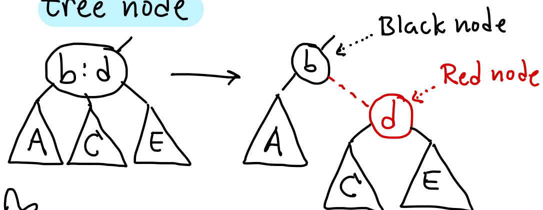
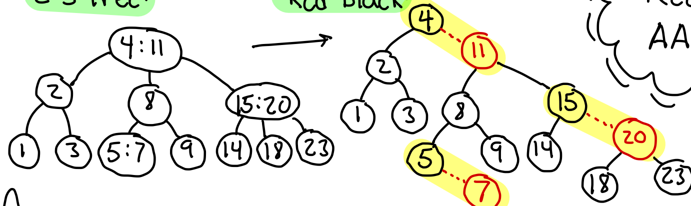
Encoding 3-node as binary tree node



Example:

2-3 Tree: → **Red-Black:**



Rules:

1. Every node labeled red/black
2. Root is black
3. Nulls treated as if black
4. If node is red, both children are black
5. Every path from root to null has same no. of black

Some history:

2-3 Trees: Bayer 1972

Red-black Trees: Guibas & Sedgewick 1978 (a binary variant of 2-3)

Rumor - Guibas had two pens - red & black to draw with

Red-Black and AA-Trees I

Lemma: A red-black tree with n keys has height $O(\log n)$

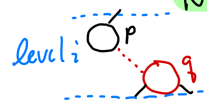
Proof: It's at most twice that of a 2-3 tree.

Q: Is every Red-Black Tree the encoding of some 2-3 tree?

AA-Trees: Simpler to code

- **No null pointers**: Create a sentinel node, nil, and all nulls point to it → nil:

- **No colors**: Each node stores level number. Red child is at same level as parent.



q is red ⟺ q.level == p.level

What we need are stricter rules!

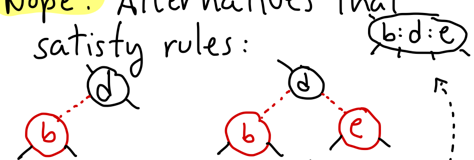
AA-tree:

Arne Anderson 1993

New rule:

6. Each red node can arise only as right child (of a black node)

Nope! Alternatives that satisfy rules:



A "left-skewed" encoding

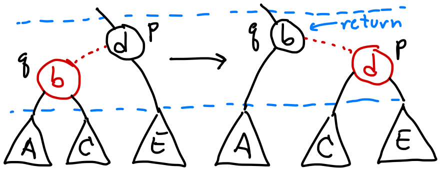
Corresponds to 2-3-4 trees

Red-Black + AA Trees II

Restructuring Ops:

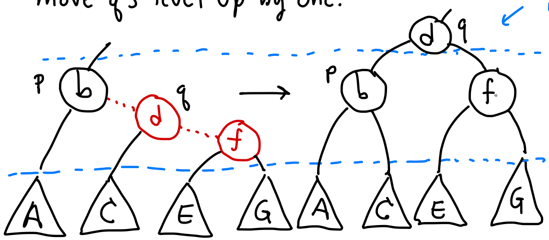
Skew: Restore right skew

→ If black node has red left child, rotate



How to test? $p.left.level == p.level$

Split: If a black node has a right-right red chain, do a left rotation on it right child q, and move q's level up by one.



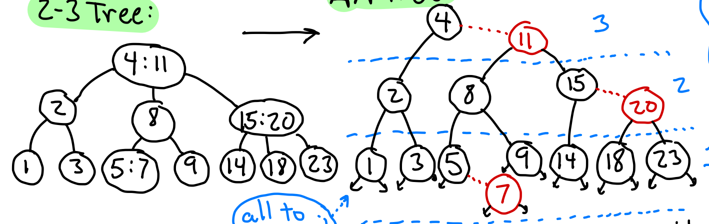
What 2-3 op does this remind you of?

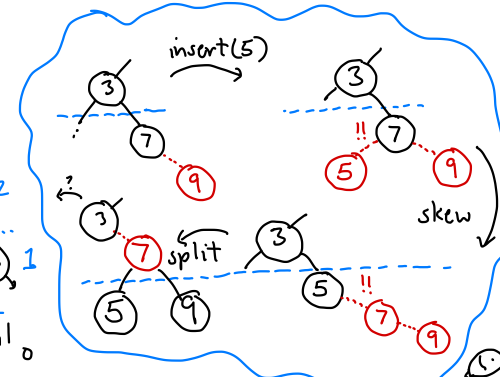
How to test?

