AVL Tree

What is an AVL Tree?

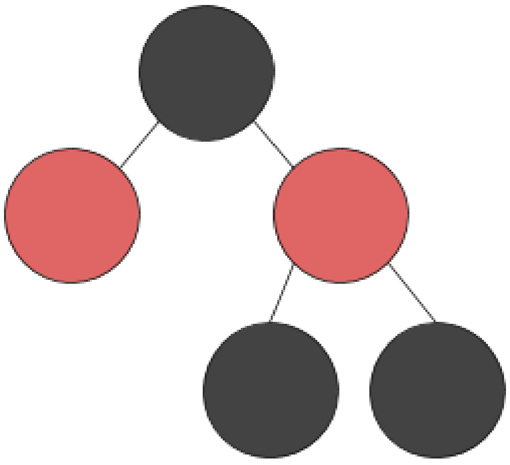
An AVL Tree is one of the types of the Binary Search Trees (BST) also known as red-black tree. It’s specific difference between other trees that it’s a self-balancing tree. This type of trees is much more efficient and faster when we talk about basic operations like search, insert, delete etc…

When it balance itself?

It starts to balance itself whenever it has two red nodes in a row in it’s right or left child nodes or when the root node of the tree is red. It can has an unlimited amount of black nodes in a row and only one red node between them. Each newly inserted node always has red color. Rotations and color changes of nodes are used to equilibrate an AVL tree.

When rotate or change color of nodes?

This is a correct tree.



There are several rules that exists to know when to swop or to rotate the nodes of an AVL tree.

To simplify the understanding of how balancing works let’s assume the next:

- **x** marked as a new node that is being inserted
- **p** the father of an x
- **pp** the grandfather of an x
- **f** the uncle of an x

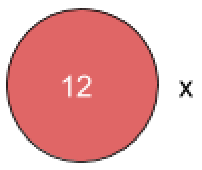
With these information we can apply the next rules:

CASE	CONDITION	WHAT TO DO
Case 1	**p** is the root node	**p** node becomes black
Case 2	**f** is red node	**p** and **f** nodes become black **pp** node becomes red
Case 3a	**f** is black node **p** node is left child **x** node is left child	rotate nodes **p** and **pp** to the right **p** becomes black **pp** becomes red
Case 3b	**f** is black node **p** node is left child **x** node is right child	rotate nodes **x** and **p** to the left rotate nodes **x** and **pp** to the right **x** becomes black **pp** becomes red
Case 3c	**f** is black node **p** node is right child **x** node is right child	rotate nodes **p** and **pp** to the left **p** becomes black **pp** becomes red
Case 3d	**f** is black node **p** node is right child **x** node is left child	rotate nodes **x** and **p** to the right **x** becomes black **pp** becomes red

In order to use this rules, let's build an another AVL tree and balance it. To not get confused, the nodes will be numbered. The tree should have the following nodes: 12, 3, 2, 5, 4, 7, 9, 10.

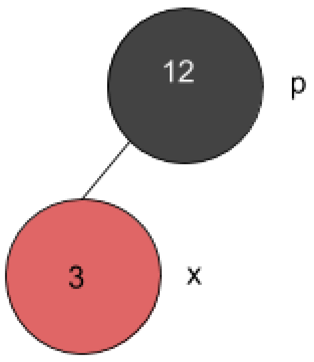
Step 1: Case 1

Placing the node 12 as a root node. Since the root node can not be red, let's change it's color to black and add a new node.



