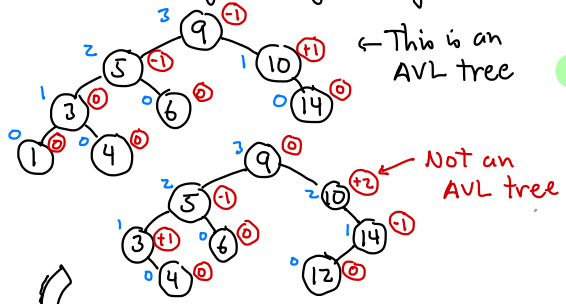
Balance factor:

bal(v) = hgt(v.right) − hgt(v.left)



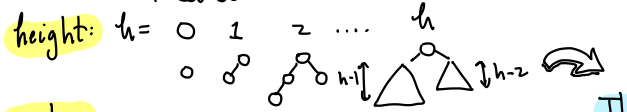
AVL Height Balance

- 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:



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 $F_{h+3} - 1$

AVL Trees I
- Basic defs
- Height props
- Rotations

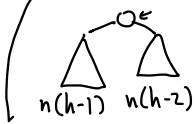
Theorem: An AVL tree of height h has at least $F_{h+3} - 1$ nodes.

Proof: (Induct. on h)

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

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

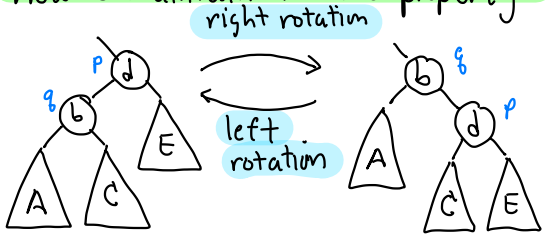
$h \geq 2$:



$$n(h) = 1 + n(h-1) + n(h-2)$$
$$= 1 + (F_{h+2} - 1) + (F_{h+1} - 1)$$
$$= (F_{h+2} + F_{h+1}) - 1 = F_{h+3} - 1 \quad \square$$

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

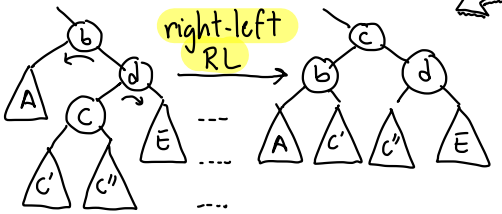
How to maintain the AVL property?



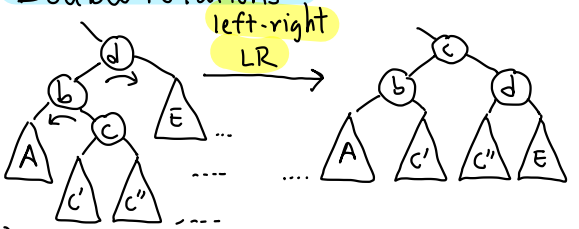
Corollary: An AVL tree with n nodes has height $O(\log n)$

Proof: Fact: $F_h \approx \varphi^h / \sqrt{5}$ where $\varphi = (1+\sqrt{5})/2$ "Golden ratio"

$$n \geq \varphi^{h+3} = c \cdot \varphi^h \Rightarrow h \leq \log_\varphi n + c'$$
$$\Rightarrow h \leq \log_2 n / \log_2 \varphi$$
$$= O(\log n) \quad \square$$



Double rotations:




```
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(balanceFact(p) > +1)
    ...(symmetrical)
  updateHeight(p); return p
```

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

simpler than bal factor

AVL Trees II
- double rotations
- insertion

AVL Tree:

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

```
AVLNode insert(Key x, Value v, AVLNode p){
  if(p == null) p = new AVLNode(x,v)
  else if(x < p.key)
    p.left = insert(x,v,p.left)
  else if(x > p.key)
    p.right = insert(x,v,p.right)
  else throw-Error-Duplicate!
  return rebalance(p)
```

Utilities:

```
int height(AVLNode p)
  return { p==null → -1
         { o.w.    → p.height
```

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

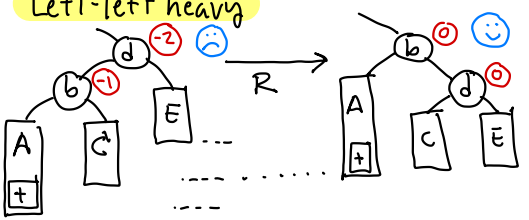
```
int balanceFactor(AVLNode p)
  return height(p.right) -
         height(p.left)
```

Find: Same as B.S.T.

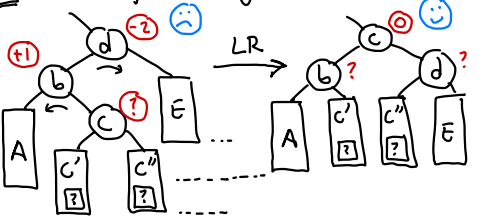
Insert: Same as BST but as we "back out" rebalance

How to rebalance? Bal = -2

Left-left heavy



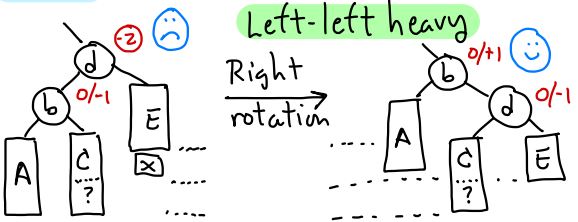
Left-right heavy:

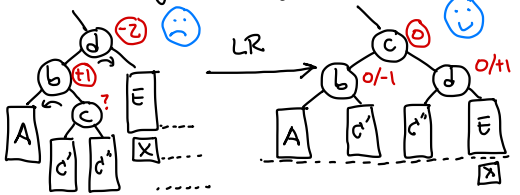


AVL Trees III
- Deletion
- Examples

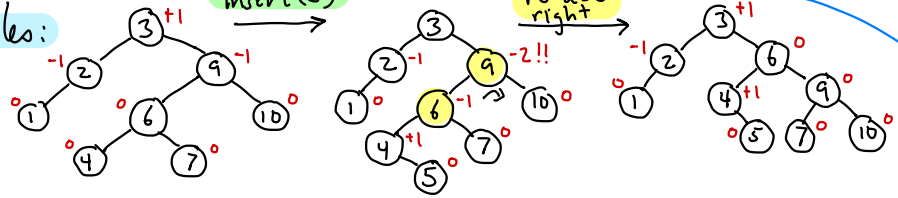
Deletion: Basic plan
- Apply standard BST deletion
  - find key to delete
  - find replacement node
  - copy contents
  - delete replacement
- rebalance

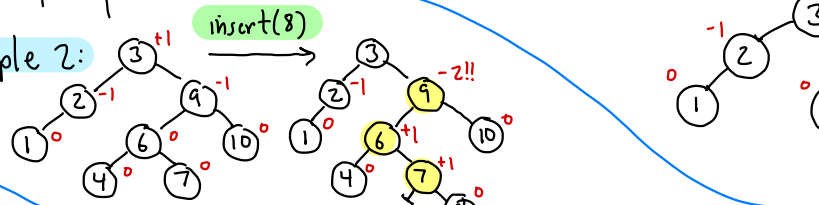
Cases: Balance factor -2

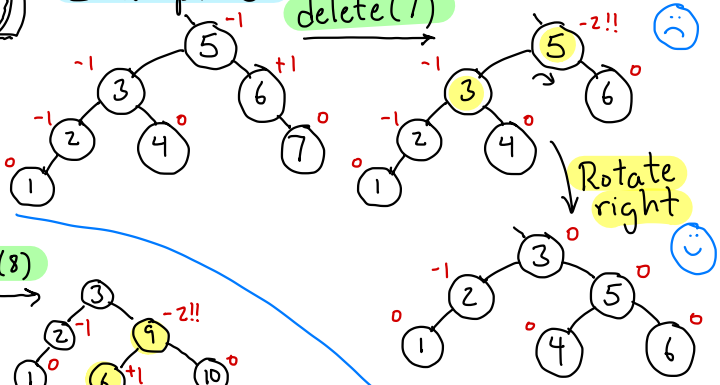
Left-left heavy


Left-right heavy


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

Examples:


Example 2:


Example 3:


Example 4: