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. If q is red then q.level == q.parent.level

What we need are stricter rules!

Red-Black and AA-Trees

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:

Red-Black and AA-Trees

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?
A: No! 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

Example:

2-3 Tree:

```
  4:11
  2 5:11
  1 3 7
```

AA tree:

```
  11
  4
   2
    1
```

Red-Black + AA Trees II

AA Insertion:

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

**AA Node split**

```java
AA Node split(AA Node p) {
  if (p.right.right.level == p.level) {
    AA Node q = p.right
    p.right = q.left
    q.left = p
    p.level = q.level + 1
    return q
  }
  else return p
}
```
Example:

```
insert(6)
```

```
split(p)
```

```
split(p)
```

```
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 and also p's right child
```

```
fix After Delete(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)
```

Red-Black and AA Trees III
Announcements - 3/2

- Homework 2 - Binary Search Trees
  - Due Tue, Mar 7, start of class
  - Final version now on handouts page
- Midterm 1 - Next Thu, Mar 9 in class
  - Tue - HW sols & Review
  - Closed-book / Closed-notes
  - 1 "cheat sheet" - front + back
- Practice Problems
  - Coming

\[ \begin{array}{cccc}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12 \\
13 & 14 & 15 & 16 \\
\end{array} \]

4 (c) Don't worry about parent
[Challenge 1]

5 (d) Show your derivation
What is total insert time

\[ \text{Free (p)} \]
\[ \begin{array}{c}
\text{source files} \\
\text{Fig} \\
hw2f1.pdf \\
\end{array} \]
Deletion example:

```
AANode fixAfterDelete(AANode p) {
    updateLevel(p);
    p = skew(p);
    p.right = skew(p.right);
    p.right.right = skew(p.right.right);
    p = split(p);
    p.right = split(p.right);
    return p;
}
```