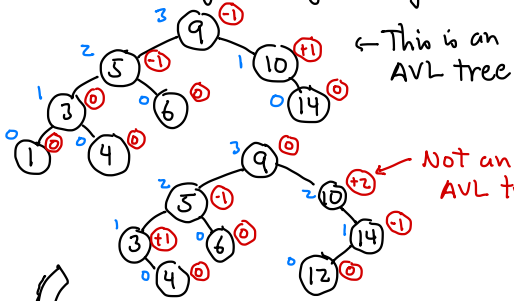


Balance factor:

$$\text{bal}(v) = \text{hgt}(v.\text{right}) - \text{hgt}(v.\text{left})$$



AVL Height Balance

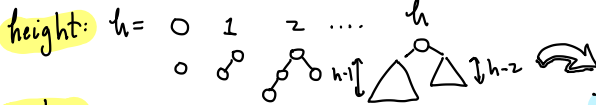
- for each node v , the heights of its subtrees differ by ≤ 1 .

AVL tree: A binary search tree that satisfies this condition



Does this imply $O(\log n)$ height?

Worst cases:



nodes: $n = 1, 2, 4, 7, 12, 20, \dots$
 $n+1 = 2, 3, 5, 8, 13, 21, \dots$

Recall: $F_0 = 0, F_1 = 1, F_h = F_{h-1} + F_{h-2}$

Conjecture: Min no. of nodes in AVL tree of height h is $F_{h+3} - 1$

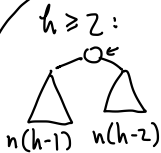
Theorem: An AVL tree of height h has at least $F_{h+3} - 1$ nodes.

Proof: (Induct. on h)

$$h=0: n(h) = 1 = F_3 - 1$$

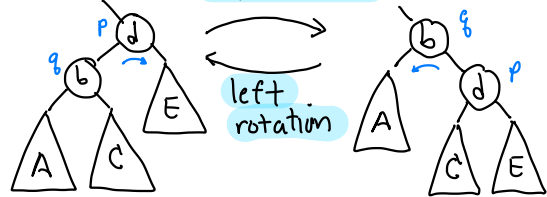
$$h=1: n(h) = 2 = F_4 - 1$$

$$\begin{aligned} h \geq 2: \\ n(h) &= 1 + n(h-1) + n(h-2) \\ &= 1 + (F_{h+2} - 1) + (F_{h+1} - 1) \\ &= (F_{h+2} + F_{h+1}) - 1 = F_{h+3} - 1 \quad \square \end{aligned}$$



```
BSTNode rotateRight(BSTNode p) {
    BSTNode q = p.left
    p.left = q.right
    q.right = p
    return q
}
```

How to maintain the AVL property?

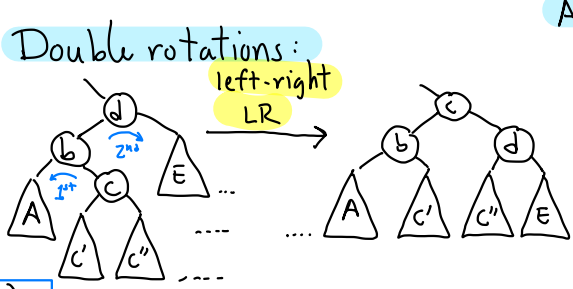
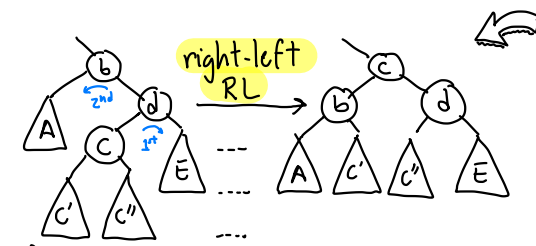


$$A < b < c < d < E$$

$$A < b < c < d < E$$

Corollary: An AVL tree with n nodes has height $O(\log n)$

Proof: Fact: $F_h \approx \varphi^h / \sqrt{5}$ where $\varphi = (1 + \sqrt{5})/2$ "Golden ratio"
 $n \geq \varphi^{h+3} = c \cdot \varphi^h \Rightarrow h \leq \log_{\varphi} n + c$
 $\Rightarrow h \leq \log_2 n / \log_2 \varphi = O(\log n) \quad \square$



```

AVLNode rebalance (AVLNode p)
if (p == null) return p
if (balanceFactor(p) < -1)
  if (ht(p.left.left) >= ht(p.left.right))
    p = rotateRight(p)
  else p = rotateLeftRight(p)
else if (balanceFactor(p) > +1)
  ... (symmetrical)
updateHeight(p); return p
  
```

```

BSTNode rotateLeftRight (BSTNode p)
p.left = rotateLeft(p.left)
return rotateRight(p)
  
```

AVL Tree:

AVL Node: Same as BSTNode (from Lect 4) but add: **int height**

Utilities:

```

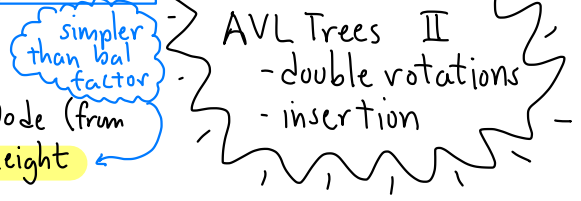
int height (AVLNode p)
return { p == null -> -1
        ow. -> p.height
  }
  
```

```

void updateheight (AVLNode p)
p.height = 1 + max (height(p.left),
                  height(p.right))
  
```

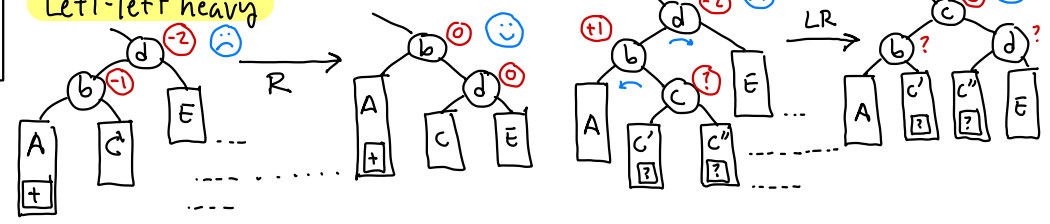
```

int balanceFactor (AVLNode p)
return height(p.right) -
       height(p.left)
  
```



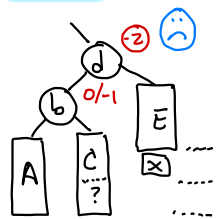
Find: Same as B.S.T.
Insert: Same as BST but as we "back out" rebalance

How to rebalance? Bal = -2
Left-left heavy



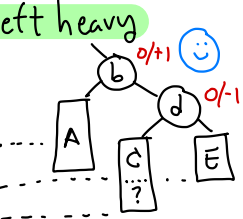
Left-right heavy:

Cases: Balance factor -2

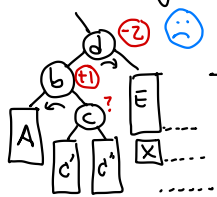


Left-left heavy

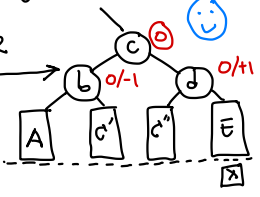
Right rotation



Left-right heavy



LR

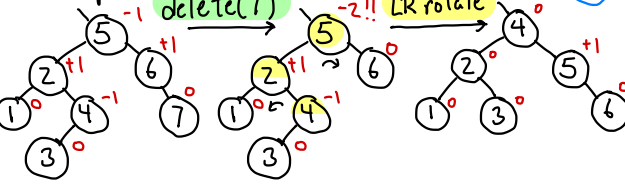


Deletion: Basic plan

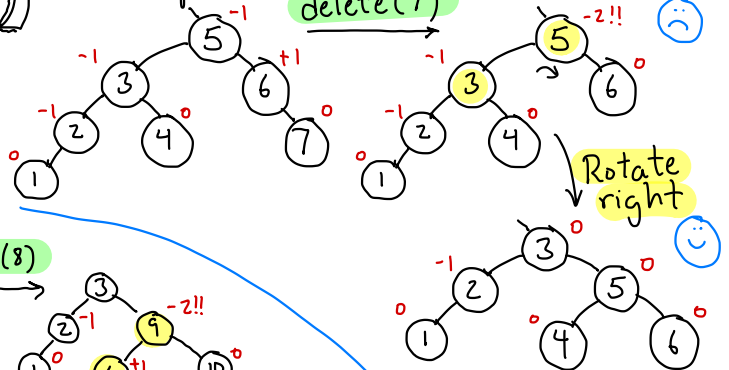
- Apply standard BST deletion
- find key to delete
- find replacement node
- copy contents
- delete replacement
- rebalance

AVL Trees III
- Deletion
- Examples

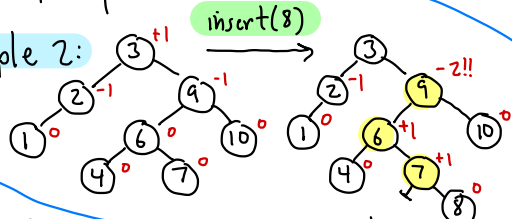
Example 4:



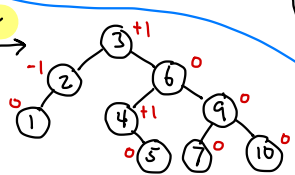
Example 3:



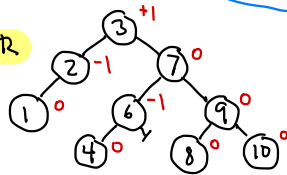
Example 2:



rotate right



rotate LR



AVLNode delete (Key x, AVLNode p)
same as BST delete
return rebalance(p)

Examples:

