2019-10-24 AVL Trees

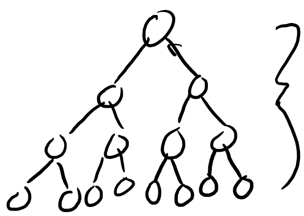
Tuesday, March 12, 2019 8:58 AM

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 differ by more than one.
- This property must always remain true, regardless of insert or removal sequence
    - Thus, adjustments to the tree's structure must be periodically made
- Unlike a standard BST, AVL trees are guaranteed to be fairly balanced
- Consider a well-balanced tree…

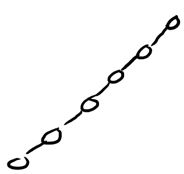


LogN height

cost of a find is $\log_2 2^{height}$

" " " = height

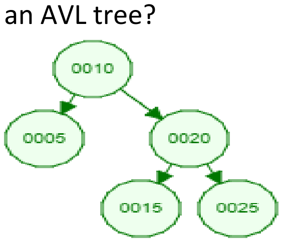
- Consider a poorly balanced tree



height = N

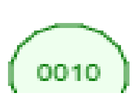
- There is a massive performance decrease as our BST's height becomes more linear in relationship to # of nodes.
- Therefore, an AVL tree is one method of guaranteeing fast performance on a tree.

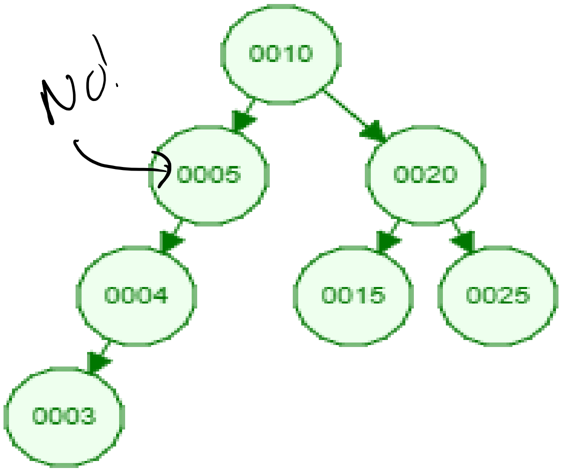
Is this an AVL tree?



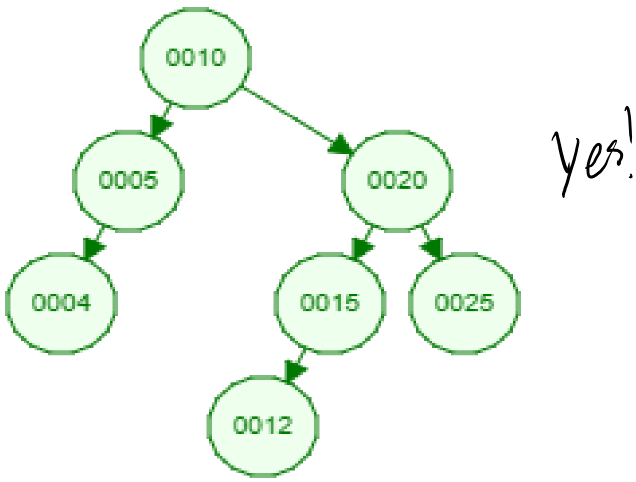
yes!

What about this one?

