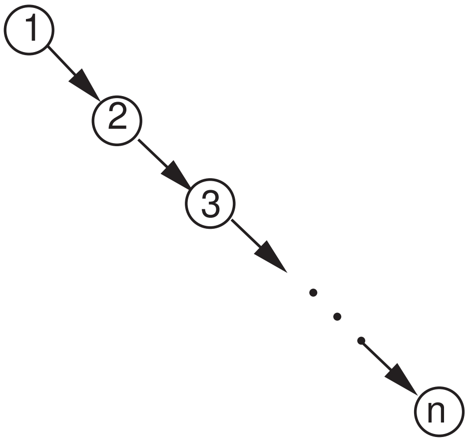
AVL-Trees

In ordinary binary trees, trees become unbalanced after many insert/delete operations.

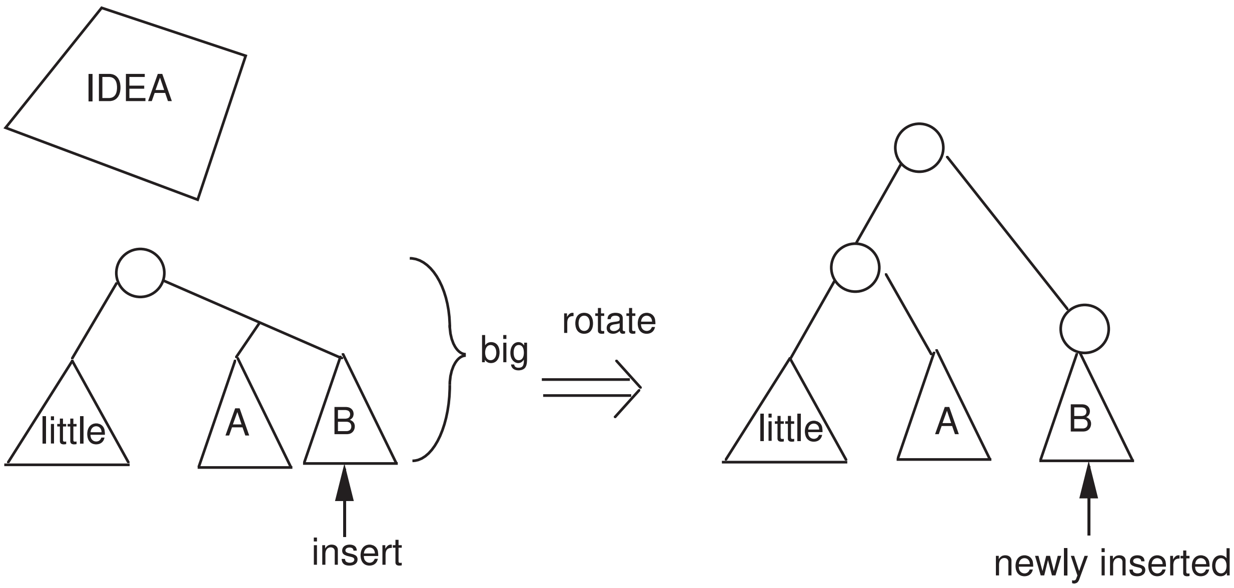
Example. insert 1; insert 2; insert 3; insert 4; ... insert n produce:



- This is nothing but a linear list.

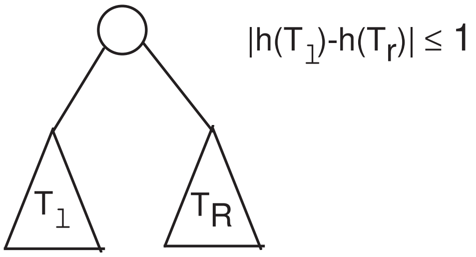
- Although: if we insert randomly, the resulting tree is usually (almost) balanced.

Question: How can we "guarantee" $O(\log n)$ performance for all of insert/delete/search?

