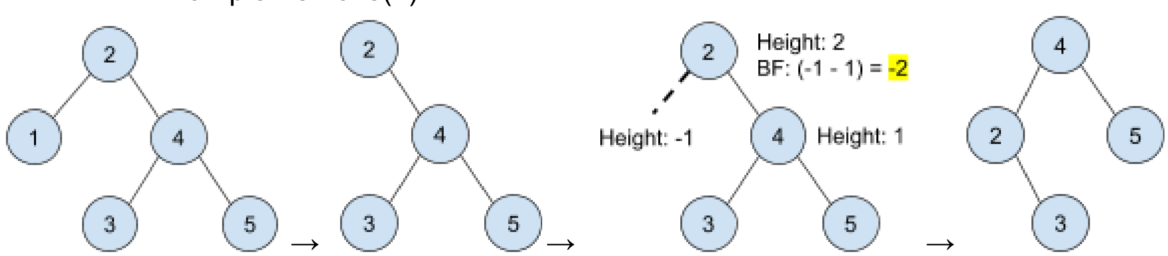
CS1332 Fall 2018 - Week 08

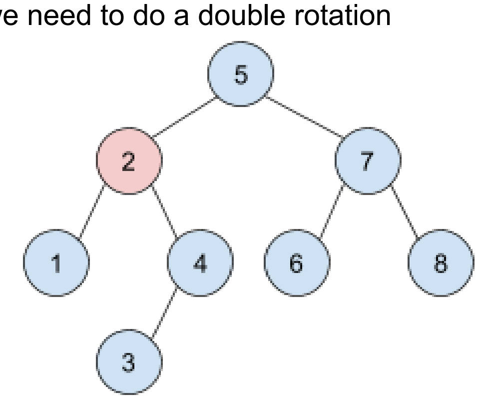
Fall Break - 10/8/18
10/10/18

<u>Topics</u>

I. Exam 2 Topic List: see announcement on Canvas for full list
   A. Coding up to (but not including) HashMaps will be fair game
II. AVL Remove
   A. The algorithm:
      1. Remove the node like you are removing from a BST (considering all the same 0, 1, and 2 child cases).
      2. When tracing back up from the node you removed, update heights and balance factors, and check for and perform rotations.
         a) If this was a two child remove case, make sure to update and rotate when coming back from removing the predecessor/successor.
   B. Example: remove(1)

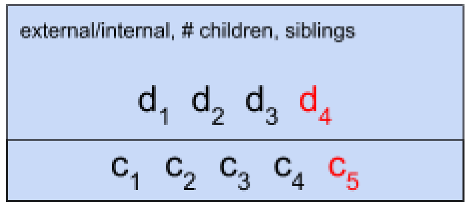


      1. Remove like you're removing from a BST
      2. Update the height and balance factor of the node containing 2 (the only node we traversed to find the node containing 1)
         a) Height = max(-1, 1) + 1
         b) Balance factor = -1 - 1 = -2
      3. Since the balance factor is -2 and its right child's balance factor is not > 0, we just need to do a single left rotation on the node containing 2
   C. Example: remove(2) using predecessor
      1. Hint: we need to do a double rotation



III. 2-4 Trees / B-Trees

A. 2-4 nodes:



1. Can have 2 to 4 children (c_i) and can store 1 to 3 data values (d_i)
    a) A node with m data values must have m + 1 children
    b) Data 4 and Child 5 are in red because they are allowed to exist momentarily, but will be immediately fixed
2. Data is stored within a node in ascending order (d1 < d2 < d3)
3. Nodes can also store whether they're internal or external, the number of children they have, and references to their siblings or parent

B. Order Property:

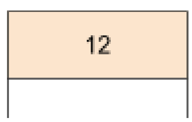
1. c_1 data < d_1 < c_2 data < d_2 < c_3 data < d_3 < c_4 data < d_4 < c_5 data

C. Shape Property:

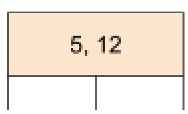
1. Every leaf MUST reside on the same level
2. Run times are guaranteed to be O(log n)

D. Example: add 12, 5, 20, 31, 2, 18, 42, 45, 49, 33, 36

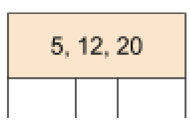
add(12):



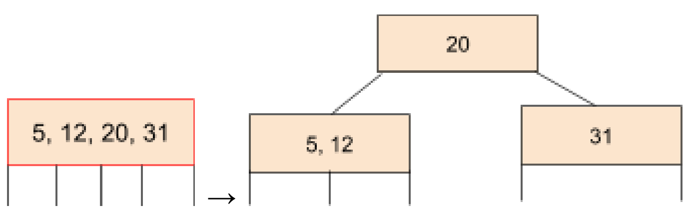
add(5):



add(20):

