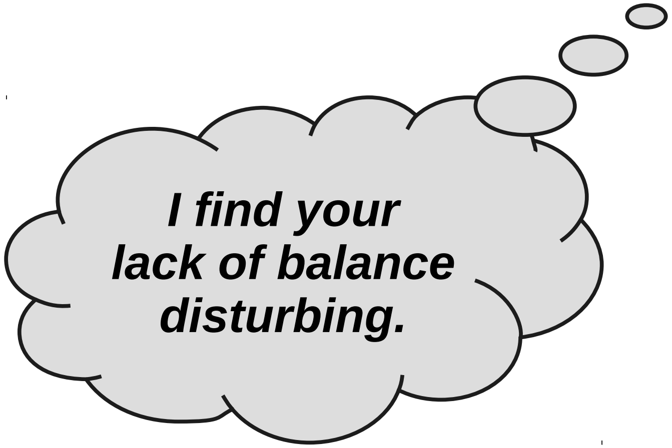
CS 240
Fall 2014

Mike Lam, Professor



Balanced (AVL) Trees

Review

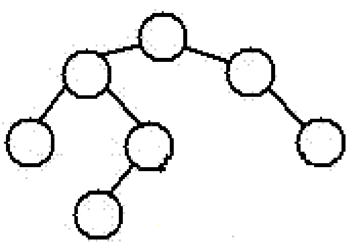
- Binary Search Trees (BSTs)
  - Ordered binary tree
  - Insertions, lookups, and removals
  - Operations are $O(h)$ where h is the height of the tree
  - For mostly-random insertions and deletions, $h \approx \log n$
  - For other situations, we need to use a more "balanced" binary tree implementation
    - For heaps, we enforced balance by enforcing completeness
    - For BSTs, this would be much more expensive
    - Tradeoff between balance and speed of operations

Issues

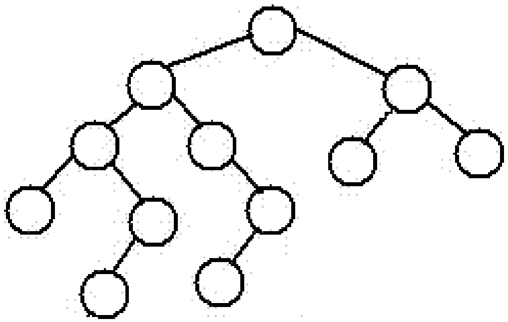
- How much should we rebalance?
  - How often? How many nodes? How strict?
  - How do we measure balance?
- We could rebalance the entire tree after every insertion
  - This would lead to *O(n log n)* insertion times
  - Essentially re-build the tree every time
- Goal: faster insertions and "good enough" balance
  - AVL trees (easiest to understand)
  - Red-Black trees
  - Many others...

AVL Trees

- Adelson-Velsky and Landis (AVL) Tree
  - Named after inventors G. M. Adelson-Velsky and E. M. Landis (1962)
  - *Height-balance property*: heights of children differ by at most one
  - Insert/remove operations enforce this property using *tree rotations*



A height-balanced Tree



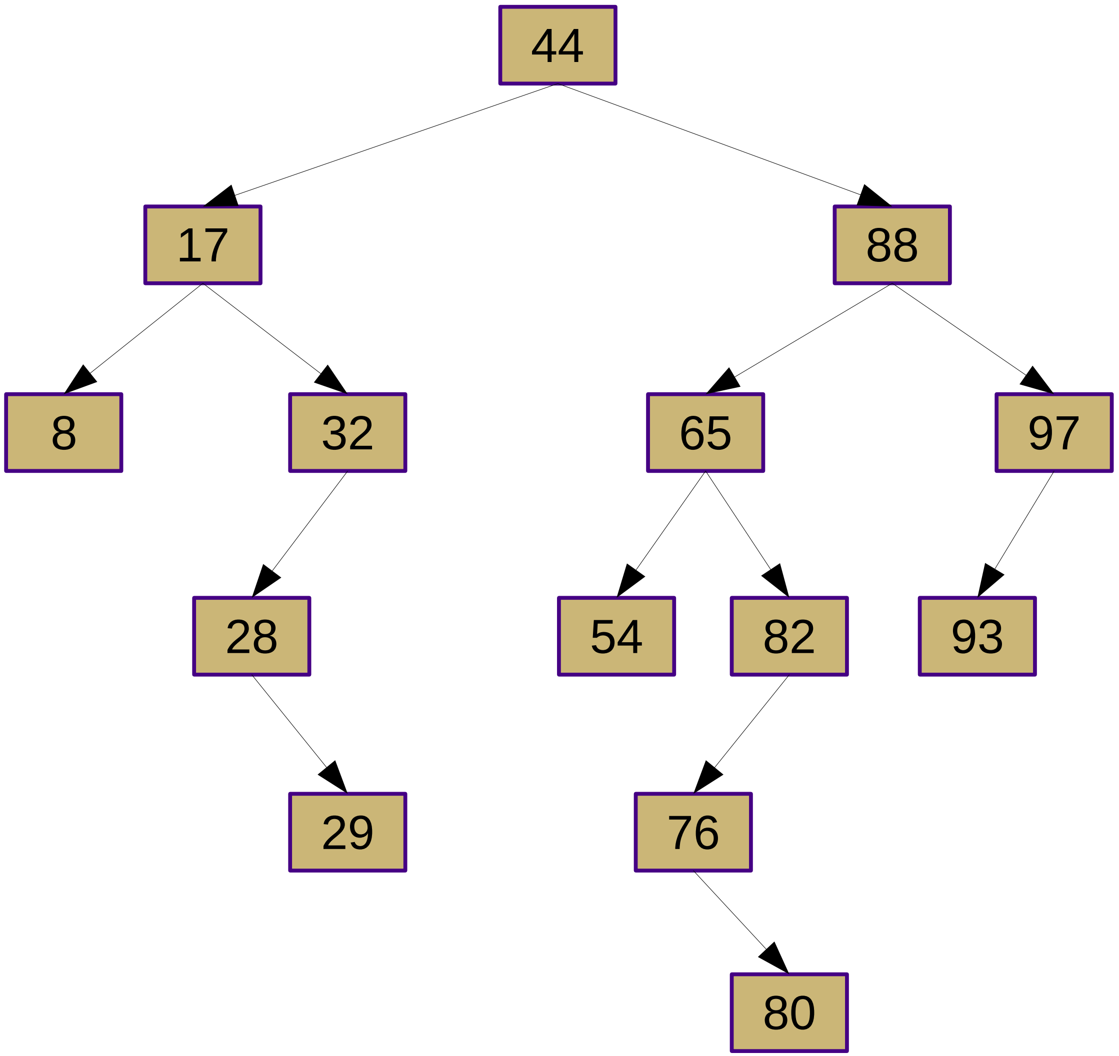
Not a height-balanced tree

Image from: `http://www.geeksforgeeks.org/how-to-determine-if-a-binary-tree-is-balanced/`

