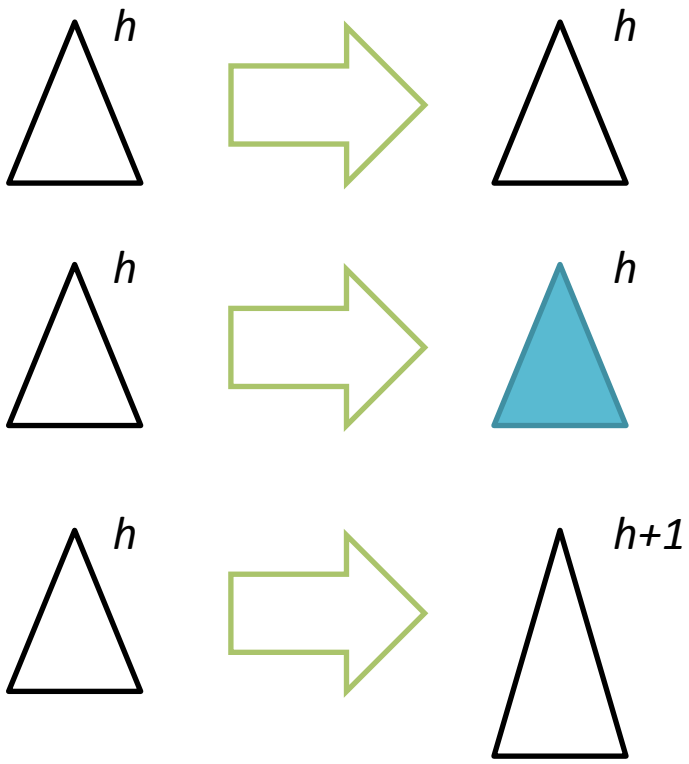
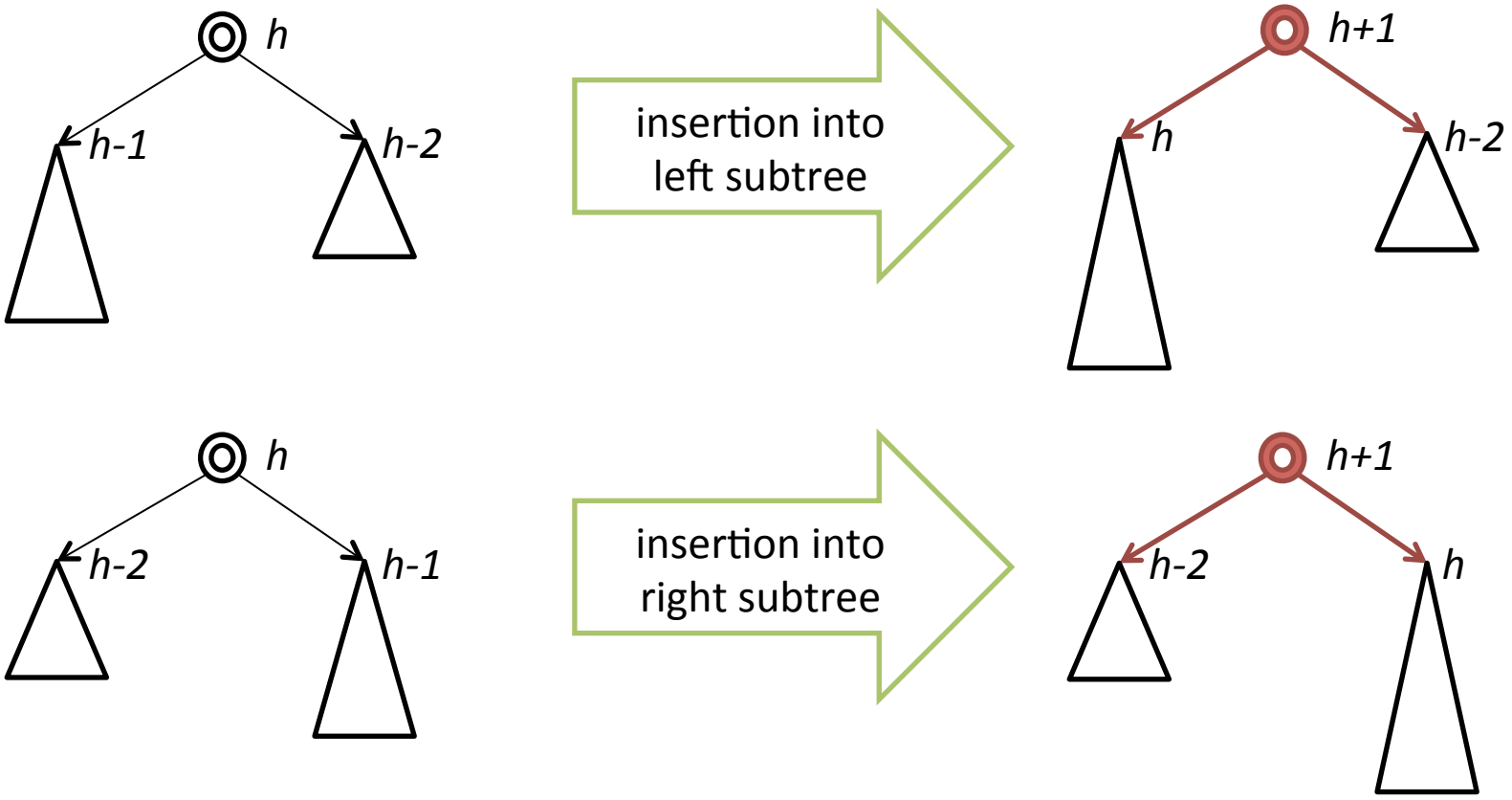
AVL insertion postcondition

When you insert into an AVL tree, *either*

- you get a new tree with the same height (which may have had rotations performed), *or*
- you get a tree that hasn't had *any rotations performed on it*, with an increased height of at most one



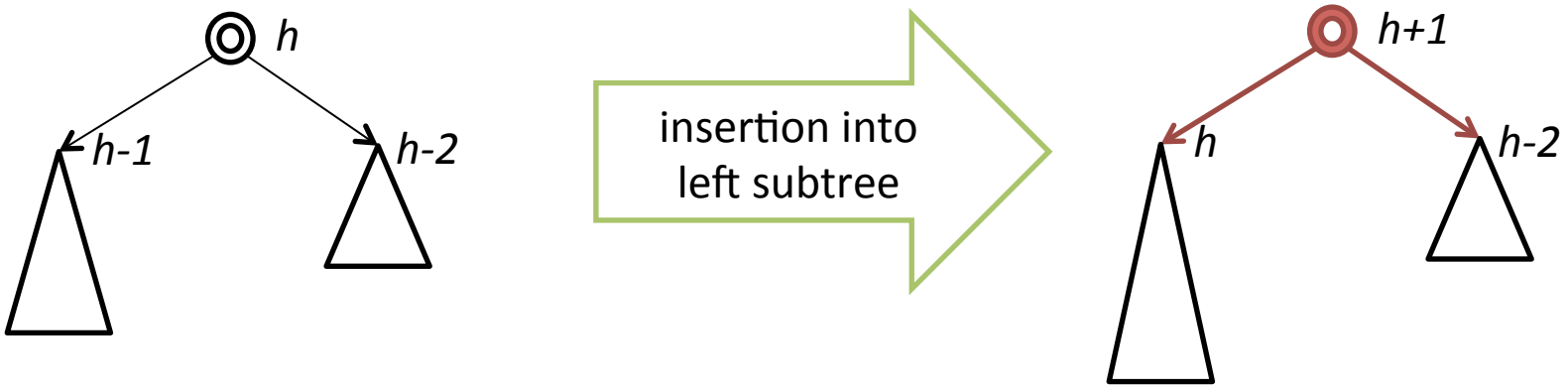
how could a violation happen?



The red nodes ◎ violate the AVL height invariant.

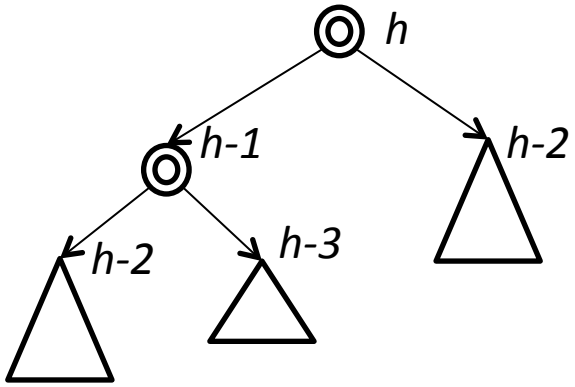
Two functions, rebalance_left and rebalance_right, will handle these two cases.

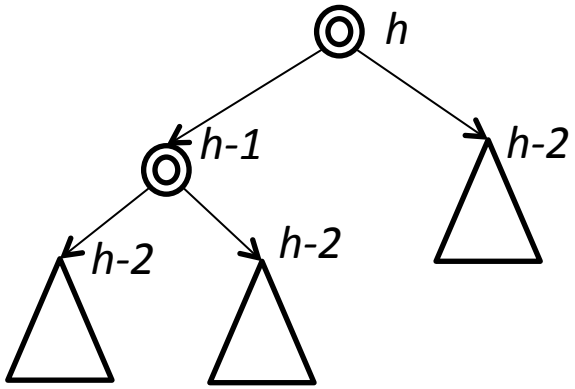
what does that left subtree look like?

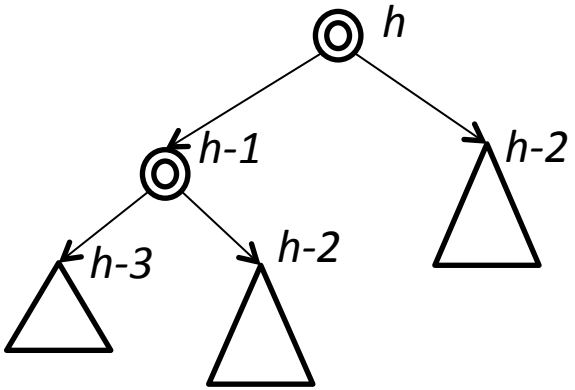


We'll just consider rebalance_left for now.

To figure out how to handle the violation, we must look at how the left subtree is constructed.

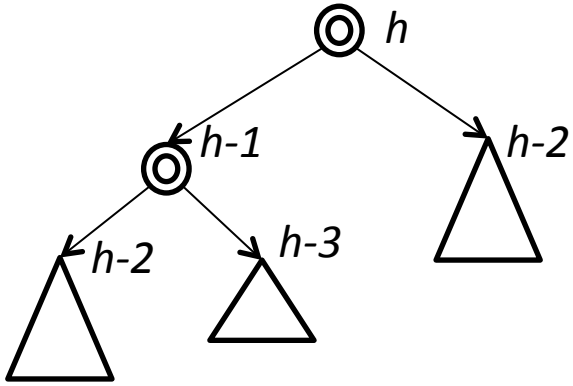


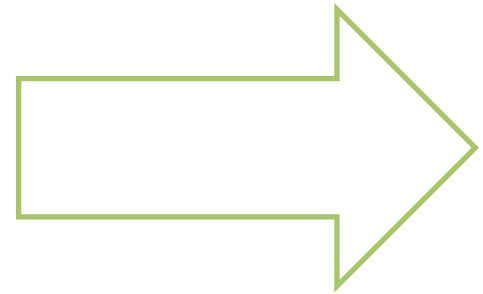


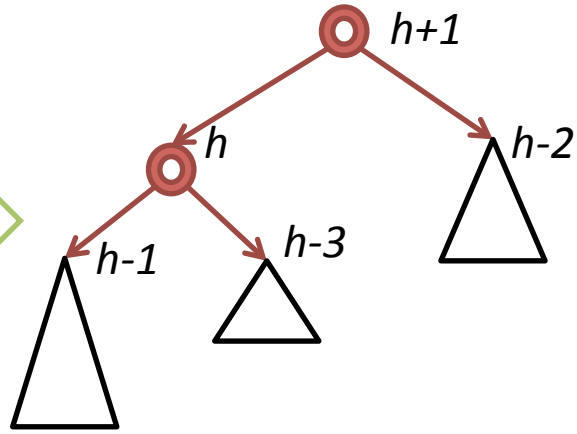


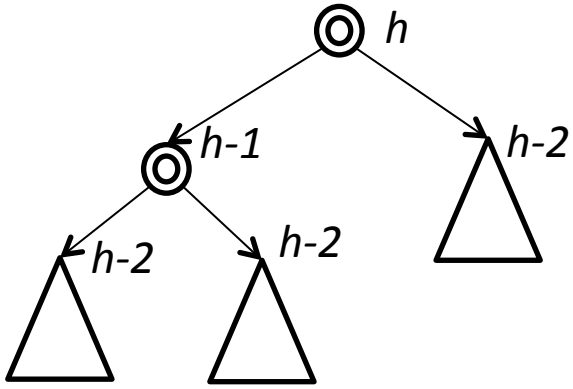
We know the AVL tree invariant held before, so the left subtree can only take one of three forms.

But that's not all we know!

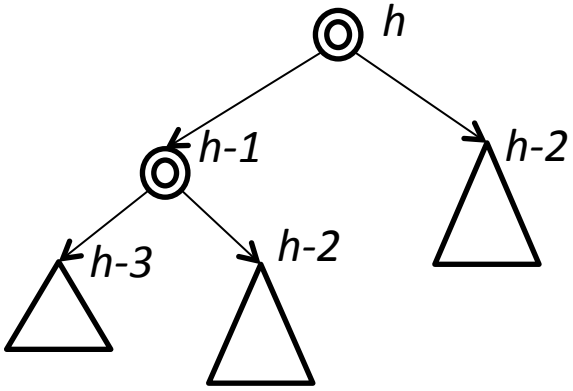


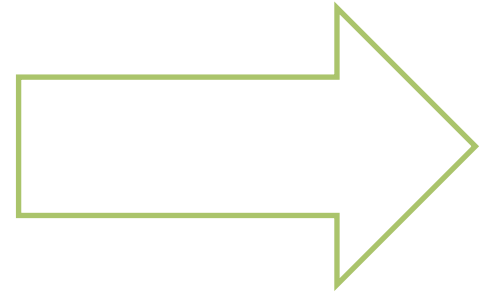


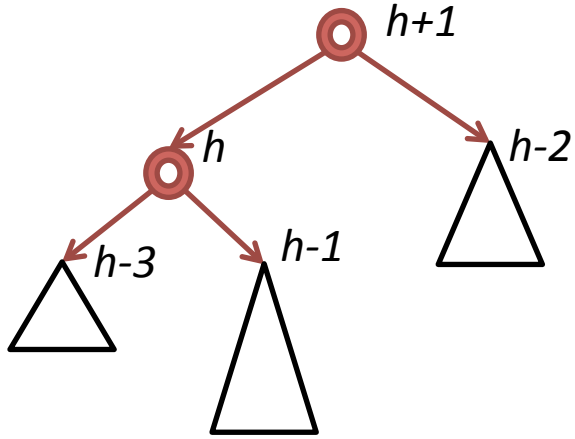


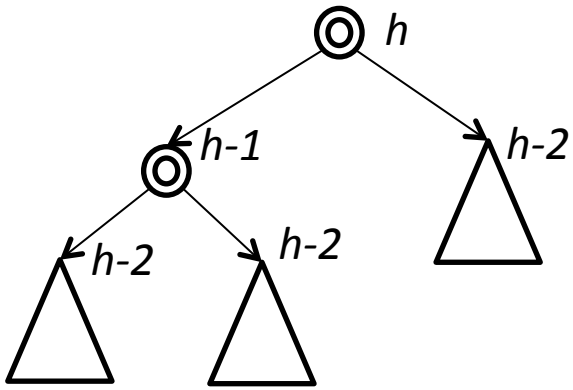


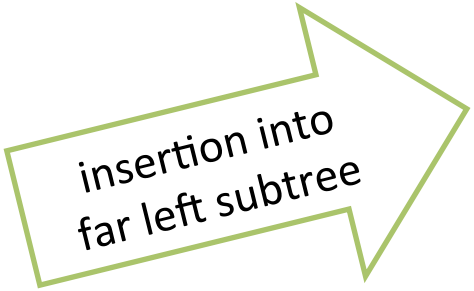
The first and the last of these three trees are *impossible* according to the original invariant, because they cause a violation lower in the tree! Or, operationally, we can say that if the tree had looked like the first or last of these three, we would have already noticed, and fixed, the violation.

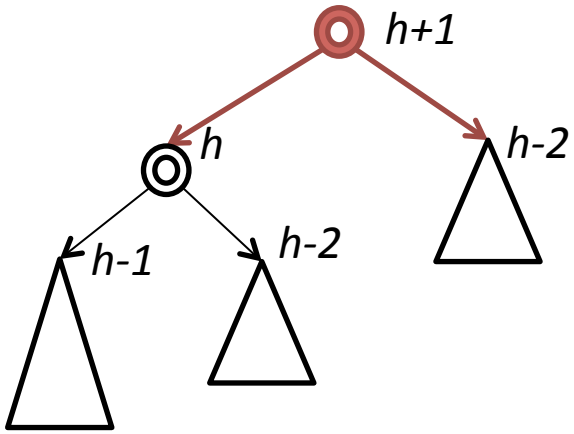


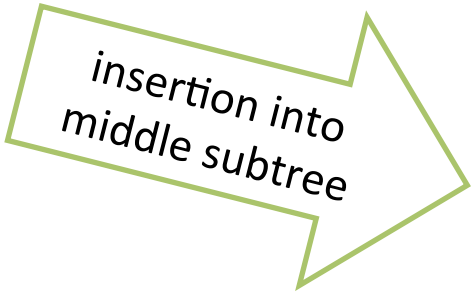


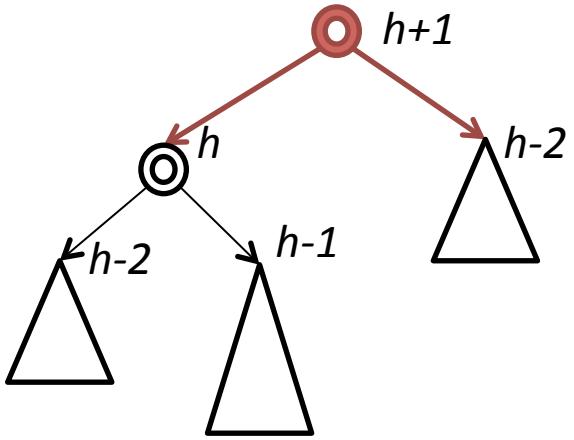




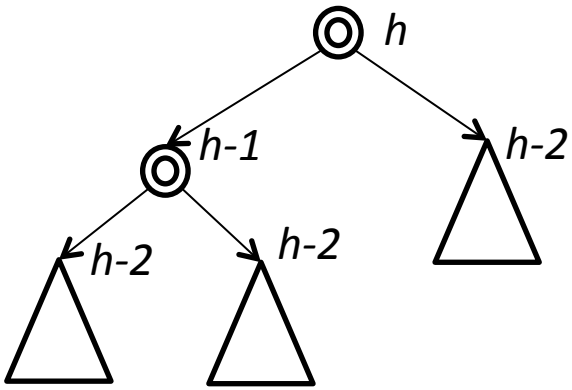


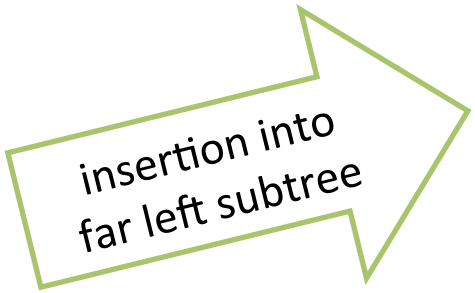


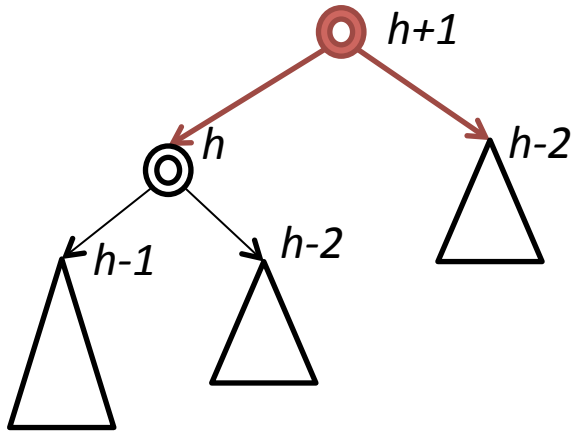




So we're left with this picture for rebalance_left:
two possible ways that BST insertion could have violated the AVL height invariant at the root.

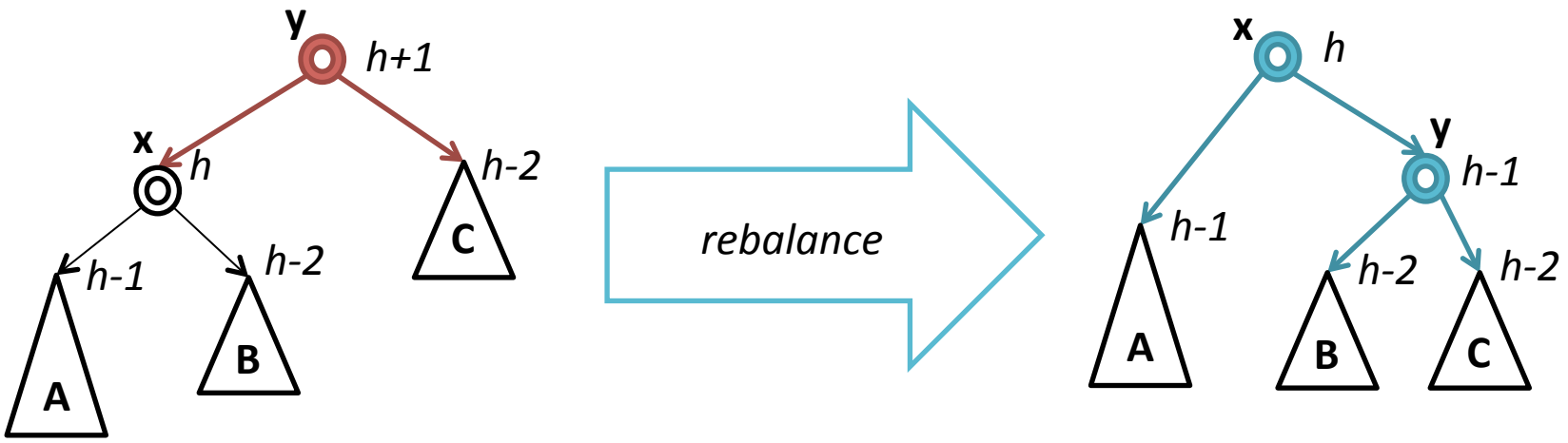




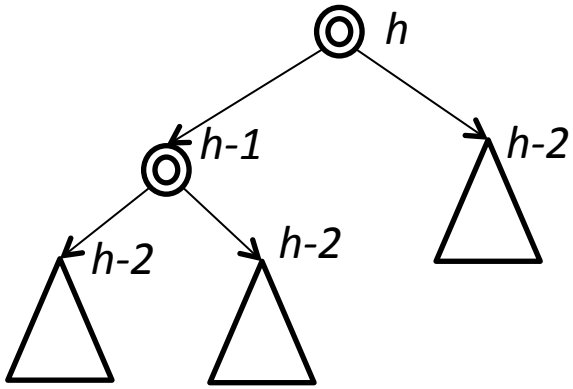


We can immediately see how to resolve the first case…

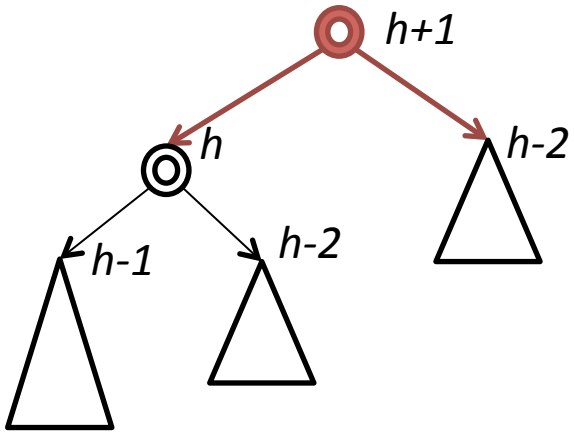
fixup: single rotation



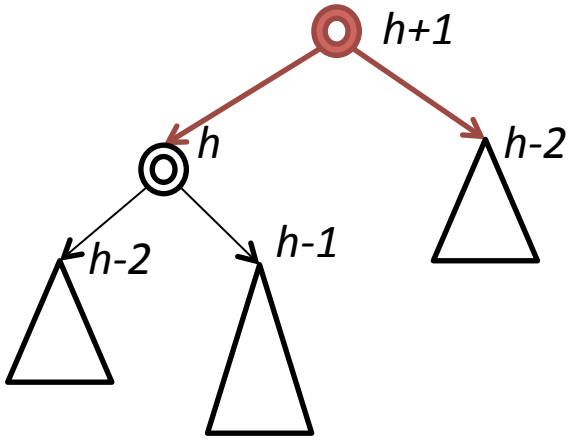
…a single right rotation at the root.



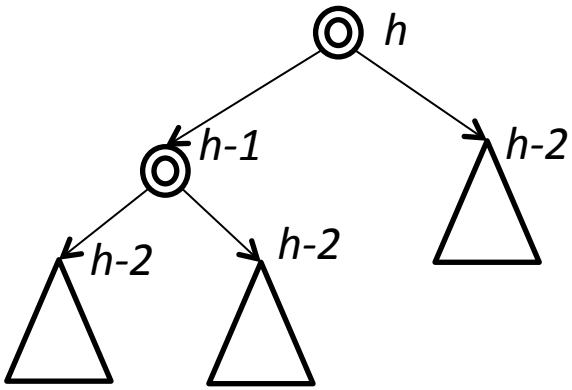
insertion into far left subtree

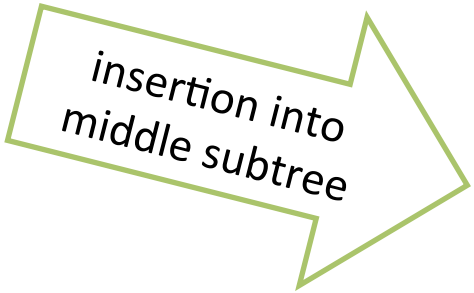


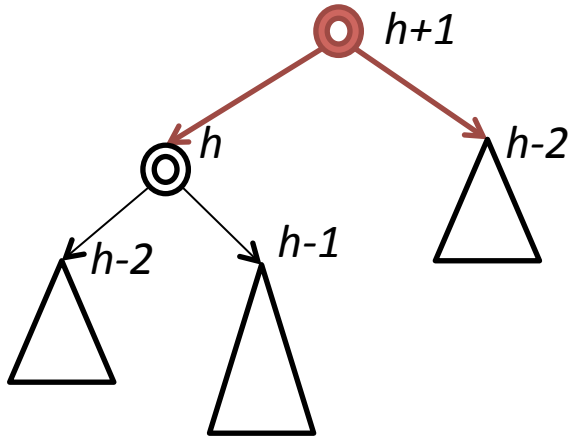
insertion into middle subtree



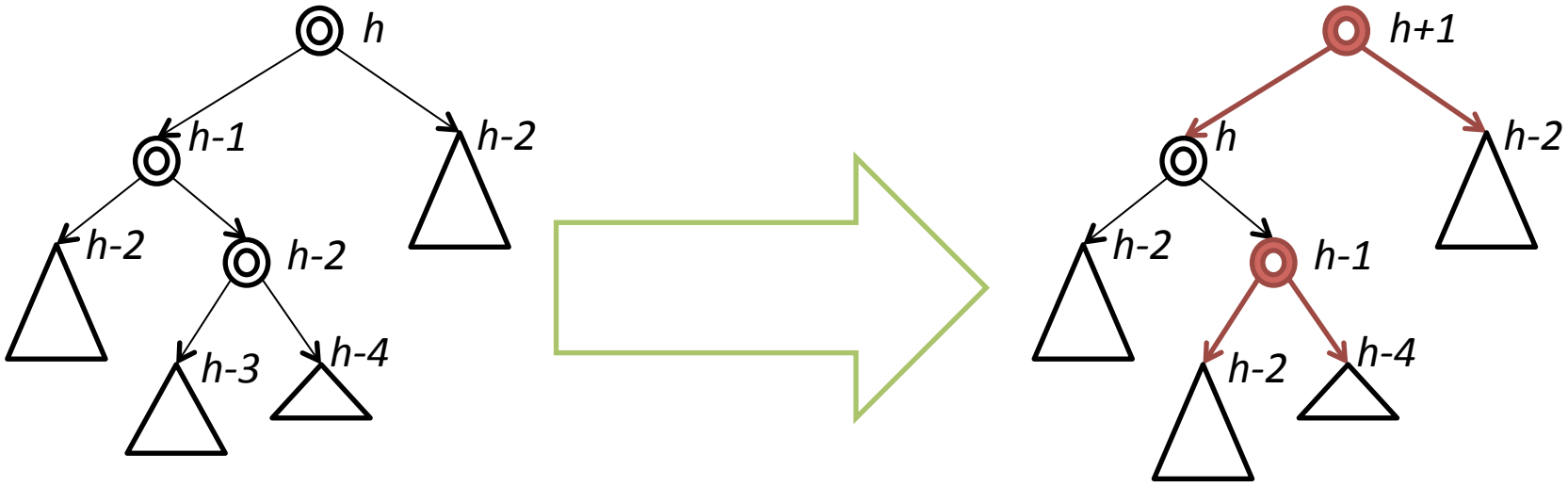
That takes care of the first case…



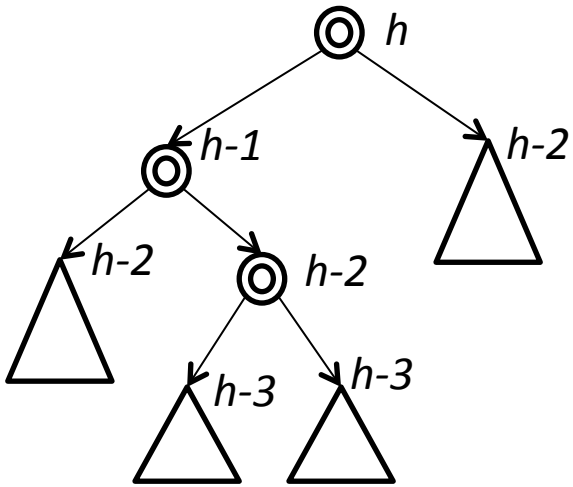


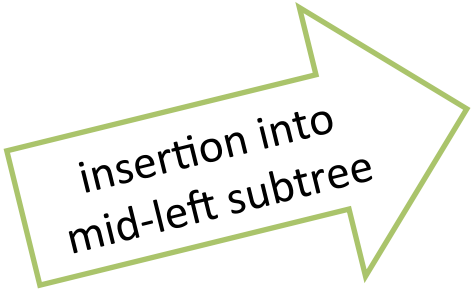


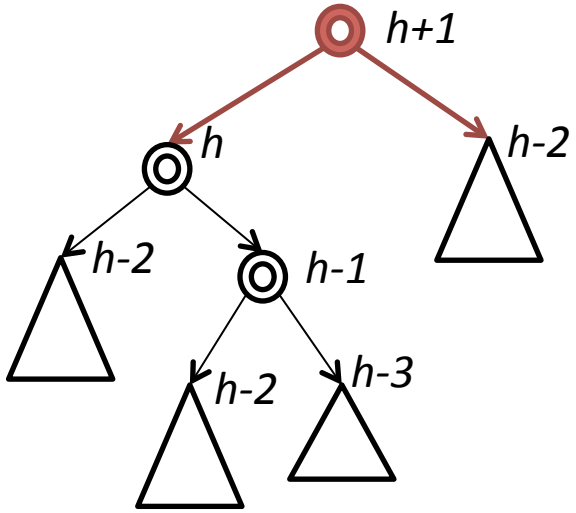
…now let's look at the second case.
To see how this works, we'll need to examine the structure of the middle tree.

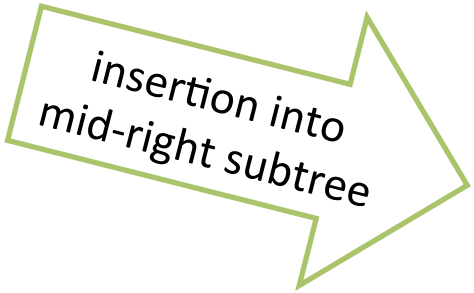


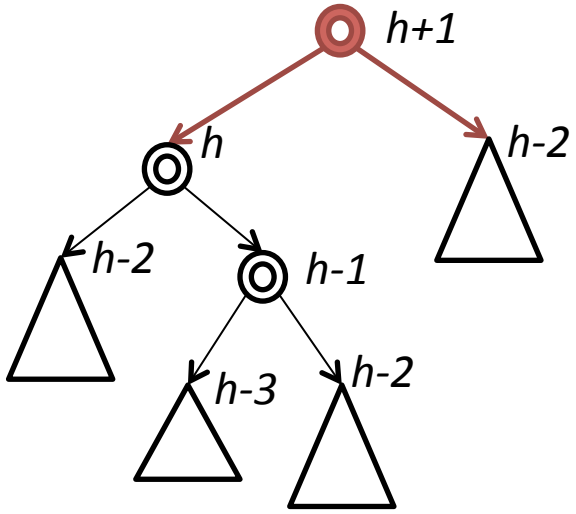
By the same reasoning as before, we know that the middle subtree can't itself have subtrees with different heights, or there would be a lower violation of the height invariant.





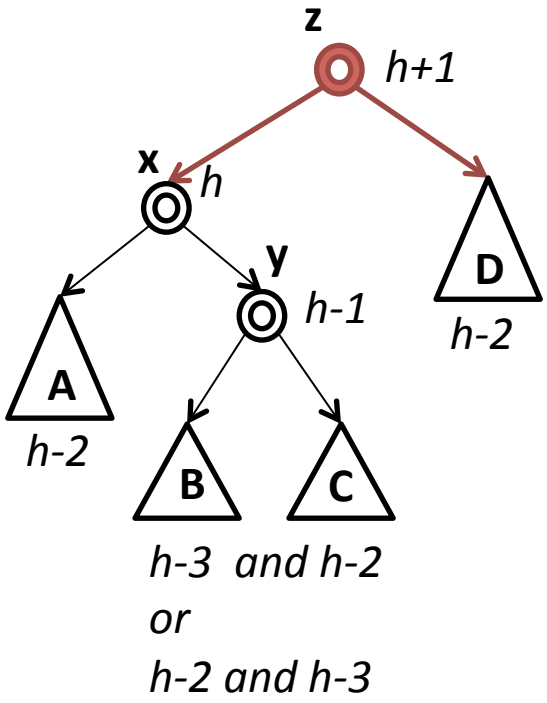


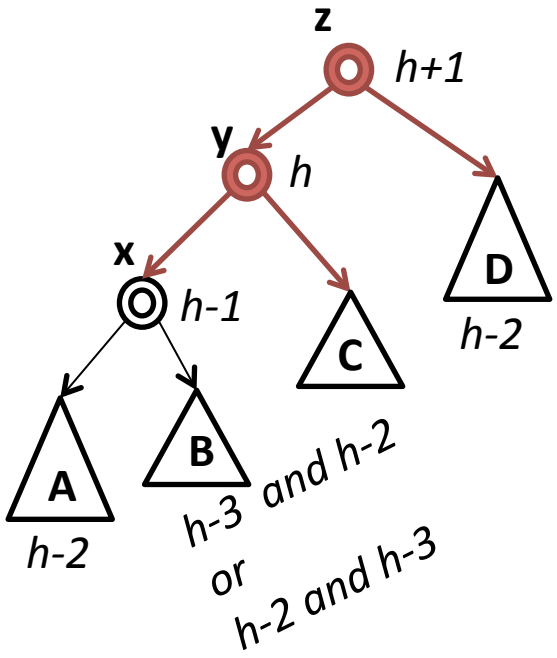


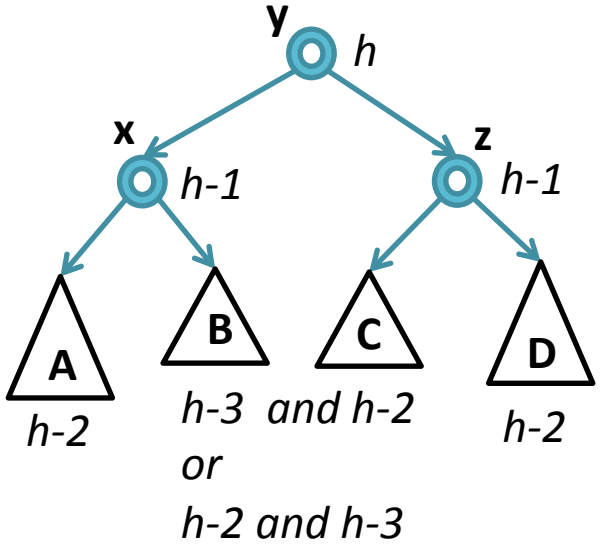


So both middle subtrees have height *h-3*, and one of them has height *h-2* after the insertion.
Again we have two cases! Luckily, we can deal with both the same way.

fixup: double rotation







The solution is the same regardless of whether the insertion went into **B** or **C**.
We rotate *twice*: first, left, at T->left, and then, right, at T.