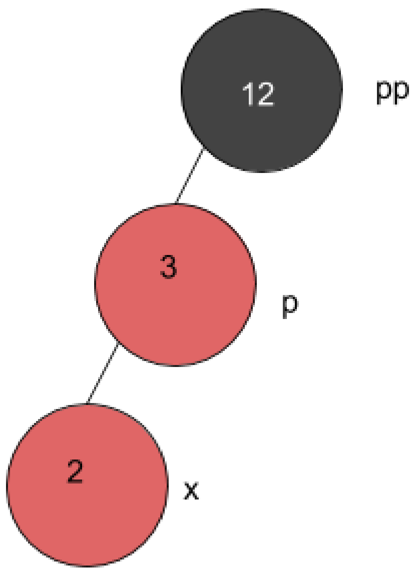
Step 2: correct tree

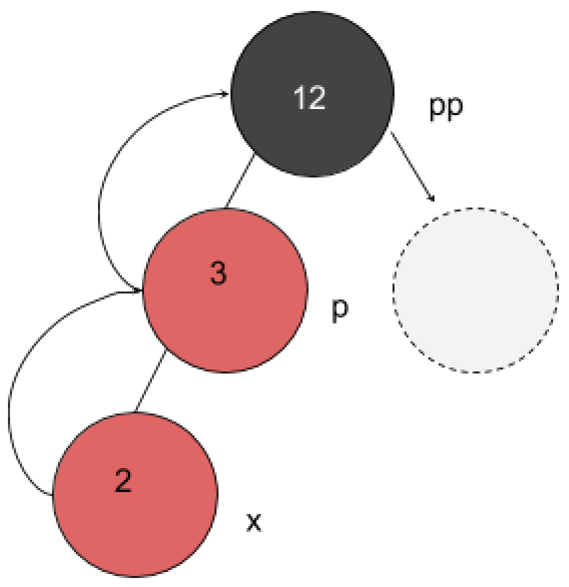
In this case an AVL tree doesn't need any changes.



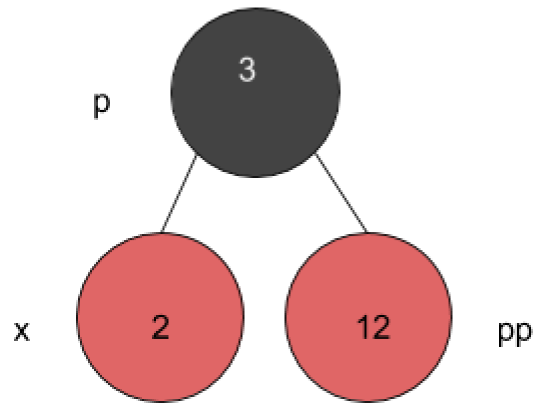
Step 3: Case 3a

After inserting node 2, an AVL tree got two red nodes in a row. Let's fix it.



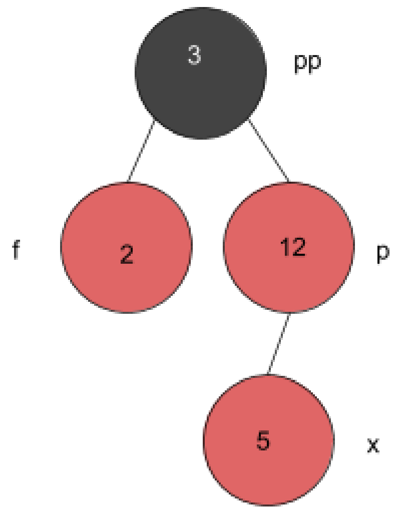


After rotation a correct built tree was obtained.