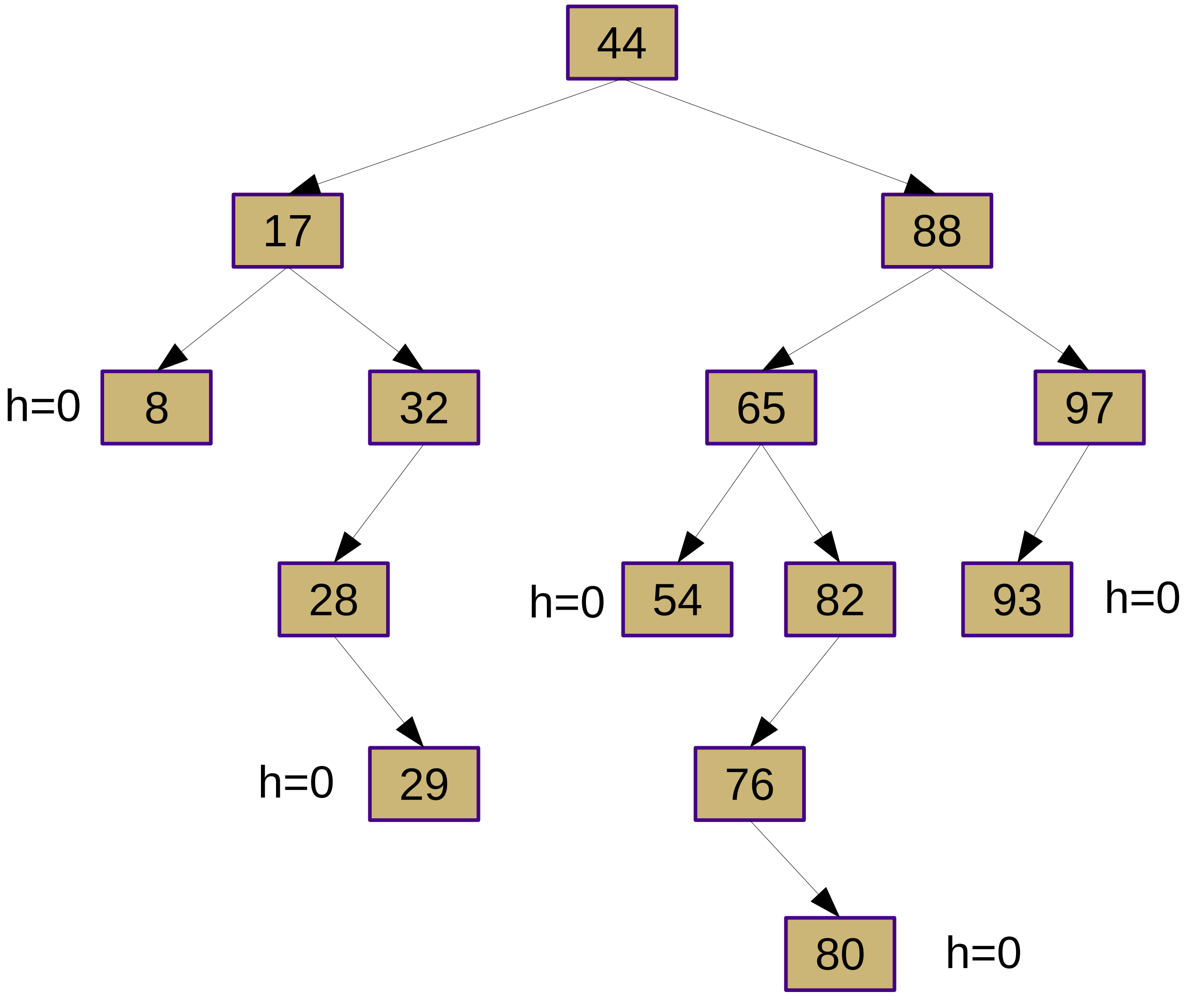
Tree Height

- Caution: Textbook changes their definition of *height*!
  - Former: # of edges from node to furthest leaf
  - Latter: # of nodes from (and including) node to furthest leaf
  - We can continue using the old definition by defining the height of an empty tree to be -1

