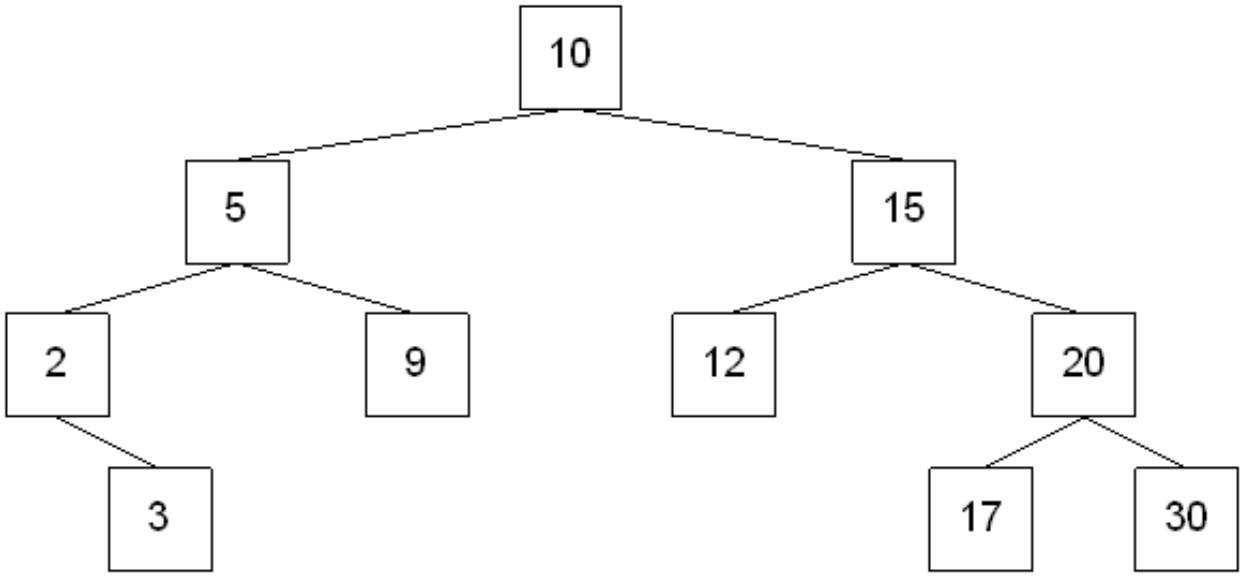
AVL Trees continued

Deletion from an AVL Search Tree

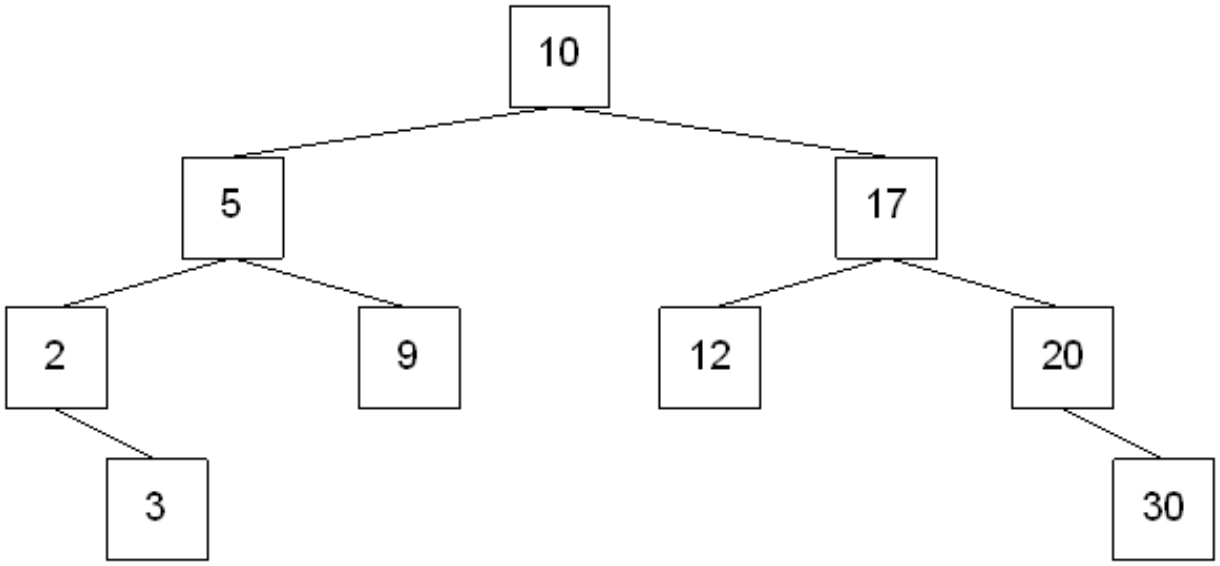
As with insertions, a node is deleted using the standard inorder successor (predecessor) logic for binary search trees. But, just like insertion, deletion can cause an imbalance, which will need to be fixed by applying one of the four rotations.

Example 1:

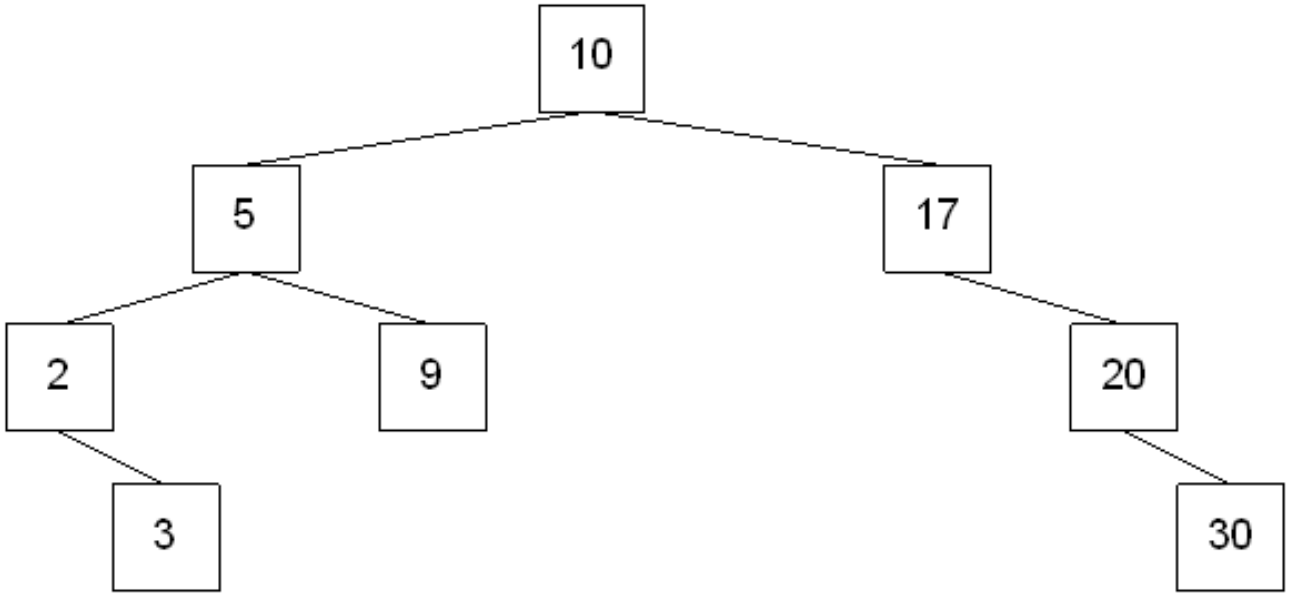


Now suppose that the nodes 15 and 12 are deleted. The resulting tree is:

Delete 15 using INORDER SUCCESSOR logic:



Delete 12 using INORDER SUCCESSOR logic:



If we identify the parent of the actual node that was deleted, then:

- If the left child was deleted, the balance factor at the parent decreased by 1.
- If the right child was deleted, the balance factor at the parent increased by 1.

The change in balance factor can move up the tree from the deleted node's parent all the way to the root. Therefore, it's possible that a node's balance

factor could become +2 or -2. If this happens, then balance needs to be restored.

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

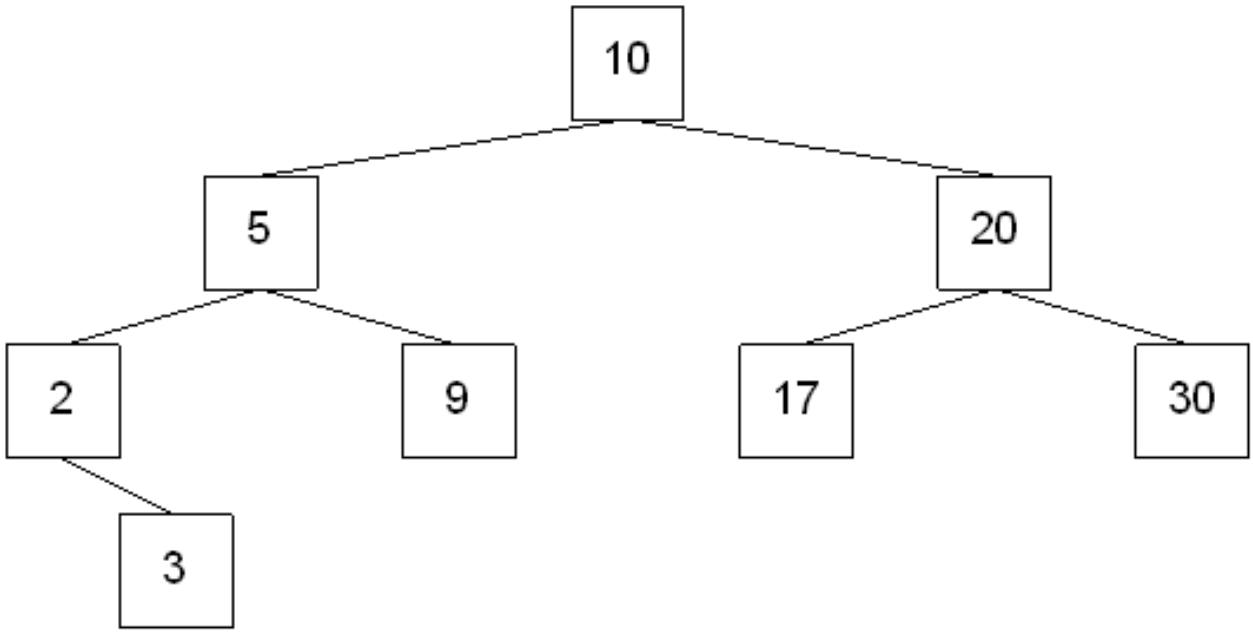
1. **B has balance factor 0 or +1 after deletion** -- then perform an **LL rotation**
2. **B has balance factor -1 after deletion** -- then perform an **LR rotation**

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

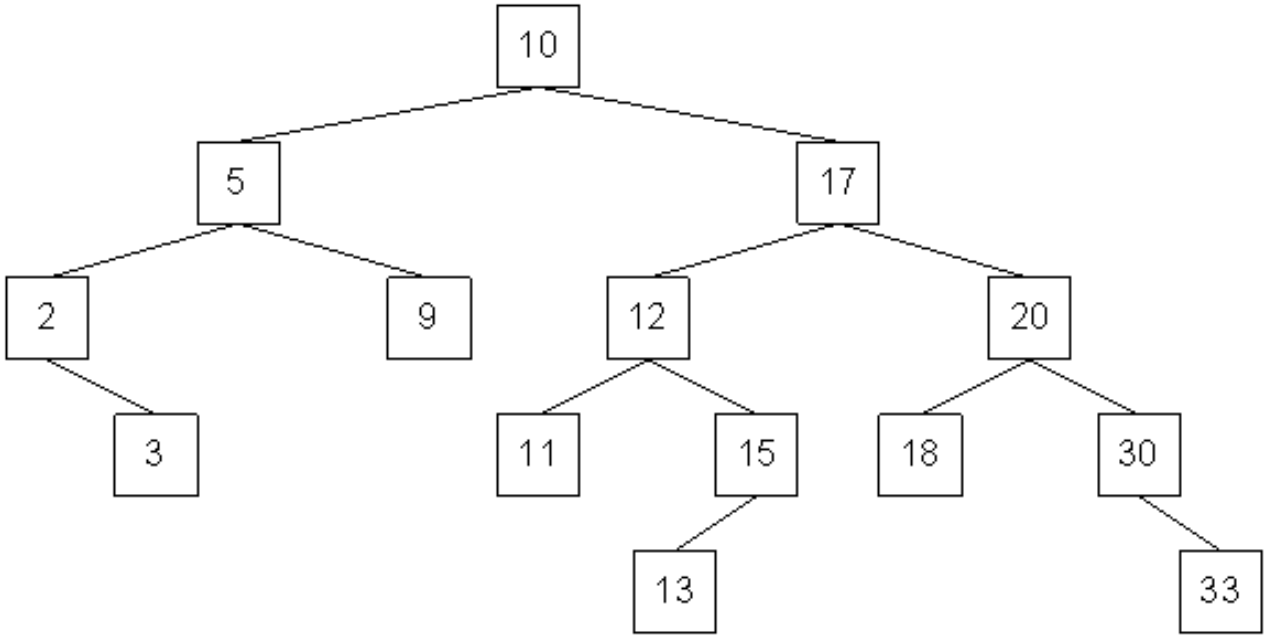
1. **B has balance factor 0 or -1 after deletion** -- then perform an **RR rotation**
2. **B has balance factor +1 after deletion** -- then perform an **RL rotation**

What steps are needed to fix the above tree?

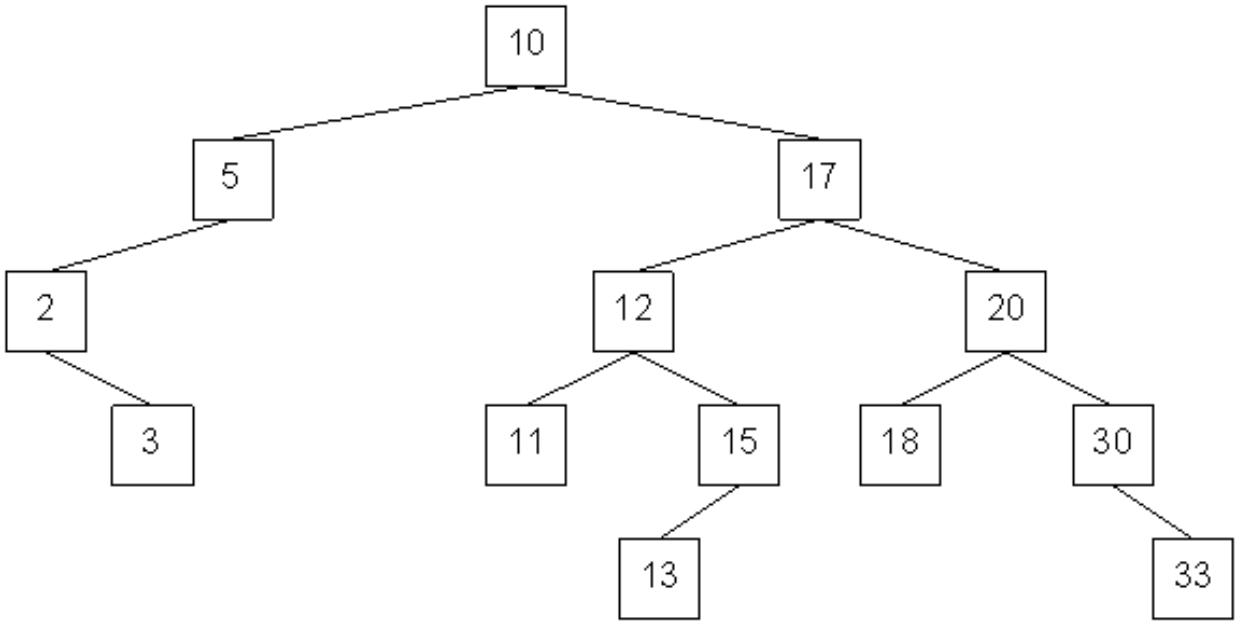
Here is the resulting tree:



Example 2:

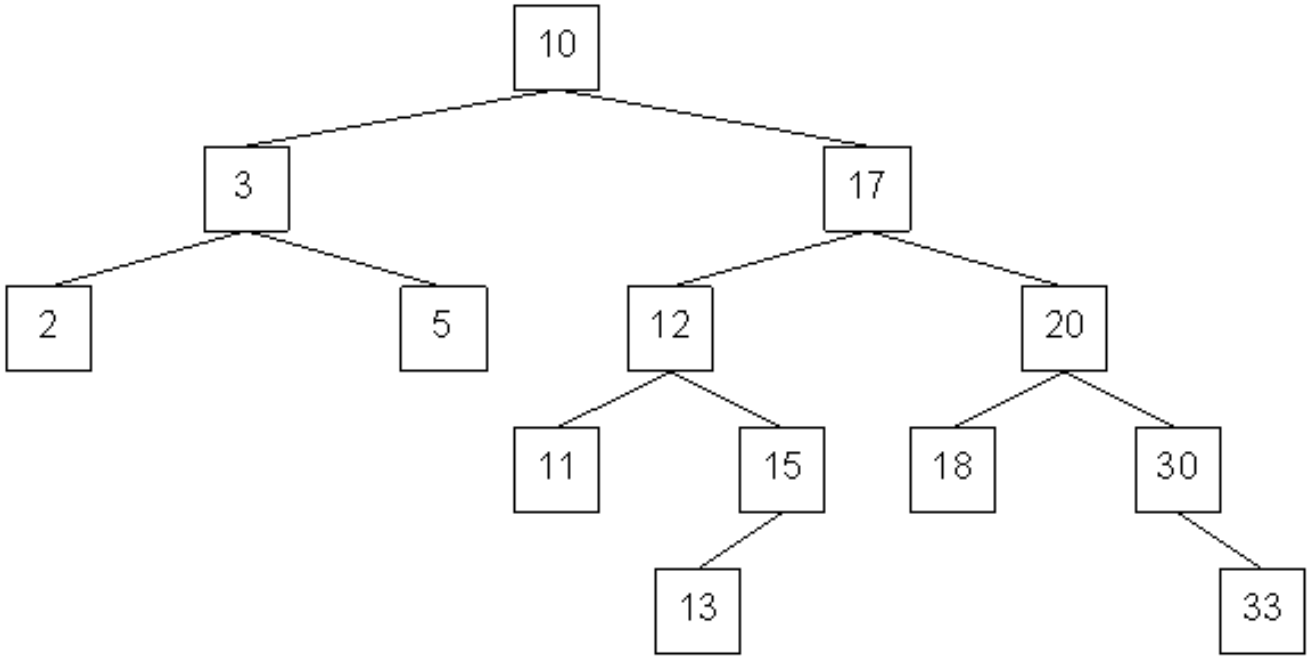


Delete 9 from the tree.



What steps do we need to take to rebalance the above tree?

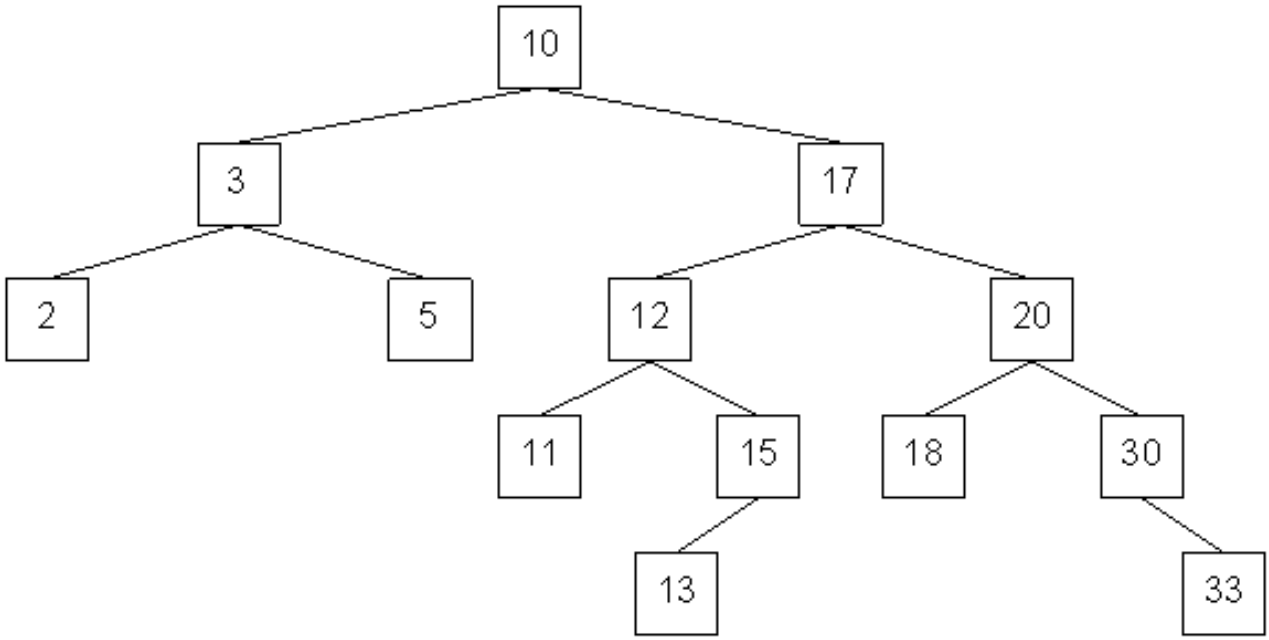
Here is the resulting tree:



As you can see, the tree is still unbalanced.

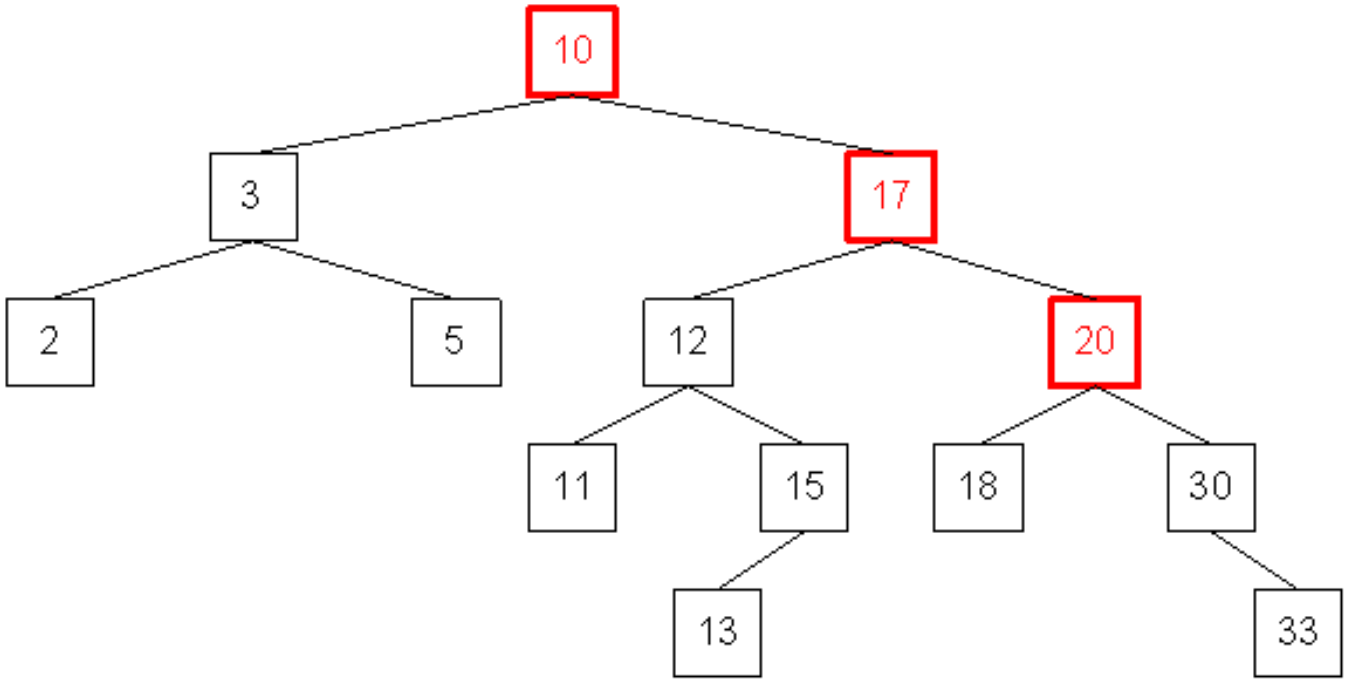
Unlike insertions, one rotation may not be enough to restore balance to the tree. If this is the case, then locate the next node where the balance factor is "bad" (call this A):

- If A's balance factor is positive, then let B be A's left child
  - If B's left subtree height is larger than B's right subtree height, then perform an **LL rotation**.
  - If B's left subtree height is smaller than B's right subtree height, then perform an **LR rotation**.
  - If B's left subtree height is equal to B's right subtree height, then perform either rotation.
- If A's balance factor is negative, then let B be A's right child
  - If B's right subtree height is larger than B's left subtree height, then perform an **RR rotation**.
  - If B's right subtree height is smaller than B's left subtree height, then perform an **RL rotation**.
  - If B's right subtree height is equal to B's left subtree height, then perform either rotation.

