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

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

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?

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

Red-Black and AA-Trees I

Height: \( 2 \cdot \log_2 n \) = \( O(\log n) \)

Corresponds to 2-3-4 trees
Restructuring Ops:

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

Example:

2-3 Tree:

- **AA Tree**:

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

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.

Red-Black + AA Trees II

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

not needed (levels are monotone)

Red-Black + AA Trees II

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

not needed (levels are monotone)

Restructuring Ops:

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

Example:

2-3 Tree:

- **AA Tree**:

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

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.

Red-Black + AA Trees II

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

not needed (levels are monotone)

Restructuring Ops:

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

Example:

2-3 Tree:

- **AA Tree**:

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

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.

Red-Black + AA Trees II

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

not needed (levels are monotone)
Example:

```java
AAANode insert(Key x, Value v, AAANode p)
```

```
if (p == nil)
    p = new AAANode(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 \( l + 1 \) 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)
Example of insert:

```
insert(6)
```

Example of delete:

```
delete(1)
```