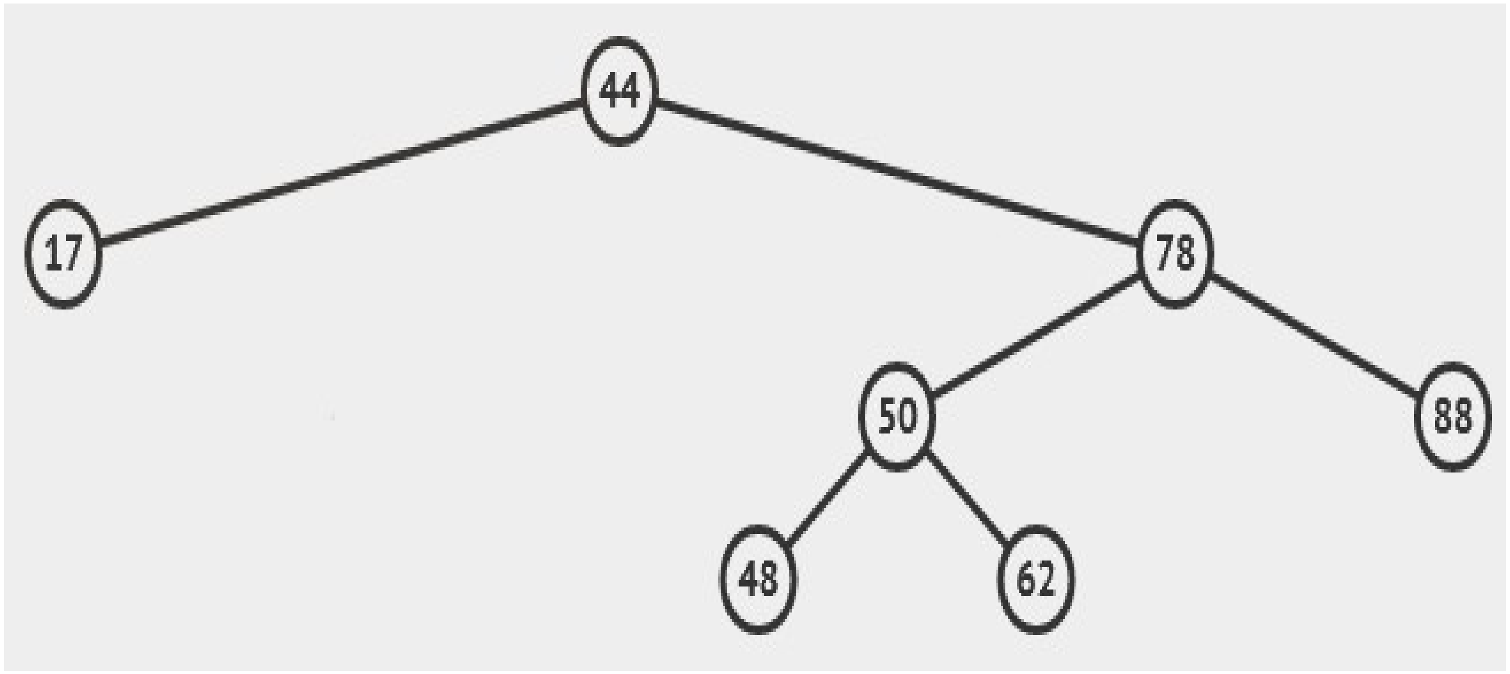
Example 1

Delete node 32 from the given tree and balance it, if required.



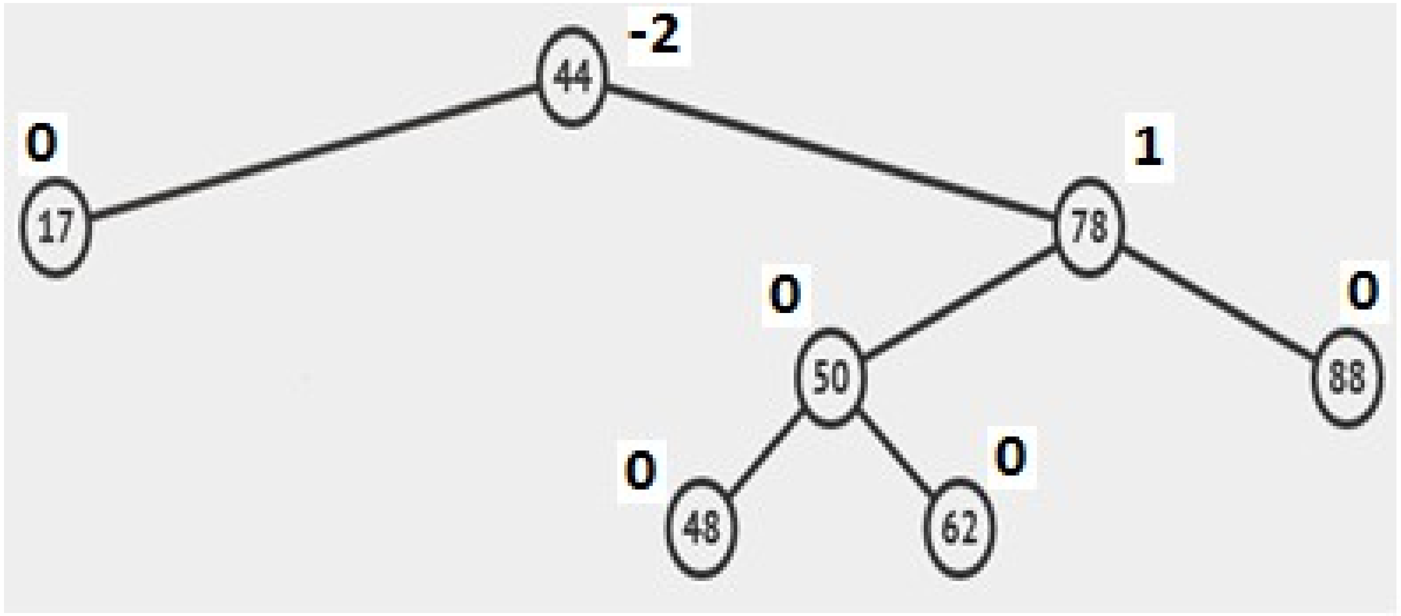
Example 1

1. Delete node 32.



Example 1

2. Check the balance factor of every node

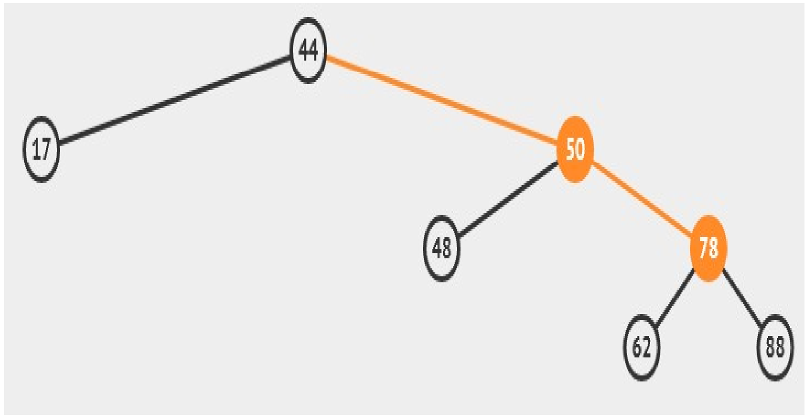


Example 1

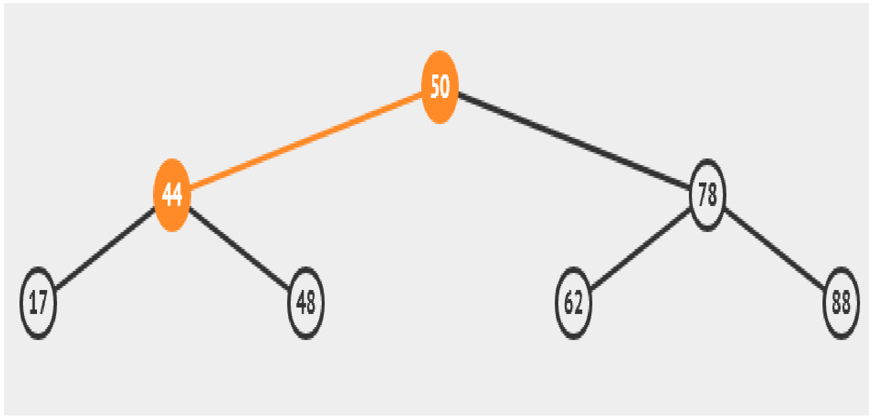
3. Balance the tree

1. 44 is a critical node
2. Deletion is performed in left-subtree of critical node
3. 78 is a root node of right-subtree of a critical node with balance factor 1, so perform Right-Left rotation.

Right rotation on 78

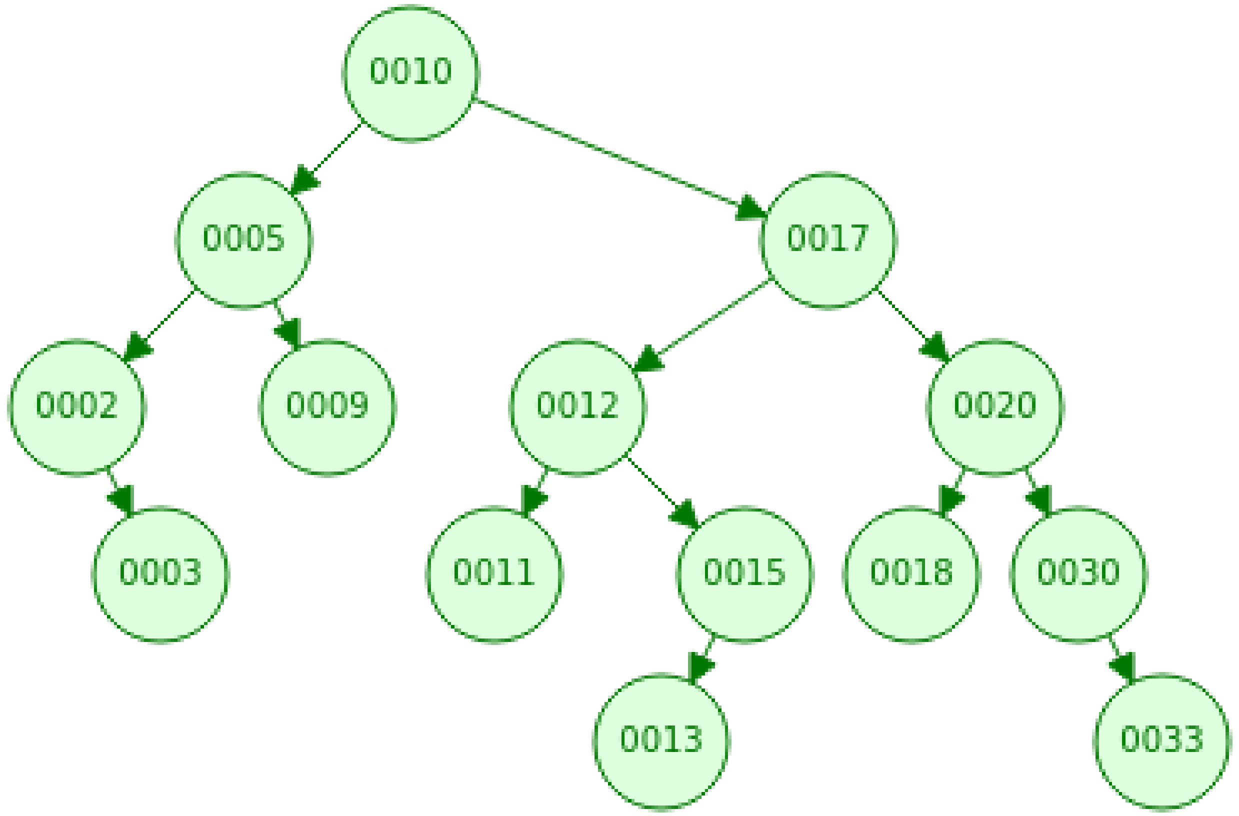


Left rotation on critical node 44



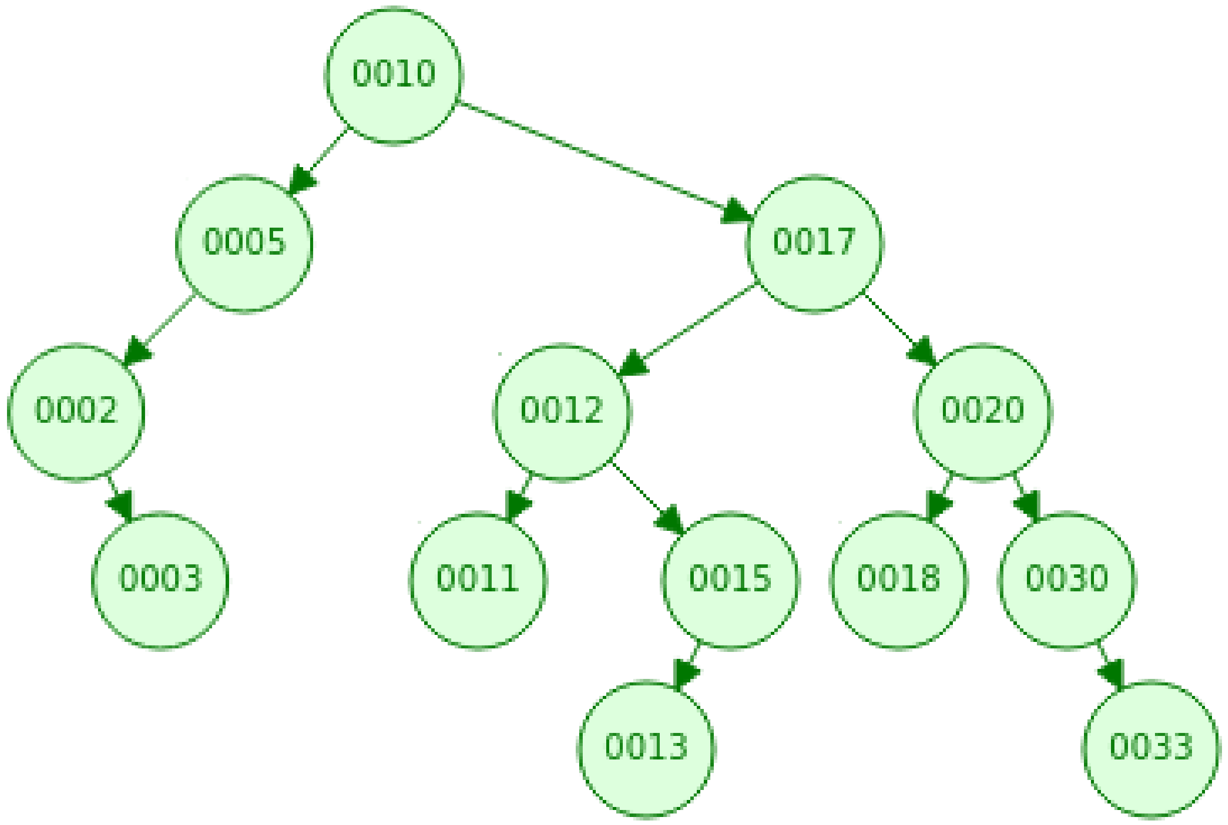
Example 2: Multiple Rotation

Delete node 9 from the given tree and balance it, if required.



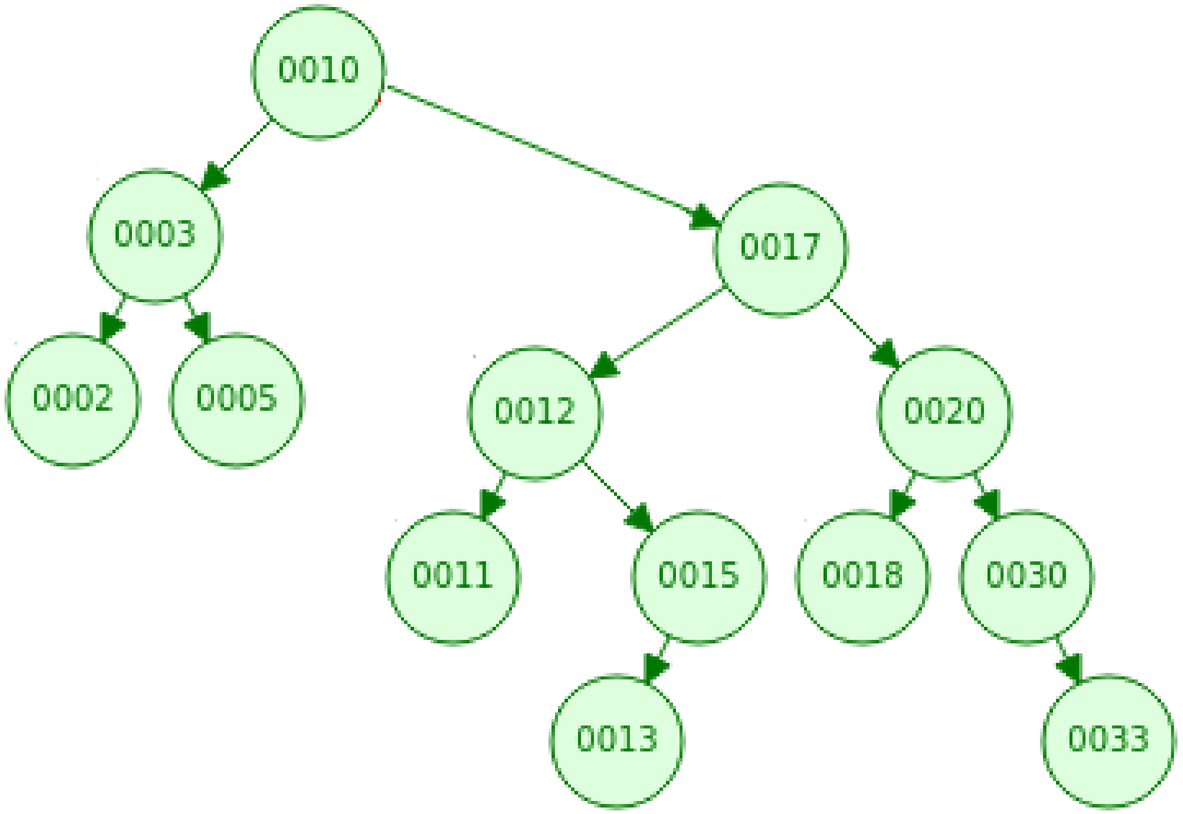
Example 2

1. Delete node 9.



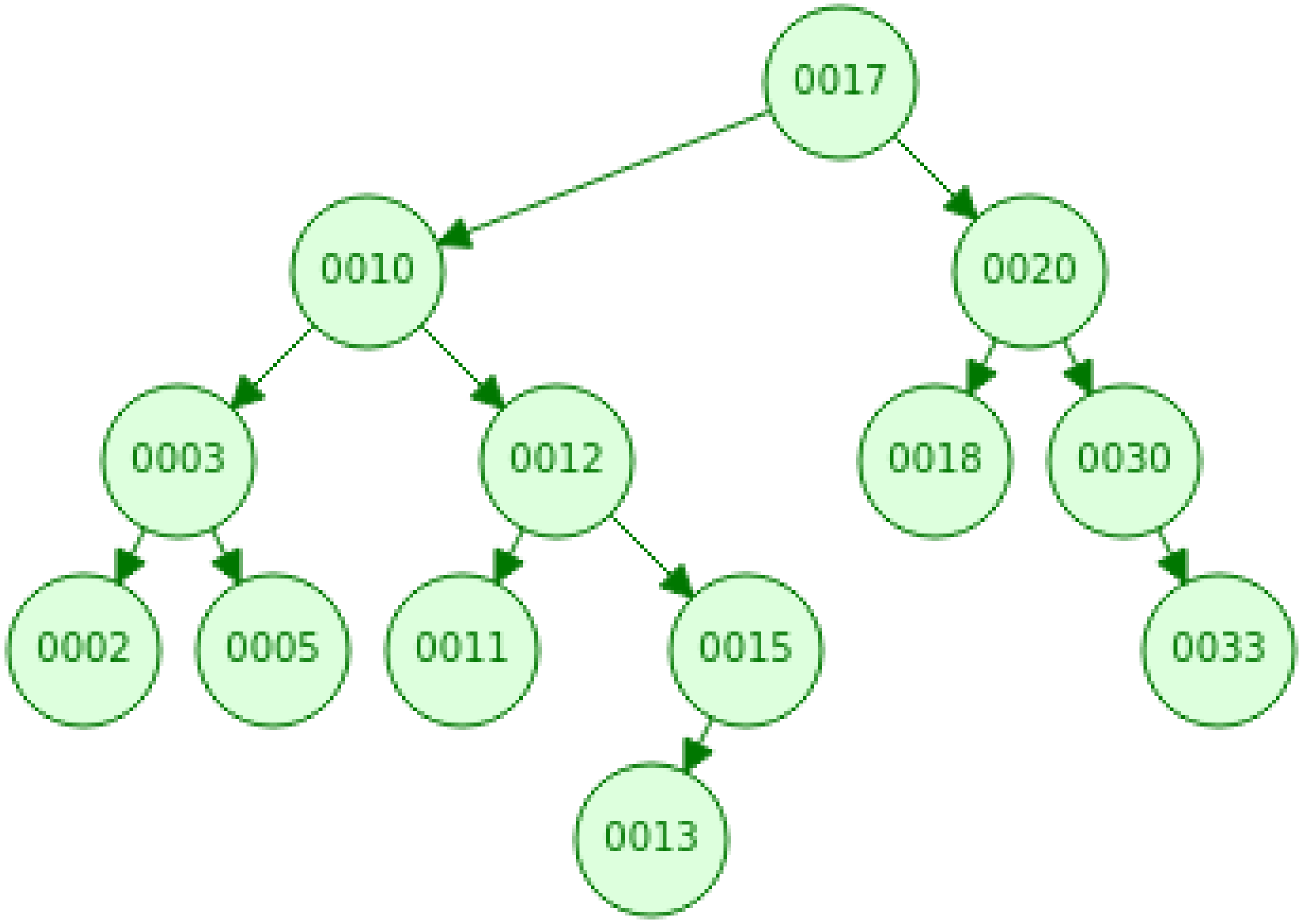
Example 2

2. Balance the tree at critical node 5



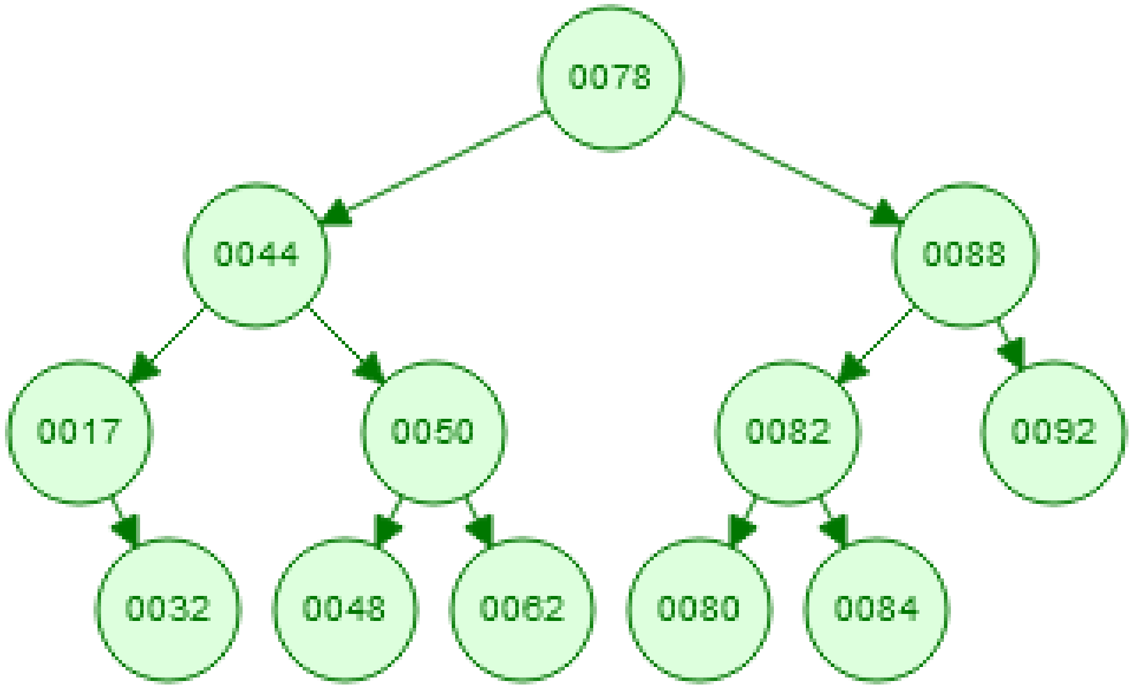
Example 2

3. Balance the tree at critical node 10



Assignment

Modify the below given AVL Tree by performing deletion of nodes in the mentioned order: **92, 80, 50, 44, 17, and 88.**


0078
0044
0088
0017
0050
0082
0092
0032
0048
0062
0080
0084