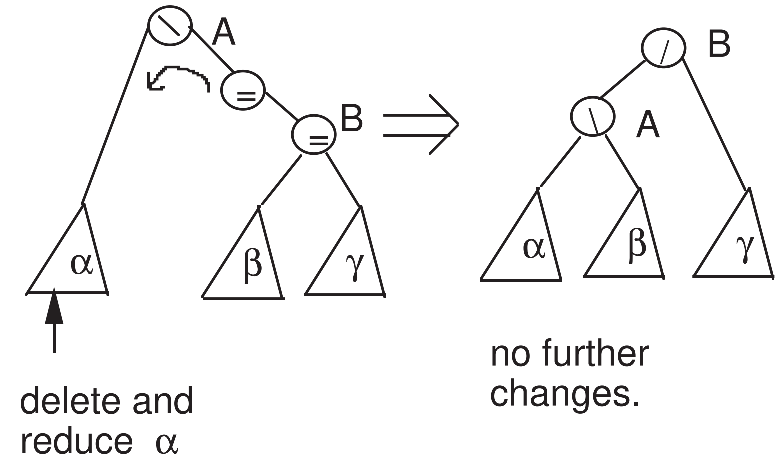
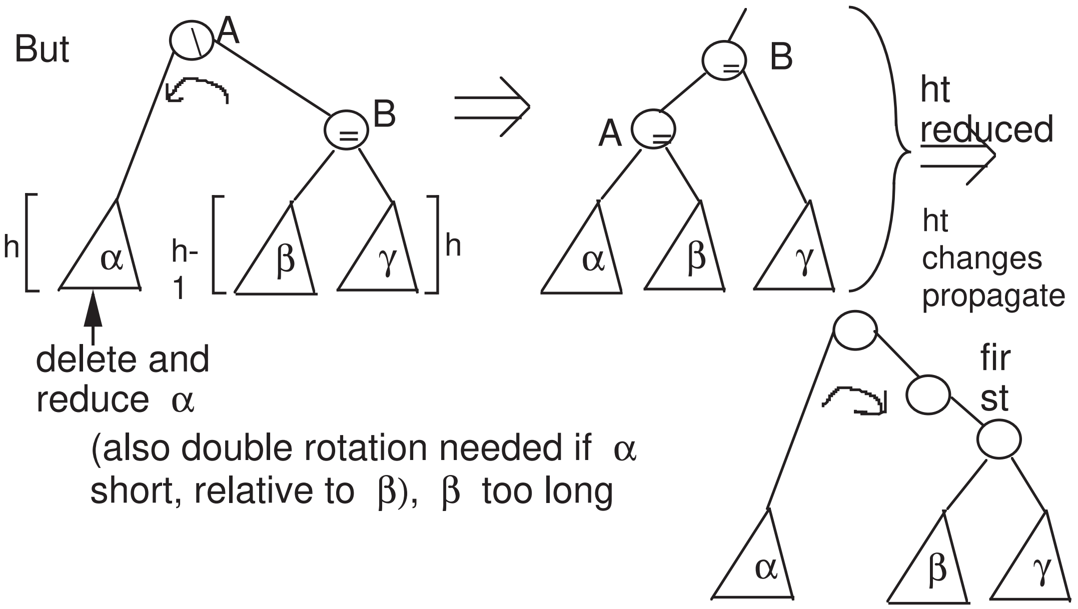
Deletion for AVL-tree

- Similar to insertion.
- But it may cause as many as $\lceil \frac{h}{2} \rceil$ rotations.

For example



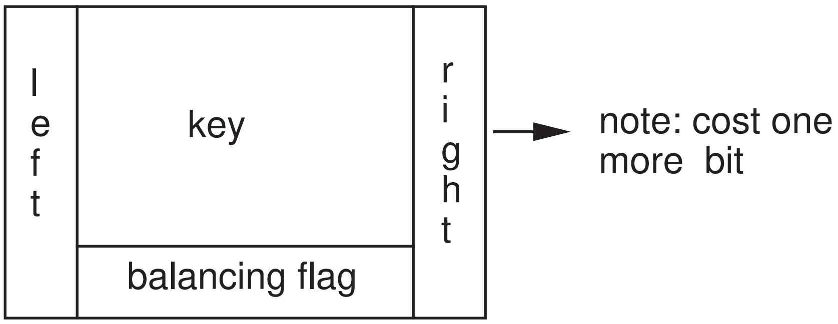


Summary

With AVL trees (ht balanced), we can perform searches in $\leq 1.44 \log n$ probes, and perform insertion/deletion in $O(\log n)$ time.

- Insert: at most 1 single or double rotation
- Delete: up to $\log n$ rotations
- Getting the code right is harder

To make it work: Each node



/	left taller by 1
=	equal
\	right taller by 1

Insert.

- Search for leaf where new value should go. Insert, with balancing flag "=".
- Back up the tree, adjusting flags as long as necessary and possible.
- If at some node, balancing is not possible (for example, left grows by 1, but flag is already "/"), then perform a single or double rotation.

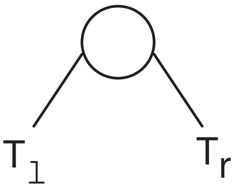
Delete

- find the node, delete it, replace the spot by "inorder successor".
- adjust the inbalance caused by deleting the "inorder successor"; back up the tree, adjusting balancing factor; if not possible do rotations.

Some comment on AVL trees: Although for each "delete", we could use $O(\log n)$ rotations, we can prove that <u>any</u> sequence of n insertions/deletions will cause only $O(n)$ rotations, instead of $O(n \log n)$.

Other balancing approaches:

Instead of "height" balanced trees, we can define "weight" balanced trees:



$$\beta(T) = \begin{cases} 1/2, & \text{if } T = \square \\ \dfrac{|T_l|}{|T_r|}, & \text{o,w.} \end{cases}$$

At each node, we require

$$\alpha \le \beta(T) \le 1 - \alpha$$

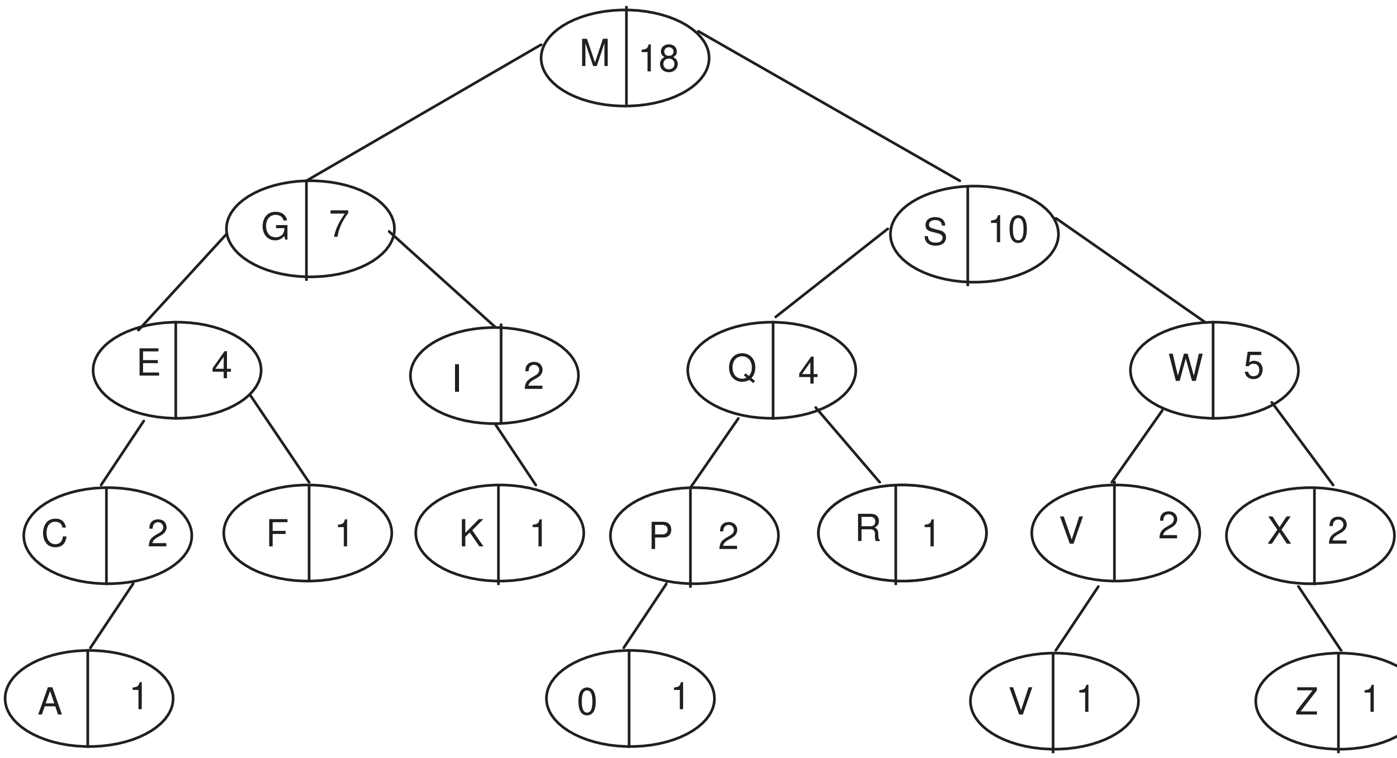
for some proper α.

Some operations on balanced trees:

- Find smallest element

  start at root, go left. ($\log n$ time)

- Find k-th smallest element, using weight balanced tree. Example:



In each node, the letter on the left is the label of the node, and the number indicates the number of elements in its subtree (inclusive).

Select (k,root):

{find the k-th smallest in a tree rooted at "root"}

```
case   root↑.left↑.weight:k-1 of
 "<"   select(k,root↑.left)
 "="   return(root)
 ">"   select(k-1-root↑.left↑.weight,root↑.right)
```