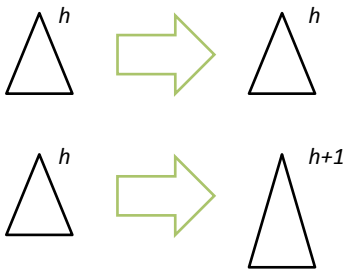
AVL insertion postcondition

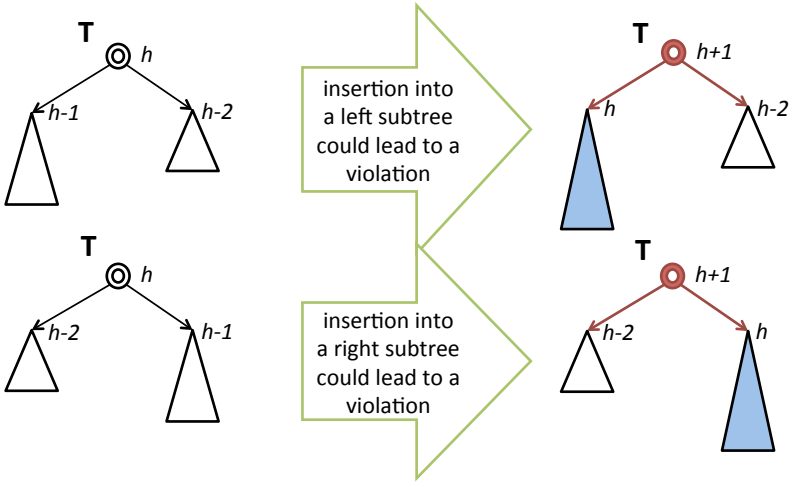
When you insert into an AVL tree of height *h*, *either*

- 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

When an insert is done in an AVL tree, nodes are checked one by one, moving up toward the root to make sure the height invariant continues to hold.



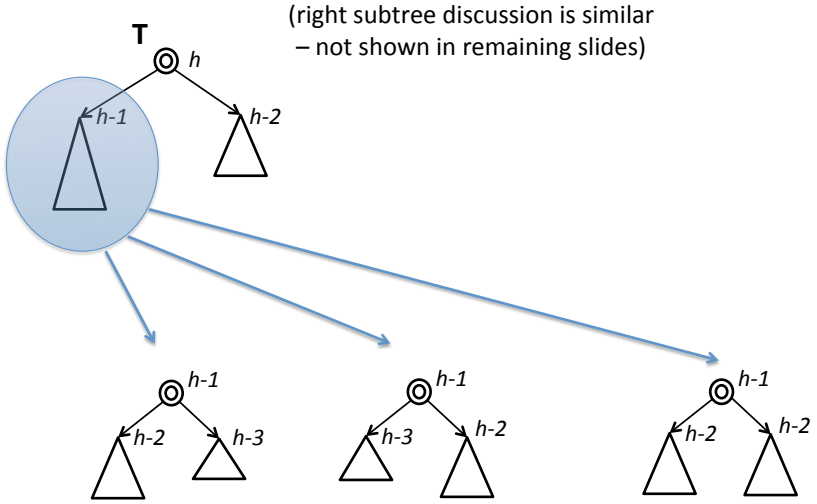
How could a violation of the AVL height invariant happen at node T?

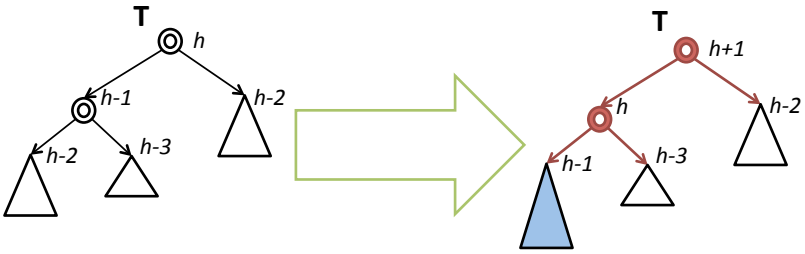


(the red nodes violate the AVL height invariant)

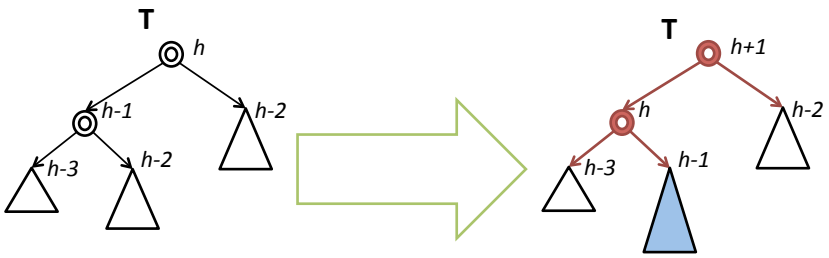
What does that left subtree look like before the insertion?

(right subtree discussion is similar
– not shown in remaining slides)

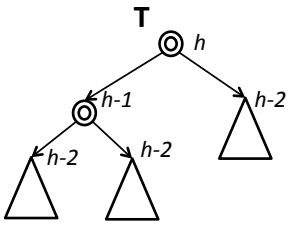




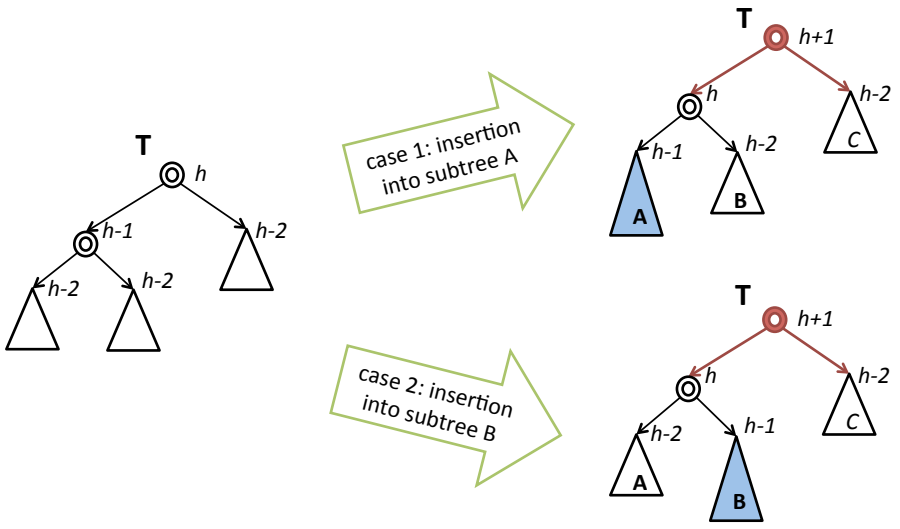
In the first situation, the AVL height invariant holds for node T. If we insert into the tree, we could violate the invariant not just at the node T but at its left child. This shouldn't happen since the height invariant should already hold for the children of the node T from a previous check. (The tree is checked bottom to top after an insertion is done.)



In the second situation, the AVL height invariant holds for the node T. If we insert into the tree, we could violate the invariant not just at node T but at its left child. This shouldn't happen since the height invariant should already hold for the children of the node T from a previous check. (The tree is checked bottom to top after an insertion is done.)

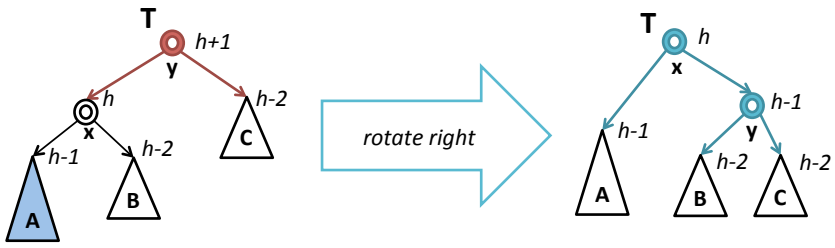


In the third situation, the AVL height invariant holds for the node T. If we insert into the left subtree of T, we could violate the invariant at only node T, so this is the only situation that we need to consider.



To violate the height invariant for node T, the inserted value must either increase the height of the left subtree of T's left child or the right subtree of T's left child.

Case 1: Single rotation fixes the invariant



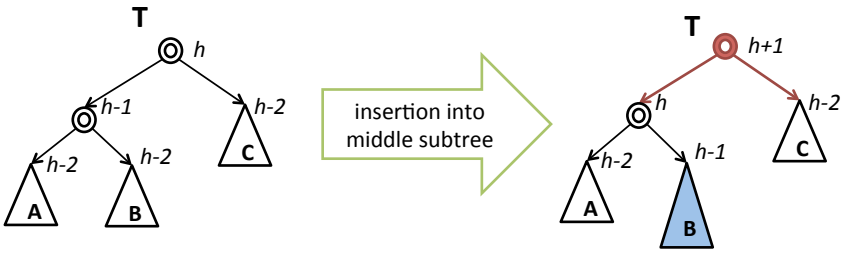
In the first case, a single rotation right fixes the tree at node T.

```
newT = T->left
T->left = T->left->right
newT->right = T
T = newT
```

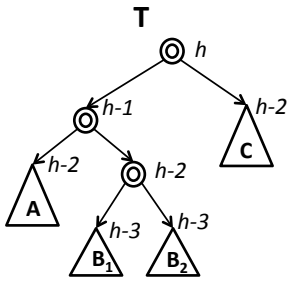
rotate right

(Note that the heights of the subtrees remains the same, but the heights of the nodes labeled x and y change.)

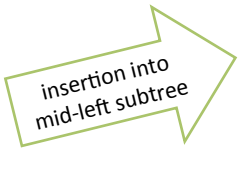
Case 2 is more involved

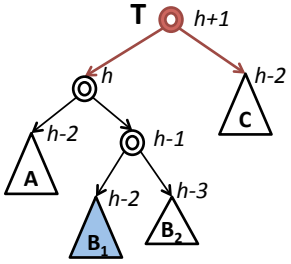


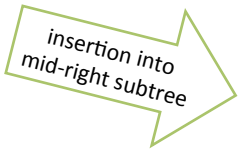
In the second case, we need to examine the structure of subtree B.

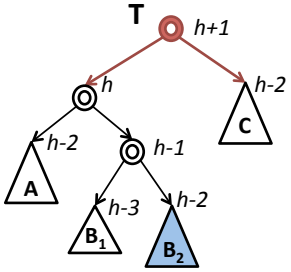


Again, these two subtrees must be the same height, otherwise we would have a violation lower in the tree as well as at T.

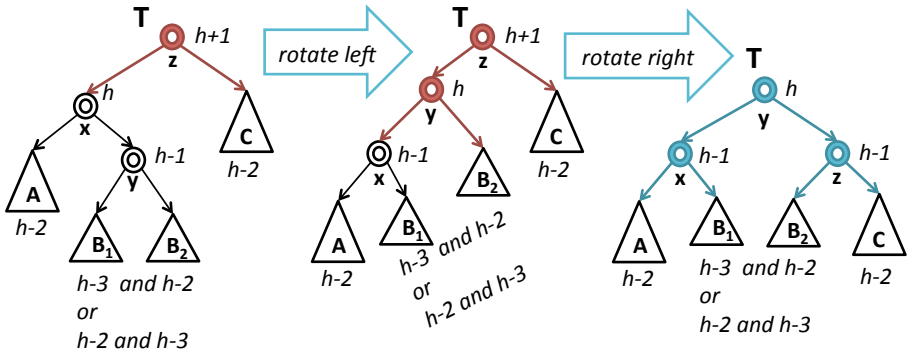








Case 2: Double rotation fixes the tree



Perform rotate left on T->left, which reverses the roles of x and y. Then perform rotate right on T, which reverses the roles of y and z, balancing the tree at node T. (Note that in the middle of this double rotation, two nodes violate the height invariant temporarily.)

More to consider

How would you write the *rotate left* operation?

How would you analyze the case where an insertion into the right subtree of T causes a height violation?

Show that in any of these rotations, the ordering invariant continues to hold.