CSE 250: Binary Search Trees (Red-Black Trees)

Lecture 28

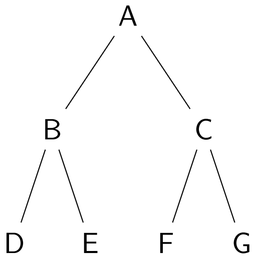
Nov 6, 2023

Reminders

- Midterm 2: Friday, Nov 10

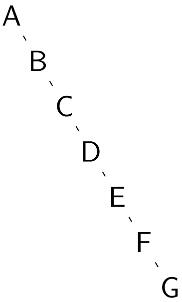
Tree height vs Size

height(*left*) $\approx$ **height**(*right*)



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

height(*left*) $\ll$ **height**(*right*)



$$d = O(N)$$

"Balanced" Trees

How do we enforce balance ($d \in O(\log(N))$)

1. Completeness ($N \in \Omega(2^d)$)
2. **balance**(*node*) $\in \{ -1, 0, +1 \}$ ($N \in \Omega(1.5^d)$)
3. ???

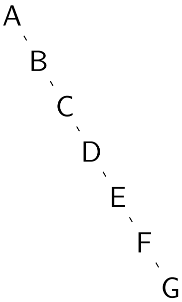
The AVL property

The AVL tree property requires **every** subtree to be balanced.

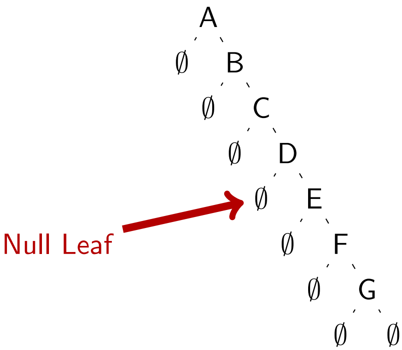
- The entire tree may be balanced ($d \in O(\log(N))$)
  ... even if one subtree is not.

- ... but the AVL property can be enforced **locally**.
  - You can tell if a node is imbalanced by looking at $O(1)$ nodes.

'Empty'/'Null' Leaves



'Empty'/'Null' Leaves

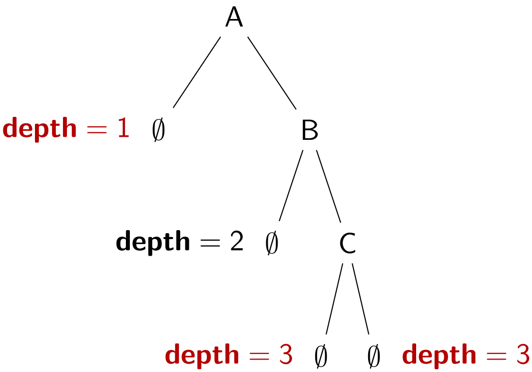


Red-Black Trees

- **Global Property**
  - The depths of any pair of null leaves are at most a factor of 2 different.
  - The deepest null leaf is at most twice the depth of the shallowest.
  - ... entails that $d \in O(\log(N))$
- **Locally-Enforceable Property**
  - Red-Black-Colorability (to be defined shortly)
  - ... entails the global property

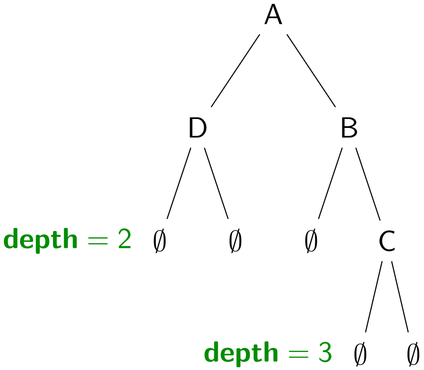
How many nodes are required?

minNodes(d=2)



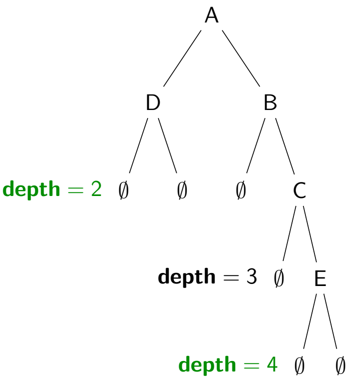
How many nodes are required?

minNodes(d=2)



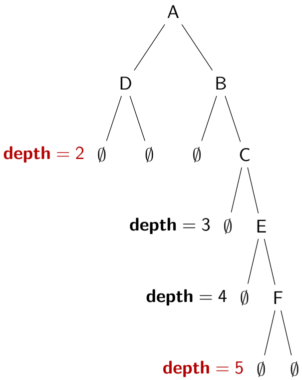
How many nodes are required?

minNodes(d=3)



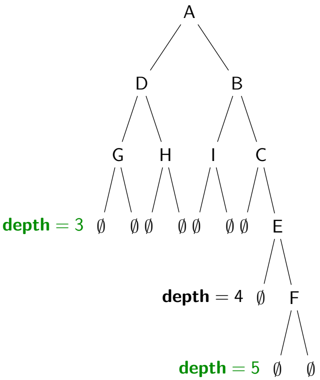
How many nodes are required?

minNodes(d=4)

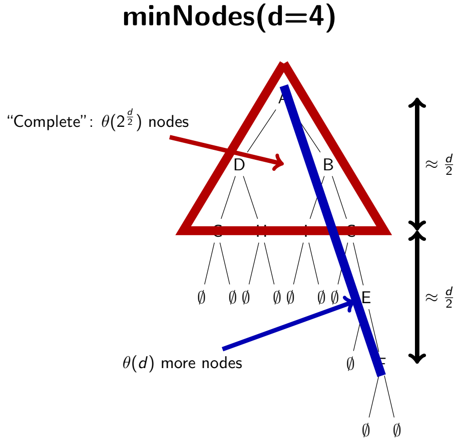


How many nodes are required?

minNodes(d=4)



How many nodes are required?



How many nodes are required?

$$\textbf{minNodes}(d) = \theta(2^{\frac{d}{2}} + d) = \theta(2^{\frac{d}{2}})$$

So...

$$N \geq \textbf{minNodes}(d) \geq c \cdot 2^{\frac{d}{2}}$$

$$\frac{N}{c} \geq 2^{\frac{d}{2}}$$

$$\log(\frac{N}{c}) \geq \frac{d}{2}$$

$$d \leq 2\log(N) - 2 \cdot \log(c)$$

$$d \in O(\log(N) + 1) = O(\log(N))$$

"Balanced" Trees

- **Faster Search**: We want **height**$(left) \approx$ **height**$(right)$
  - **Formalization 1**: $|\mathbf{height}(left) - \mathbf{height}(right)| \leq 1$ (Left, right height differ by at most 1)
  - **Formalization 2**: Each null leaf at least $\frac{d}{2}$ edges from 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

Red-Black Trees

We Enforce (high-level)...

- Every node is colored **Red** or **Black**.
- The number of **Black** nodes on a path from the root to *every* null leaf node is the same.
  - Call this number the **Black**-depth of the tree.
- The number of **Red** nodes on a path from the root to every null leaf node is never bigger than the **Black**-depth of the tree.

Claim 1: Every null leaf is at least the **Black**-depth away from the root. **Claim 2:** No null leaf is ever $2\times$ the **Black**-depth away from the root.

Red-Black Trees

We Enforce (low-level)...

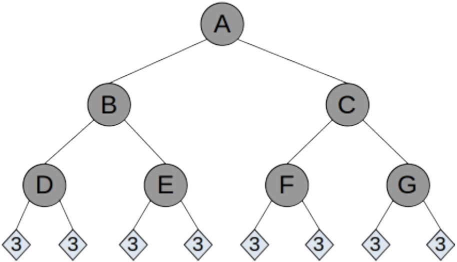
- Every node is colored **Red** or **Black**.
- Every insertion/deletion preserves the **Black**-depth of the tree (or modifies it uniformly for the entire tree).
- No **Red** node can have a **Red** parent.
- The root is always **Black**.

Red-Black Trees

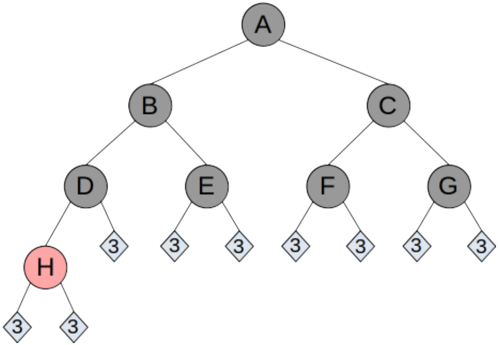
To insert a node:

1. Find the insertion point
2. Insert the new node, colored **Red**.
   - Inserting the node as **Red** doesn't affect the **Black**-depth of the tree.
3. If the parent of the insertion point is also **Red**, fix it.
   - The fix must preserve the **Black**-depth of the tree.

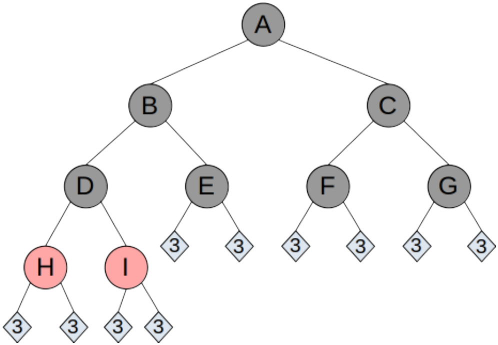
Red-Black Insertion Example



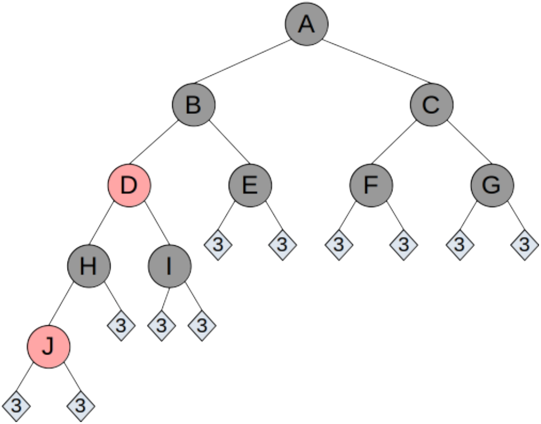
Red-Black Insertion Example



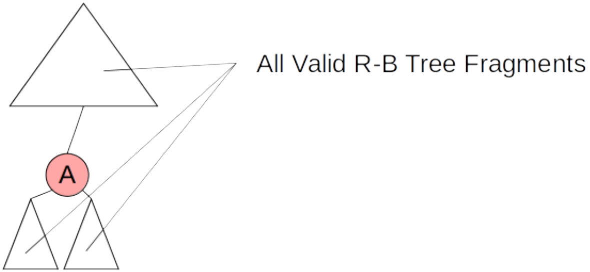
Red-Black Insertion Example



Red-Black Insertion Example

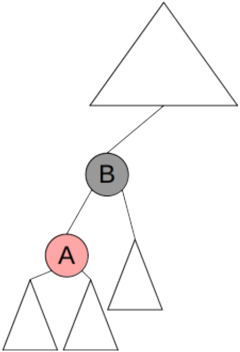


Repairing a Red-Black Tree



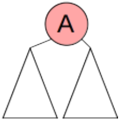
Goal: We want to 'fix' **Red** Node A.

Repairing a Red-Black Tree



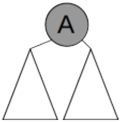
Case 1: If A's parent is **Black**, we're done.

Repairing a Red-Black Tree



Case 2: If A is the root, make it **Black**.

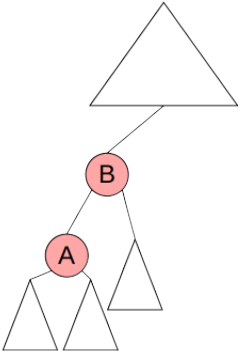
Repairing a Red-Black Tree



Case 2: If A is the root, make it **Black**.

The **Black**-depth of the *entire* tree changes (with $O(1)$ work).

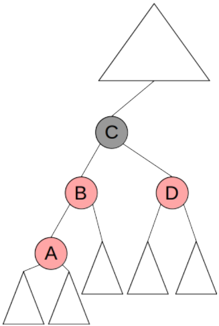
Repairing a Red-Black Tree



We have a problem if A's parent is also **Red**.

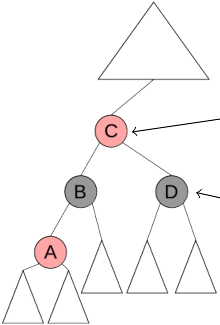
... but A's grandparent *must* be **Black**.

Repairing a Red-Black Tree



Case 3: A's parent's sibling (aunt) is also **Red**.
B and D swap colors with C.

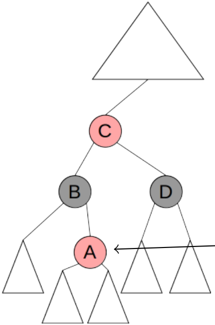
Repairing a Red-Black Tree



C now red, need to repeat repair, but at grandparent of A

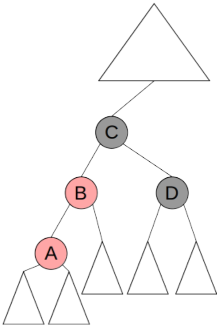
of **Black** nodes on any path unchanged.

Repairing a Red-Black Tree



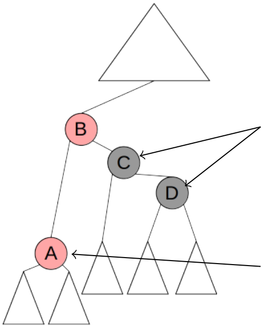
Also works if A is right child of B, (or a child of D).

Repairing a Red-Black Tree



Case 4: A's parent's sibling (aunt) is **Black**.
Rotate B, C

Repairing a Red-Black Tree

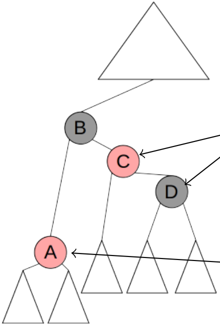


of **Black** nodes on paths through C, D unchanged.

1 fewer **Black** node on paths going through A

Swap B and C's colors

Repairing a Red-Black Tree

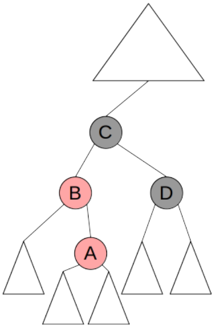


\# of **Black** nodes on paths through C, D unchanged.

Same # of **Black** nodes on paths going through A

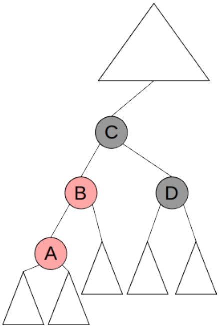
Root of affected subtree now **Black**, so repair can stop.

Repairing a Red-Black Tree



Case 4b: A's parent's sibling (aunt) is **Black**, and A is inner leaf.
Rotate A, B

Repairing a Red-Black Tree



Rotate A, B

Then proceed as **Case 4**.

If A is D's child, the cases are symmetric.

Insertions into a Red-Black Tree

1. Find insertion point. $O(d) = O(\log(N))$
2. Insert **Red** node. $O(1)$
3. Fix if necessary.
   - At most 3 color changes per fix. $O(1)$
   - At most 2 rotations per fix. $O(1)$
   - May need to repeat at grandparent. $O(d) = O(\log(N))$ times

Total: $O(\log(N)) + O(\log(N) \cdot 1) = O(\log(N))$

Removals

... are similar, but with more cases

BST Overview

	General BST	**AVL Tree**	**R-B Tree**
`find`	$O(N)$	$O(\log(N))$	$O(\log(N))$
`insert`	$O(N)$	$O(\log(N))$	$O(\log(N))$
`remove`	$O(N)$	$O(\log(N))$	$O(\log(N))$

Note 1: R-B Trees are like AVL Trees, but with a better constant.

Note 2: $\log(N)$ is great, but can we do better?