Potential Balance Conditions

What are some candidate balance conditions on BST's?
1. Left and right subtrees of the root have equal number of nodes

2. Left and right subtrees of the root have equal height

Potential Balance Conditions

3. Left and right subtrees of every node have equal number of nodes

4. Left and right subtrees of every node have equal height

Balancing Trees

- Many algorithms exist for keeping trees balanced
  - Adelson-Velskii and Landis (AVL) trees
  - Splay trees and other self-adjusting trees
  - B-trees and other multiway search trees (for very large trees)

- Today we will talk about AVL trees…
The AVL Tree Data Structure

Ordering property
- Same as for BST

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 height: $O(\log n)$

Recursive Height Calculation

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

What is the height at A?

$$\max(h_{\text{left}}, h_{\text{right}}) + 1$$

Note: height(null) = -1

Minimum Size of an AVL Tree

- $m(h) =$ minimum number of nodes in an AVL tree of height $h$
- Base cases
  - $m(0) = 1, m(1) = 2, m(2) = 4$
- Induction
  - $m(h) = m(h-1) + m(h-2) + 1$
- Can prove:
  - $m(h) > \phi^h - 1$
  - $\phi$ is the golden ratio, $(1+\sqrt{5})/2$
Aside: The Golden Ratio

Since the Renaissance, many artists and architects have proportioned their work (e.g., length/height) to approximate the golden ratio.

\[
\frac{a + b}{a} = \frac{a}{b} = \phi
\]

The golden section:

\[
a + b = a = \phi
\]

\[a + b \text{ is to } a \text{ as } a \text{ is to } b\]

Substitute \(a = b\phi\) in left eqn, solve:

\[
\phi = \frac{1 + \sqrt{5}}{2} \approx 1.62
\]

Note: \(\phi^2 = \left(\frac{1 + \sqrt{5}}{2}\right)^2 = \frac{1 + 2\sqrt{5} + 5}{4} = \frac{6 + 2\sqrt{5}}{4} = \frac{3 + \sqrt{5}}{2} = 1 + \frac{\sqrt{5}}{2} = 1 + \phi\)

Proof that \(m(h) > \phi^h - 1\)

- Base cases \(h=0,1:\)
  \[m(0) = 1 > \phi^0 - 1 = 0 \quad m(1) = 2 > \phi^1 - 1 = 0.62\]

- Assume true for \(h-2\) and \(h-1:\)
  \[m(h-2) > \phi^{h-2} - 1 \quad m(h-1) > \phi^{h-1} - 1\]

- Induction step:
  \[m(h) = m(h-1) + m(h-2) + 1 > (\phi^{h-1} - 1) + (\phi^{h-2} - 1) + 1\]
  \[= \phi^{h-1} (\phi + 1) - 1 = \phi^{h-2} (\phi^2) - 1 = \phi^h - 1\]
  \[\Rightarrow m(h) > \phi^h - 1\]

Maximum Height of an AVL Tree

Suppose we have \(n\) nodes in an AVL tree of height \(h\).
We can now say:

\[n \geq m(h) > \phi^h - 1 \Rightarrow \phi^h < n + 1\]

What does this say about the complexity of \(h\)?

\[h \leq \log_\phi(n + 1) \in \Theta(\log n)\]

\[h = 2^{\frac{\log_\phi{n + 1}}{\log_\phi{\phi}}} \geq \frac{\log_2(n + 1)}{\log_2(\phi)} \in \Omega(\log n)\]

Testing the Balance Property

We need to be able to:

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

Is this AVL tree balanced?
How about after insert(30)?
**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.

We will consider the 4 fundamental insertion cases...

**Case #1: Left-Left Insertion (zig)**

Insert on left child’s left

**Case #1: Repair with Single Rotation**

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

Single rotation

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

Case #2: left-right insertion

Insert on left child’s right

Case #2: repair with single rotation?

Case #2: trying again

Let’s break subtree Y into pieces:

Insert on left child’s right (at U or V)

Are we better off now?
Case #2: trying again

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

Second rotation

Double rotation example
Double rotation, step 1

Double rotation, step 2

Case #3: right-left insertion

Case #4: right-right insertion
AVL tree case summary

Let \( \alpha \) be the node where an imbalance occurs.

Four cases to consider. The insertion below \( \alpha \) is in the

1. left child’s left subtree. (zig)
2. left child’s right subtree. (zig-zag)
3. right child’s left subtree. (zig-zag)
4. right child’s right subtree. (zig)

Cases 1 & 4 are solved by a single rotation:

1. Rotate between \( \alpha \) and child

Cases 2 & 3 are solved by a double rotation:

1. Rotate between \( \alpha \)’s child and grandchild
2. Rotate between \( \alpha \) and \( \alpha \)’s new child

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?

Insertion procedure:

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:
   cases #1,#4: Perform single rotation and exit
   cases #2,#3: Perform double rotation and exit

Both rotations restore subtree height to value before insert. Hence only type of rotation is sufficient per insert!

More insert examples

Insert(33)
Single Rotation

More insert examples

Suppose we didn’t do that last insert.

Now do: Insert(18)

Unbalanced?
How to fix?

Single Rotation (oops!)

Double Rotation
More insert examples

Insert(3)

Unbalanced?
How to fix?

AVL complexity

What is the worst case complexity of a find?

$O(\log n)$

What is the worst case complexity of an insert?

BST insert $O(\log n)$ + repair $O(\log n)$

$O(\log n)$

What is the worst case complexity of buildTree?

$O(n \log n)$