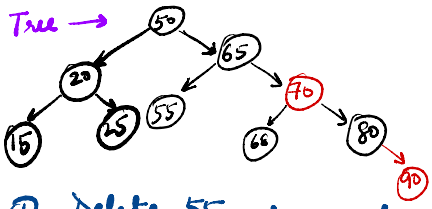
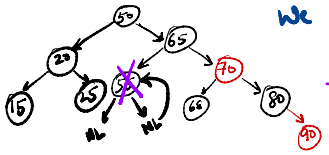
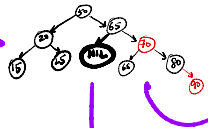
Deletion Example

Tree →



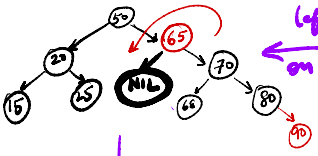
① Delete 55 :→ 55 is a black node. Removing it will alter the red-black tree properties. So we need to correct them. It does not have any children. The thing that replaces it is its child which is NIL which black. We replace 55 by the sentinel NIL.

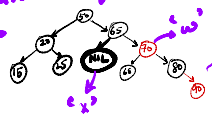




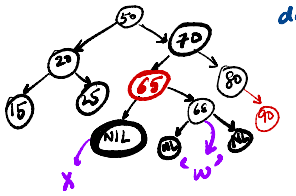
the NIL node here is 'x' and it is double black since we add another black count to an already black node. — this way the property that all tree paths from root to leaf have same number of black nodes.

left rotation on x.p



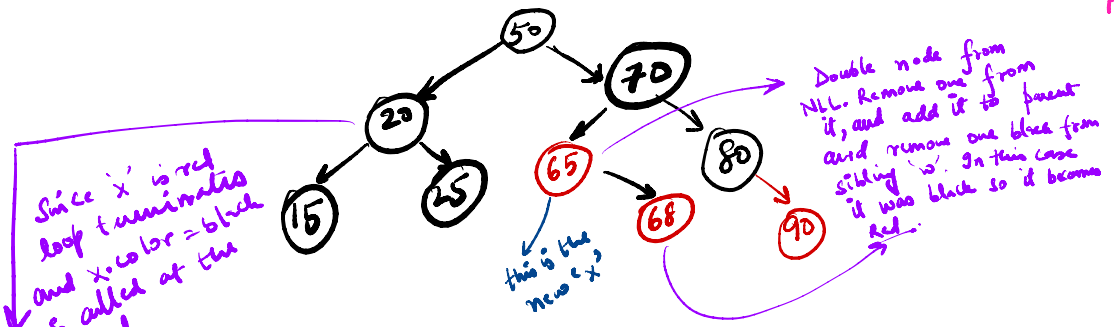


the sibling 'x' here is 'w' which is red. So this is the first case. here we make x's parent red, 'w' to black and then left rotate on x's parent



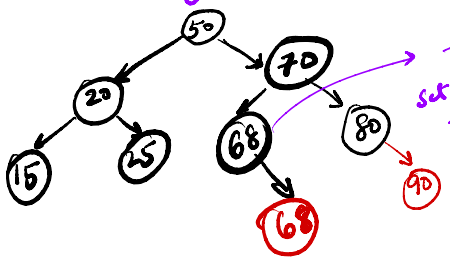
Now 'x' is still on the double black 'NIL' node and 66 is the new sibling. Now 66 is a leaf node. Its children are NIL nodes which by default are black nodes.

So right now, 'x' is a left child with a black sibling whose both children are black. This is case 2. So we remove the double black node on the x and make the sibling Red and transfer the double black node 'NIL' pointer to its parent x.p.



Since 'x' is red loop terminates and x.color = black is added at the end.

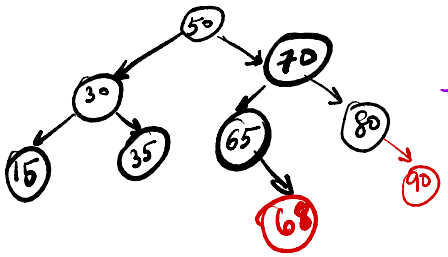
Double node from NIL. Remove one from it, and add it to parent and remove one black from sibling 'w'. In this case it was black so it becomes Red.



this gets set to black at the end.

