# Scapegoat Trees:

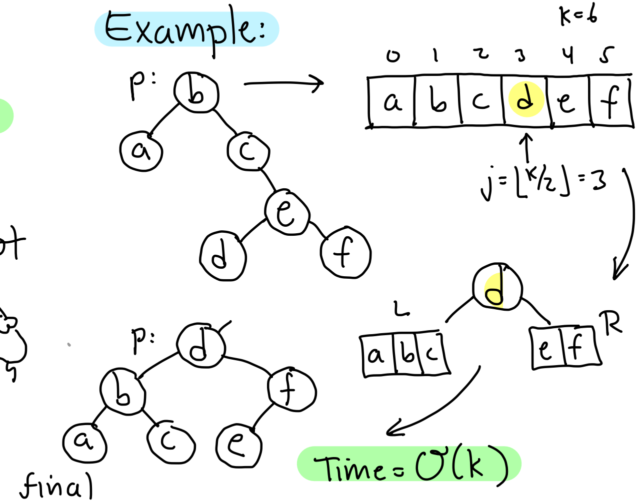
- Arne Anderson (1989)
- Galperin + Rivest (1993) rediscovered/extended
- Amortized analysis
  - $O(\log n)$ for dictionary ops amortized (guaranteed for find)
  - Just let things happen
  - If subtree unbalanced – rebuild it

# Recap:

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

# Scapegoat Trees I

# Example:




# Overview:

## Insert:

- same as standard BST
- if depth too high
  - trace search path back
  - find unbalanced node – scapegoat
  - rebuild this subtree

## Find:

Same as std BST

- Tree height $\leq \log_{3/2} n \approx 1.71 \lg n$

## 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: $n{-}{-}$ ⋯→ If $m > 2n$ rebuild

# How to rebuild?

**rebuild(p):**

- inorder traverse p's subtree → 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)

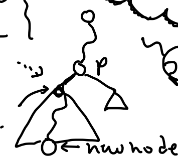
# Scapegoat Trees II

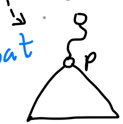
## Insert:

- $n$++; $m$++
- 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 $\frac{\text{size}(p.\text{child})}{\text{size}(p)} > \frac{2}{3}$

    $p \leftarrow$ rebuild($p$)
  - break

  }




## How to compute size(p)?

- Can compute it on the fly
- While backing out, traverse "other sibling"
- Too slow? No!
  → Charge to rebuild.


## Details of Operations:

**Init:** $n \leftarrow m \leftarrow 0$ root ← null

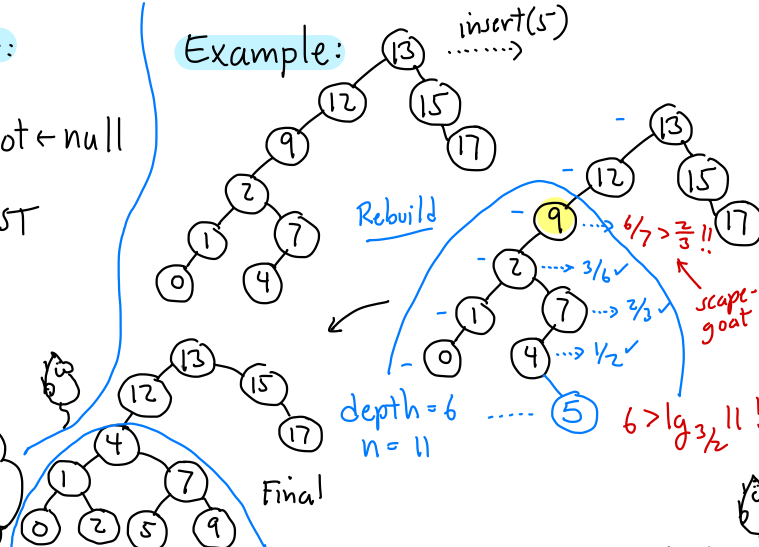
**Delete:**
- Same as std BST
- $n$--
- if $m > 2n$,
  rebuild(root)

Time: $O(n)$

## Must there be a scapegoat? yes!

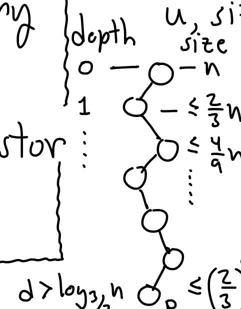
**Lemma:** Given a binary tree with $n$ nodes, if ∃ node $p$ of depth > $\log_{3/2} n$, then ∃ ancestor of $p$ that satisfies scapegoat condition

## Example:




## Proof: By contradiction

- Suppose $p$'s depth > $\log_{3/2} n$ but ∀ ancestors $u$, size($u$.child) ≤ $2/3 \cdot$ size($u$)




⇒ Since $p$ has 1 node:

$1 \leq \text{size}(p) \leq \left(\frac{2}{3}\right)^d n$

$\Rightarrow (3/2)^d \leq n$

$\Rightarrow d \leq \log_{3/2} n$ □

# Scapegoat Trees

# III

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

**Proof:** (Sketch)

**Find:** $O(\log n)$ guaranteed [Height $= O(\log n)$]

**Delete:** In order to induce a rebuild, number of deletes $\sim$ number of nodes in tree

→ Amortize rebuild time against delete ops

**Insert:** Based on potential argument

→ It takes $\sim k$ ops to cause a subtree to size $k$ to be unbalanced.

→ Charge rebuild time to these operations