Scapegoat Trees:
- Arne Anderson (1989)
- Galperin & Rivest (1993) rediscovered/extended
- Amortized analysis
  - $O(\log n)$ for dictionary ops amortized (guaranteed for find)
  - If things happen
  - If subtree unbalanced
    - rebuild it

Recap:
- Seen many search trees
- Restructure via rotation
- Today: Restructure via rebuilding
- Sometimes rotation not possible
- Better mem. usage

Example:

```
    p: b
   / \   \\
  a   c   e
```

```
    j = \lfloor \frac{k}{2} \rfloor = 3

    p: d
   / \   \\
  a   c   e
```

```
    Time = O(k)
```

Overview:
Insert:
- same as standard BST
- if depth too high
  - trace search path back
  - find unbalanced node
  - rebuild this subtree

Find:
Same as std. BST
- Tree height $\leq \log_{3/2} n \approx 1.71 \log n$

Delete:
- Same as std. BST
- If num. of deletes is large rel. to $n$
  - rebuild entire tree!

How? Maintain $n, m \leftarrow 0$

Insert: $n++, m++$
Delete: $n--$ …\text{If} \quad m > 2n \text{ rebuild}

How to rebuild $p$:
  - inorder traverse $p$'s subtree $\rightarrow$ array $A[]$
  - buildSubtree($A[]$)

```
buildSubtree($A[0..k-1]$):
  - if $k = 0$ return null
  - $j \leftarrow \lfloor k/2 \rfloor$; $x \leftarrow A[j]$ median
  - $L \leftarrow buildSubtree(A[0..j-1])$
  - $R \leftarrow buildSubtree(A[j+1..k-1])$
  - return Node($x, L, R$)
**Insert:**
- \( n++ \), \( m++ \)
- Same as std BST but keep track of inserted node's depth \( d \)
- if \( (d > \log_2 m) \) \{ /* rebuild event */
  - trace path back to root
- for each node \( p \) visited, \( \text{size}(p) = \text{no. of nodes in } p\text{'s subtree} \)
- if \( \frac{\text{size}(p \text{'s child})}{\text{size}(p)} > \frac{2}{3} \)
  \( p \leftarrow \text{rebuild}(p) \)
  - break

**Details of Operations:**

**Init:**
- \( n \leftarrow m \), \( root \leftarrow \text{null} \)

**Delete:**
- Same as std BST
- if \( m > 2n \), rebuild(\( root \))

**Example:**

```
Example:
```

**Time:** \( O(n) \)

**Scapegoat Trees II**

**Must there be a scapegoat?** Yes!

**Lemma:** Given a binary tree with \( n \) nodes, if \( \exists \) node \( p \) of depth \( d \) \( > \log_2 n \), then \( \exists \) an ancestor \( p \) that satisfies scapegoat condition

**Proof:** By contradiction

- Suppose \( p \text{'s depth} > \log_2 n \) but \( \forall \) ancestors

**How to compute \( \text{size}(p) \)?**
- Can compute it on the fly
- While backing out, traverse "other sibling"
- Too slow? No!
  \( \rightarrow \) Charge to rebuild.

**Trap:**

```
Trap:
```

```
```
**Theorem:** Starting with an empty tree, any sequence of $m$ dictionary operations on a scapegoat tree take time $O(m \log m)$ \([\text{Amortized: } O(\log m)]\)

**Proof:** (Sketch)
- **Find:** $O(\log n)$ guaranteed \([\text{Height: } O(\log n)]\)
- **Delete:** In order to induce a rebuild, number of deletes $\sim$ number of nodes in tree
  $\rightarrow$ Amortize rebuild time against delete ops
- **Insert:** Based on potential argument
  $\rightarrow$ It takes $\sim k$ ops to cause a subtree to size $k$ to be unbalanced.
  $\rightarrow$ Charge rebuild time to these operations