Keep black height

Insert

Text

Description automatically generated

There are three cases by which the insertion might violate the properties of an RBT.

Diagram, shape

Description automatically generated

As explained in the slide, black height, that is the number of black nodes following the path from a node to a leaf, stays the same since pushing the colors down does not affect bh’s, expect of g’s, which is incremented by 1 since its children became black.

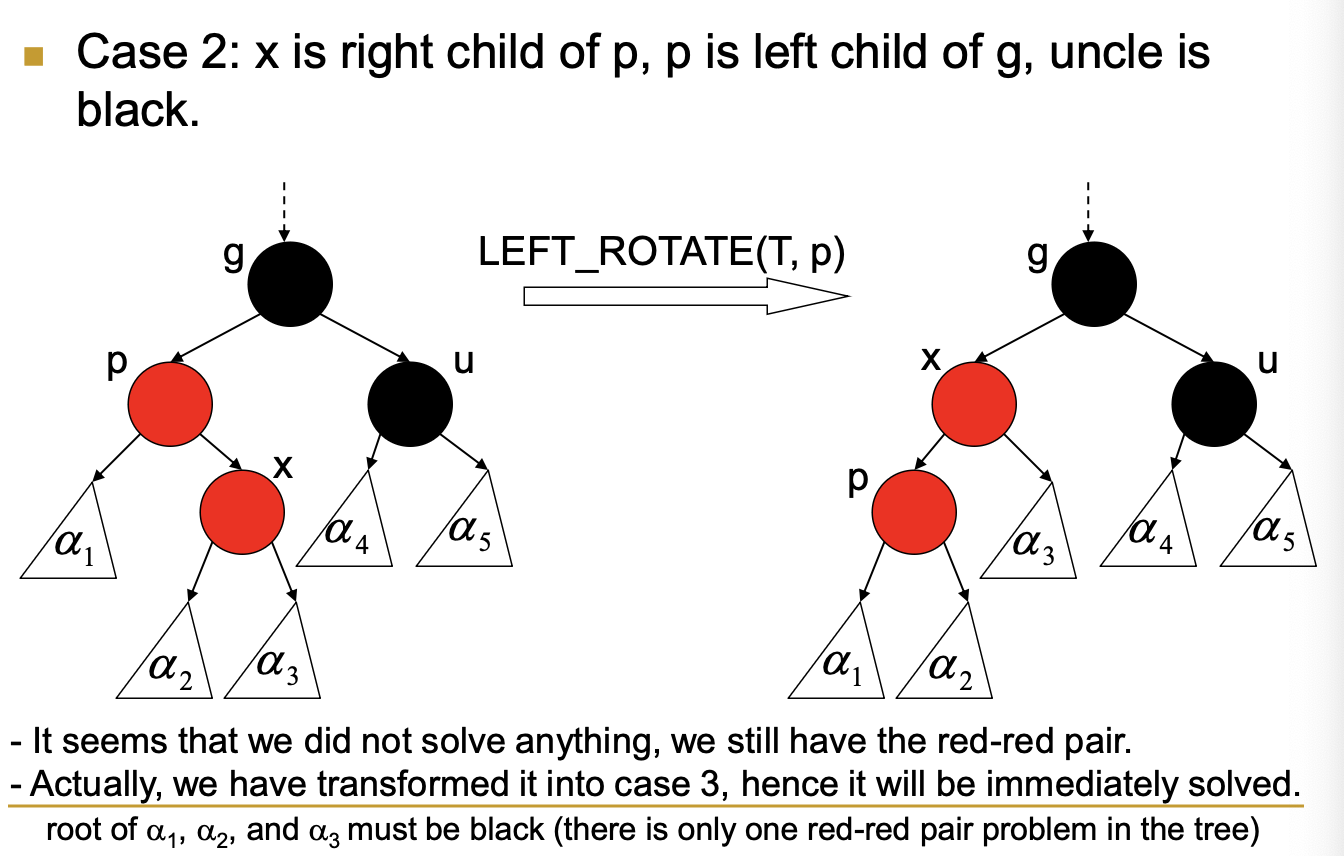
After recoloring, however, the problem (red-red nodes) might continue to exist as g is now red, and it might have a red parent and uncle. In that case, we repeat the same steps until reaching the root. If the root becomes red because of recoloring, then it is colored to black at the end.

As explained above, we can observe that in the worst-case every node’s black height above (and including) g will be incremented by 1. Fortunately, we can achieve this without increased time complexity by simply incrementing the grandparent’s black height by one each time recoloring is applied. Thus, both grandparent’s children and g itself preserve bh property.

Diagram

Description automatically generated

If case 3 exists after insertion, one can observe that black height of any node above g (before rotation) remains the unchanged. Examining the nodes that have moved, let’s first check bh’s of them. Assume that bh(x) = n; then bh(p) = n, bh(g) = n, bh(u) = n-1, where n is an arbitrary number. Considering these values, after the rotation we can see that bh(x) is the same since it has the same children. Also, bh(g) is the same since it has u and a\_3 subtree as children, and u has the same children before the rotation as well, thus unchanged. Therefore, we can say there is no need to traverse the tree, or even change any values of bh.



In this case, same observations as above can be done. Assume that bh(x) = n; then bh(p) = n, bh(g) = n, bh(u) = n-1. After the rotation: bh(x) = n since p’s new child a\_2 was of x’s and x had bh(x) = n, meaning that bh(p) is unchanged. Also, bh(g) and bh(u) remains the same since bh’s of nodes underneath them is the same as explained. Thus, we don’t need to change the bh value of any node in this rotation. After that the problem is turned into case 3 (previous case), in which there is no need to change bh values as explained previously.

In other cases, in which a red node is inserted as a child of a black node, there is no violation of RBT properties that can occur and trivially black height of any of nodes in the tree is not effected.

In all these cases there is no need to traverse the tree or a subtree to fix bh values. Only change is that when case 1 applies, there a constant time operation [bh(g) += 1], and this is repeated for new grandparent of g if case 1 shows up because of recoloring. Yet since grandparent is recolored anyway, so this operation does not add to the worst-case time complexity of FixColoring, and it is still O(h).

Diagram, shape

Description automatically generated

Considering case 1, since there is no rotation is done, nodes’ depth is not changed, thus there is no need to traverse the tree to fix the depth values.

Diagram

Description automatically generated

In case 3, because of rotation some nodes’ depth value is changed. Assume that depth(x) = d; then depth(p) = d – 1, depth(g) = d – 2, depth(u) = d – 1. After rotation, depth(x) = d – 1, depth(p) = d – 2, depth(g) = d – 1, depth(u) = d. Moreover, depth of nodes in a\_1 and a\_2 is decreased by 1, nodes in a\_4 and a\_5 is increased by 1. Only nodes in a\_3 subtree keep their depth values. Because we need to update the values of each node within subtrees (except a\_3), and also we know that the tree is balanced, meaning that there cannot be considerable more amount of nodes in a\_3 than of other subtrees, we can say that there needs to be O(k) amount of computation, that is we need to traverse and fix depth values of each node within these subtrees (and x, p, g, u) where k is the amount of nodes below g (in the original state). Therefore, at worst-case this situation would occur when g is the root of the tree, and the number of nodes that is needed to be traversed and fixed would-be O(n) where n is the number of nodes in the tree.

Diagram, shape

Description automatically generated

In case 2, after the rotation depth values changed are of node p, x and subtrees a\_1, a\_3. Furthermore, in this state problem is turned into case 3, in which O(n) computation for traversing and fixing nodes’ depth values is required. Also, since a\_3 is also changed in this case (in only case 3 it was unchanged), simply all the nodes below and including g is required to be fixed. Therefore, at worst-case scenario g would be the root meaning that all the nodes in the tree is required to be fixed. Thus, at worst-case fixing adds O(n) time complexity to insertion.