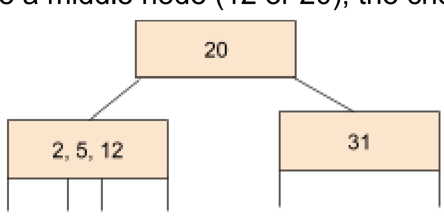


add(31):



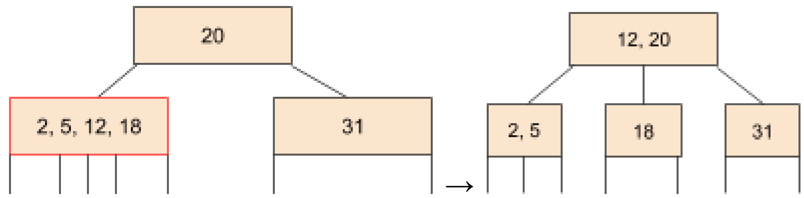
Promotion: occurs when we have overflow in a node; we will always promote a middle node (12 or 20), the choice is arbitrary

add(2):

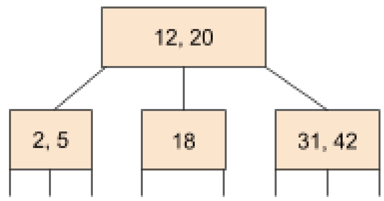


We always add new data to a leaf node.

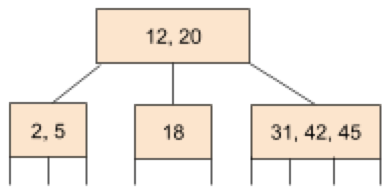
add(18):



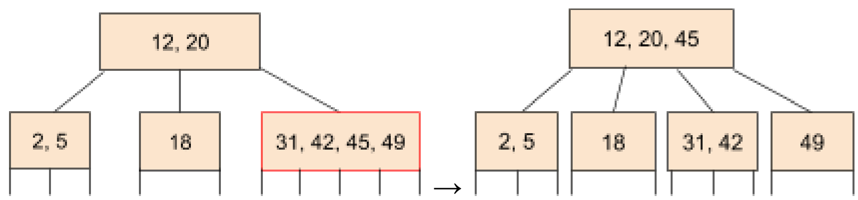
add(42):



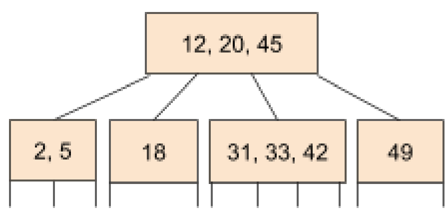
add(45):



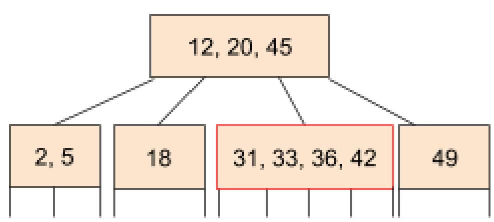
add(49):



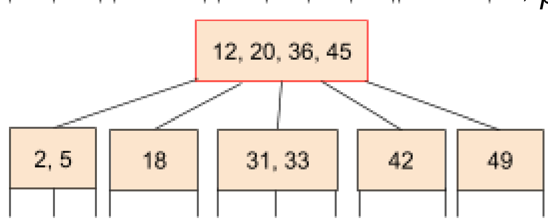
add(33):



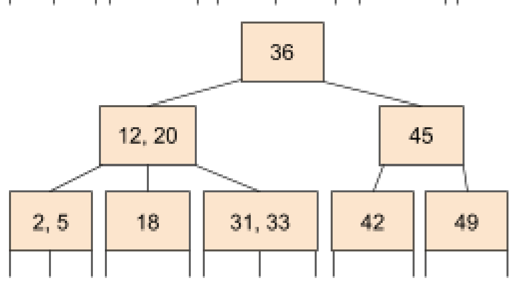
add(36):



