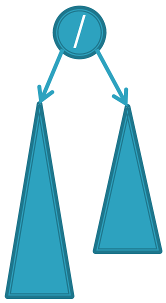


CSSE 230 Day 13

AVL trees and rotations

This week, you should be able to…
…perform rotations on height-balanced trees,
on paper and in code
… write a rotate() method
… search for the kth item in-order using rank

Announcements

- Term project partners posted
  - Sit with partner(s) now.
  - Read the spec before tomorrow and start planning.
- Test 2a next class

Test 2a next class: Recursive tree methods all follow this format

- Consider an arbitrary method named foo()

foo()

If base case, return the appropriate value

- 1. Compute a value for the node
- 2. Call left.foo() and right.foo()
- 3. Combine the results and return them

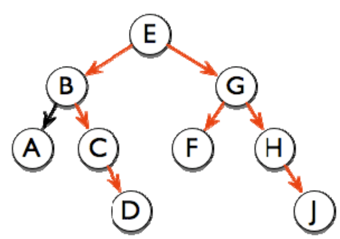
- This is O(n) if the computation on the node is constant-time
- But when searching in a BST, you only need to call left.foo() **or** right.foo(), so it is O(height)
- Style: pass info through parameters and return values.
  - Not extra instance variables (fields).

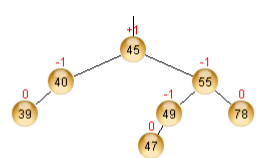
If you submitted HW4 TreePractice, you should have received a solution in your repo.

Summary: for fast tree operations, we must keep tree somewhat balanced in O(log n) time

Q1

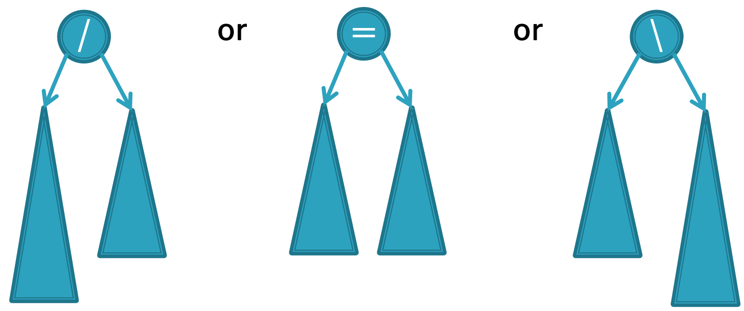
- Total time to do insert/delete =
  - Time to find the correct place to insert = O(height)
  - + time to detect an imbalance
  - + time to correct the imbalance
- If don't bother with balance:
- If try to keep perfect balance:
  - Height is O(log n) BUT …
  - But maintaining perfect balance is O(n)
- Height-balanced trees are still O(log n)
  - For T with height h, N(T) ≤ Fib(h+3) – 1
  - So H < 1.44 log (N+2) – 1.328 *
- AVL (Adelson-Velskii and Landis) trees maintain height-balance using rotations
- Are rotations O(log n)? We'll see…





AVL nodes are just like BinaryNodes, but also have an extra "balance code"

Q2

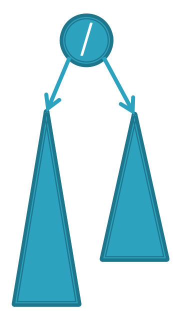


Different representations for / = \ :

- Just two bits in a low-level language
- Enum in a higher-level language

Using balance codes makes AVL Tree rebalancing efficient: O(log n)

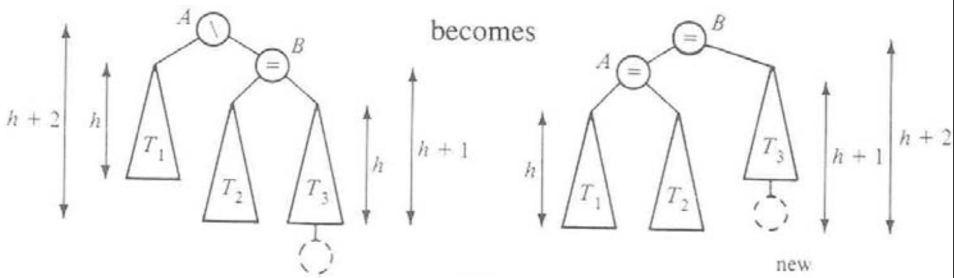
Q3



- Assume tree is height-balanced before insertion
- Insert as usual for a BST
- Move up from the newly inserted node to the **lowest** "unbalanced" node (if any)
  - Use the balance code to detect unbalance – how?
  - Why is this O(log n)?
    - We move up the tree to the root in worst case, NOT recursing into subtrees to calculate heights
- Do an appropriate rotation (see next slides) to balance the subtree rooted at this unbalanced node