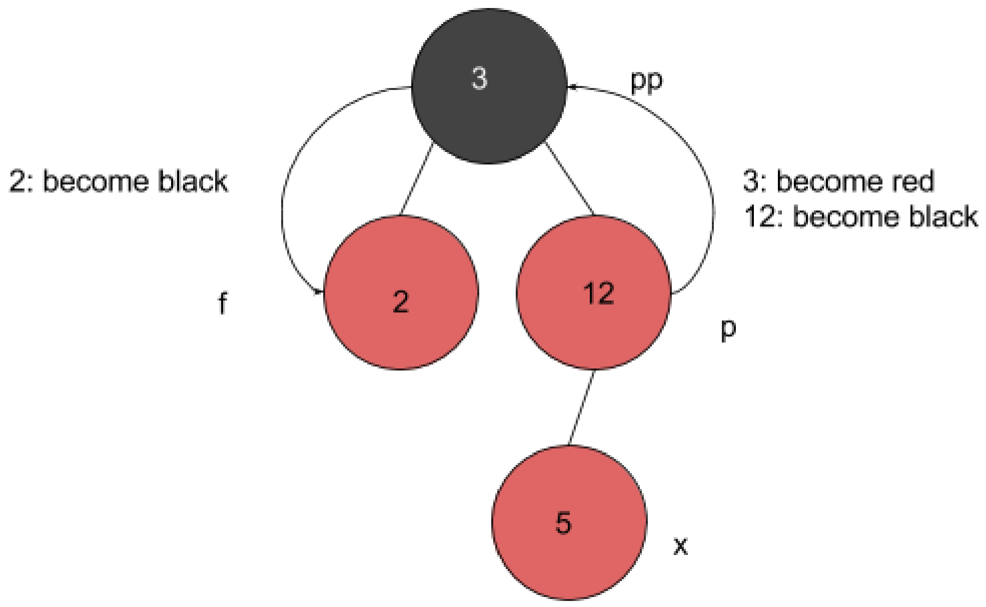


Step 4: Case 2 and Case 1

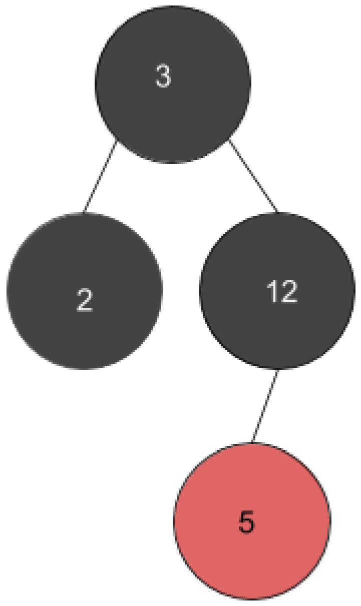
Let’s insert the next node, which is 5:



Again, an AVL tree got two red nodes in a row, so to fix it, Case 2 should be used:

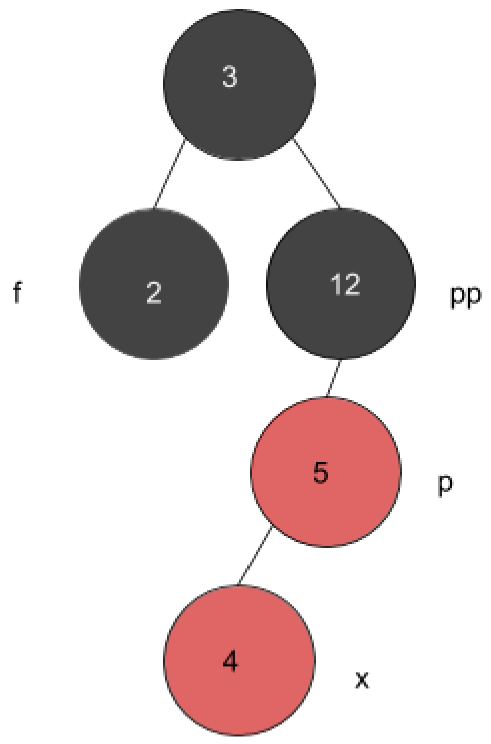


After using Case 2, the node 3 became red, so it has been changed to black by using Case 1, it allowed us to get the following tree:

