CSE 250: Binary Search Trees (AVL Trees)

Lecture 27

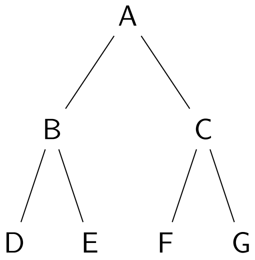
Nov 3, 2023

Reminders

- PA2: Implement **Map Routing**
  1. Create an adjacency list (discussed today)
  2. Find a path from A to B with the fewest intersections
  3. Find a path from A to B with the shortest distance
- PA2 implementation due Sun, Nov 5 at 11:59 PM
- UB Hackathon: Sat/Sun, Nov 4-5.
- Midterm 2: Friday, Nov 10

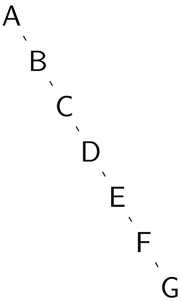
Tree height vs Size

$\textbf{height}(\textit{left}) \approx \textbf{height}(\textit{right})$



$d = O(\log(N))$

$\textbf{height}(\textit{left}) \ll \textbf{height}(\textit{right})$



$d = O(N)$

"Balanced" Trees

- **Faster Search**: We want **height**(*left*) ≈ **height**(*right*)
  - **Formalization 1**: $|\textbf{height}(\textit{left}) - \textbf{height}(\textit{right})| \leq 1$ (Left, right height differ by at most 1)
  - **Formalization 2**: Each leaf at least $\frac{d}{2}$ edges from the root.
- **Question**: How do we keep the tree balanced?
  - **Challenge 1:** Detecting an imbalanced tree.
    - Track the 'imbalance' of each node. (AVL Trees)
    - Track the 'height' of each leaf. (Red-Black Trees)
  - **Challenge 2:** Restoring balance to the tree.
    - Tree Rotations

AVL Trees

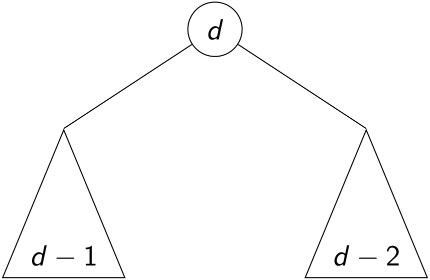
- An AVL Tree (Adelson-Velsky and Landis) is a BST where every node is "height balanced"
  - $|\textbf{height}(\textit{left}) - \textbf{height}(\textit{right})| \leq 1$

- $\textbf{balance}(v) = \textbf{height}(\textit{left}) - \textbf{height}(\textit{right})$

  Maintain $\textbf{balance}(v) \in \{ -1, 0, 1 \}$
  - $\textbf{balance}(b) = 0 \rightarrow$ "v is balanced"
  - $\textbf{balance}(b) = -1 \rightarrow$ "v is left-heavy"
  - $\textbf{balance}(b) = 1 \rightarrow$ "v is right-heavy"

- $\textbf{balance}(v) \in \{ -1, 0, 1 \}$ is the AVL tree property

Enough Nodes?



$$\textbf{minNodes}(d) = 1 + \textbf{minNodes}(d-1) + \textbf{minNodes}(d-2)$$

Enough Nodes?

For $d > 1$:

- $\textbf{minNodes}(d) = 1 + \textbf{minNodes}(d-1) + \textbf{minNodes}(d-2)$
- This is (almost) the Fibonacci Sequence!
  - $\textbf{minNodes}(d) = \text{Fib}(d+3) - 1$
  - $\text{Fib}(0), \text{Fib}(1), \text{Fib}(2), \ldots = 0, 1, 1, 2, 3, 5, 8, \ldots$
- $\textbf{minNodes}(d) \in \Omega(1.5^d)$

Enough Nodes?

$$\textbf{minNodes}(d) = \Omega(1.5^d)$$
$$\textbf{minNodes}(d) \geq c \cdot 1.5^d$$
$$N \geq \textbf{minNodes}(d) \geq c \cdot 1.5^d$$
$$\frac{N}{c} \geq 1.5^d$$
$$\log_2\left(\frac{N}{c}\right) \geq \log_2(1.5^d)$$
$$\log_2\left(\frac{N}{c}\right) \geq \log_{1.5}(1.5^d)\log_2(1.5)$$
$$\log_2\left(\frac{N}{c}\right) \geq d\log_2(1.5)$$
$$\log_2(N) - \log_2(c) \geq d\log_2(1.5)$$
$$\frac{1}{\log_2(1.5)}\log_2(N) - \frac{\log_2(c)}{\log_2(1.5)} \geq d$$
$$d \leq \frac{1}{\log_2(1.5)}\log_2(N) - \frac{\log_2(c)}{\log_2(1.5)}$$
$$d \in O(\log_2(N))$$

Enforcing the AVL Constraint

- Computing **balance**() as-needed is expensive
  - **balance**() computes **height**() twice ($O(N)$ each)
- **Idea:** Precompute the balance factor and store it at each node.