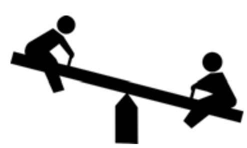
Four types of rotations are required to remove different cases of tree imbalances

- For example, a *single left rotation*:



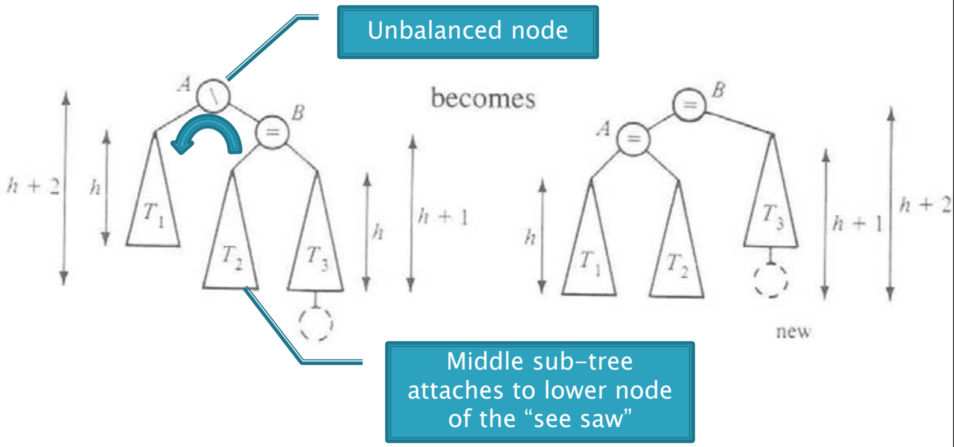
We rotate by pulling the "too tall" sub-tree up and pushing the "too short" sub-tree down

- Two basic cases
  - "Seesaw" case:
    - Too-tall sub-tree is on the outside
    - So tip the seesaw so it's level
  - "Suck in your gut" case:
    - Too-tall sub-tree is in the middle
    - Pull its root up a level