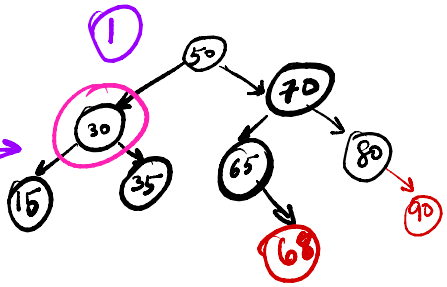
This is the tree at the end after deleting 55.

(2) Delete 30



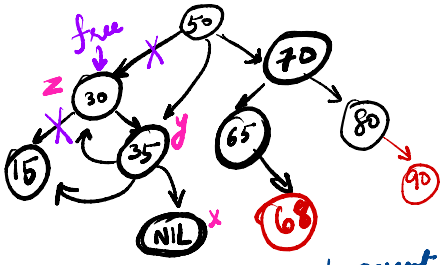
30 is deleted replaced by its inorder successor 35

(1)



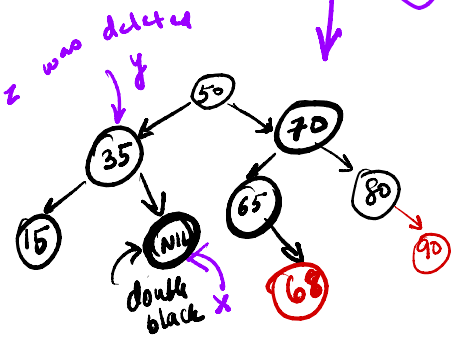
here, 30 has to be deleted. and has two children. it gets replaced by its inorder successor. (35)
So '30' is z, y is '35', and x is y.right which is null (black).
y's original color is black. so it will alter the RB tree (so it needs fixing). y gets the actual color (the new color set) as z's color which is again black but that doesn't matter. what matters is y's old color. which if black calls the fix-up routine.

(2)

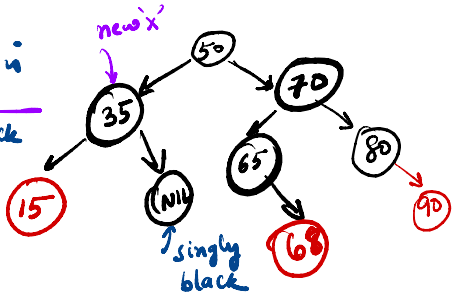


30's parent is now 35's parent.
30's left child is 35's left child.
free '30' ← Removal complete.
'x' is considered double black
y's original color was black so fixup will be called.

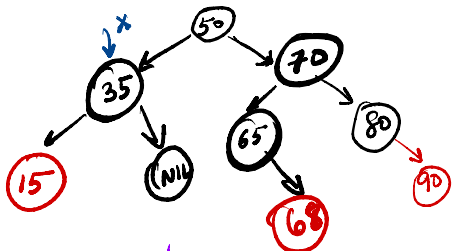
(3)



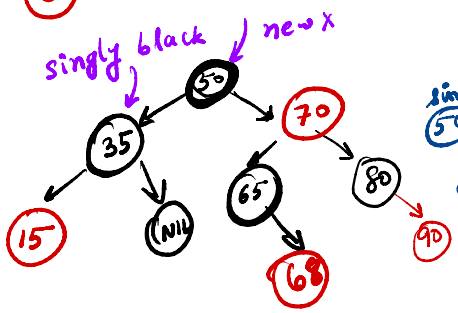
We check the sibling of x which is (15) and it is black. It has two children, both are (NIL) Sentinels which are considered black. So this is case 2. The nil node is considered double black. So we remove 1 black from both x and its sibling. x becomes singly black and its sibling becomes red. the new 'x' node for the next loop iteration is the parent (35)



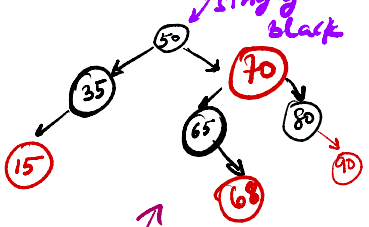
2nd loop iteration. x is at 35 and its black and it is not ROOT



again this is case 2
x's sibling (70) is black has two black children.
x is considered doubly black.
we remove 1 black from both x and its sibling. and set x as x.parent.

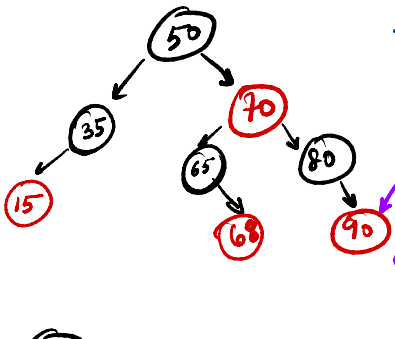


since the new x (50) is the root, the loop terminates. and at end (50)'s color is set to black (Singly black)

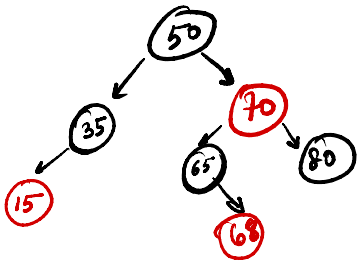


Tree at the end of deleting 30

(3) Delete (90)



here (90) is a leaf node, so it gets replaced by its child which is (NIL). (NIL) is black by default. since (90)'s original color is red, the tree is still a RED BLACK tree after deleting (90), so we return without any fixup.



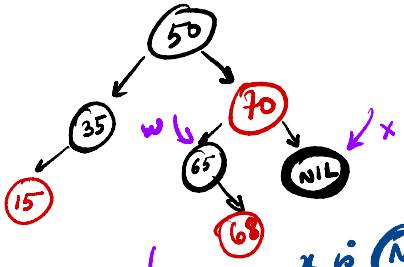
→ Tree after deleting (90)

(4) Delete (80)

x is at 80. It has no children. it is black so we have to fix R-B Tree after deleting it.

z is (80), y is (80), x is (NIL)

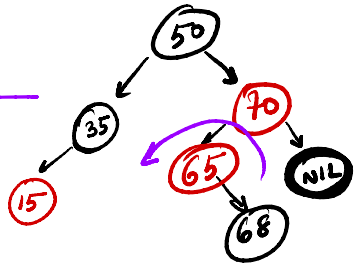
Replace y ((80)) with double black node (NIL).

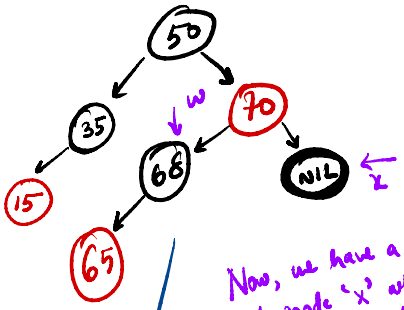


x is (NIL) its sibling is black. It has one Red child which is near child to 'x'.

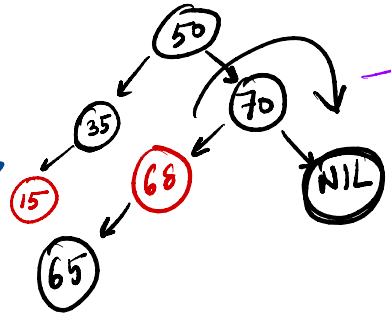
This is <u>case 3.</u> →

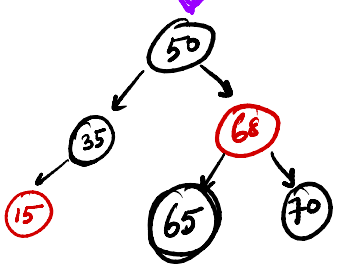
we change near child (68) color to black and left rotate on (65) after making (65) Red.



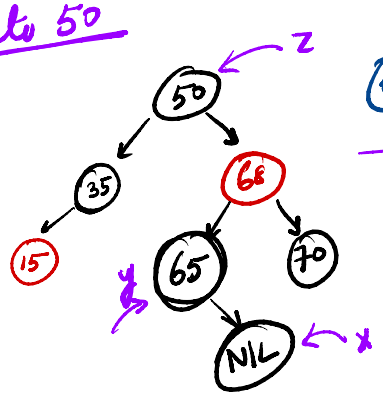


Now, we have a double black node 'x' whose sibling 'w' is black and the FAR child is red, this is <u>Case 4.</u> Here, 'w' takes color of parent (which is red) (68) → red. both parent & child of 'w' get black. tree is right rotated on (70) the parent.