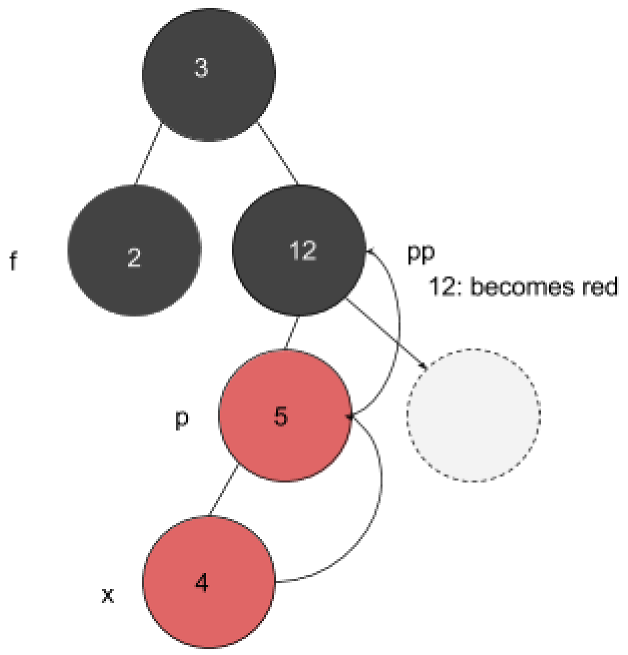


Since two black nodes or more in a row are allowed this tree is considered correct. Let's add next node.

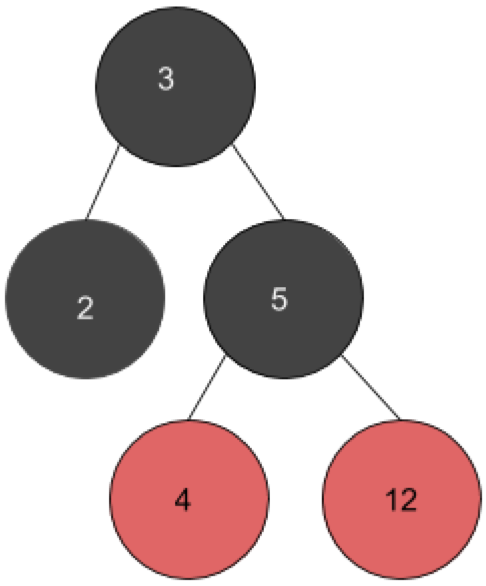
Step 5: Case 3a



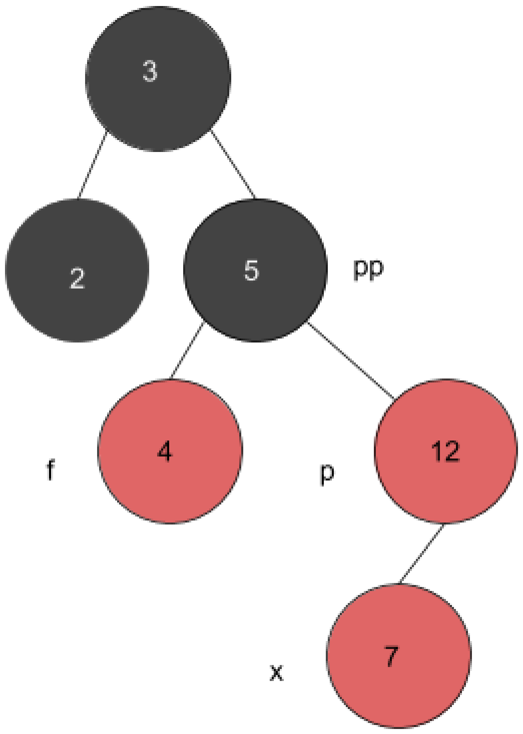
The node **x** is red as well as his parent node **p**. Rotation using Case 3a is needed:



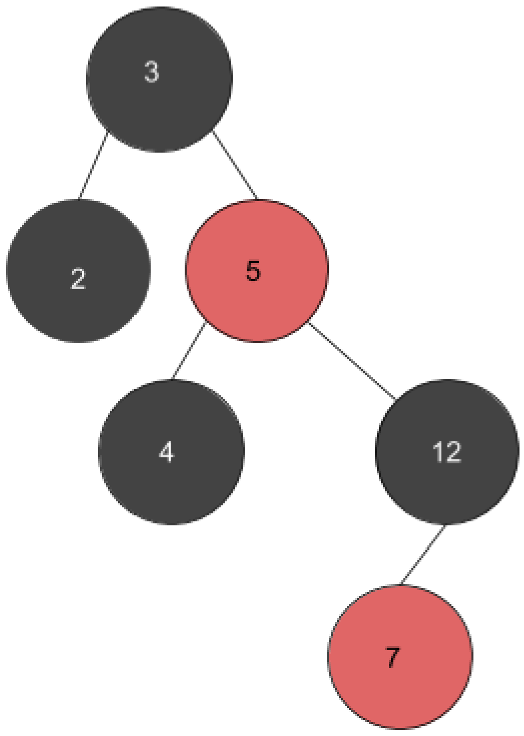
We get the following tree:



Step 6: Case 2

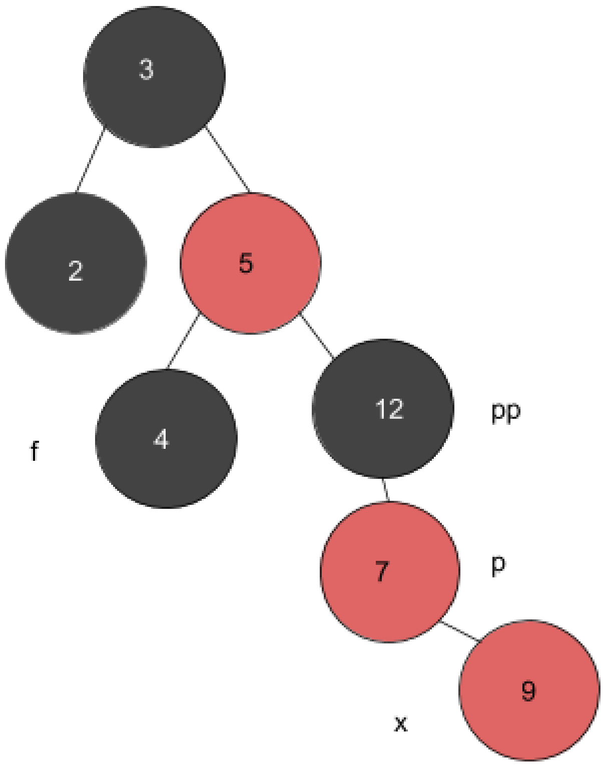


We have to apply Case 2 and change the colors of **p, pp** and **f** nodes because the **f** node is red. We get the following tree:

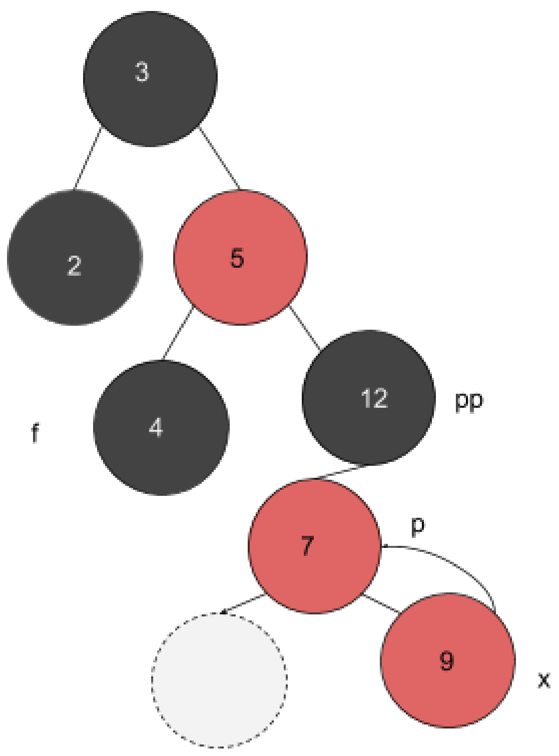


Step 7: Case 3b

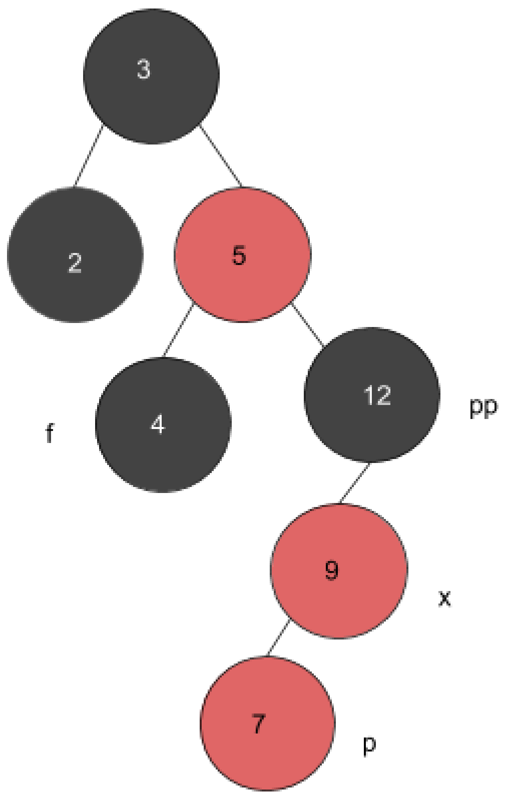
In this case double rotation should be used. Rotation to the left of nodes **x** and **p** and then rotation to the right of **x** and **pp** nodes. Also the colors of nodes **p** and **pp** will be changed.



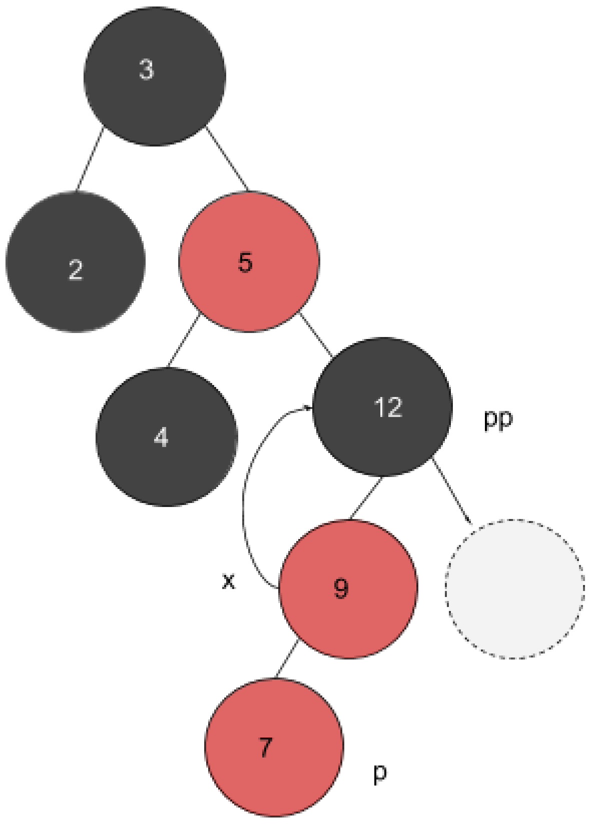
First rotation between **x** and **p** nodes:



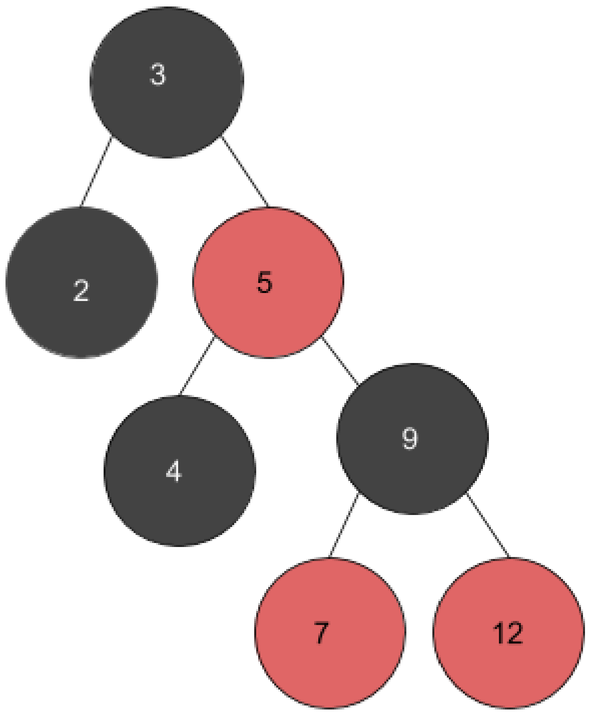
So we get the following tree:



Second rotation between **x** and **pp** nodes:

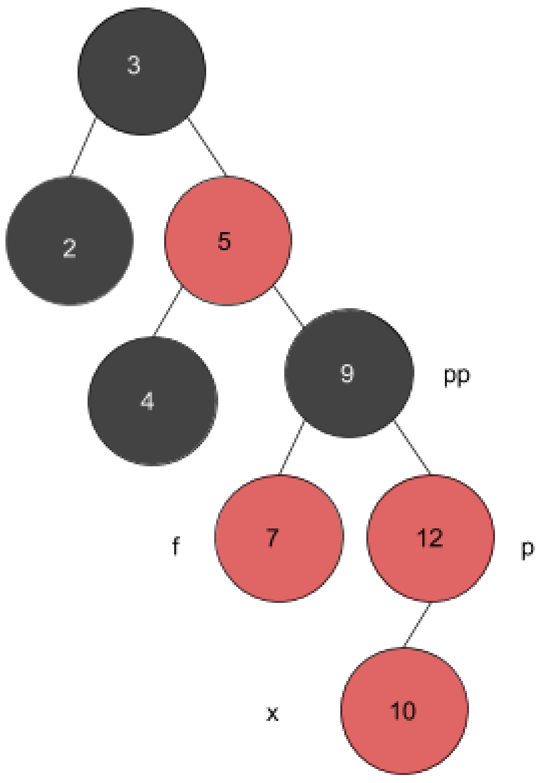


Finally, the correct AVL tree is obtained:

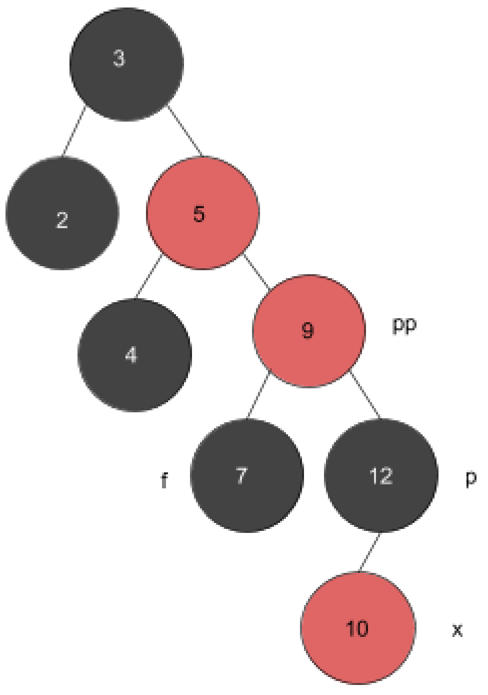


Step 8: Case 2 and Case 3c and Case 1

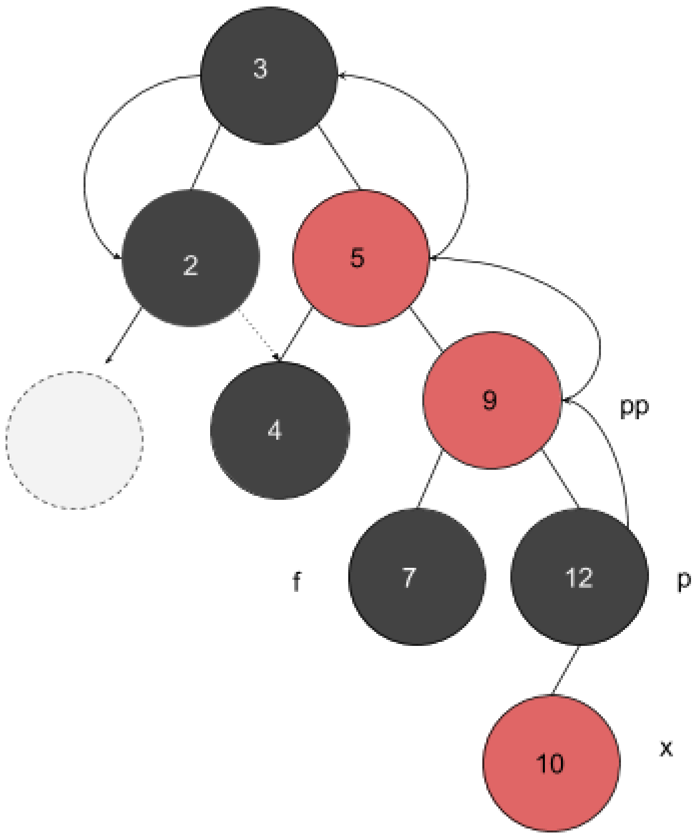
Firstly, the Case 2 should be used because the node **f** is red. So the nodes p and f become black and the node pp become red.



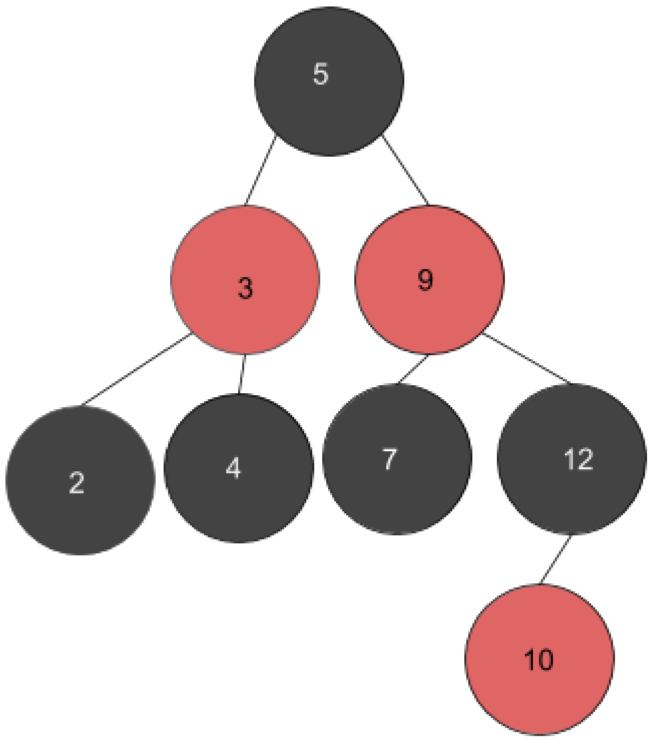
After applying Case 2, we get the following tree:



Now Case 3c should be applied. Rotation of the nodes **p** and **pp** to the left. The node **p** become black and the node **pp** become red.



Since after Case 3c, the root node becomes red, we apply Case 1 to change its color to black. Now we’ve got a proper AVL tree:


5
3
9
2
4
7
12
10