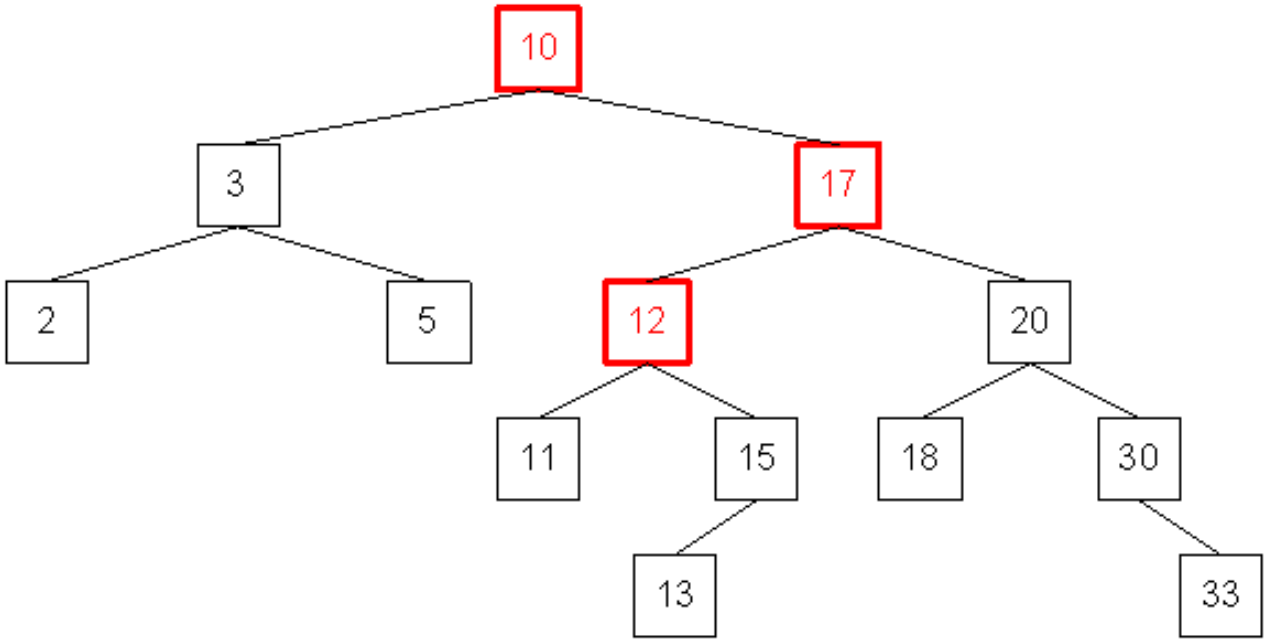


What type of rotation should be performed to fix the imbalance?

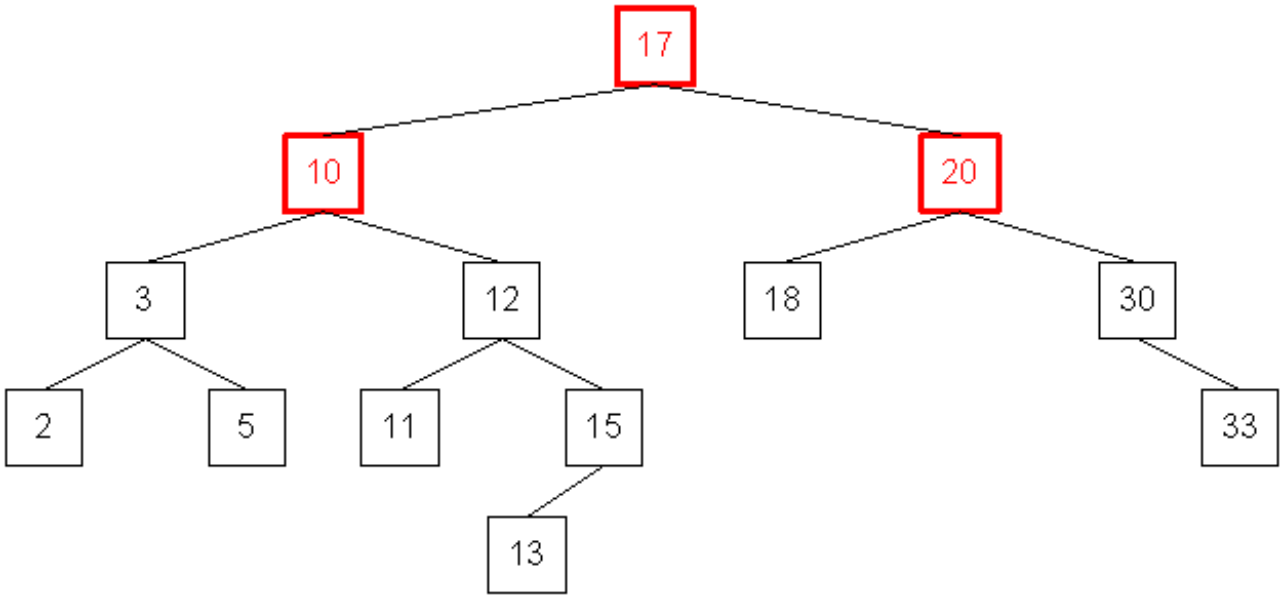
An RR rotation with the highlighted nodes