→ *promotion*



→ *promotion*

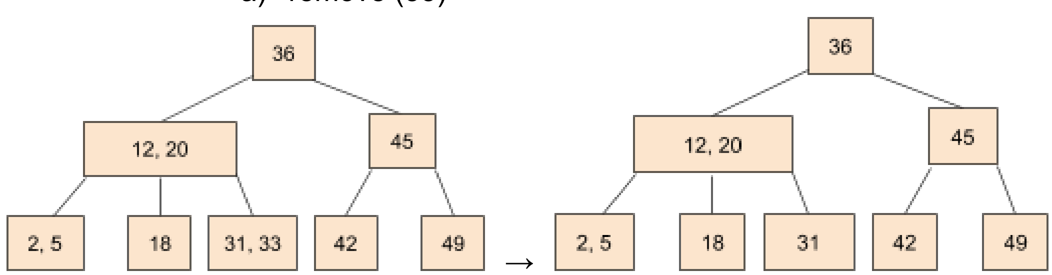


Activities
I. Big O reality check

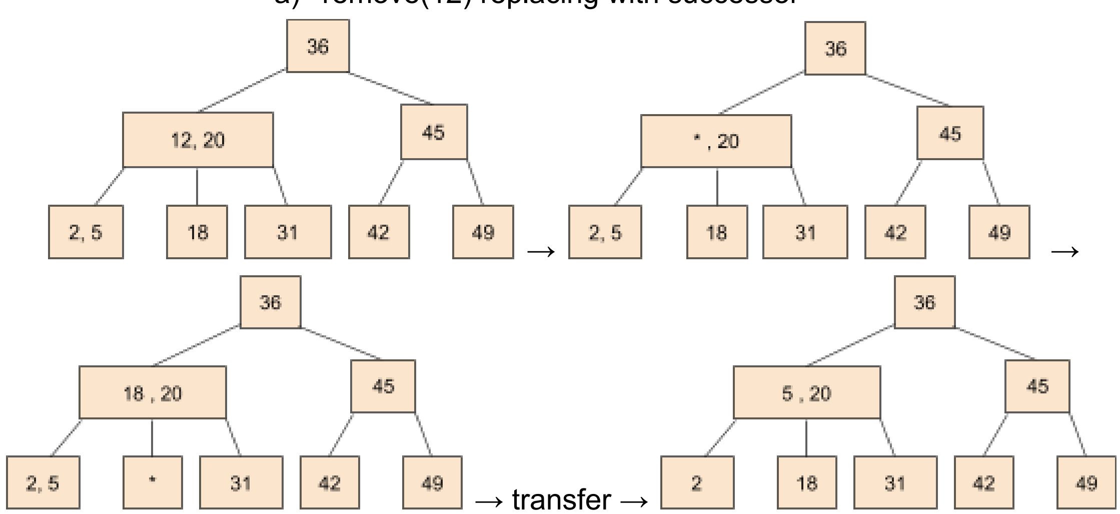
10/12/18

Announcements
I. Exam 2
   A. 2 coding questions
   B. 1 recursion tracing - show your work so we can award partial credit!
   C. Big-O multiple choice/matching
II. Practice exams
   A. 2 for each exam
   B. We provide hints not answers to help you arrive at the answers yourselves
   C. If you want your work checked, post privately to Piazza or come to office hours

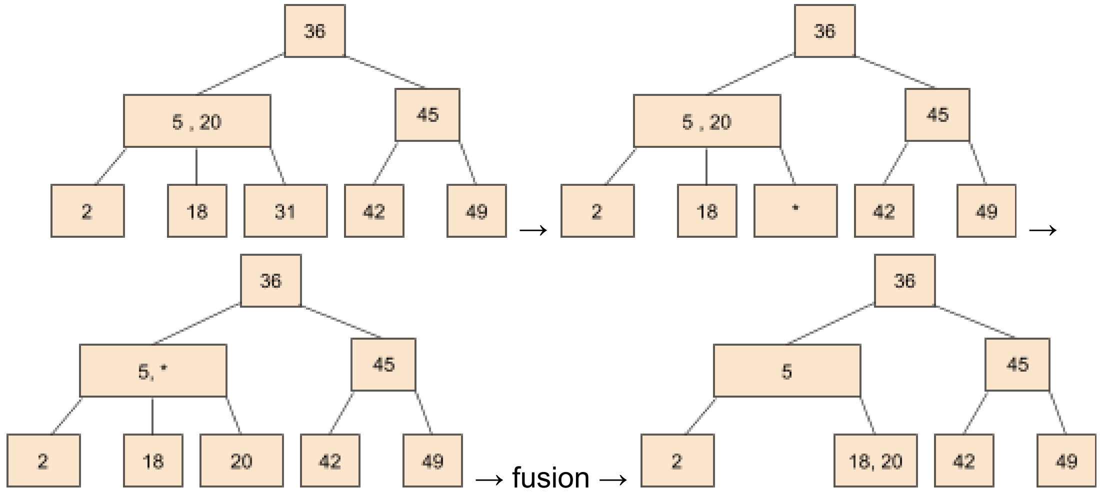
Topics
I. 2-4 Trees
   A. Add - can cause *overflow*, we fix overflow with *promotion* (move a middle value to its parent or create a new node if the overflow is in the root)
   B. Remove - can cause *underflow*, we fix underflow with *transfers* and *fusion*
      1. Removing from a leaf with > 1 values (easiest case)
         a) remove (33)



      2. Removing from an internal node with a child with > 1 value
         a) remove(12) replacing with successor



3. Removing from a leaf with a parent with multiple data
    a) remove(31)



*for a flowchart of the remove cases, see Canvas Files > Recitation > 2_4_remove.jpg

<u>Activities</u>

I. Tree review reality check