AVL Trees

CSE 373
Data Structures and Algorithms

The AVL Balance Condition

Left and right subtrees of every node have equal heights differing by at most 1

Define: \( \text{balance}(x) = \text{height}(x\text{.left}) - \text{height}(x\text{.right}) \)

AVL property: \(-1 \leq \text{balance}(x) \leq 1, \text{ for every node } x\)

- Ensures small depth
  - Will prove this by showing that an AVL tree of height \( h \) must have a lot of (i.e. \( \Theta(2^h) \)) nodes
- Easy to maintain
  - Using single and double rotations

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 \( \Theta(\log n) \)

Ordering property
- Same as for BST

Proving Shallowness Bound

Let \( S(h) \) be the min # of nodes in an AVL tree of height \( h \)
Claim: \( S(h) = S(h-1) + S(h-2) + 1 \)
Solution of recurrence: \( S(h) = \Theta(2^h) \)
(like Fibonacci numbers)

Testing the Balance Property

We need to be able to:
1. 
2. 
3. 

NULLs have height \(-1\)
AVL trees: find, insert

- **AVL find:**
  - same as BST find.

- **AVL insert:**
  - same as BST insert, *except* may need to “fix” the AVL tree after inserting new value.

AVL tree insert

Let $x$ be the node where an imbalance occurs.

Four cases to consider. The insertion is in the
1. left subtree of the left child of $x$.
2. right subtree of the left child of $x$.
3. left subtree of the right child of $x$.
4. right subtree of the right child of $x$.

*Idea:* Cases 1 & 4 are solved by a single rotation.
Cases 2 & 3 are solved by a double rotation.

Fix: Apply Single Rotation

AVL Property violated at this node ($x$)

Single Rotation:
1. Rotate between $x$ and child

Single rotation in general

Height of tree before? Height of tree after? Effect on Ancestors?
Single rotation example

Bad Case #2

Insert(1)
Insert(6)
Insert(3)

Fix: Apply Double Rotation

AVL Property violated at this node (x)

Double rotation in general

Double rotation, step 1

Double rotation, step 2
Imbalance at node X

Single Rotation
1. Rotate between x and child

Double Rotation
1. Rotate between x’s child and grandchild
2. Rotate between x and x’s new child

Insert into an AVL tree: a b e c d

Insertion into AVL tree
1. Find spot for new key
2. Hang new node there with this key
3. Search back up the path for imbalance
4. If there is an imbalance:
   case #1: Perform single rotation and exit
   case #2: Perform double rotation and exit
   Both rotations keep the subtree height unchanged.
   Hence only one rotation is sufficient!

Easy Insert
Insert(3)

Hard Insert (Bad Case #1)
Insert(33)
**Single Rotation**

![Diagram of Single Rotation]

**Hard Insert (Bad Case #2)**

Insert(18)

Unbalanced?

How to fix?

![Diagram of Hard Insert (Bad Case #2)]

**Single Rotation (oops!)**

![Diagram of Single Rotation (oops!)]

**Double Rotation (Step #1)**

![Diagram of Double Rotation (Step #1)]

**Double Rotation (Step #2)**

![Diagram of Double Rotation (Step #2)]