Trees  
(Today: AVL Trees)

Chapter 4 in Weiss

Balanced BST

Observation
- BST: the shallower the better!
- For a BST with \( n \) nodes
  - Average height is \( O(\log n) \)
  - Worst case height is \( O(n) \)
- Simple cases such as insert(1, 2, 3, ..., \( n \)) lead to the worst case scenario

Solution: Require a Balance Condition that
1. ensures depth is \( O(\log n) \) – strong enough!
2. is easy to maintain – not too strong!

Potential Balance Conditions
1. Left and right subtrees of the root have equal number of nodes
2. Left and right subtrees of the root have equal height
3. Left and right subtrees of every node have equal number of nodes
4. Left and right subtrees of every node have equal height

The AVL Balance Condition
Adelson-Velskii and Landis (AVL)
AVL balance property:
Left and right subtrees of every node have heights differing by at most 1
- Ensures small depth
  - Will prove this by showing that an AVL tree of height \( h \) must have a lot of (i.e. \( O(2^h) \)) nodes
- Easy to maintain
  - Using single and double rotations

The AVL Tree Data Structure

Structural properties
1. Binary tree property (0, 1, or 2 children)
2. Heights of left and right subtrees of every node differ by at most 1
Result:
Worst case depth of any node is: \( O(\log n) \)

Ordering property
- Same as for BST

Is this an AVL Tree?

NULLs have height −1
Deciding AVLness

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: $S(h) = O(2^h)$ (like Fibonacci numbers)

Proving Shallowness Bound

AVL tree of height $h=4$ with the min # of nodes (12)

An AVL Tree

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.

Bad Case #1

Insert(6)
Insert(3)
Insert(1)
Fix: Apply Single Rotation

AVL Property violated at this node (x)

Single Rotation:
1. Rotate between x and child

Single rotation in general

Single rotation example

Bad Case #2

Fix: Apply Double Rotation

AVL Property violated at this node (x)

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

Double rotation in general
Double rotation, step 1

Double rotation, step 2

Imbalance at node X

Insert into an AVL tree: a b e c d

Single and Double Rotations:

Insertion into AVL tree

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

Inserting what integer values would cause the tree to need a:
1. single rotation?
2. double rotation?
3. no rotation?

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)

Unbalanced?

How to fix?

Hard Insert (Bad Case #1)

Insert(33)

Unbalanced?

How to fix?

Single Rotation

Hard Insert (Bad Case #2)

Insert(18)

Unbalanced?

How to fix?

Single Rotation (oops!)

Double Rotation (Step #1)
Double Rotation (Step #2)