Encoding 3-node as binary tree node

Example:
- 2-3 Tree
  - Red-Black

Some history:
- 2-3 Trees: Bayer 1972
- Red-black Trees: Guibas and Sedgewick 1978 (a binary variant of 2-3)
- 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.
- Rumor - Guibas had two pens - red & black to draw with

Red-Black and AA-Trees I

What we need are stricter rules!

AA-tree:
- Arne Anderson 1993
- New rule:
  - 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

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: 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 on its right child q, and move q's level up by one.

Example:

- 2-3 Tree:

  - AA tree:

    - Level:

      - 3

      - All to nil

Red-Black + AA Trees II

AA Insertion:

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

AA Node split (AA Node p)

```c
AA Node split (AA Node p) {
  if (p == nil) return p;
  if (p.right.right == p) {
    AA Node q = p.right;
    p.right = q.left;
    q.left = p;
    return q;
  } else return p;
}
```

AA Node skew (AA Node p)

```c
AA Node skew (AA Node p) {
  if (p == nil) return p;
  if (p.left.level == p.level) {
    AA Node q = p.left;
    p.left = q.right;
    q.right = p;
    return q;
  } else return p;
}
```

How to test?

- p.left == p.right == p.right.right == p.right.right.right

How to test?

- p.level == p.right.level == p.right.right.level

Not needed (levels are monotone)
Example:

```
void insert(int x, int 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.left.level, p.right.level) then set p's level to l + also p's right child.

**fix AfterDelete:**
- 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)