Advanced Data Structures

Scapegoat Trees
Recap
- HW1 up - due Feb. 14
  #3 is essay-type

- Sub on Friday next Wednesday
  (No class Monday)

- Office hours:
  Monday 10-11am
  Wed 4-5pm
  or by appt: stop in or email
Binary Search Trees:

What is the "best" one?

Recap:

0(\text{height})

Search:

\text{start at root}

if \ v = \ target

\text{return \ YES}

else if \ \leq \ target

recurse \ left

else recurse \ right

Insert:

while (\ v \ has \ children)

if \ x \leq \ v

\text{go \ left}

else \ \text{go \ right}
Data Structures Class

- "Vanilla" BSTs (no rotations or balancing)
  
  Runtime: $O(n)$

How can it get this bad?
BSTrees: balancing
Rotation/pivot:

unbalanced: left (or right) is too big
- AVL trees
- Red-Black trees

Today:
- Scapegoat Trees

This week:
- Splay Trees

\[ h(l(v)) - h(r(v)) \leq 1 \]

\[ \text{height}(y) = \text{height}(x) + 1 \]
Terminology I'll assume:
- search key
- node
- left/right child, parent
- internal/leaf node
- root
- ancestor/descendant
- preorder, inorder, postorder

Recap:
- Height(v): distance to furthest leaf in v's subtree
- Depth(v): distance from v to the root
- Size(v): # of nodes in v's subtree
Scapegoat Trees:

[Anderson '89, Galperin-Rivest '93]

Supports amortized $O(\log n)$. 

Basic idea:
- Standard BST search
- Delete: mark "deleted" node. When tree is half dirty, rebuild into perfect tree.

Runtime:

Claim: rebuild a perfect tree in linear time

$\Rightarrow O(h)$ amortized time
And insert:

- Standard insert
- But: if imbalanced, rebuild a subtree containing new leaf

**Def**: Fix any \( \alpha > 2 \).

A node in imbalanced

if height \( (v) > \alpha \lg(\text{size}(v)) \)

So here:

![Diagram](diagram.png)
Let: \( I(v) = \max \{ 0, |\text{size}(\text{left}(v)) - \text{size}(\text{right}(v))| - 17 \} \)

Example:

```
Ex: v
    /   \
   /     \left
   /       \right
  15       84
```
Lemma:
Just before rebuilding at \( v \),
\[ I(v) = \sum_{v \in \text{size}(ll(v))} - \text{size}(r(v)) \geq cN \]

Proof:
If imbalanced, \( h(v) > \alpha(\lg \text{size}(v)) \)
(by dfn of imbalanced)
but \( \text{left}(v) + \text{right}(v) \)
were not imbalanced:
\[ h(\text{left}(v)) \leq \alpha(\lg \text{size}(\text{left}(v))) \]
\[ h(\text{right}(v)) \leq \alpha(\lg \text{size}(\text{right}(v))) \]

Wlog: Assume insert on left: so:
\[ h(v) = h(\text{left}(v)) + 1 \]
\[ \leq \alpha(\lg \text{size}(\text{left}(v))) + 1 \]
Some intense math:
\[
\alpha \lg (\text{size}(l(v))) + 1 \geq \alpha \lg (\text{size}(v))
\]

raise both sides to power of 2

\[
2^{\alpha \lg (\text{size}(l(v))) + 1} \geq 2^{\alpha \lg (\text{size}(v))}
\]

rule: \(2^{ab} = (2^a)^b = (2^b)^a\)

\[
2^{\alpha \lg (\text{size}(l(v)))} \geq (\text{size}(l(v)))^\alpha
\]

take root:

\[
\sqrt[\alpha]{2^{\alpha \lg (\text{size}(l(v)))}} \geq \sqrt[\alpha]{(\text{size}(l(v)))^\alpha}
\]

\[
\Rightarrow \text{size}(l(v)) \geq \frac{\text{size}(v)}{2^{x/\alpha}}
\]
\[ \text{size}(r(v)) = \text{size}(v) - \text{size}(l(v)) + \sum \text{size}(e(v)) \leq (1 - \frac{1}{2^{\frac{1}{\alpha}}}) \text{size}(v) + 1 \]

So

\[ I(v) \geq \left( \frac{2}{2^{\frac{1}{\alpha}}} - 1 \right) \text{size}(v) \]

\[ \text{goal: size}(v) \]

\[ \text{constant} \]

\[ \geq \text{size}(v) \]
So: Takeaway

$I(v) = \Omega(size(v))$

This means ~$size(v)$ insertions since the last rebuilding.

So: rebuild! How?

Several ways to do this in $O(size(v))$ time.

(HW question!)

$k$ inserts to trigger $O(k)$ rebuild

$\Rightarrow$ amortizes to $O(1)$
Claim: \( \leq 1 \) tree rebuild for each insertion.

When rebuild to "perfect" tree in \( \ell(v) \) height goes down.
Final runtime:

**Find:** no worse than

\[ h = \alpha \log n \]

\[ O(\log n) \]

**Delete:** \( O(\log n) \) to find a mark
rebuild when \( \geq \frac{1}{2} \) dirty

\[ \Rightarrow \text{amortized } O(\log n) \]

**Insert:** find, so \( O(\log n) \)
amortized
Next Topics

- Fractional Cascading
- Splay Trees
- (a,b)-Trees