CSE373: Data Structures & Algorithms
Optional Slides: AVL Delete

Dan Grossman
Fall 2013
The AVL Tree Data Structure

Structural properties

1. Binary tree property
2. Balance property: balance of every node is between -1 and 1

Result:

Worst-case depth is \( O(\log n) \)

Ordering property

- Same as for BST
AVL Tree Deletion

- Similar to insertion: do the delete and then rebalance
  - Rotations and double rotations
  - Imbalance may propagate upward so rotations at multiple nodes along path to root may be needed (unlike with insert)

- Simple example: a deletion on the right causes the left-left grandchild to be too tall
  - Call this the left-left case, despite deletion on the right
  - `insert(6) insert(3) insert(7) insert(1) delete(7)`
Properties of BST delete

We first do the normal BST deletion:
- 0 children: just delete it
- 1 child: delete it, connect child to parent
- 2 children: put successor in your place, delete successor leaf

Which nodes’ heights may have changed:
- 0 children: path from deleted node to root
- 1 child: path from deleted node to root
- 2 children: path from deleted successor leaf to root

Will rebalance as we return along the “path in question” to the root
Case #1 Left-left due to right deletion

- Start with some subtree where if right child becomes shorter we are unbalanced due to height of left-left grandchild.

A delete in the right child could cause this right-side shortening.
Case #1: Left-left due to right deletion

- Same single rotation as when an insert in the left-left grandchild caused imbalance due to X becoming taller
- But here the “height” at the top decreases, so more rebalancing farther up the tree might still be necessary
Case #2: Left-right due to right deletion

- Same double rotation when an insert in the left-right grandchild caused imbalance due to c becoming taller
- But here the “height” at the top decreases, so more rebalancing farther up the tree might still be necessary
No third right-deletion case needed

So far we have handled these two cases:
left-left

But what if the two left grandchildren are now both too tall (h+1)?
• Then it turns out left-left solution still works
• The children of the “new top node” will have heights differing by 1 instead of 0, but that’s fine
And the other half

• Naturally two more mirror-image cases (not shown here)
  – Deletion in left causes right-right grandchild to be too tall
  – Deletion in left causes right-left grandchild to be too tall
  – (Deletion in left causes both right grandchildren to be too tall, in which case the right-right solution still works)

• And, remember, “lazy deletion” is a lot simpler and might suffice for your needs