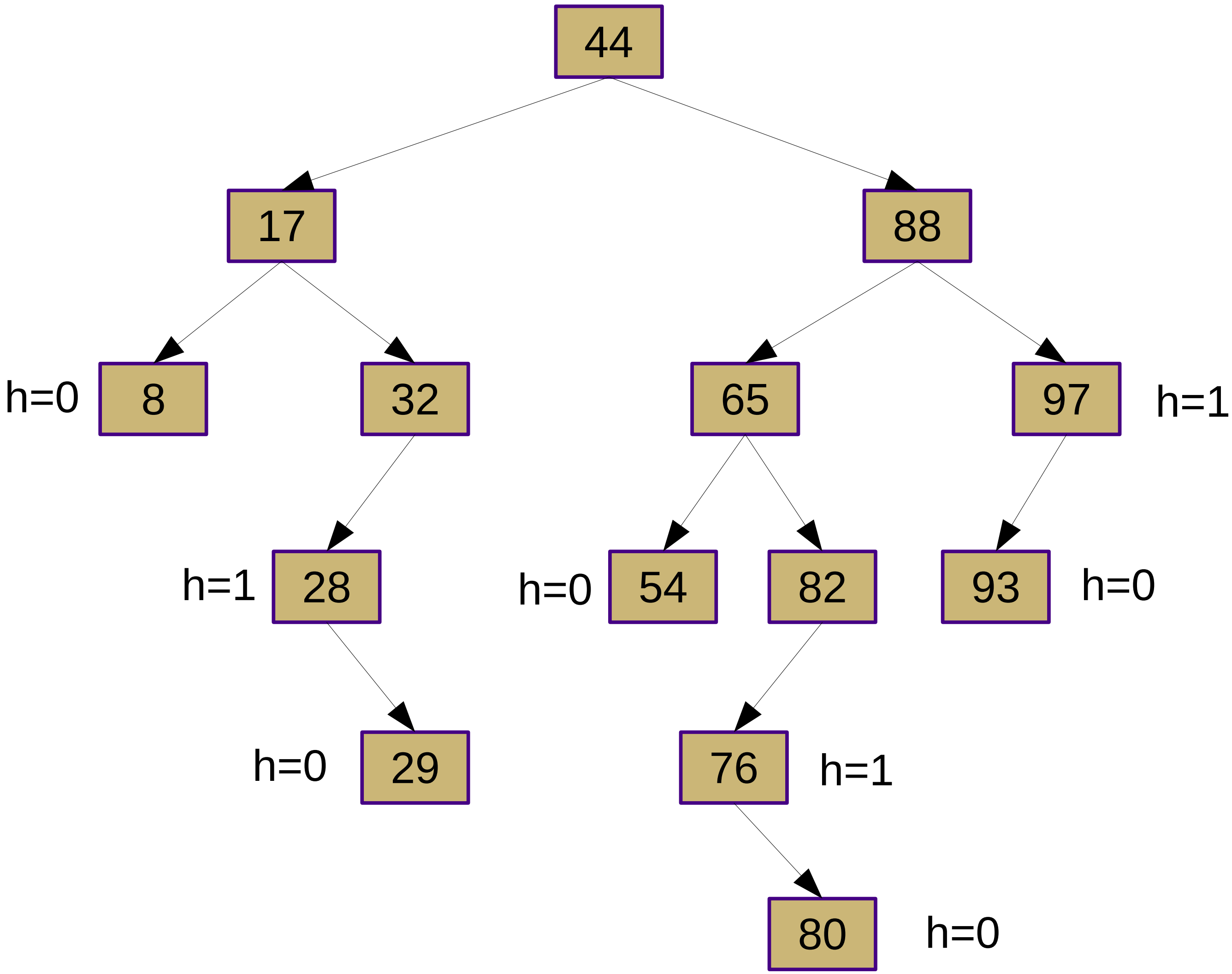
Binary Search Tree



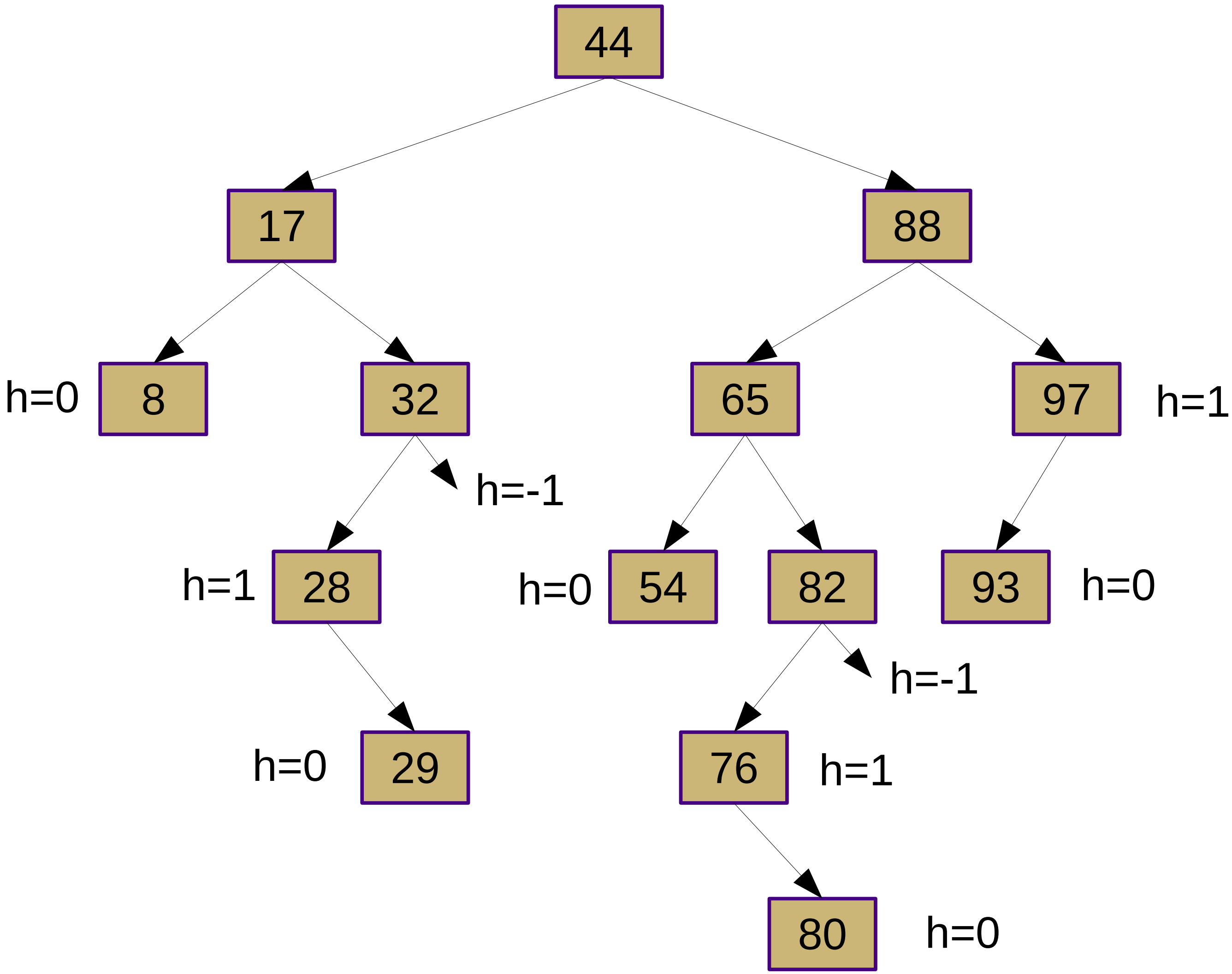
Binary Search Tree



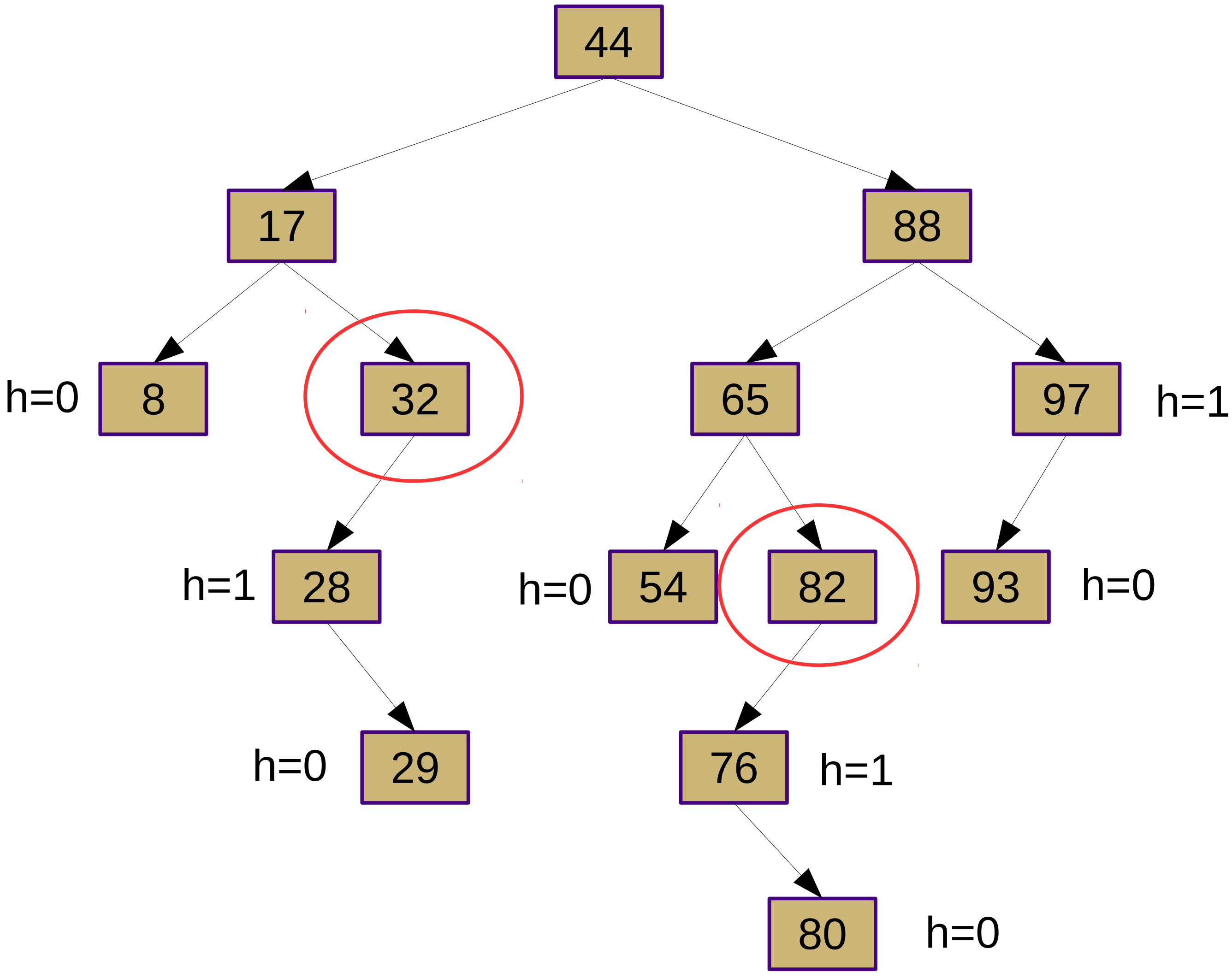
Binary Search Tree



Binary Search Tree

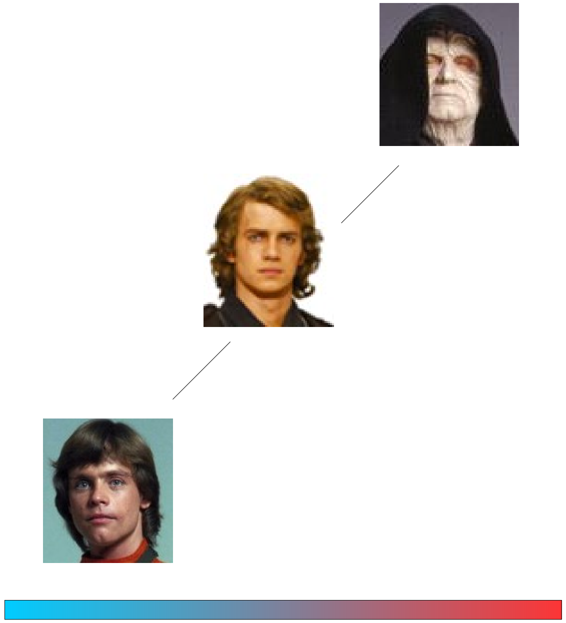


Binary Search Tree



Imbalances

- How do we fix imbalances?
  - Need to re-arrange tree
- Solution: Rotations!
- Example:



Imbalances

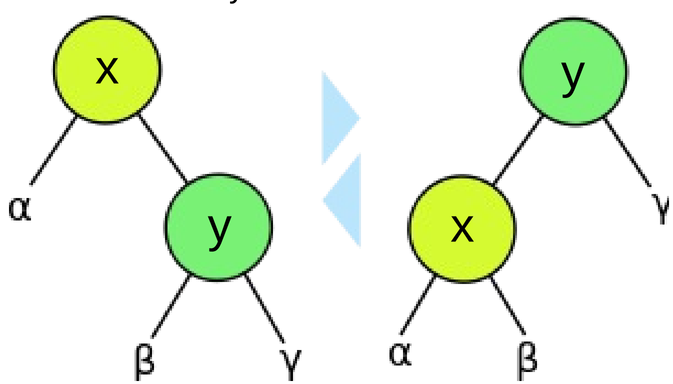
- How do we fix imbalances?
  - Need to re-arrange tree
- Solution: Rotations!
- Example:

Rotations

- Rotations
  - Single rotation (below)
  - Single/double rotations (right)
    - Four cases of *trinode restructuring*

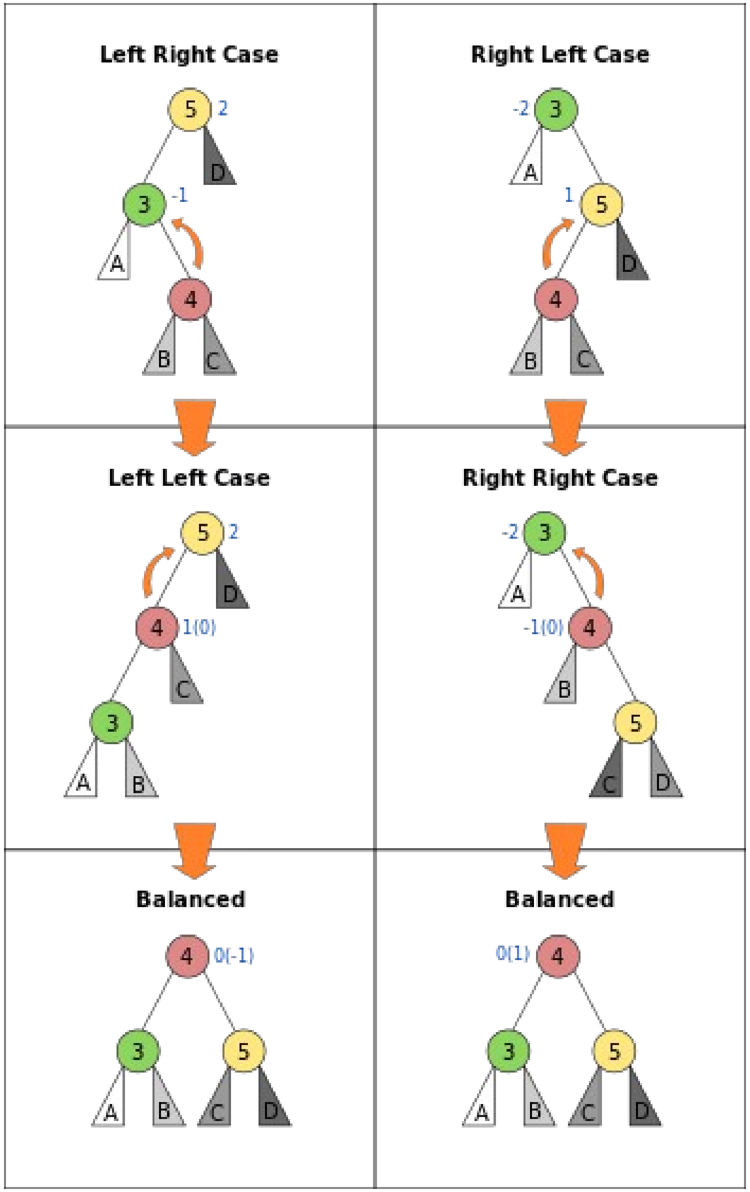
"left rotation"

```
x.right = y.left
y.left = x
```




```
y.left = x.right
x.right = y
```

