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 \text{insert}(1, 2, 3, \ldots, n) \) lead to the worst case scenario

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

Not Strong Enough
1. Left and right subtrees of the root have equal number of nodes
2. Left and right subtrees of the root have equal height

Too Strong
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 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 nodes (i.e., $O(2^h)$)

• 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
Recursive Height Calculation

*Recall:* height is max number of edges from root to a leaf

What is the height at A?

Note: height(null) = -1
AVL Tree?
AVL Tree?
Proving Shallowness Bound

Let \( S(h) \) be the min # of nodes in an AVL tree of height \( h \)

Trees of height \( h = 1, 2, 3 \ldots \)

Claim: \( S(h) = S(h-1) + S(h-2) + 1 \)

Solution of recurrence: \( S(h) = O(2^h) \) (like Fibonacci numbers)

AVL tree of height \( h=4 \) with the min # of nodes (12)
Testing the Balance Property

We need to be able to:

1. Track Balance
2. Detect Imbalance
3. Restore Balance

NULL has a height of $-1$
An AVL Tree

Track height at all times.
AVL trees: find, insert, delete

• AVL find:
  – Same as BST find.

• AVL insert:
  – Same as BST insert, except you need to check your balance and may need to “fix” the AVL tree after the insert.

• AVL delete:
  – We’re not going to talk about it, but same basic idea. Delete it, check your balance, and fix it.
Bad Case #1

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

AVL Property violated at this node (x)

Intuition: 3 must become root

Single Rotation:
1. Rotate between x and child
Bad Case #2

Insert(1)
Insert(6)
Insert(3)
Single Rotation Does Not Work

AVL Property violated at this node (x)

That ain’t even a BST!!!!!
Single Rotation Does Not Work

AVL Property violated at this node (x)

It’s a BST, but it’s unbalanced
Fix: Apply Double Rotation

AVL Property violated at this node (x)

Intuition: 3 must become root

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

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!
Single rotation in general

\[
X < b < Y < a < Z
\]

Height of tree before?  Height of tree after?  Effect on Ancestors?
Double rotation in general

\[ h \geq 0 \]

\[
\begin{array}{c}
\text{a} \\
\text{b} \\
\text{c} \\
\text{W} \\
\text{X} \\
\text{Y} \\
\text{Z}
\end{array}
\]

\[
\begin{array}{c}
\text{W} \ < \ b \ < \ X \ < \ c \ < \ Y \ < \ a \ < \ Z
\end{array}
\]

Height of tree before? Height of tree after? Effect on Ancestors?
Easy Insert

Insert(3)
Easy Insert

Insert(3)
Easy Insert

Insert(3)
Insert(3)
Easy Insert

Insert(3)

Unbalanced?  No
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 $x$. 
Case #1: left-left insertion

Insert on left child’s left
Case #1: repair with single rotation

\[ X < b < Y < a < Z \]

single rotation
Single rotation example
Single rotation example
Single rotation example
Case #2: left-right insertion

Insert on left child’s right
Case #2: left-right insertion

Let’s break subtree Y into pieces:

Insert on left child’s right (at U or V)
Case #2: left-right insertion

Let’s break subtree Y into pieces:

Insert on left child’s right (at U or V)
Case #2: left-right insertion

Let’s break subtree Y into pieces:

Insert on left child’s right (at U or V)
Can also do this in two rotations

First rotation

\[
X < b < U < c < V < a < Z
\]
Second rotation
Double rotation example
Double rotation example
Double rotation example
Double rotation example
Double rotation example
Double rotation example
Case #3: right-left insertion

Double rotation
Case #4: right-right insertion

Single rotation
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.
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
     [Illustration of Zig-zig rotation]
   - case #2: Perform double rotation and exit
     [Illustration of Zig-zag rotation]

Both rotations keep the subtree height unchanged. Hence only one rotation is sufficient!
Single and Double Rotations:

Consider inserting one of \{1, 4, 6, 8, 10, 12, 14\}
Which values require:

1. single rotation?

2. double rotation?

3. no rotation?
AVL complexity

What is the worst case complexity of a find?

O(log n)

What is the worst case complexity of an insert?

O(log n)

What is the worst case complexity of buildTree?

O(n log n)