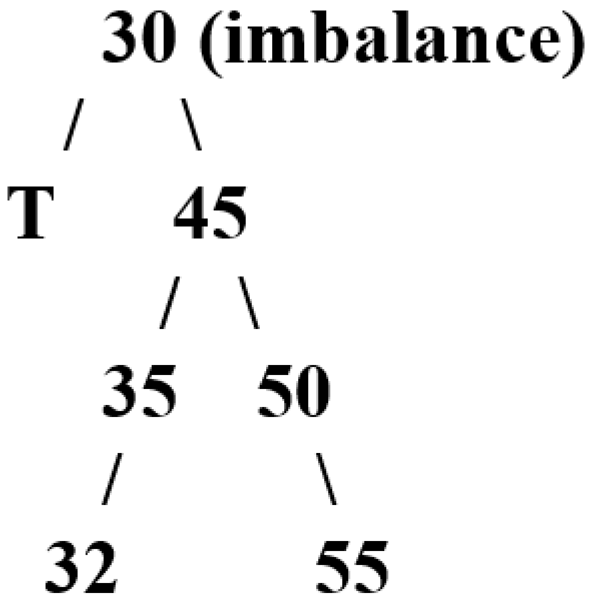
Example of node deletion from AVL Tree

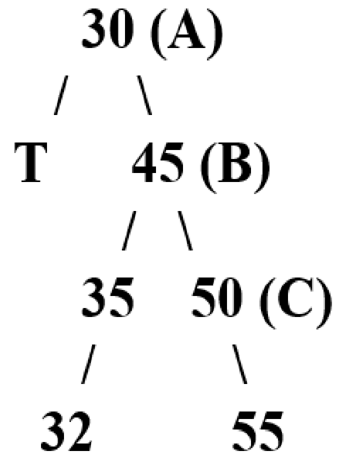
Example 1

- **Let’s say that the node deleted was in the subtree T shown below (which will store a single node in this instance) and the imbalance is caused at the node storing 30**

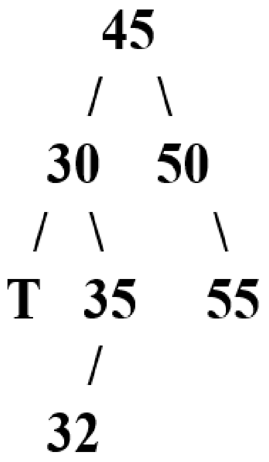


- **We know that 30 will be one of A, B, and C. So will 45.**

- **Now we must choose between 35 and 50.**
- **Since both subtrees of 45 are of the same height, and since 45 is the RIGHT child of 30, we will choose 45’s right child, 50 to be the third node of the group.**
- **Thus, our labels are as follows:**

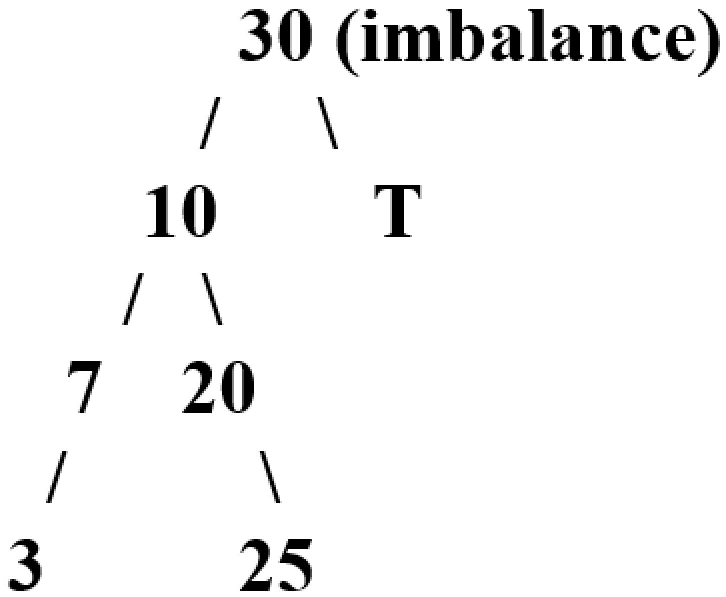


and our restructuring is as follows:



Example 2

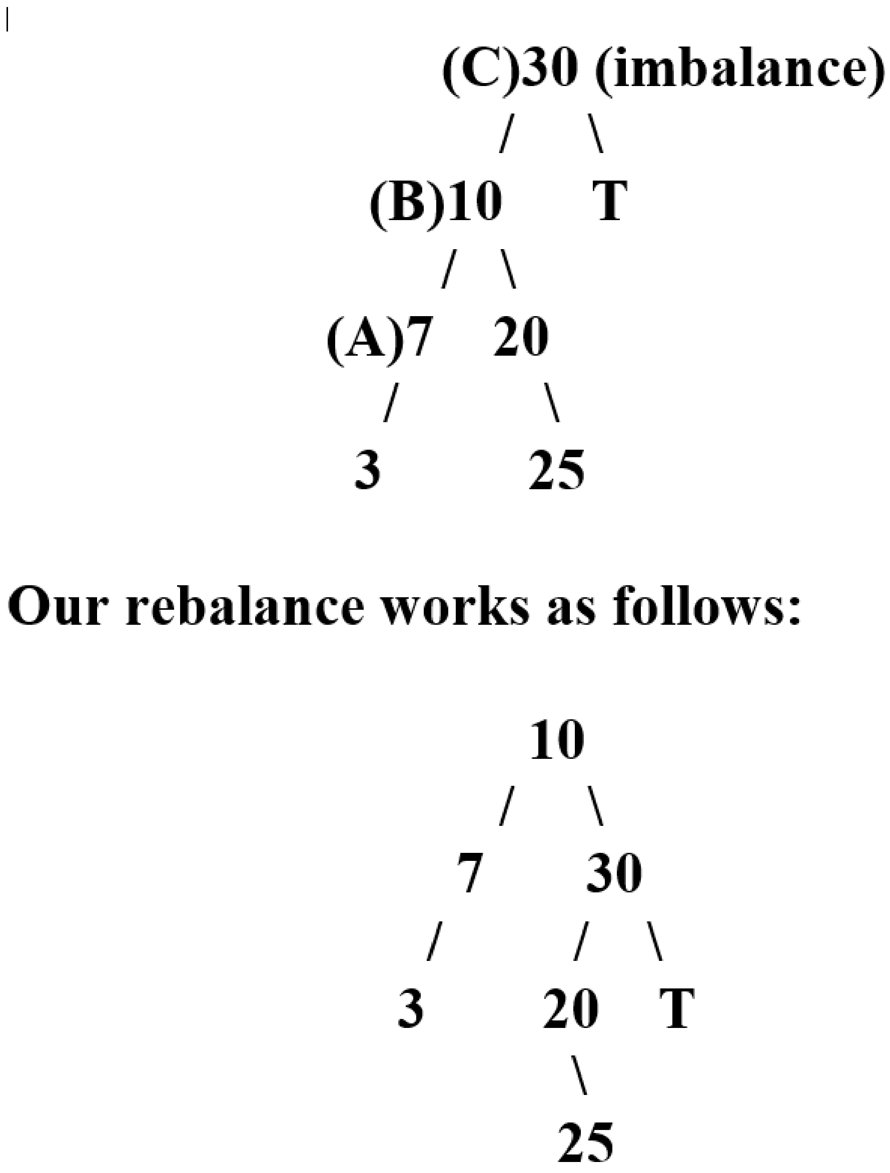
- If we have the following situation:



- **Because 10 is the left child of 30, we must choose the left child of 10, 7 to comprise our three nodes:**

- **Because 10 is the left child of 30, we must choose the left child of 10, 7 to comprise our three nodes:**

If you remember- here,
T0: 3,
T1: NULL,
T3: 20-25,
T4: T



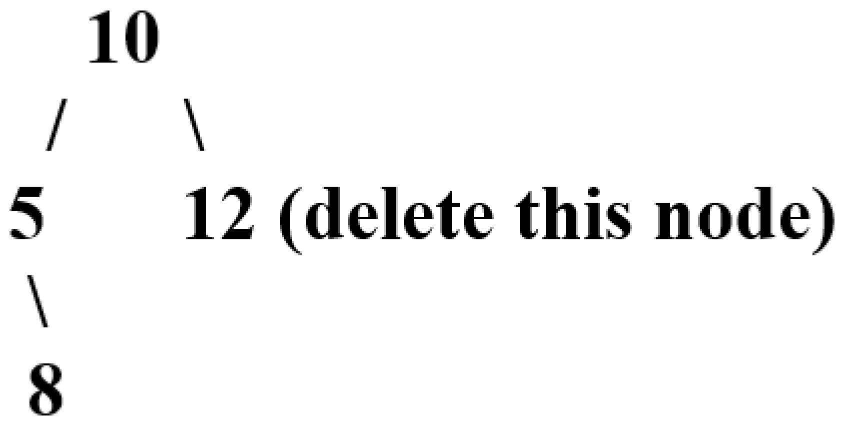
```
        (C)30 (imbalance)
            /    \
     (B)10       T
         /   \
   (A)7    20
     /          \
    3           25
```

Our rebalance works as follows:

```
            10
           /   \
         7       30
        /       /   \
      3       20    T
                \
                 25
```

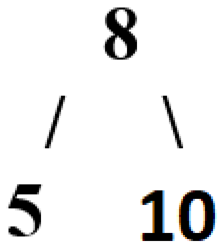
Example 3. Simple Example

The most simple example is formed when a node from a tree with four nodes gets deleted. In this example, consider the value 12 getting deleted:



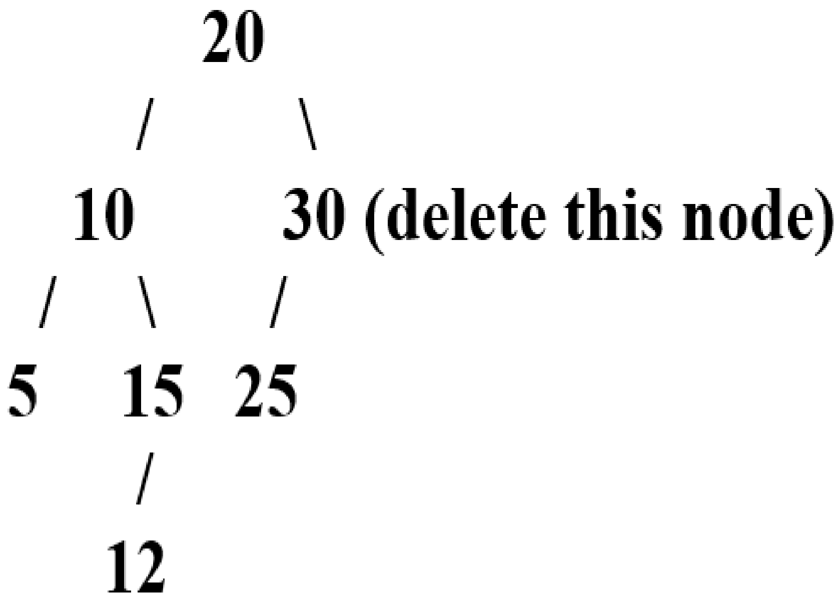
Who is A, B , C here? A: 5, B:8, C: 10

In this instance, after the node storing 12 is deleted, we move up to the parent, 10. From here, it’s fairly obvious that our nodes A, B and C will be the numbers 5, 8 and 10, respectively. Our resulting tree is:

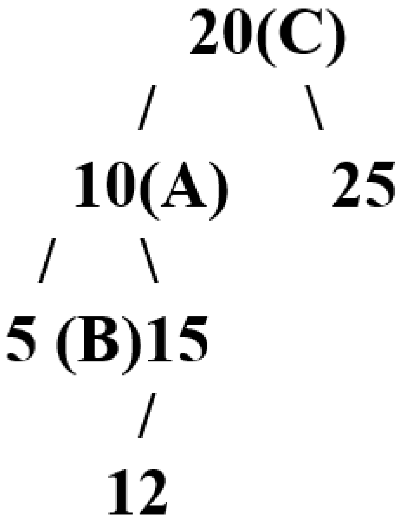


Example 4

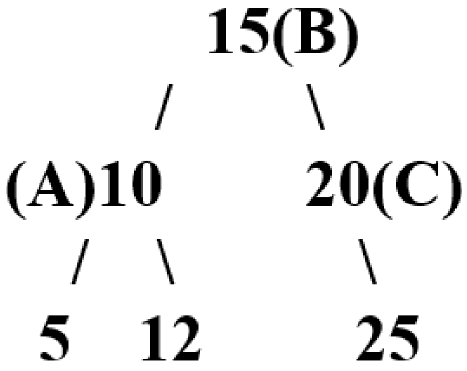
- **Consider deleting 30 from the tree:**
- **Using the rules for a regular binary search tree delete,**
  - **we would make the 25 the right child of 20. The 25 is balanced, so then we trace up to the 20. This is unbalanced.**
- **To determine A, B and C, we know that our first choice must be to go left from 20.**
- **Thus, two of the three values are 20 and 10**
- **Then to choose between 5 and 15, we choose 15 because that subtree is strictly TALLER than the subtree rooted at 5:**



Example4 continue

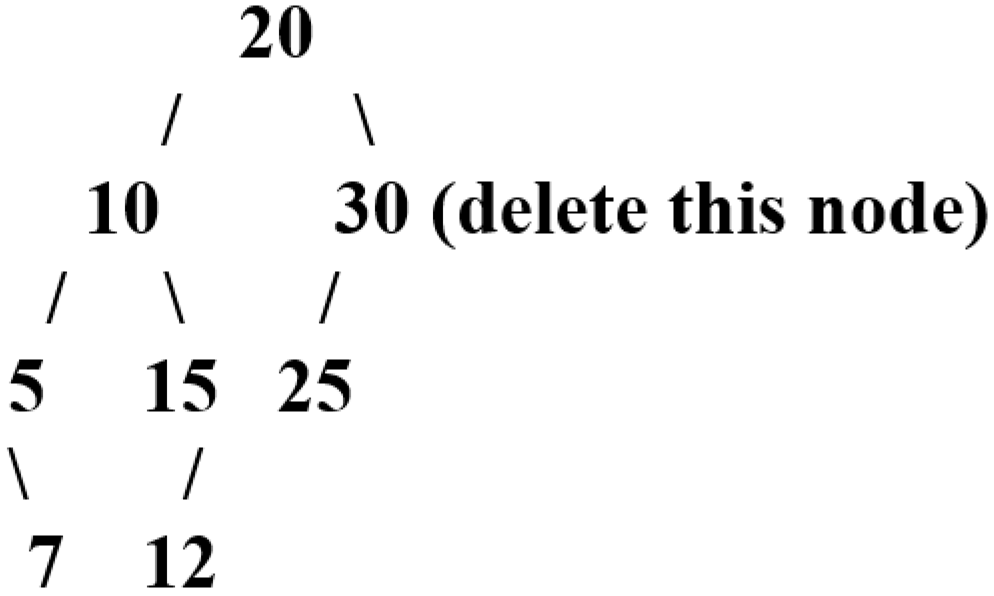


The restructure is as follows:



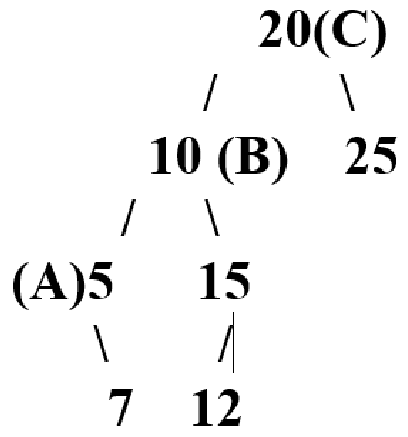
Example 5

- **Now consider deleting the value 30 from the following slightly different tree:**
- **This situation, what has changed is that both 5 and 15 have subtrees of the exact same height.**
- **In this case, we must choose the direction of the first node to the second. Since 10 is the left child of 20, we must choose 5, the left child of 10 instead of 15.**
- **The resulting labels are as follows in the next slide**



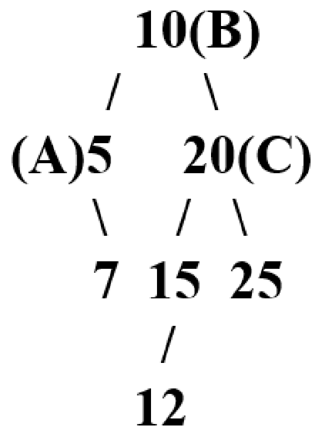
Example 5 continue

Remember about T0, T1, T2, T3



```
            20(C)
           /     \
      10 (B)    25
      /    \
  (A)5      15
     \      /
      7   12
```

The resulting tree is:

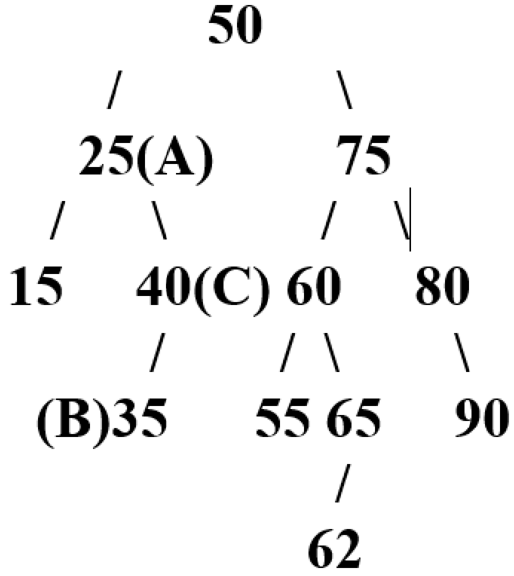


```
           10(B)
          /     \
      (A)5     20(C)
          \    /  \
           7  15  25
              /
            12
```

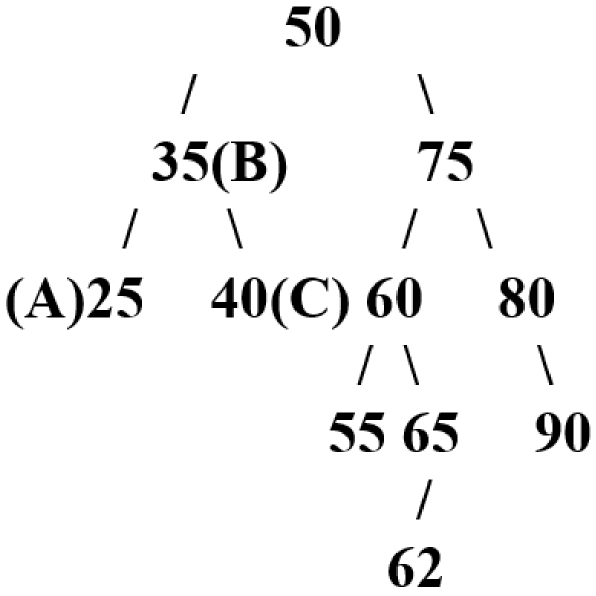
Example 6

- Consider deleting 15 from the tree:

- **First, we can see that when we get to 25, we will have an imbalance.**



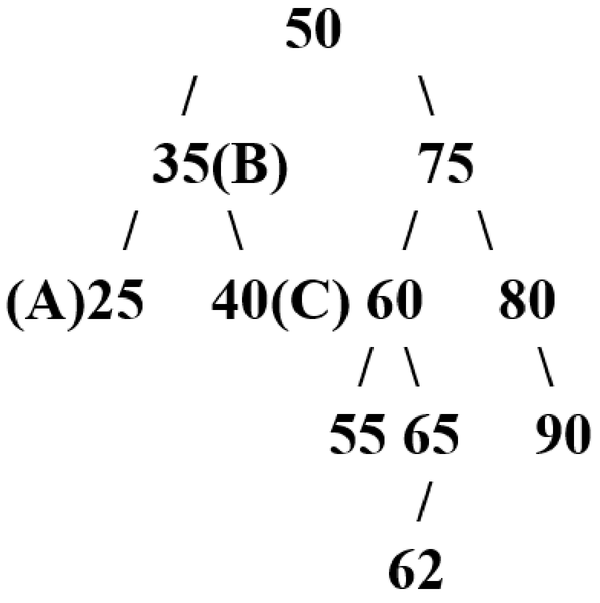
- **This situation is similar to the first case (example 3) we saw, so we label A = 25, B = 35 and C = 40. Following this restructuring, we have:**

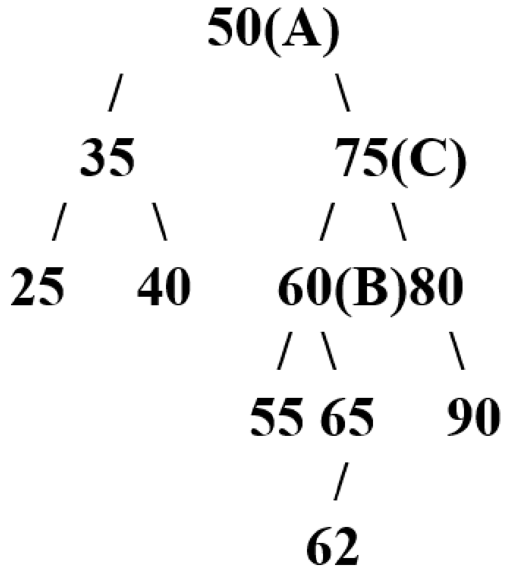


Example 6 continue

- **The problem, of course, is that now, 35 is balanced, but as we continue up the tree to 50, we've introduced a problem here, since we made the left of 50 shorter by one level.**

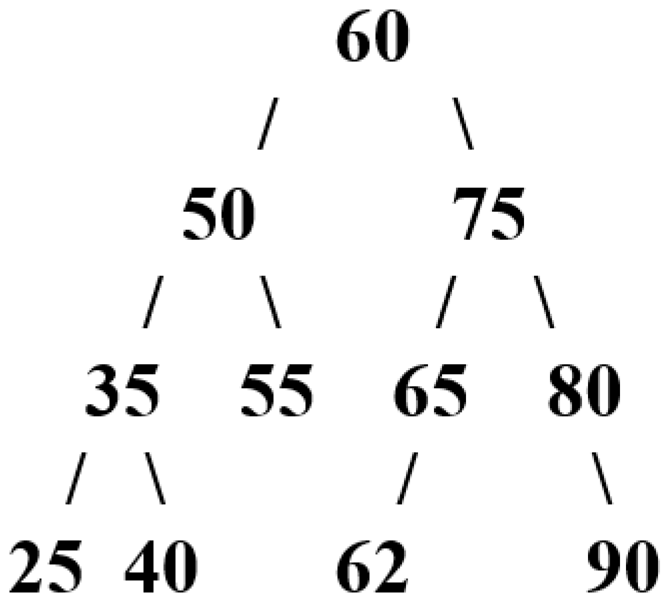
- **In this case, we will choose 50 and 75 to be two of our three nodes.**
- **To choose the third, we see that the tree rooted at 60 is strictly taller than the one rooted at 80. Thus, we have that A = 50, B = 60 and C = 75:**





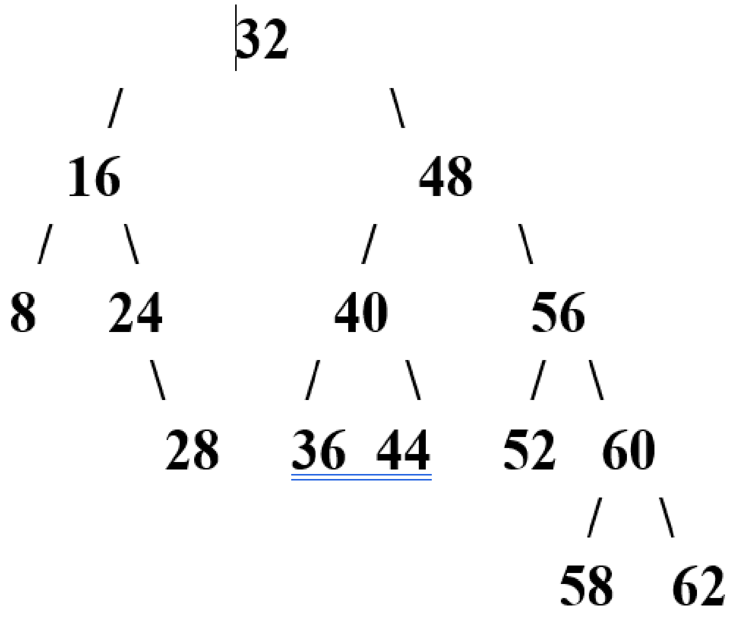
Example 6 continue

- **Finally, to do the restructuring, we’ll have 60 be the new root of the tree, 50 to the left and 75 to the right:**

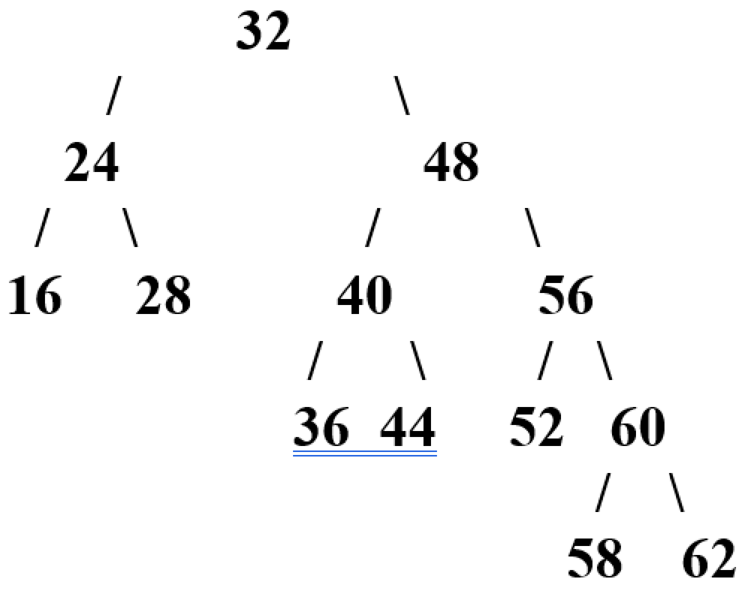


Example 7

For this example, we will delete the node storing 8 from the AVL tree.



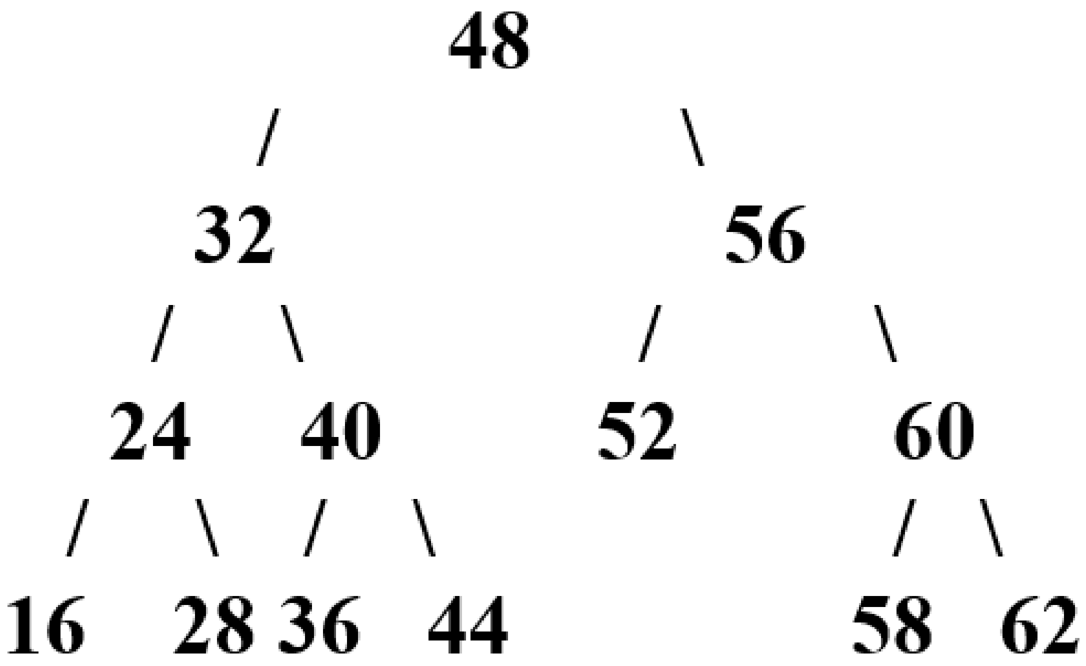
- **After deleting, We must first rebalance the node storing 16, resulting in:**



Example 7 continue

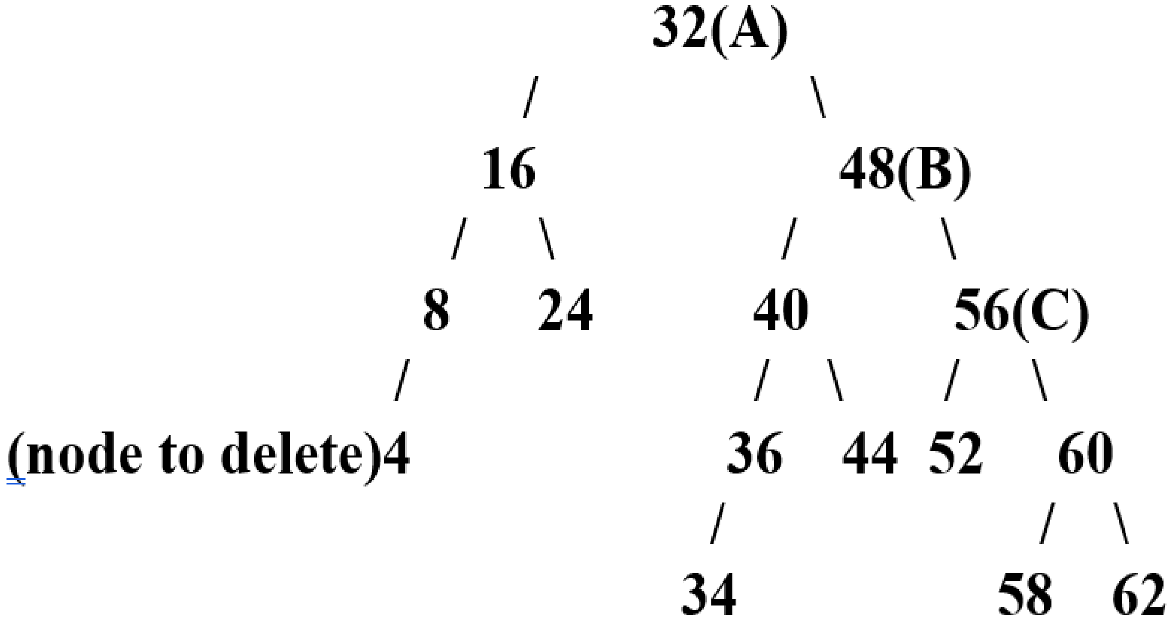
- Next, we march up to the parent of the node storing 24, the node storing 32. Once again, just like the previous example, this node is imbalanced. The reason for this is that the restructuring of the node with a 16 reduced the height of that subtree. By doing so, there was an INCREASE in the difference of height between the subtrees of the old parent of the node storing 16. This increase could propagate an imbalance in the AVL tree.

- When we restructure at the node storing the 32, we identify the node storing the 56 as the tallest grandchild.
- **Following the steps we've done previously, we get the final tree as follows:**



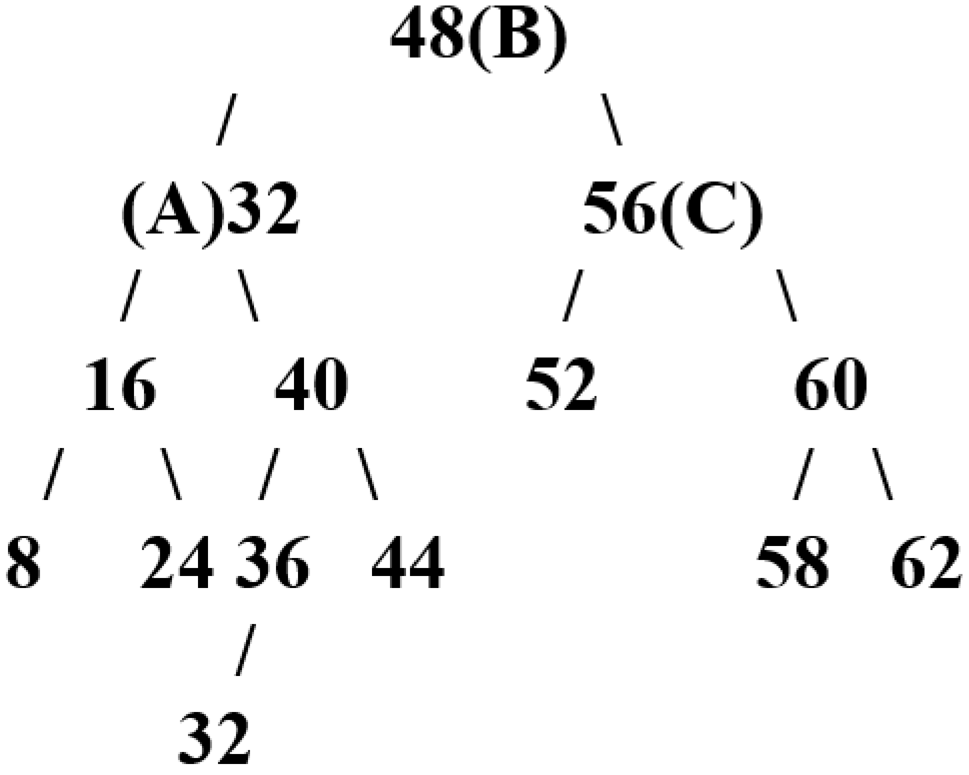
Example 8

- **In the final example, we will delete the node storing 4 from the AVL tree.**
- **When we call rebalance on the node storing an 8, (the parent of the deleted node), we do NOT find an imbalance at an ancestral node until we get to the root node of the tree. Here we identify both 32 and 48, like before as two of our three nodes. Now, when we decide between 40 and 56, we find the height of both subtrees to be the same. In this case, since 48 is the right child of 32, we must pick the right child of 48, which is 56, to be the third node. Thus A = 32, B = 48 and C = 56.**



Example 8 continue.

- **Accordingly, we restructure as follows:**



Reference

- This examples where taken from Prof. Arup's note:
- http://www.cs.ucf.edu/~dmarino/ucf/transparency/cop3502/lec/AVLTrees-Del-2.doc

- There were some typo in the note, and some of them were corrected while preparing the slide