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

**Overview**:
- **Insert**:
  - same as standard BST
    - if depth too high
    - trace search path back
    - find unbalanced node - scapegoat
    - rebuild this subtree
- **Delete**:
  - same as std. BST
    - if num. of deletes is large rel. to n
    - rebuild entire tree!
  - How? Maintain $n, m = 0$
- **Find**:
  - same as std BST
  - Tree height $\leq \log_{3/2} n \approx 1.71\log n$

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

**Example**:
- Restructure via rotation
  - $p$:
    - $T$:
      - $T(k) = 1 + 2T(\frac{k}{2}) = O(k)$
  - $j = \lceil \frac{k}{2} \rceil = 3$

**How to rebuild?**
- **Inorder traverse** $p$'s subtree $\to$ array $A[]$
- **buildSubtree** $(A)$
- $buildSubtree(A[0..k-1])$:
  - if $k = 0$ return null
  - $j = \lfloor k/2 \rfloor$; $x = A[j]$ median
  - $L \leftarrow buildSubtree(A[0..j-1])$
  - $R \leftarrow buildSubtree(A[j+1..k-1])$
  - return Node$(x, L, R)$

**Final**
- $T(k) = 1 + 2T(\frac{k}{2}) = O(k)$
**Details of Operations:**

**Insert:**
- `n++`; `m++`
- Same as std BST but keep track of inserted node's depth → `d`
- If `(d > \log_{3/2} m)`:
  - *rebuild event*
  - `trace path back to root`
  - For each node `p` visited, `size(p) =` no. of nodes in `p`'s subtree
    - If `size(p.child) > \frac{2}{3} size(p)`
  - `p` rebuild event

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

**Example:**

**Scapegoat Trees II**

**Insert:**
```
\text{init: } n \leftarrow m \leftarrow 0 \quad \text{root} \leftarrow \text{null}
```

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

**Proof:** By contradiction

- Suppose `p`'s depth > \log_{3/2} m
  - but \(4\) ancestors

**Lemma:** Given a binary tree with \(n\) nodes, if \exists node `p` of depth > \log_{3/2} \(n\), then \exists ancestor of `p` that satisfies scapegoat condition

**Proof:** By contradiction

- Suppose `p`'s depth > \log_{3/2} m
  - but \(4\) ancestors

**Example:**

- `rebuild`
  - `no more rebuild`
  - `trigger rebuild`

**Proof:** By contradiction

- Suppose `p`'s depth > \log_{3/2} m
  - but \(4\) ancestors

**Lemma:** Given a binary tree with \(n\) nodes, if \exists node `p` of depth > \log_{3/2} \(n\), then \exists ancestor of `p` that satisfies scapegoat condition

**Proof:** By contradiction

- Suppose `p`'s depth > \log_{3/2} m
  - but \(4\) ancestors

**Lemma:** Given a binary tree with \(n\) nodes, if \exists node `p` of depth > \log_{3/2} \(n\), then \exists ancestor of `p` that satisfies scapegoat condition

**Proof:** By contradiction

- Suppose `p`'s depth > \log_{3/2} m
  - but \(4\) ancestors

**Lemma:** Given a binary tree with \(n\) nodes, if \exists node `p` of depth > \log_{3/2} \(n\), then \exists ancestor of `p` that satisfies scapegoat condition

**Proof:** By contradiction

- Suppose `p`'s depth > \log_{3/2} m
  - but \(4\) ancestors
Theorem: Starting with an empty tree, any sequence of $m$ dictionary operations on a scapegoat tree take time $O(m \cdot \log m)$ \([\text{Amortized: } O(\log m)]\)

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

Rebuild

Many inserts before $p$ again is scapegoat

$O(m \cdot \log n)$ $\rightarrow$ $n = \text{max size of tree}$
Amortized time: $O(\log n)$
Geometric Search:
- Nearest neighbors
- Range searching
- Point Location
- Intersection Search

Sofar: 1-dimensional keys
- Multi-dimensional data
- Applications:
  - Spatial databases + maps
  - Robotics + Auton. Systems
  - Vision/Graphics/Games
  - Machine Learning

Partition Trees:
- Tree structure based on
  hierarchical space partition
- Each node is associated w. a region - cell
- Each internal node stores a splitter - subdivides the cell
- External nodes store pts.

Multi-Dim vs. 1-dim Search?

Similarities:
- Tree structure
- Balance $O(\log n)$
- Internal nodes - split
- External nodes - data

Differences:
- No (natural) total order
- Need other ways to discriminate + separate
- Tree rotation may not be meaningful

External nodes store pts.

Point: A $d$-vector in $\mathbb{R}^d$
$p = (p_1, \ldots, p_d)$ $p_i \in \mathbb{R}$
Java: $(p_0, \ldots, p_{d-1})$

Class Point

float[] coord // coords
Point(int d)
... coord = new float[d]
int getDim() -> coord.length
float get(int i) -> coord[i]
... others: equality, distance
toString...

Quadtrees & KD Trees I

Representations:
- Scalars: Real numbers for coordinates, etc.
  - $\mathbb{R}^d$ $(x,y)$
- Points: $p = (p_1, \ldots, p_d)$ in real $d$-dim space $\mathbb{R}^d$
- Other geom objects: Built from these

Radius