2019-03-12 AVL Trees

Tuesday, March 12, 2019 8:58 AM

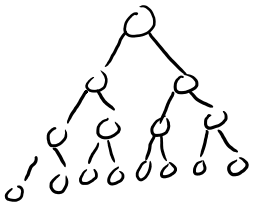
Visualization Resources

- BST Visualization: http://www.cs.usfca.edu/~galles/visualization/BST.html
- AVL Visualization: http://www.cs.usfca.edu/~galles/visualization/AVLtree.html
- Visualization homepage: http://www.cs.usfca.edu/~galles/visualization/Algorithms.html

AVL Tree Properties

- An AVL tree is a BST with one additional rule:
  - For each node, the difference between the height of the right subtree and the left subtree cannot be greater than one
- This property must always remain true. This means that the tree might have to be adjusted on an insert or removal
- Unlike a BST, an AVL tree is guaranteed to be fairly balanced
  - Why does this matter?

- Consider a well-balanced tree

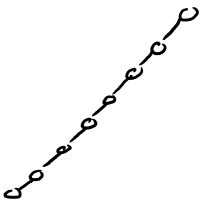


# of nodes $= (2^{\text{height}}) - 1$

cost of find $\log_2 2^{\text{height}}$

" $=$ height

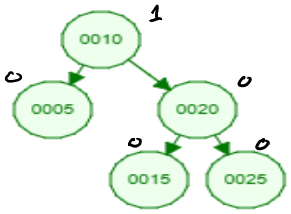
Consider a poorly-balanced tree



cost of find $= N$

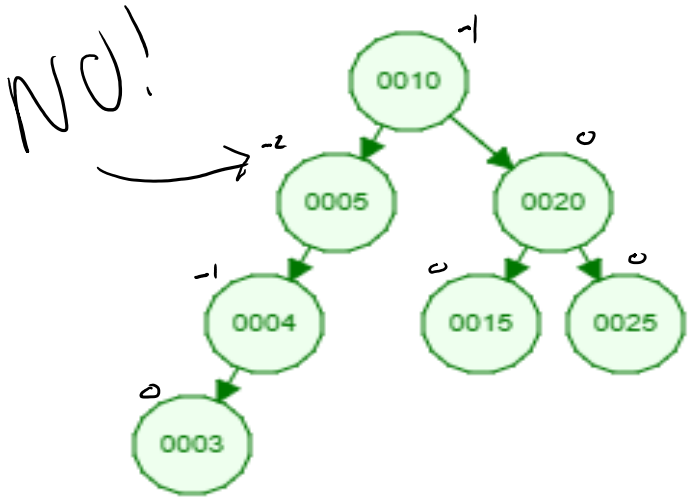
- There are massive performance implications when a tree becomes greatly imbalanced.
- Thus, an AVL tree is a way to guarantee good performance regardless of insert sequence. Unlike BSTs, AVL tree guarantee O(LogN) performance

Is this an AVL tree?

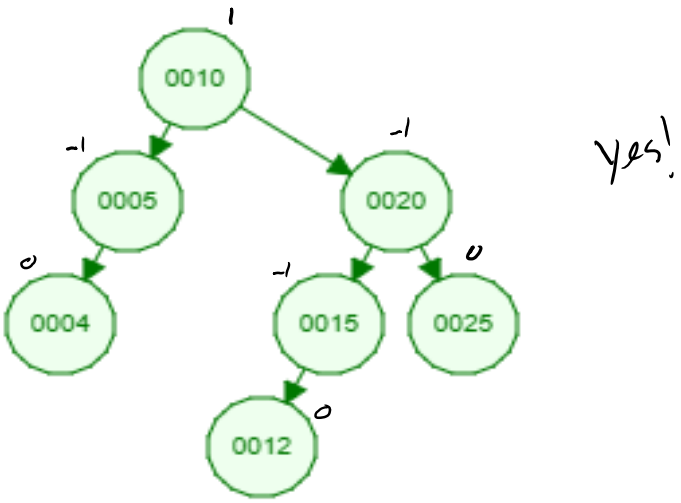


Yes!

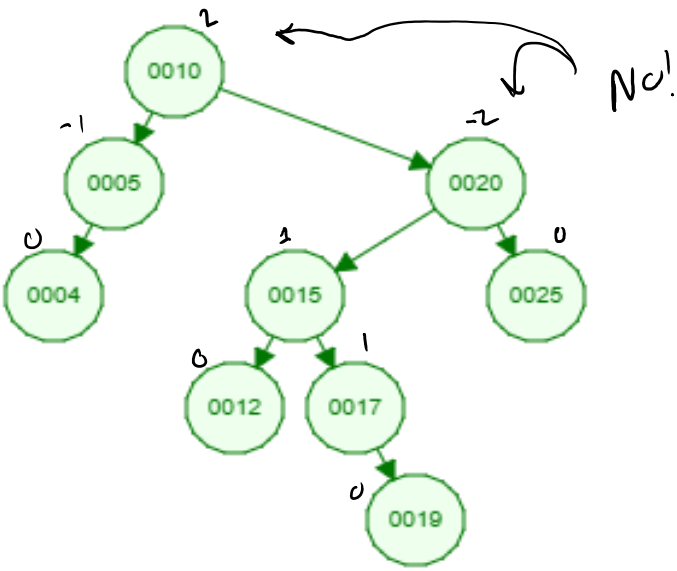
What about this one?



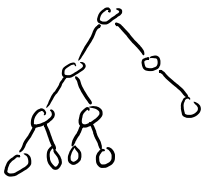
This one?



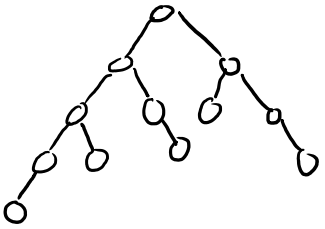
This one?



How unbalanced can an AVL tree of height 3 be?



Draw an AVL tree of height 4 that has the fewest possible nodes in it



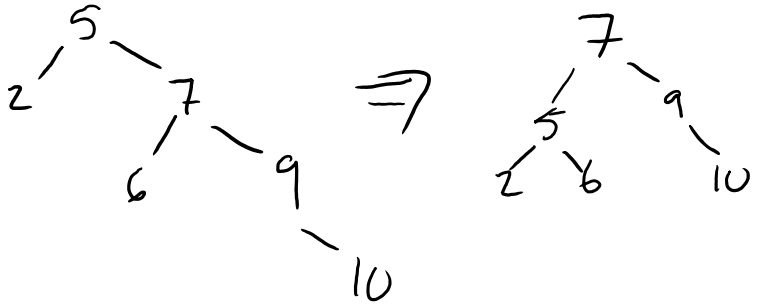
Constructing an AVL tree

- Observation: a tree with 0 nodes is AVL compliant
  - Meaning all trees start out as AVL trees
- If we start with an AVL tree, it is relatively "easy" to maintain AVL correctness if we verify AVL correctness after every tree operation (i.e. insert and remove)

Verifying AVL correctness

1. Working up from where the tree was modified back up to root, find where the tree is imbalanced
2. If we find an imbalanced node (abs(balance factor) > 1), we need to adjust the tree at that point. This is called a rotation.

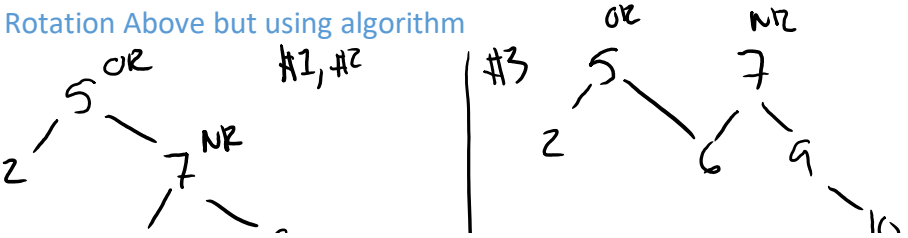
When the balance factor is +2, we do a left rotation

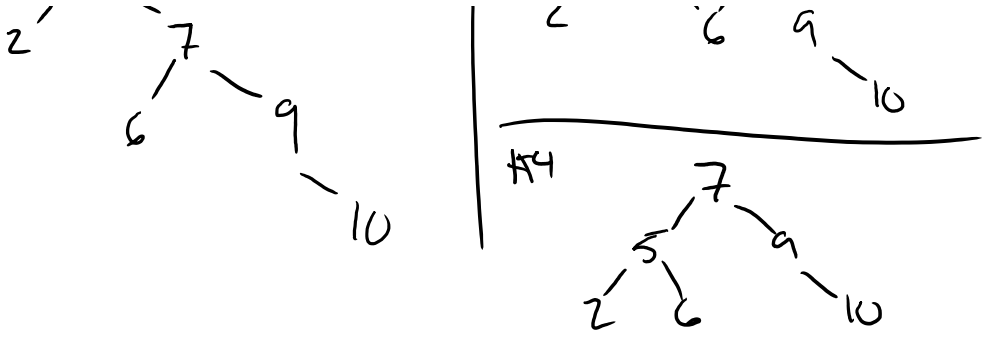


- At the node whose balance factor is equal to 2, do the following:

1. Let OriginalRoot = the imbalanced node (5 in the case above)
2. Let NewRoot = OriginalRoot->RightChild() (7 in case above)
3. Set OriginalRoot's right child equal to NewRoot's left child
4. Set NewRoot's left child equal to OriginalRoot

Rotation Above but using algorithm



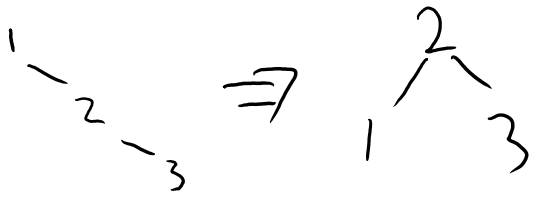


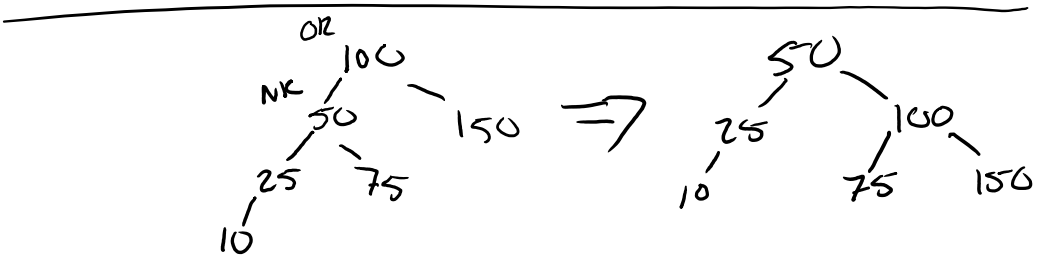
Right (clockwise rotation) Algorithm

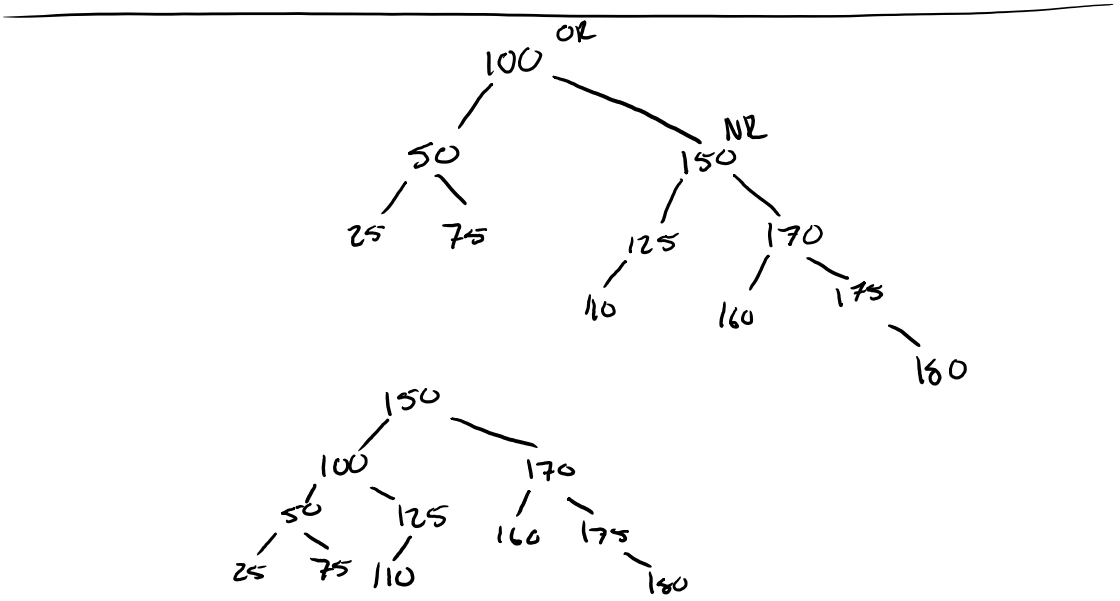
- At the location where balance factor equals -2

1. Let OriginalRoot = the imbalanced node
2. Let NewRoot = OriginalRoot->Left()
3. Set OriginalRoot's left child equal to NewRoot's right child
4. Set NewRoot's right child equal to OriginalRoot

Examples







Add values 1 through 10 to an empty AVL tree

Add values 1 through 10 to an empty AVL tree