Enforcing the AVL Constraint

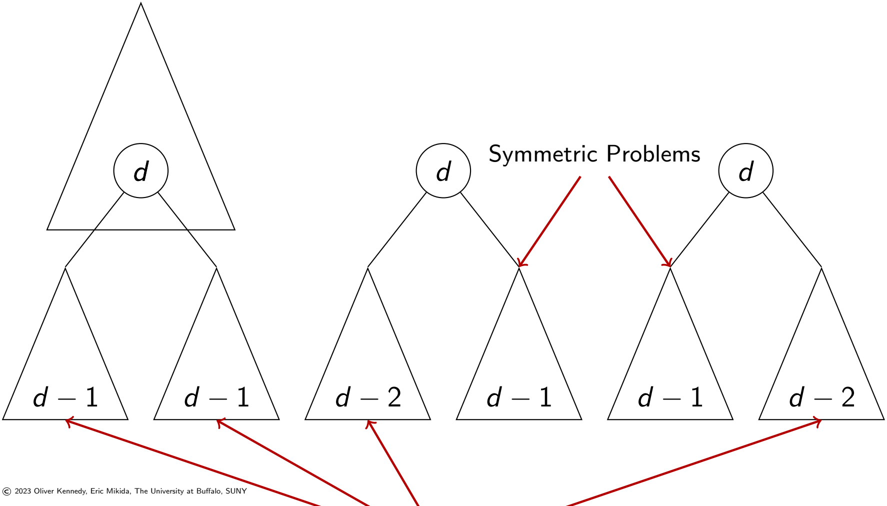
```
public class AVLNode<E> {
  E element;
  Optional<AVLNode<E>> parent = Optional.empty();
  Optional<AVLNode<E>> left   = Optional.empty();
  Optional<AVLNode<E>> right  = Optional.empty();

  boolean isLeftHeavy = false;  // t if balance(this) == -1
  boolean isRightHeavy = false; // t if balance(this) == 1
  /* ... */
}
```

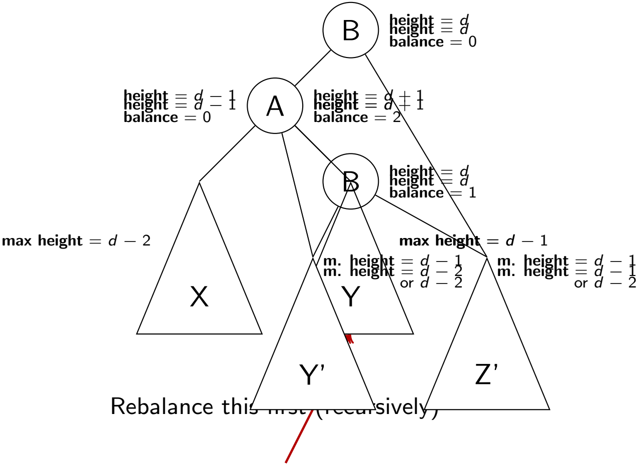
`parent` makes it possible to traverse up the tree.

$$balance(n) = \begin{cases} -1 & \textbf{if } \texttt{n.isLeftHeavy == true} \\ 1 & \textbf{if } \texttt{n.isRightHeavy == true} \\ 0 & \textbf{otherwise} \end{cases}$$

Enforcing the AVL Constraint

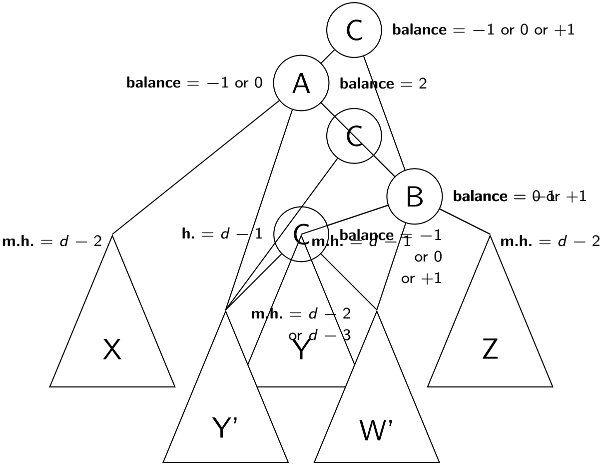


Enforcing the AVL Constraint



Only a problem if we insert here

Enforcing the AVL Constraint



Inserting Records

1. Find Insertion Point. $O(\log N)$
2. Insert Node (Current $\leftarrow$ Parent). $O(1)$
3. Is Current Node Imbalanced? $O(1)$
   - Rotate Left
     OR $O(1)$
   - Rotate Right
     OR $O(1)$
   - Rotate Left-Right
     OR $O(1)$
   - Rotate Right-Left $O(1)$
4. Current $\leftarrow$ Current's Parent.
5. Repeat from step 3. $O(\log N)$ times

Inserting Records

Claims:

1. If the balance factor of one node is off by at most one, at most two rotations will fix it for that node.
   - See preceding slides.
2. If an AVL tree is balanced, then after an insertion, no nodes will have balance factors worse than ± 2.
   - An insertion can increase the depth of a subtree by at most 1.
3. If an AVL tree is balanced, then after an insertion, at most $O(\log(N))$ no nodes will have balance factors of ± 2.
   - An insertion can only change the balance factor of the insertion point's ancestors.
   - The number of ancestors of a node is at most the depth.
   - The depth of a balanced binary search tree $(+1)$ is $O(\log(N))$

Inserting Records

1. Find Record's Node. $O(\log(N))$
2. Clean Up Children $O(\log(N))$
   - If no children, done
   - If one child, replace node with child
   - If two children, replace node's value with child's
     - ... then 'delete' child, repeating from step 1.
3. Fix Imbalance Up The Tree. $O(\log(N))$

Total: $O(\log(N))$