Encoding 3-node as binary tree node

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

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)

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

Example:

2-3 Tree:
Red-Black:

Example:

Red-Black:

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?

Nope! Alternatives that satisfy rules:

A "left-skewed" encoding corresponds to 2-3-4 trees
Restructuring Ops:

**Skew:** Restore right skew
→ If black node has red left child, rotate

**Split:** If a black node has a right-right red chain, do a left rotation at p (bringing its right child q up) and move q up one level.

How to test?
\[ \text{p}\text{.level} = \text{p}\text{.right}\text{.level} = \text{p}\text{.right}\text{.right}\text{.level} \]
not needed (levels are monotone)

Example:

2-3 Tree:

Inserts:

AA tree:

AA Insertion:

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

AA Node split (AA Node p):

```plaintext
if (p == nil) return p
if (p.right.right.level == p.level) {
    AANode q = p.right
    p.right = q.left
    q.left = p
    q.level += 1  // move q up a level
    return q  // all okay
} else return p  // everything's fine
```
Example:

```java
AANode insert(Key x, Value v, AANode p)
if (p == nil)
    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))
```

**Red-Black and AA Trees III**

**Deletion**:

Two more helpers:

- **updateLevel**: If p's level exceeds \( l = 1 + \min(p \text{.left \text{.level}}, p \text{.right \text{.level})} \)
  
  then set p's level to \( l + 1 \) also p's right child

**fix AfterDelete** (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 fix AfterDelete (p)