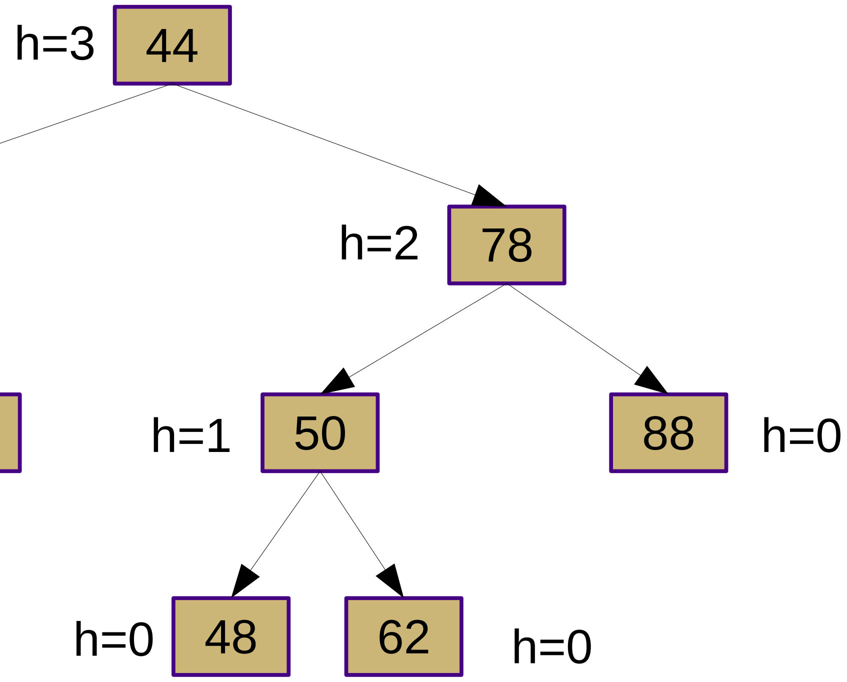
"right rotation"



AVL Trees

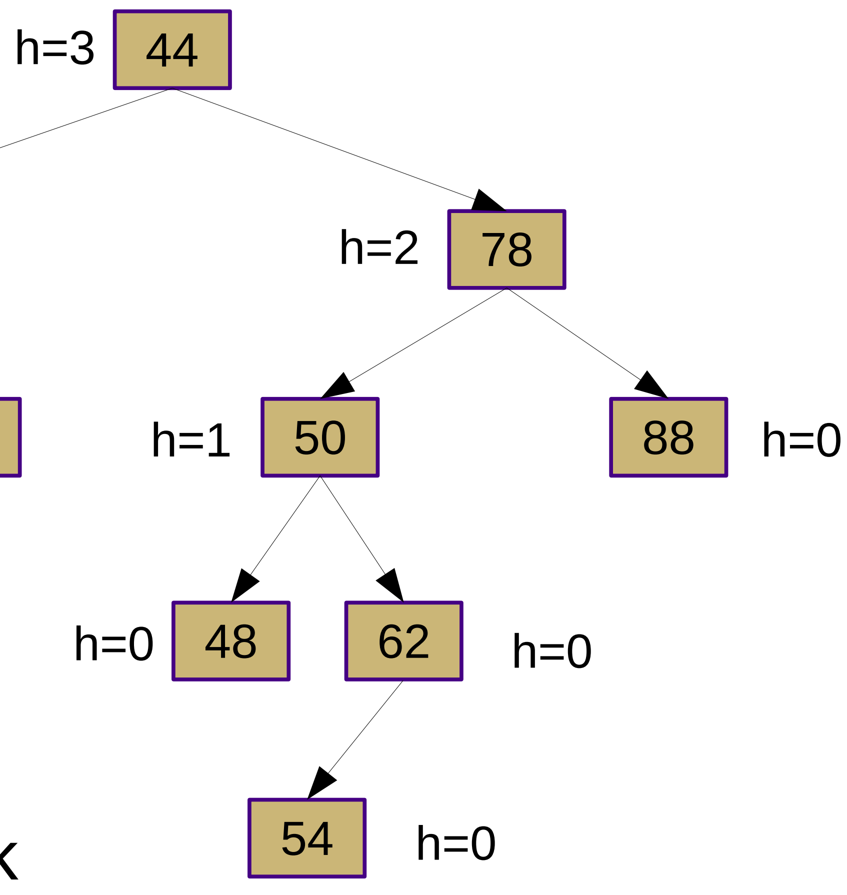
- Insertion
  - Insert into BST as usual
  - Check ancestors of new node for imbalances
  - Fix imbalances via trinode restructuring
- Removal
  - Remove from BST as usual
  - Check ancestors of removed node for imbalances
  - Fix imbalances via trinode restructuring

AVL Tree



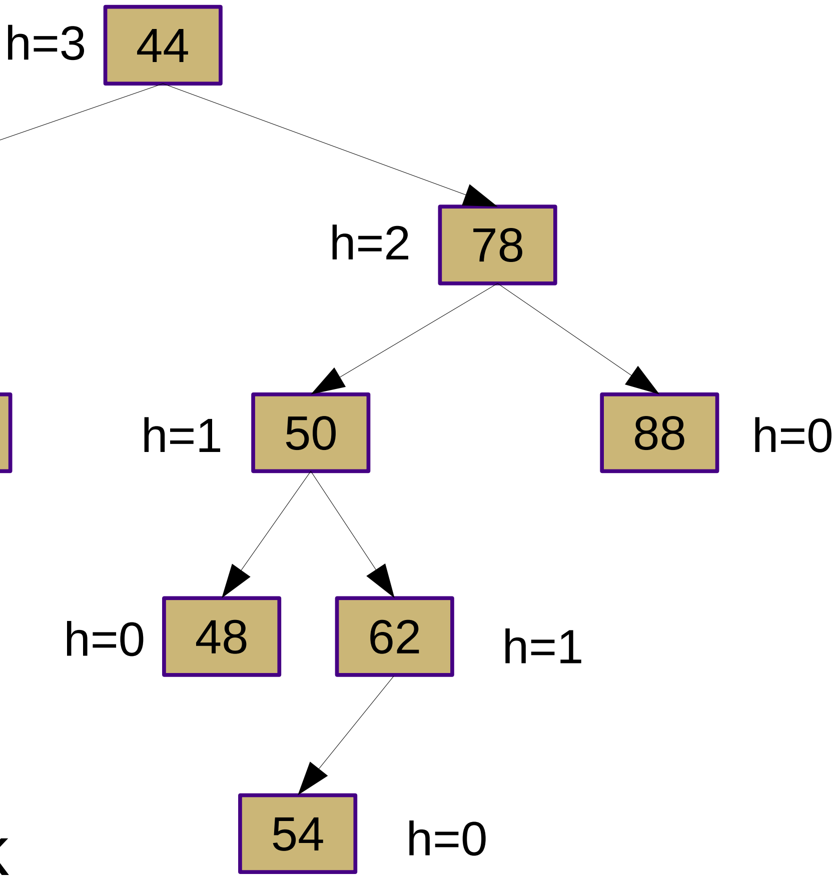
Insert 54

AVL Tree



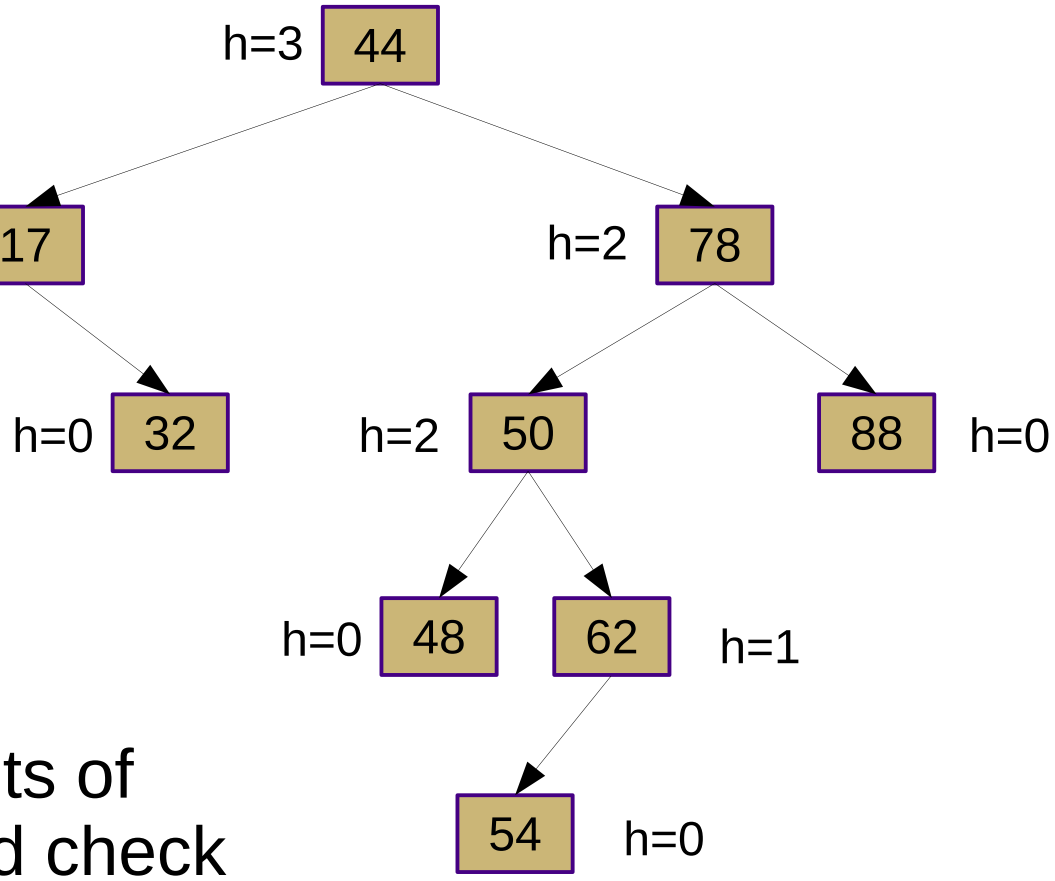
Update heights of ancestors and check for imbalances

AVL Tree



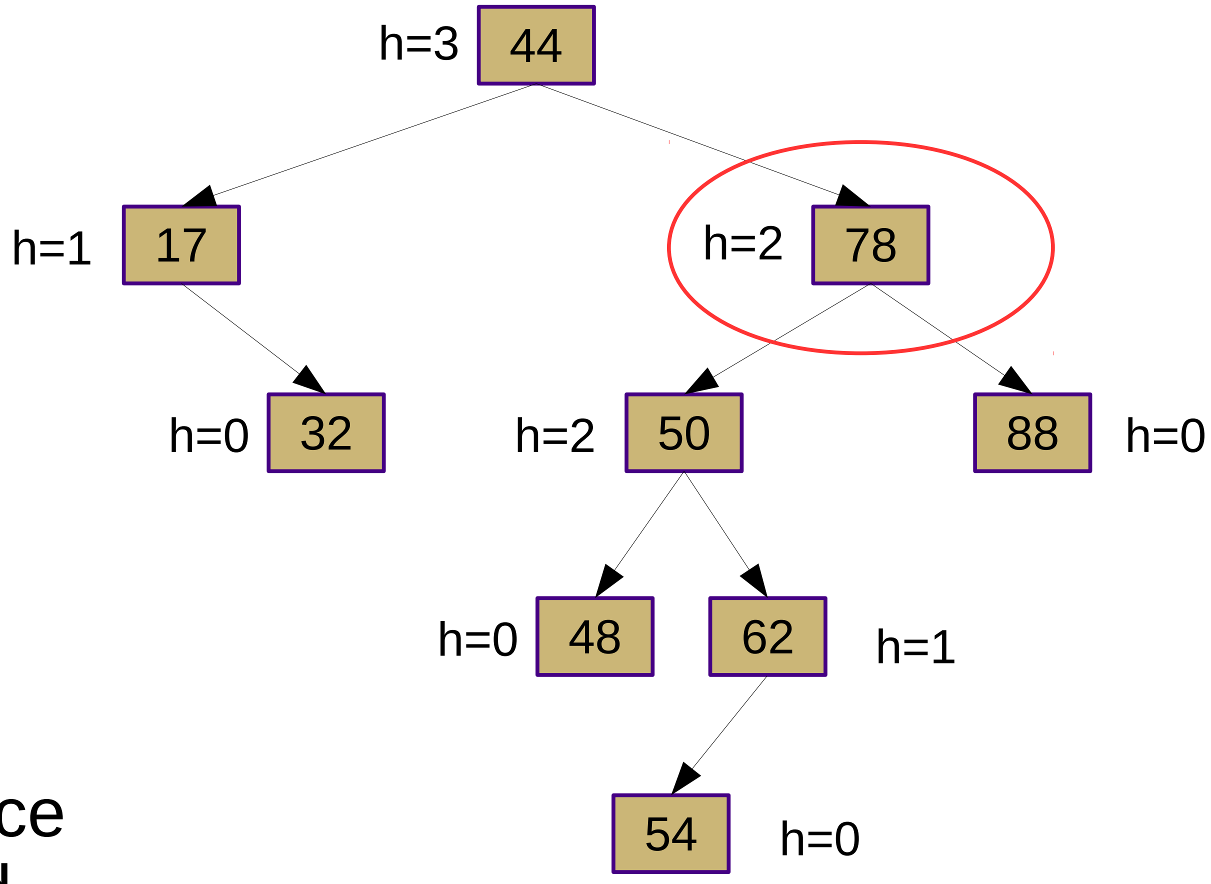
Update heights of ancestors and check for imbalances

AVL Tree



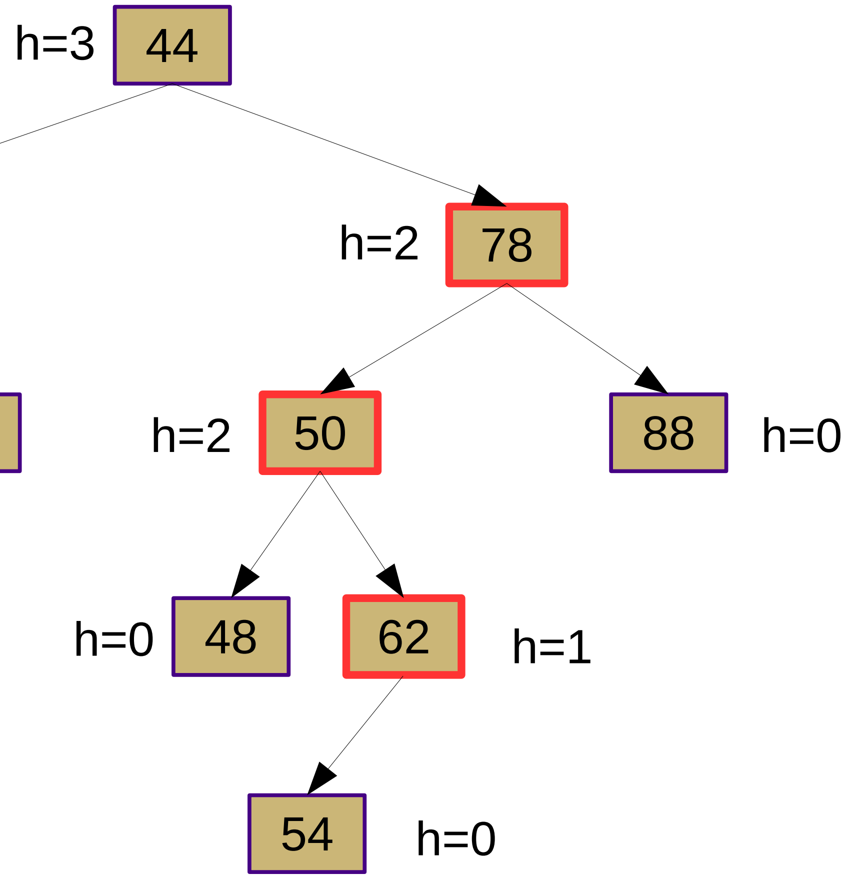
Update heights of ancestors and check for imbalances

AVL Tree



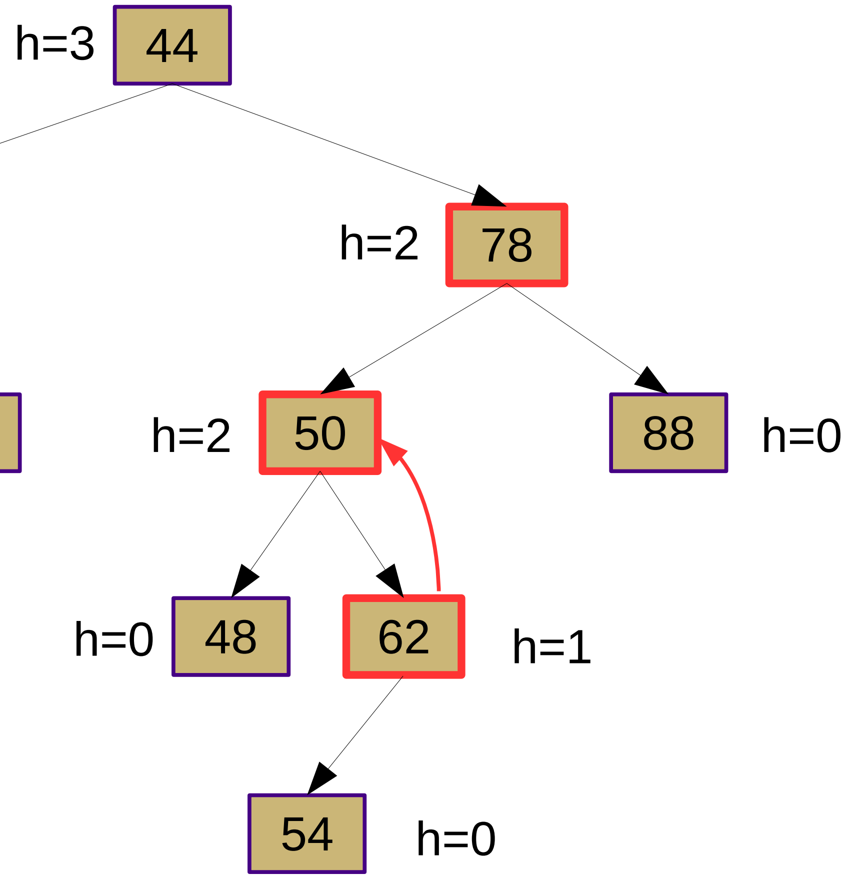
Imbalance detected

AVL Tree



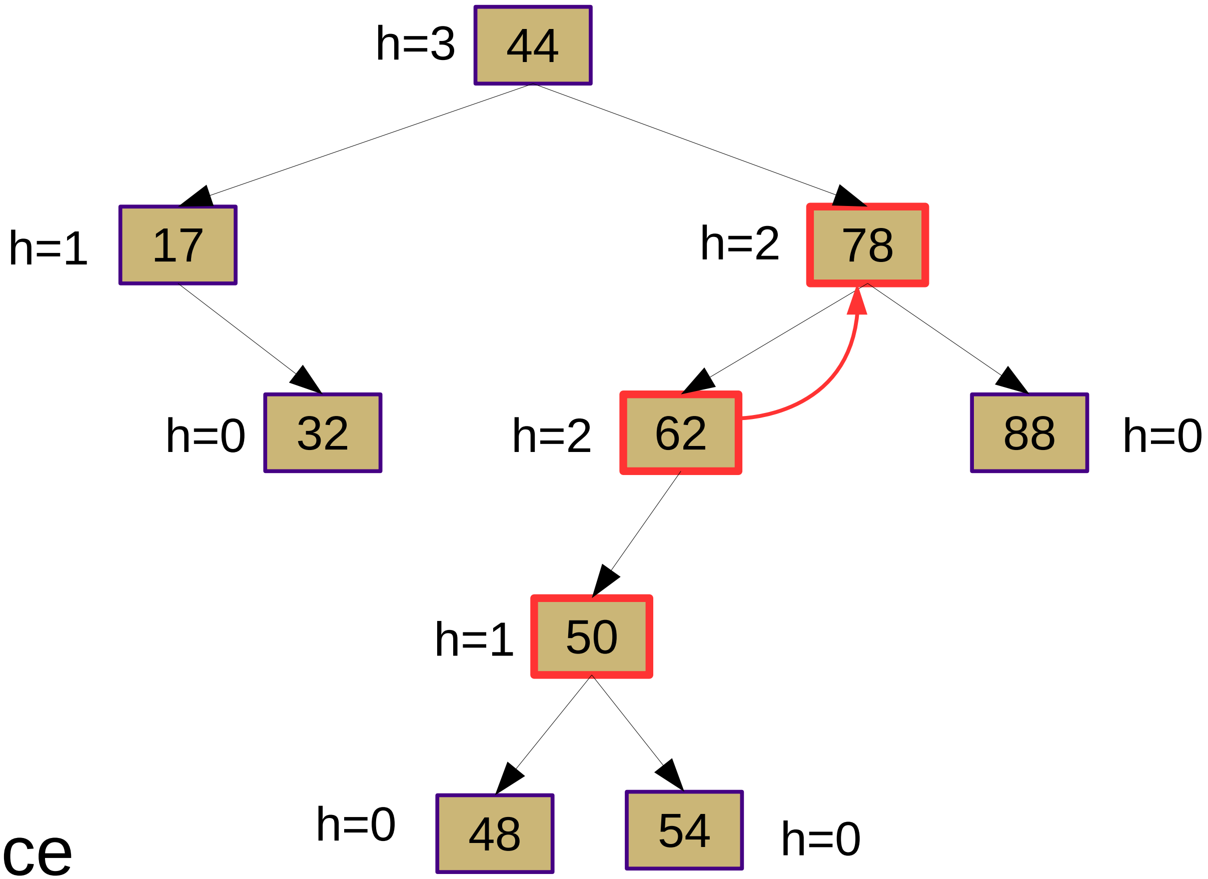
Nodes involved

AVL Tree



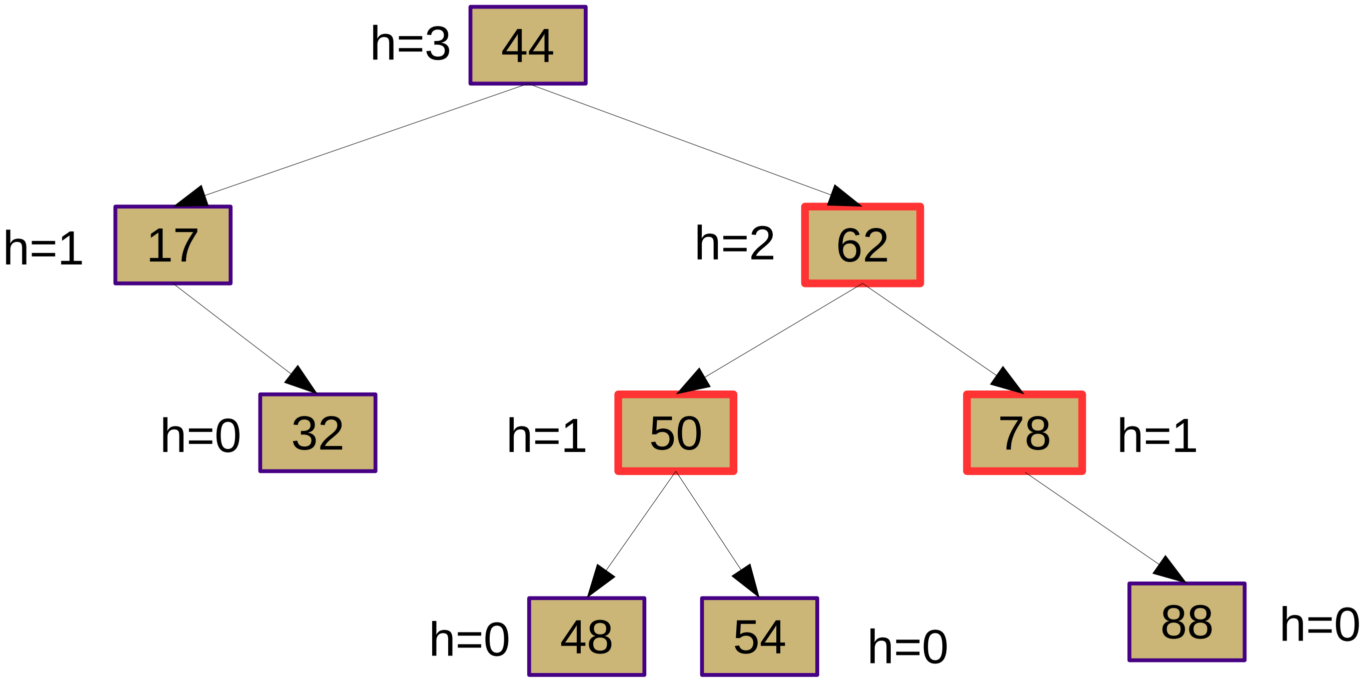
Rebalance

AVL Tree



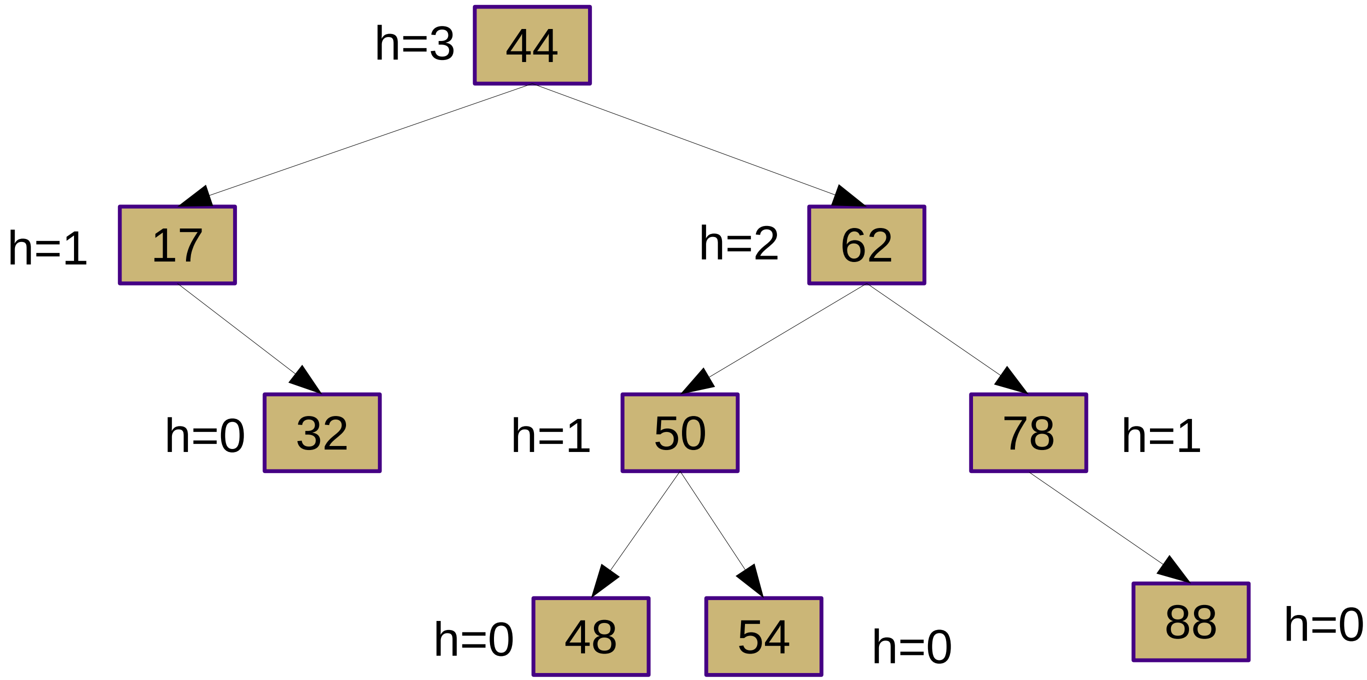
Rebalance

AVL Tree



Balanced subtree;
continue up tree to root

AVL Tree



Done!

Exercises

- 3, 9, 12, 5, 4, 1
- 10, 5, 7, 15, 9, 25

To check your answers:

`http://webdiis.unizar.es/asignaturas/EDA/AVLTree/avltree.html`

AVL Tree Analysis

- In general: $n(h) = 1 + n(h-1) + n(h-2)$
  - AVL tree with minimal number of nodes has one node and two subtrees: one with height $h-1$ and one with height $h-2$
- This is a Fibonacci progression
  - Exponential w.r.t. height: $n(h)$ *is* $\Omega(2^h)$
  - Thus, h is $O(\log n)$
- See Section 11.3 for formal justification
  - Stricter bound: $h < 2 \log n + 2$

Alternative: Red-Black Trees

- Coloring scheme
  - Root is colored black
  - All children of a red node must be colored black (no "double reds")
  - All nodes with zero or one children have the same number of black-colored ancestors
- Path from root to furthest leaf is no more than twice as long as the path from root to nearest leaf
- Less-strictly balanced
- Faster insertion/removal but slower lookups