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

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:

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)

Lemma: A red-black tree with n keys has height O(log n) → Ign...2^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 up) and move q up one level.

Red-Black + AA Trees II

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

Example:

2-3 Tree:

- AA tree:

Example Insertion:

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

```plaintext
AA Node split(AA Node p)
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
else return p  # all okay
```

```plaintext
AA Node skew(AA Node p)
if (p == nil) return p
if (p.left.level == p.level)
    AANode q = p.left
    p.left = q.right
    q.right = p
    return q  # new subtree root
else return p  # everything's fine
```

Slight correction to the AA Node skew(AA Node p) function, as it was not needed (levels are monotone). For a black node with a red left child, we don't need to rotate.
Example:
`insert(6)`

```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) p.left = insert(x, p.left)
else if (x > p.key) ... insert on right
else Duplicate Key: return split(skew(p))
return 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

- **split(p), split(p.right)**

fix AfterDelete(p):
- update p's level
- skew(p), skew(p.right)
- skew(p.right.right)

deletion: Same as AVL deletion, but end with:
return fix AfterDelete(p)

Notes for details
Announcements - 10/4
- HW 1 - Grading in progress
  - Done this weekend
- Prog Assignment 1: Leftist Heap
  - Due Tue, Oct 11
  - New test case.
  - Autograder - Waiting for some people to validate my result
  - If your output does/does not match ours, let me know (Piazza)
delete(1):

```
update Level(4)
```