$\underbrace{p.level == p.right.level}_{\text{not needed (levels are monotone)}} == p.right.right.level$

Example:

2-3 Tree: → AA tree:





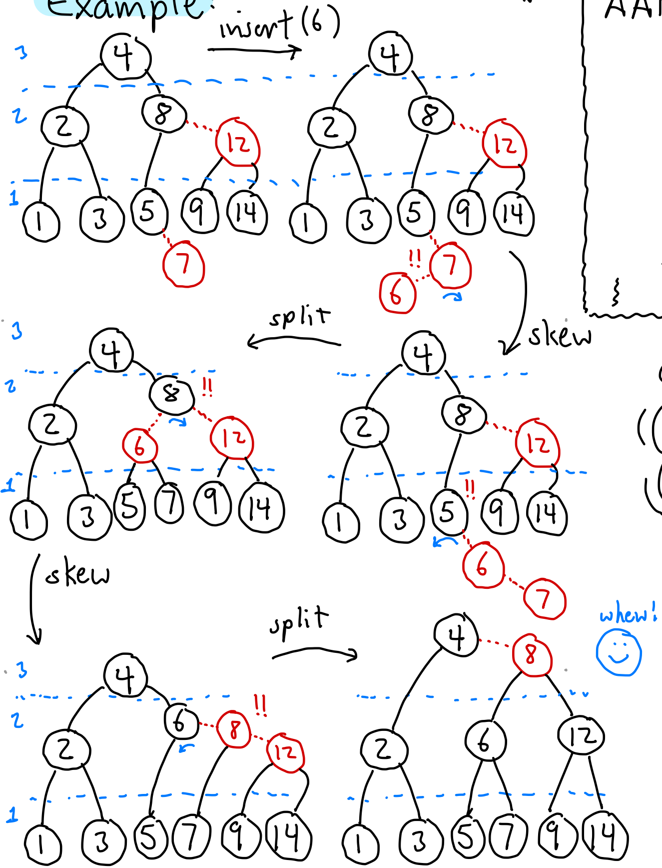
AA Insertion:
- Find the leaf (as usual)
- Create new red node
- Back out applying skew + split

```
AANode skew(AANode p) {
    if (p == nil) return p
    if (p.left.level == p.level) {        // right rotate p
        AANode q = p.left
        p.left = q.right ; q.right = p
        return q                          // ← new subtree root
    } else return p                       // ← everything's fine
}
```

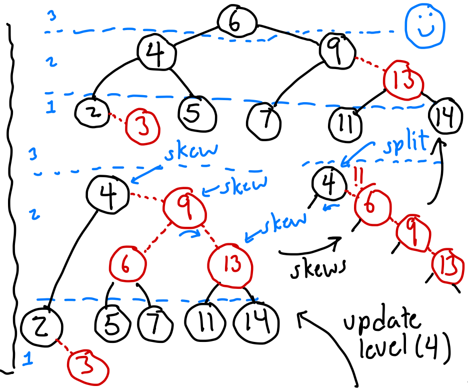
```
AANode split(AANode p) {
    if (p == nil) return p
    if (p.right.right.level == p.level) {
        AANode q = p.right
        p.right = q.left                  // left rotation at q
        q.left = p
        q.level += 1                      // ← move q up a level
        return q
    } else return p                       // ← all okay
}
```

Red-Black and AA Trees III

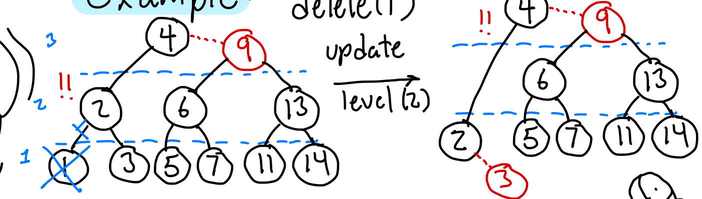
Example:




```
AANode insert(Key x, Value v, AANode p) {
    if (p == nil)                                  leaf level   left,right
        p = new AANode(x, v, 1, nil, nil)
    else if (x < p.key) ... insert on left
    else if (x > p.key) ... insert on right
    else Duplicate Key!
    return split(skew(p))
}
```



Example: delete(1)



Deletion:

Two more helpers:

updateLevel: If p's level exceeds $\ell = 1 + \min(p.left.level, p.right.level)$ then set p's level to ℓ & also p's right child

fixAfterDelete(p):

- update p's level
- skew(p), skew(p.right), skew(p.right.right)
- split(p), split(p.right)

deletion: Same as AVL deletion, but end with: **return fixAfterDelete(p)**