CSE 250: Binary Search Trees (contd.)

Lecture 26

Nov 1, 2023

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

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: Nov 4-5.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Binary Search Trees

- **Binary** Tree
  - Each element has (at most) 2 children.
- **Binary Search Tree** Constraint
  - Each node has a value.
  - Each node's value is greater than its left descendants
  - Each node's value is lesser than (or equal to) its right descendants
- **Set** Constraint [optional]
  - Each node's value is unique.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Binary Search Trees

Operation	**Runtime**
`find`	$O(d)$
`insert`	$O(d)$
`remove`	$O(d)$

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Tree Traversals

- Pre-order (top-down)
  - visit **root**, visit **left** subtree, visit **right** subtree
- In-order
  - visit **left** subtree, visit **root**, visit **right** subtree
- Post-order (bottom-up)
  - visit **left** subtree, visit **right** subtree, visit **root**

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Tree Traversals

```
1   public class TreeNode<T>
2   {
3     T value;
4     Optional< TreeNode<T> > left = Optional.empty();
5     Optional< TreeNode<T> > right = Optional.empty();
6     /* ... */
7     public value preorderVisit(List<T> accumulator)
8     {
9       accumulator.add(value)
10
11      if(left.isPresent()){
12                left.get().inorderVisit(accumulator); }
13
14      if(right.isPresent()){
15                right.get().inorderVisit(accumulator); }
16    }
17  }
```

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Tree Traversals

```
1   public class TreeNode<T>
2   {
3     T value;
4     Optional< TreeNode<T> > left = Optional.empty();
5     Optional< TreeNode<T> > right = Optional.empty();
6     /* ... */
7     public value inorderVisit(List<T> accumulator)
8     {
9       if(left.isPresent()){
10                left.get().inorderVisit(accumulator); }
11
12      accumulator.add(value)
13
14      if(right.isPresent()){
15                right.get().inorderVisit(accumulator); }
16    }
17  }
```

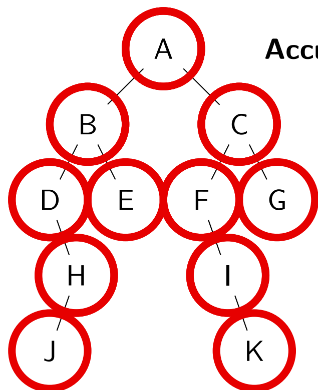
© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Tree Traversals

```
1   public class TreeNode<T>
2   {
3     T value;
4     Optional< TreeNode<T> > left = Optional.empty();
5     Optional< TreeNode<T> > right = Optional.empty();
6     /* ... */
7     public value postorderVisit(List<T> accumulator)
8     {
9       if(left.isPresent()){
10                left.get().inorderVisit(accumulator); }
11
12      if(right.isPresent()){
13                right.get().inorderVisit(accumulator); }
14
15      accumulator.add(value)
16    }
17  }
```

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Tree Iteration: In-Order



Accumulator: D J H B E A F I K C G

Call Stack

inorderTraversal(A)
inorderTraversal(B)
inorderTraversal(D)
visit(D)
inorderTraversal(H)
inorderTraversal(J)
visit(J)
visit(H)
visit(B)
inorderTraversal(E)
visit(E)
visit(A)

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Binary Search Trees

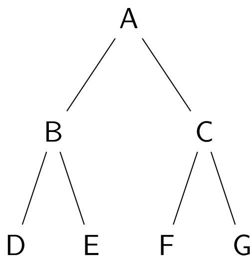
Operation	Runtime
`find`	$O(d)$
`insert`	$O(d)$
`remove`	$O(d)$

$$\log(N) \leq d \leq N$$

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

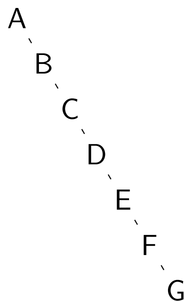
Tree Depth vs Size

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



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

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



$d = O(N)$

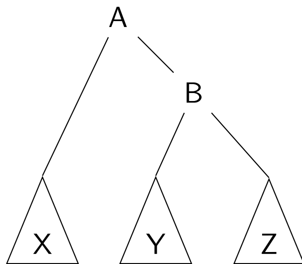
© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

"Balanced" Trees

- **Faster Search**: We want **depth**(*left*) $\approx$ **depth**(*right*)
  - **Formalization 1**: $|$**depth**(*left*) $-$ **depth**(*right*)$| \leq 1$
    (Left, right depth 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 'depth' of each leaf. (Red-Black Trees)
  - **Challenge 2:** Restoring balance to the tree.
    - **Tree Rotations**

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Tree Rotation

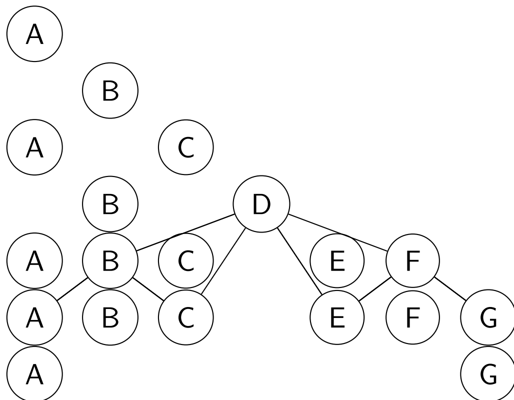


$X < A < B$ $A < Y < B$ $A < B < Z$

RotateLeft(A, B)

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Rebalancing Trees



[illegible]

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

AVL Trees

- An AVL Tree (Adelson-Velsky and Landis) is a BST where every node is "depth 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

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

AVL Trees

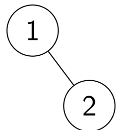
- **Goal**: AVL Tree Property maintains a nearly balanced tree.
  - If **balance**$(v) \in \{ -1, 0, 1 \}$ for all nodes, $d \in O(\log(N))$
- **Proof Idea**: An AVL Tree with Depth d must have at least $N = \Omega(2^d)$ nodes.
  - ...thus $d \in O(\log(N))$

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

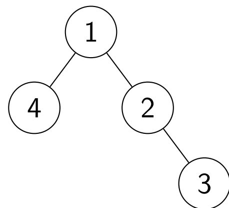
AVL Trees

Let **minNodes**(d) be the minimum number of nodes in an AVL tree of depth d.

minNodes$(1) = 1$



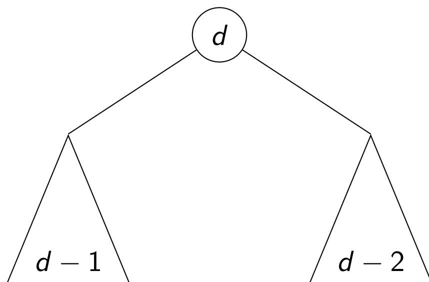
minNodes$(2) = 2$



minNodes$(3) = 4$

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Enough Nodes?



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

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Enough Nodes?

For $d > 1$:

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

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Enough Nodes?

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

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY