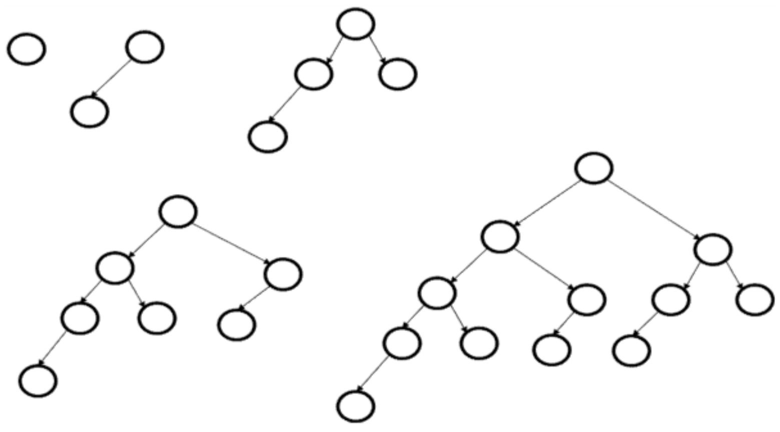
# Section 4 SOLUTION: AVL Trees & B-Trees

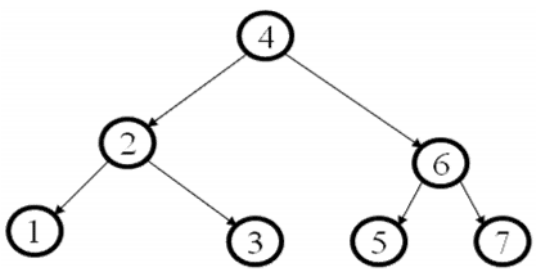
1. **What 3 properties must an AVL tree have?**
   a. Be a binary tree
   b. Have Binary Search Tree ordering property (left children < parent, right children > parent)
   c. Be a balanced tree: | n.left.height – n.right.height | ≤ 1 for all nodes n in the tree.

2. **Draw an AVL tree of height 4 that contains the *minimum* possible number of nodes.**

   Construct a minimum size AVL tree of height h by creating a new root, and making one of its children a minimum AVL tree of height h-1, and the other a minimum AVL tree of h-2. So to get a minimum AVL tree of height 4, we need to build up minimum AVL trees of heights 0-3 first. The image below shows each of these, and finally a minimum AVL tree of height 4. Values are left out here, but any valid BST values could be filled in. Note that there are many different possible orderings of branches (left versus right); the ones shown below are one way to do it. The minimum number of nodes is 12.

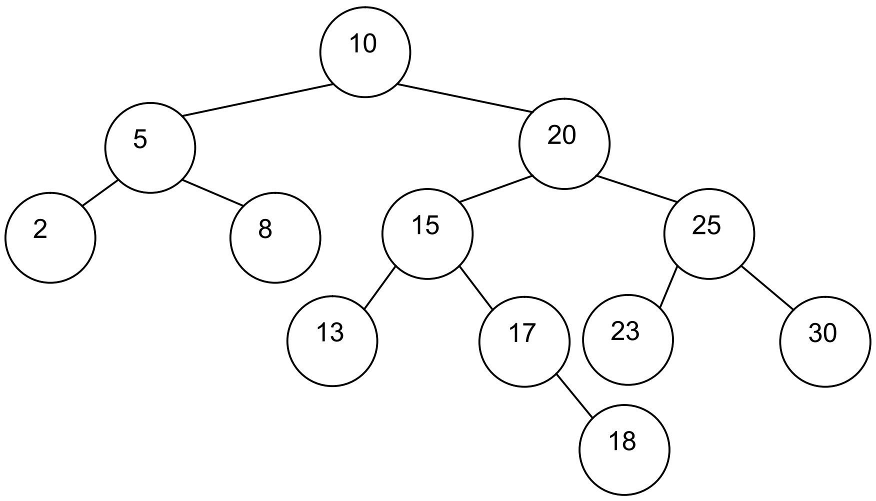


3. **In a typical BST, inserting keys in order result in a worst-case height. Show the result when an initially empty AVL tree has keys 1 through 7 inserted in order.**

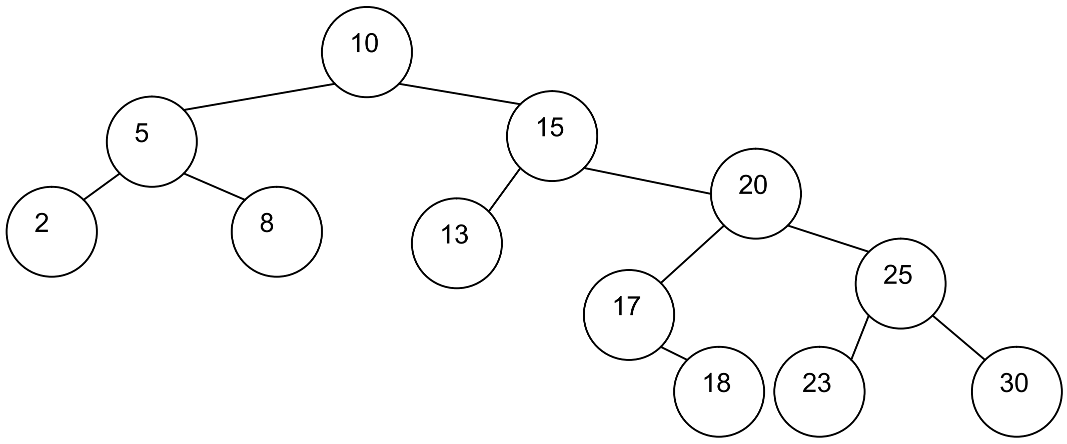


4. **Insert 18 into the following AVL tree. What type of imbalance does it cause? Show the result after balancing**.
   a. Inserting any value greater than 20 causes a right-right imbalance at node 10, the root, because the root's left child now has height 1 and the right child has height 3.
   b. Inserting any value LESS than 20 causes a right-left imbalance also at the root, for the same reason.

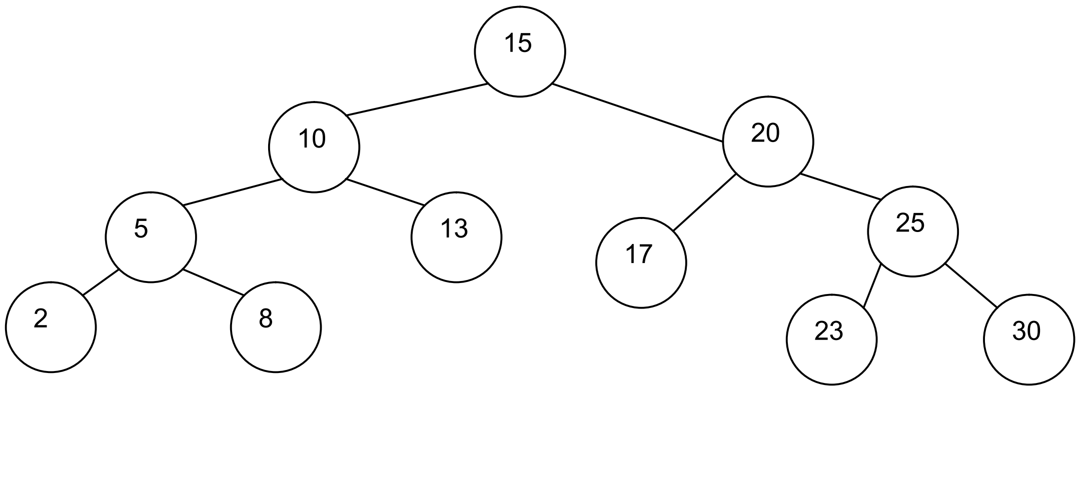
c. Inserting 18 causes a right-left imbalance at the root node 10, because its left child is of height 1 and its right child is of height 3, meaning the difference is 2, which is greater than the allowed difference of 1.



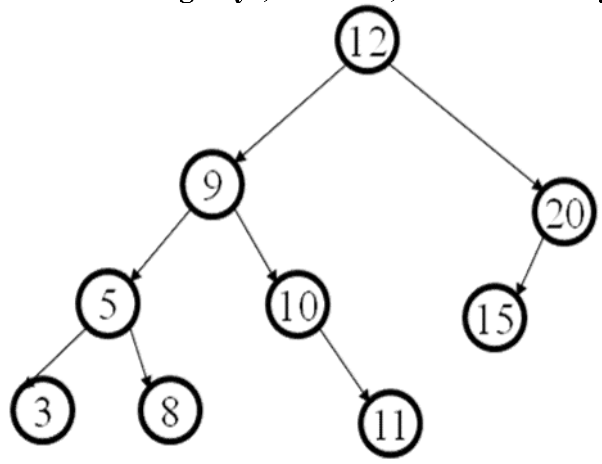
To fix this node, you need to do a double rotation. First, rotate grandchild of 10 with child of 10, that is, rotate 15 with 20 clockwise, making sure to re-assign 13, subtree starting at 17, and subtree starting at 25 correctly.



Now do your second rotation, rotate the node counterclockwise where the imbalance was found (node 10) with its imbalance inducing child 15.



5. **AVL tree balance violation cases:**
   a. **Insert the following keys, in order, into an initially empty AVL tree**: 12, 8, 9, 20, 10, 15, 3, 11, 5.



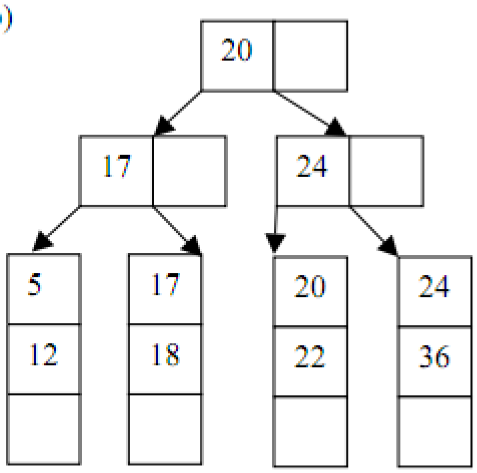
   b. **Find a key we could insert into your resulting tree that would result in a case 1 balance violation (left-left).**

      There are several keys we could insert to get a case 1 rotation; inserting '1', for instance, will cause a height imbalance to be detected at the root.

6. **Given a binary search tree, describe how you could convert it into an AVL tree with worst-case time O(nlogn) and best case O(n).**
   We can traverse the tree in O(n) time and insert each element into an initially empty AVL tree; this will take O(nlogn) time overall. To get O(n) 'best-case' performance we can do something that's a bit of a hack: try to verify that the BST is a valid AVL tree, which we can do in O(n) time; if it is, return it as it is. Otherwise, create a new AVL tree as described. It's questionable as to whether this is really a 'best-case' result, as we don't actually do any conversion, only verification.

7. B-Tree
   a. Each node can have at most M children, and must have a minimum of M/2 children, and each leaf can have at most L data items, and at least L/2 data items. (rounding-up) So, a tree with M=32 and L=16 must have 16-32 children at each internal node, and must have 8-16 items at each leaf. (excepting the first 7 insertions

      b)



   b.

c.

| |
|---|
| 18 |
| 20 |
| 24 |