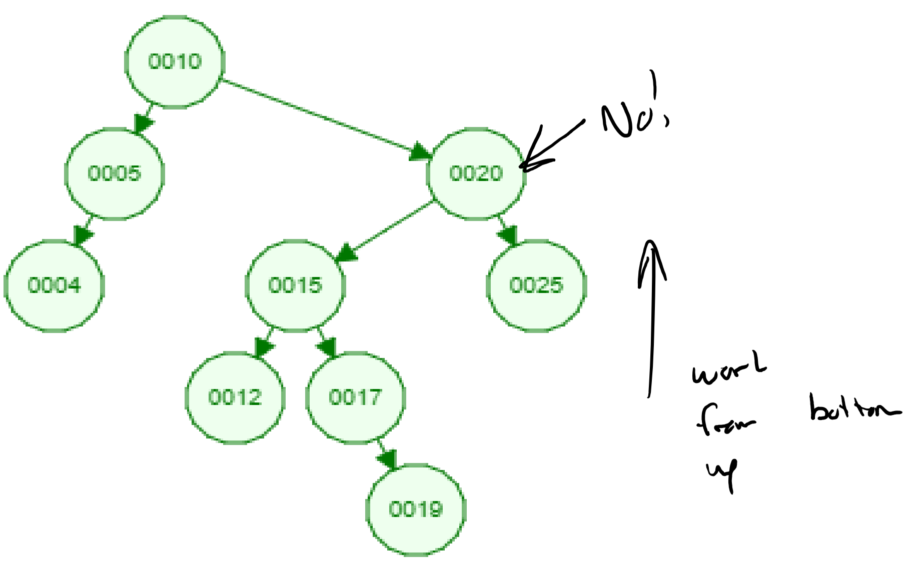




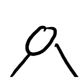
This one?



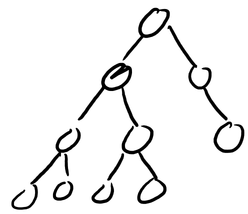
This one?



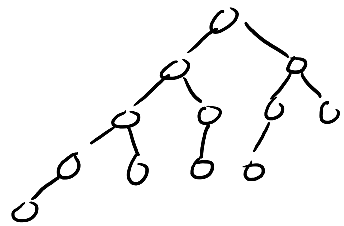
How unbalanced can an AVL tree of height 3 be?



How unbalanced can an AVL tree of height 3 be?



What would an AVL tree of height 4 look like that has the fewest nodes possible?



How do we construct and maintain an AVL tree

- Observation: an empty tree is AVL compliant
- Observation: a tree of size one is balanced (leaf node tree)
- Implication: all trees start out balanced
- Inserts and removes can cause a tree to become unbalanced
  - Thus, we may need to periodically rebalance after such an operation

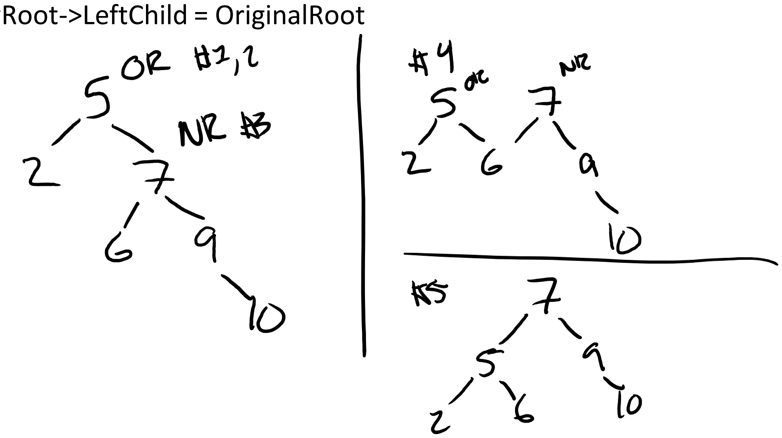
Verifying AVL correctness

1. Working up from where the tree was modified (insert or remove), find the imbalance
2. Term: balance factor = (height of right subtree) - (height of left subtree)
3. If we find a node whose abs(balance factor) > 1, we need to adjust the tree
   a. This is called an AVL rotation

A left rotation occurs when the balance factor > +1

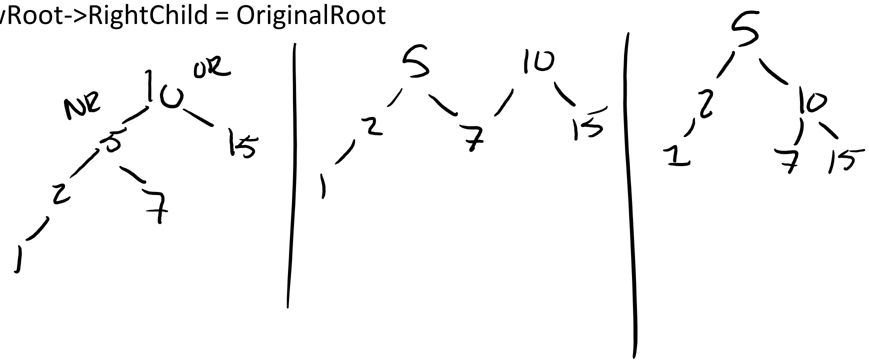
- AKA counter-clockwise rotation (or anti-clockwise)

1. At the node whose balance factor is 2, do the following:
2. Let OriginalRoot = the imbalanced node (node identified in #1)
3. Let NewRoot = OriginalRoot->RightChild
4. Set OriginalRoot->RightChild equal to NewRoot->LeftChild
5. Set NewRoot->LeftChild = OriginalRoot

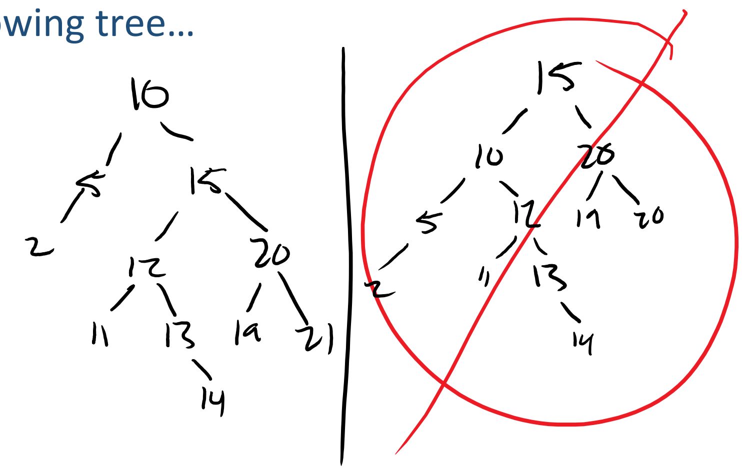


Right (clockwise) rotations occur when BF < -1

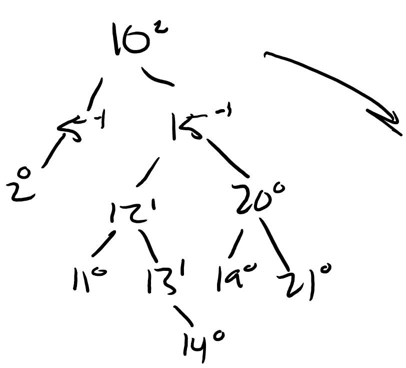
1. Let OriginalRoot = the node that is imbalanced
2. Let NewRoot = OriginalRoot->LeftChild
3. Let OriginalRoot->LeftChild = NewRoot->RightChild
4. Let NewRoot->RightChild = OriginalRoot



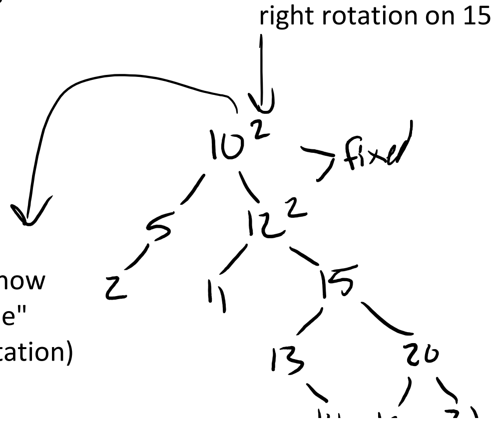
Consider the following tree…



- The above left rotation didn't solve the imbalance
- In order to solve this, we need to track balance factors at each node



- Single rotations **will not work** when OriginalRoot and NewRoot have differing signs on their balance factors
- To fix this requires a "pre rotation" that will align the balance OR and NR's balance factor
- In this example, this means doing a right rotation on 15



With signs of BF aligned, we can now perform the "true" Rotation (left rotation)

perform the "true"
Rotation (left rotation)