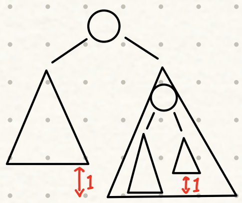
AVL Trees

An AVL tree is a BST where the height of each subtree of an internal node differs by at most 1.



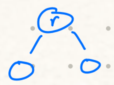
The height of each of a nodes subtree must be the same or differ by at most 1

The maximum height of an AVL tree

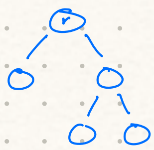
Let $n(h)$ = min # nodes in an AVL tree with height h

$n(0) = 1$

$n(1) = 3$



$n(2) = 5$



$n(h) = 1 + n(h-1) + n(h-2)$

root

one of the subtrees, as tall as $h-1$, this is the tallest possible subtree.

For minimum # of nodes, subtree 2 will be as short as possible while still satisfying AVL conditions.

So recursively, for each internal node count 1 (root), then one subtree height $h-1$ and a corresponding subtree $n-2$ to find lowest possible # of nodes within each subtree, repeat

Solve Recurrence

<u>Assume h is even</u>

$n(0) = 1$
$n(1) = 3$
$n(h) = 1 + n(h-1) + n(h-2)$

we want to show that $n(h)$ grows exponentially with h, and thus h grow logaritmically with n, this is difficult to expand with both $(h-1)$ and $(h-2)$ so we'll look for a simpler inequality

Since $h-2$ is less than $h-1$, replace $h-1$ with $h-2$ and change it from equality to inequality

$n(h) > 1 + 2n(h-2)$ for $h > 1$ → from here, expand recurrence

$n(h) > 1 + 2(1 + 2n(h-4))$

$n(h) > 1 + 2(1 + 2(1 + 2n(h-6)$ → Simplify to identify pattern

$n(h) > 1 + 2(1 + 2 + 4n(h-6)$

$n(h) > 1 + 2 + 4 + 8n(h-6)$ → Find pattern

$n(h) > 1 + 2^1 + 2^2 + 2^3 + \ldots + 2^i + 2^{i+1} n(h - 2(i+1))$ → after i times

The reason the last term is to the $i+1$ is because after 1 iteration $(i=1)$ of unrolling the term is 2^2 which is $i+1$

Stop unrolling when height hits 0, assume this is after i iterations

At that point: $h - 2(i+1) = 0$

$2(i+1) = h$

$i = \frac{h}{2} - 1$ → substitute back in

$n(h) > 1 + 2^1 + 2^2 + \cdots + 2^i + 2^{i+1} n(0)$ → $n(0) = 1$

$n(h) > 1 + 2^1 + 2^2 + \cdots + 2^i + 2^{i+1}$ → Simplify geometric series $(a = 1, r = 2, n = i+2)$

$n(h) > 2^{i+2} - 1$ → sub in i

$n(h) > 2^{\frac{h}{2} - 1 + 2} - 1$

$n(h) > 2^{\frac{h}{2} + 1} - 1$ → Solve for h

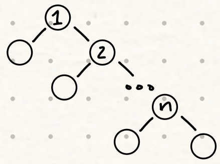
$n(h) + 1 > 2^{\frac{h}{2} + 1}$ → log both sides

$\log_2(n(h) + 1) > \frac{h}{2} + 1$

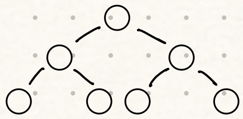
$h < 2(\log_2(n(h) + 1) - 1$ $\therefore$ $h = O(\log n)$

Balanced Vs. Unbalanced Trees

Unbalaced



Balanced



If an ordered dictionary ADT is implemented with a binary search tree, every operation will be O(height). The maximum height of a BST with n nodes is n-1, occurring if the tree is fully unbalanced

Therefore, an ordered dictionary is more efficiently implemented with an AVL which has O(logn) vs BST O(n).