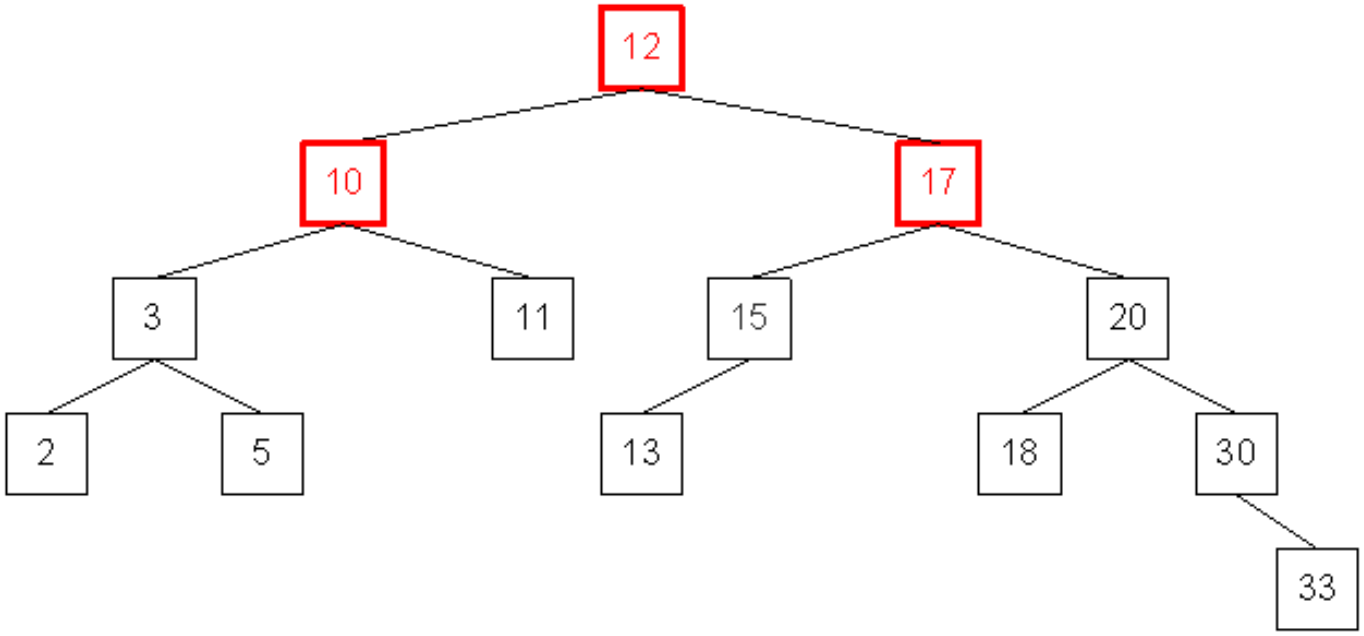
OR an RL rotation with the highlighted nodes



Result after the RR rotation:



Result after the right-left rotation:



Your turn.

Delete the node containing (S) from the following tree. Make sure that you document each step that you took.