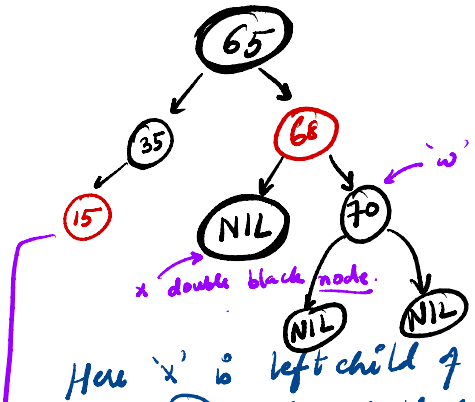


Delete 50

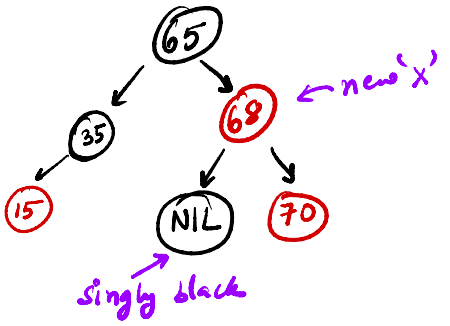


50 has two children, so get the inorder successor which is 65 which 'y'. 'x' is right child of 'y'.

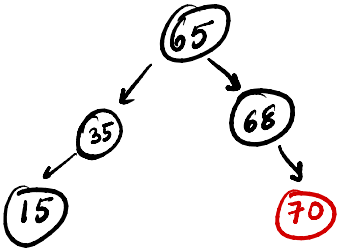
1. Here 65 gets replaced by its right child NIL
2. this NIL will be a double black node.
3. y's original color is black so we have to check treed validity.
4. 65 will replace 50 and its new color will be the same as 50 which is black



Here 'x' is left child of its parent. 'w' is 70 which is black with 2 black children. Case 2 : We remove 1 black count from 'x' and 'w' and make x.p black. set new x to x.p.

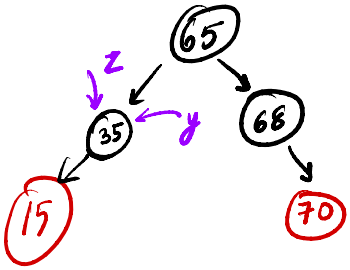


Since 'x' is red, loop terminates. At the end x gets set to black.

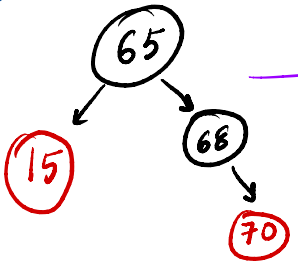


final tree

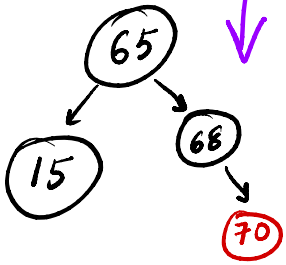
Delete 35



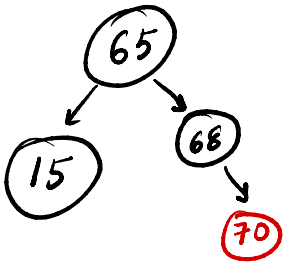
y is black, x is y → left. Replace 35 with 15



Since y's original color was black, we call fixup. the loop does not run since 'x' is not 'black' at the end it gets set to black.

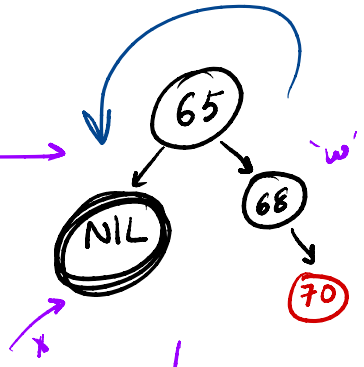


Delete 15



15 is black (y, z)
it has no child.
x is NIL

Replace 15 with double black node NIL and call fixup.



'w' is black 68
70 is a RED child farthest from 'x'
this is case 3.

'w' gets parent's color
parent and child get black
and left rotate about the parent 65