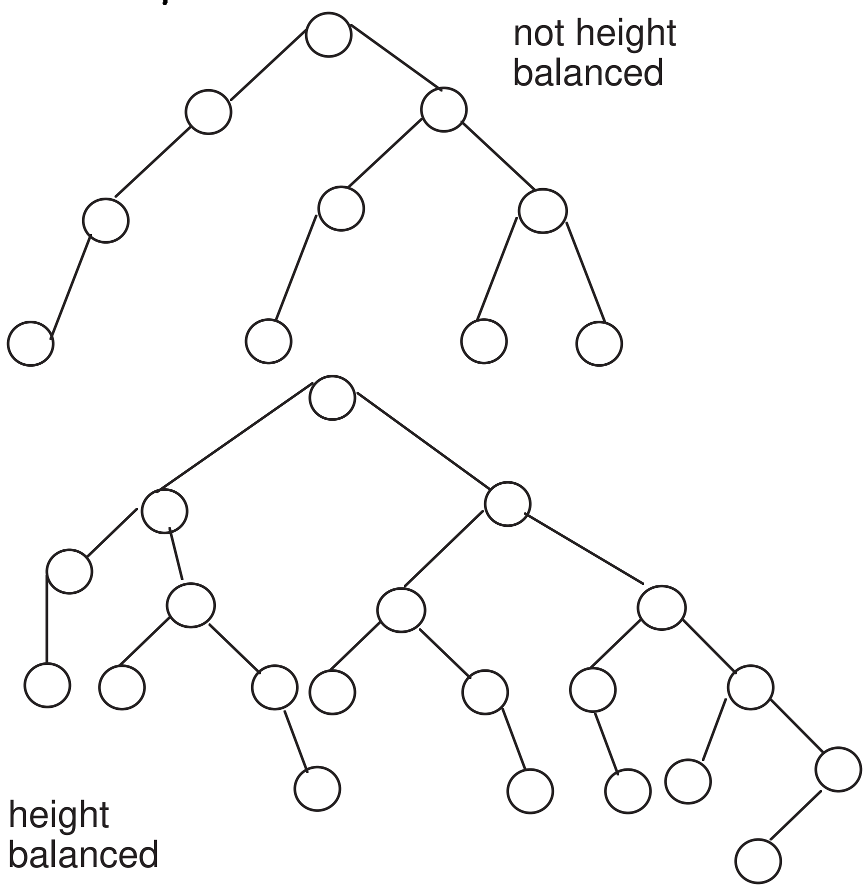


Definition. An AVL-tree (Height-Balanced tree) is a binary search tree such that for each node v, the heights of the left subtree and the right subtree of v differ by at most 1.

For every node v,

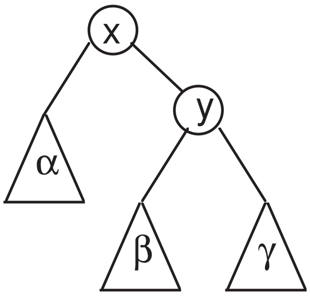


Examples.

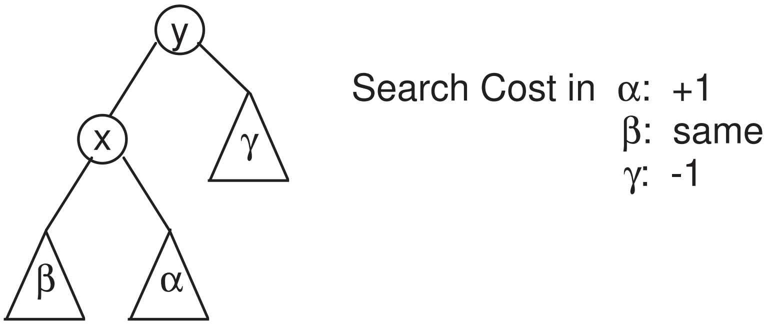


Balancing

Single Rotation



To move γ up at cost of moving α down

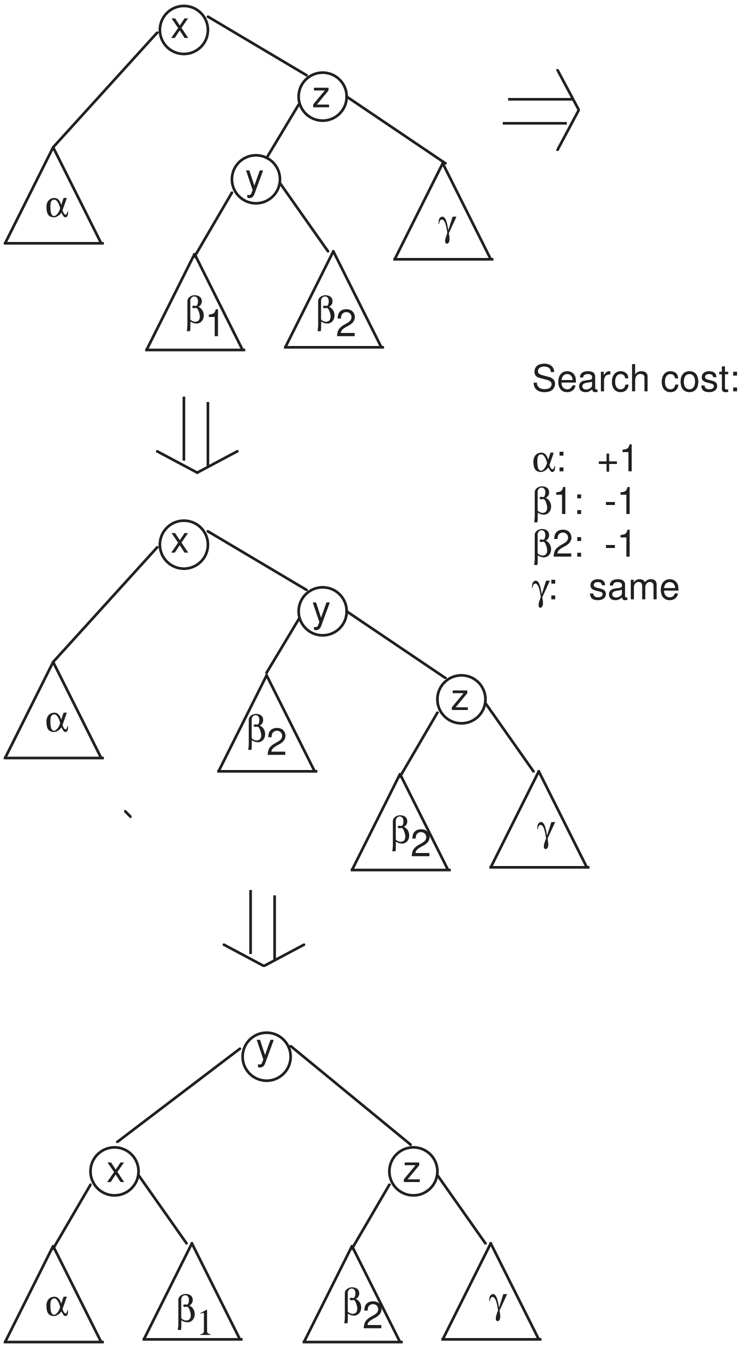


Apply this when

- γ tree already too tall and we insert to γ, or

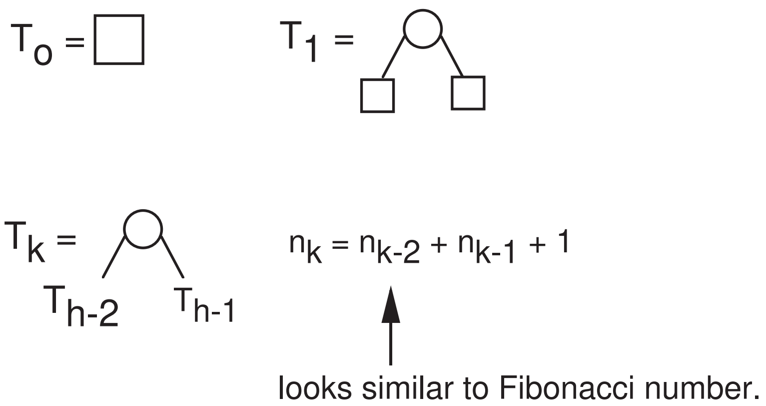
- α tree already too short and we delete from α.

<u>Double Rotation</u>. Move β up at cost of α



Use double rotation when the β tree is already "too tall" and when we need to (1) insert to β, or (2) delete from α.

- How high can a height balanced tree on n internal nodes be? or: How few internal nodes can a height balanced tree of height h have?

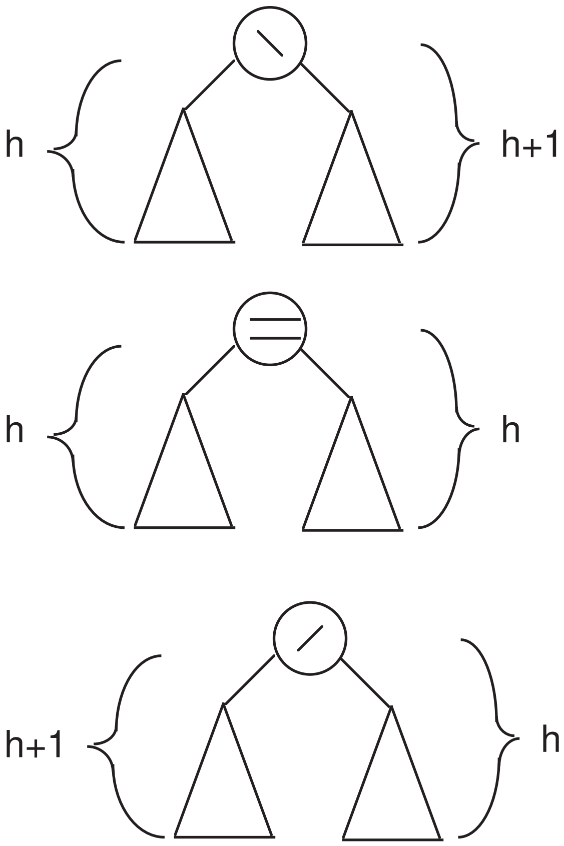


In fact, n_k is the $(k+2)$nd Fibonacci number minus 1. And we have (calculation omitted) $\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+2} + O(1)$

Solving for h in terms of n, we have

$$h \leq \frac{1}{\log\left(\frac{1+\sqrt{5}}{2}\right)} \log(n+1) + O(1) \approx 1.44 \log n$$

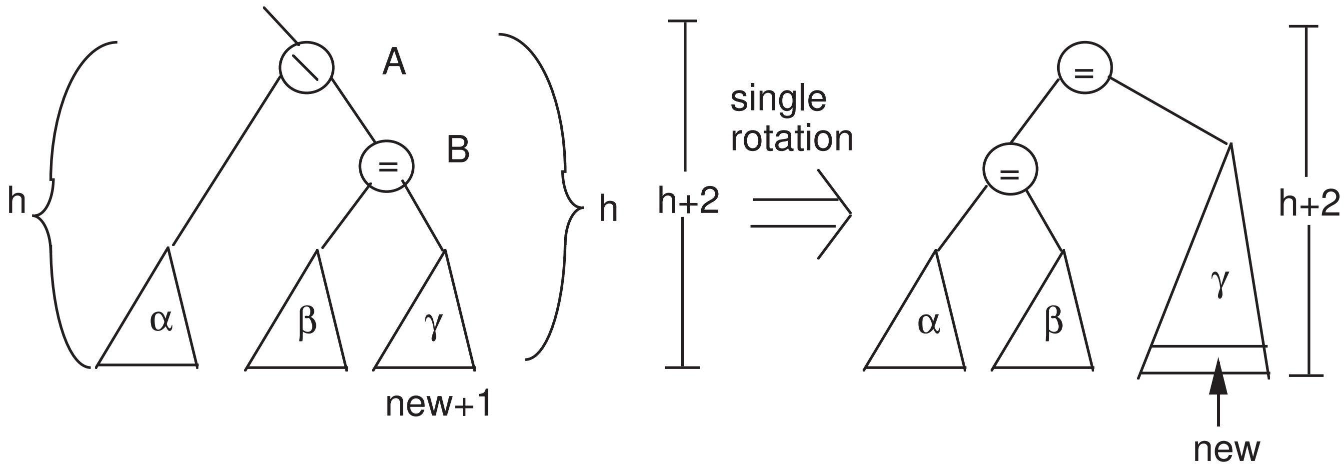
We will use the following convenient notations.



- Can set a flag, for each node.

Two cases may cause rotation to occur.

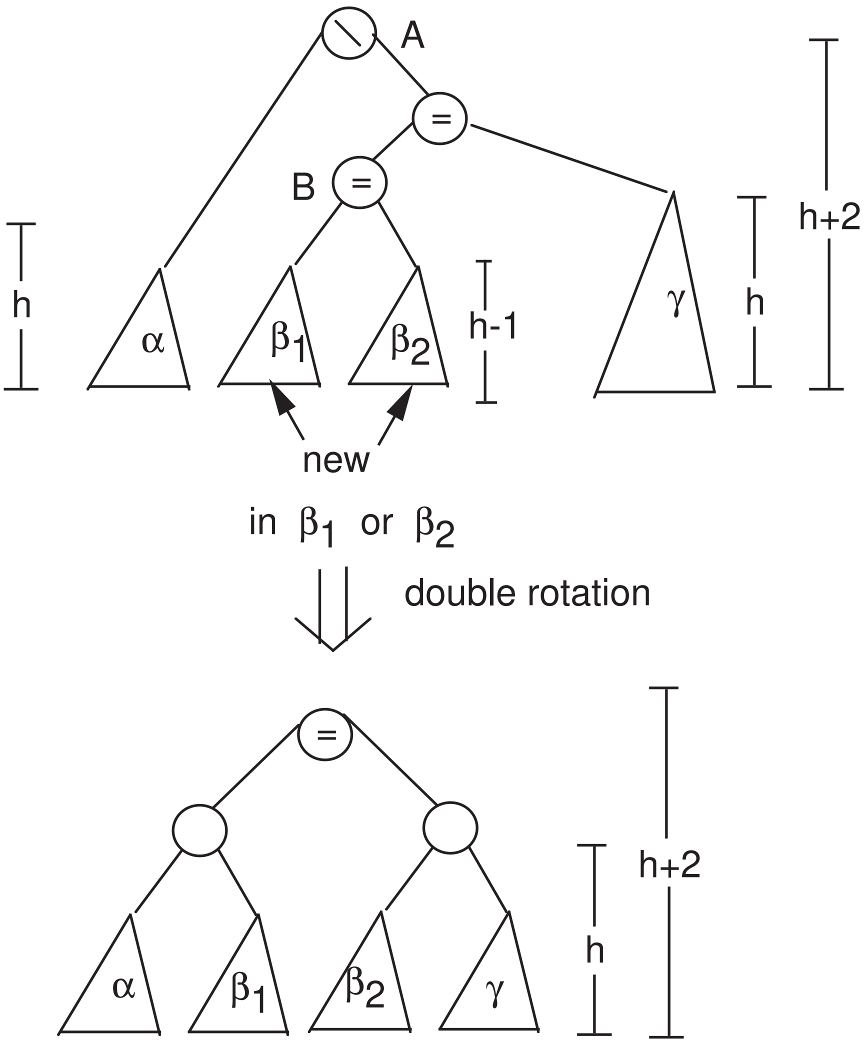
- "Outside insertion"



α, β, γ each has height h originally. γ grows.

This whole subtree doesn't grow in height as a consequence of the insert/rotation action. So the balance factors *do not change* at other places (outside this subtree) in the tree!!

- "Inside insertion"



- Again, this subtree does not increase in height, so no need for change balance factors outside this subtree (at point A)