

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

See schedule page

Test summary: 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()
- 3. Call right.foo()
- Combine the results and return them
- This is O(n) if the computation on the node is constant-time
- When searching in a BST, you only need to recurse left or right, so it is O(height)

If you submitted HW4, you will receive a solution in your repo. Look at it before Monday.

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

- 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, $N(T) \leq Fib(h+3) 1$
 - 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"



Different representations for $/ = \setminus$:

- 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)

- 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 sub-tree 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

- "See saw" case:
 - Too-tall sub-tree is on the outside
 - So tip the see saw 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



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

Q4-5

Double Left Rotation



Weiss calls this "right-left double rotation"

Your turn — work with a partner



- 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)?

- 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
 indexing

- How would we do findK_{th}?
- Insert and delete start similarly



Get with your EditorTrees team

Read the specification and check out the starting code

Milestone 1 due soon. Coordinate Fall break as a team Get started before next class!