Node types:
- 2-Node
  - 1 key
  - 2 children
- 3-Node
  - 2 keys
  - 3 children

Recap:
- AVL: Height balanced
- Binary
- 2-3 tree: Height exact
- Variable width

2-3 Trees

Def: A 2-3 tree of height $h$ is either:
- Empty ($h = -1$)
- A 2-Node root and two subtrees, each a 2-3 tree of height $h-1$
- A 3-Node root and three subtrees, each a 2-3 tree of height $h-1$

Thm: A 2-3 tree of $n$ nodes has height $O(\log n)$

Roughly: $\log_3 n \leq h \leq \log_2 n$

How to maintain balance?
- Split
- Merge
- Adoption (Key rotation)

Adoption (Key-Rotation):
- $1 + 3 = 2 + 2$

Merge:
- $1 + 2 / 2 + 1 \rightarrow 3$

Split:
- $4 \rightarrow 2 + 2$

Example:
2-3 tree of height 2

Conceptual tool:
- We'll allow 1-nodes
- 1-node
- 4-nodes temporary
Dictionary operations:
- **Find**: straightforward
- **Insert**: find leaf node where key "belongs" + add it (may split)
- **Delete**: find/replacement/root

**Implementation?**

```java
class TwoThreeNode {
    int nChildren;
    TwoThreeNode children[3];
    Key key[2];
}
```

**Example (continued)**

```
4:8
/  \
2   6
/  \
1   3
```

**Deletion remedy:**
- Have a 3-node neighboring sibling → adopt
- O.w.: Merge with either sibling + steal key from parent
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 \( \rightarrow \) 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)

Nope! Alternatives that satisfy rules:

A "left-skewed" corresponds to 2-3-4 trees

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

Lemma: A red-black tree with \( n \) keys has height \( O(\log n) \) \( \Rightarrow \) \( \log n \) \( \leq \) \( 2n \)

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

Example:

2-3 Tree:

AA tree:

Red-Black + AA Trees

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.

How to test?
- $p.level = p.right.level = p.right.right.level$
  not needed (levels are monotone)

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

AA Node split(AANode 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
  return q
} else return p = everything's fine
```
Announcements - 9/29

- HW 1 - Due today 11:59 pm
  - I'll be available 2-3 pm this afternoon

- Prog Assignment 1: Leftist Heap
  - Due Tue, Oct 11
  - Autograder - Try to get it done this weekend
  - If your output does/does not match ours, let me know (Piazza)