Single Left Rotation

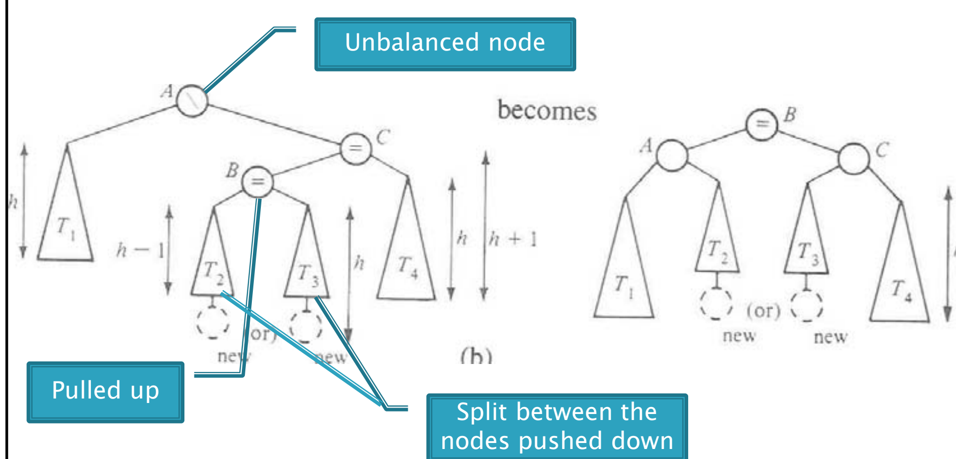
Q4-5



Diagrams are from *Data Structures* by E.M. Reingold and W.J. Hansen

Double Left Rotation

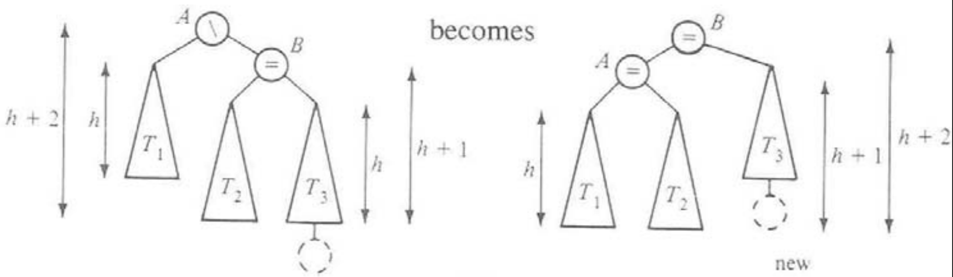
Q6-7



Weiss calls this “right-left double rotation”

Your turn — work with a partner

Q8



- Write the method:
- ```
  static BalancedBinaryNode singleRotateLeft (
      BalancedBinaryNode parent,    /* A */
      BalancedBinaryNode child      /* B */   ) {

  }
  ```
- Returns a reference to the new root of this subtree.
- Don't forget to set the balanceCode fields of the nodes.

More practice— (sometime after class)

- Write the method:
- ```
  BalancedBinaryNode doubleRotateRight (
    BalancedBinaryNode parent,     /* A */
    BalancedBinaryNode child,      /* C */
    BalancedBinaryNode grandChild  /* B */ ) {


  }
  ```
- Returns a reference to the new root of this subtree.
- Rotation is mirror image of double rotation from an earlier slide

O(log N)?

Q9,Q1,Q10-11

- If you have to rotate after insertion, you can stop moving up the tree:
  - Both kinds of rotation leave height the same as before the insertion!

- Is insertion plus rotation cost really O(log N)?

Insertion/deletion in AVL Tree:	O(log n)
Find the imbalance point (if any):	O(log n)
Single or double rotation:	O(1)
(looking ahead) *for deletion, may have to do O(log N) rotations*	
Total work:	O(log n)

Term Project: EditorTrees

Like BST, except:

1. Keep height-balanced
2. Insertion/deletion by **index**, not by comparing elements. So not sorted

Examples:

- EditorTree et = new EditorTree()
- et.add('a') // append to end
- et.add('b') // same
- et.add('c') // same. Rebalance!
- et.add('d', 2) // where does it go?
- et.add('e')
- et.add('f', 3)

- Notice the tree is height-balanced (so height = O(log n)), but not a BST

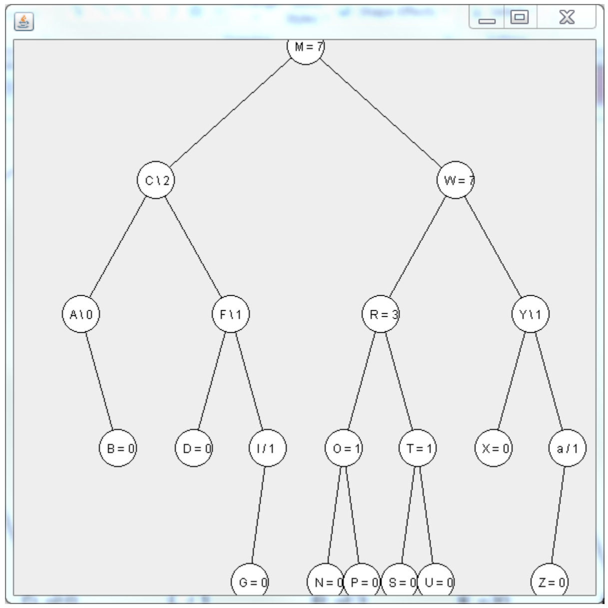
To find index quickly, add a **rank** field to BinaryNode

- Gives the in-order position of this node within its own subtree
  - i.e., the size of its left subtree

0-based indexing

- How would we do **`get(pos)`**?

- **`Insert`** and **`delete`** start similarly



With your EditorTrees team

Milestone 1 due in 1 week.

Start soon!

Read the specification and check out the starting code