Lecture 8a: 2-3-4 and Red-Black Trees

CS 70: Data Structures and Program Development

Tuesday, March 10, 2020

The Story So Far

Lookup and insert with n nodes:

- naive BST: worst-case $\Theta(n)$
- randomized BST: expected $\Theta(\log n)$ and worst-case $\Theta(n)$

Future preview:

- splay tree: *amortized* $\Theta(\log n)$ and worst-case $\Theta(n)$.

How can we get worst-case $\Theta(\log n)$?

- Need to ensure the tree has height $\Theta(\log n)$?

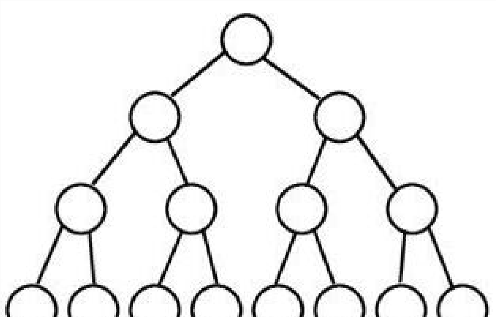
A Perfect Tree

All levels are full.

Has $2^{h+1} - 1$ nodes, where h is height.

So $h \in \Theta(\log n)$.

This is a lower bound! For any BST, $h \in \Omega(logn)$.



Red-Black Trees

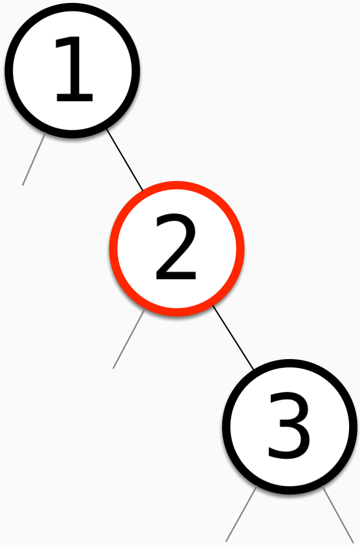
Red-Black Trees (definition 1)

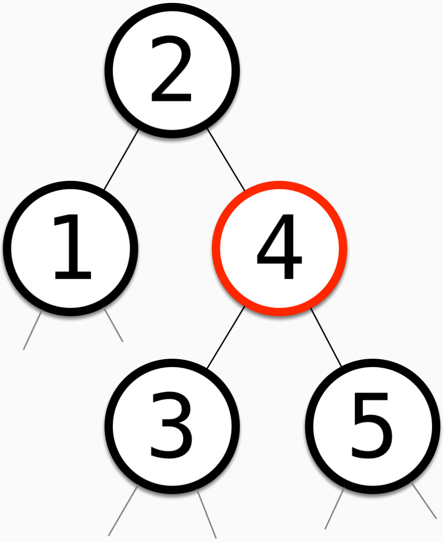
A red-black tree is a BST such that:

- Every node is red or black
- The root is black
- No red parent has a red child
- Every path from the root to `nullptr` passes through the same number of black nodes.

Claim: an n-node red-black tree has height $\Theta(\log n)$

Valid Red-Black Trees?





Red-Black Trees have $\Theta(\log n)$ height.

Suppose every path to `nullptr` hits b black nodes.

1. The height is less than $2b$.
2. The tree has at least $2^b - 1$ nodes, i.e., $2^b - 1 \leq n$.

Red-Black Trees have $\Theta(\log n)$ height.

Suppose every path to `nullptr` hits b black nodes.

1. The height is less than $2b$.
2. The tree has at least $2^b - 1$ nodes, i.e., $2^b - 1 \leq n$.
3. $h < 2b \leq 2\log_2(n+1)$, so $h \in O(\log n)$.

Red-Black Trees have $\Theta(\log n)$ height.

Suppose every path to `nullptr` hits b black nodes.

1. The height is less than $2b$.
2. The tree has at least $2^b - 1$ nodes, i.e., $2^b - 1 \leq n$.
3. $h < 2b \leq 2\log_2(n+1)$, so $h \in O(\log n)$.

Also we know that $h \in \Omega(\log n)$.

Red-Black Trees have $\Theta(\log n)$ height.

Suppose every path to `nullptr` hits b black nodes.

1. The height is less than $2b$.
2. The tree has at least $2^b - 1$ nodes, i.e., $2^b - 1 \leq n$.
3. $h < 2b \leq 2\log_2(n+1)$, so $h \in O(\log n)$.

Also we know that $h \in \Omega(\log n)$.

Since $n \in O(\log n)$ and $n \in \Omega(\log n)$, $n \in \Theta(\log n)$.

The Problem

Red-Black Trees can be a pain to implement.

- Lookup is easy - same as in a BST
- But how to update the colors when we insert?
    - Not immediately obvious...

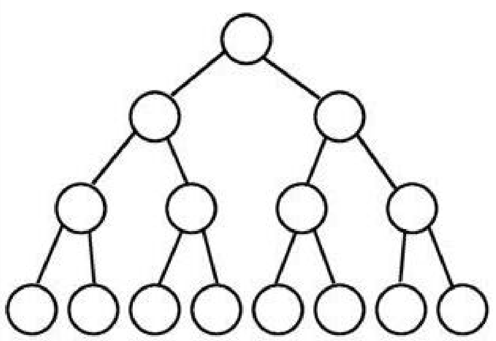
Maybe we can find a balanced tree that is more intuitive?

2-3-4 Trees

Why not require perfection at all times?

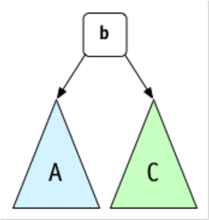
All levels are full.

Has $2^{h+1} - 1$ nodes, where h is height.

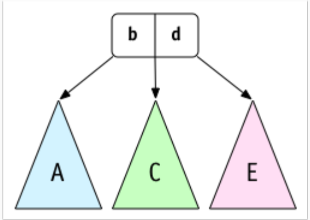


2-3-4 Tree

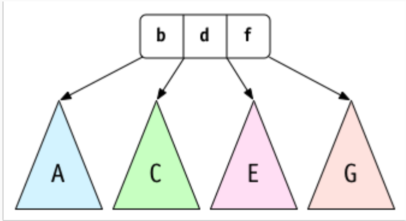
1. Allow a node to store more than one key.
2. All leaves at the same depth



2-node

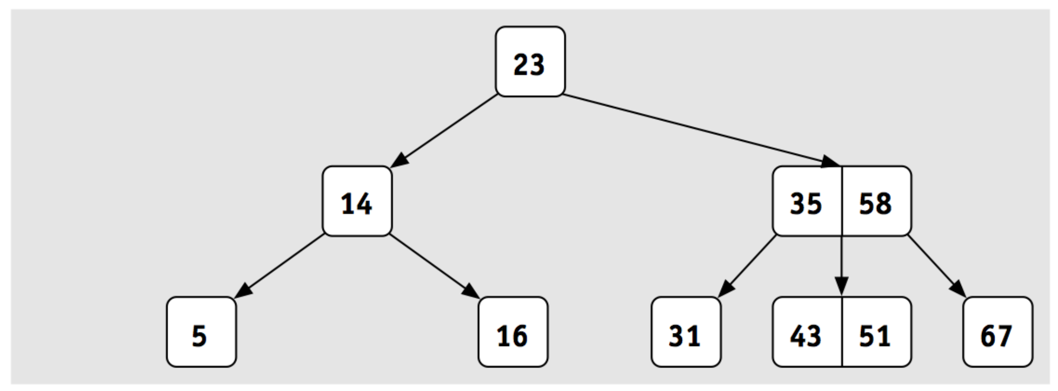


3-node



4-node

Typical 2-3-4 tree



2-3-4 trees: insertion

Rules for insert-as-leaf

- Always keep the leaves at the same level
- If leaf is a 2 node: Make it a 3 node
- If leaf is a 3 node: Make it a 4 node
- If we encounter a 4 node on the way down, "promote" the middle (why)?

Exercise: insert 1,2,3,4,5,6,...

Your Turn: 2-3-4 Trees

- In groups of 4
- Starting from an empty tree
- Insert the month numbers and day numbers of your birthdays (without repeats) into a 2-3-4 Tree.

Advantages and Disadvantages

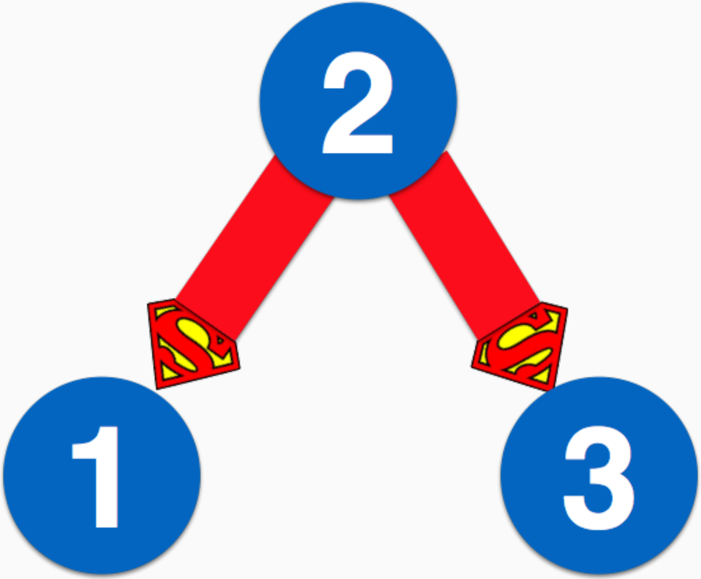
Advantages of 2-3-4 Trees

- The tree is always "balanced"
- Worst case for insert and lookup is $\Theta(\log n)$ for a tree with n values
- Simple algorithm; no rotations required.
- Smaller height than a typical binary tree. (why?)

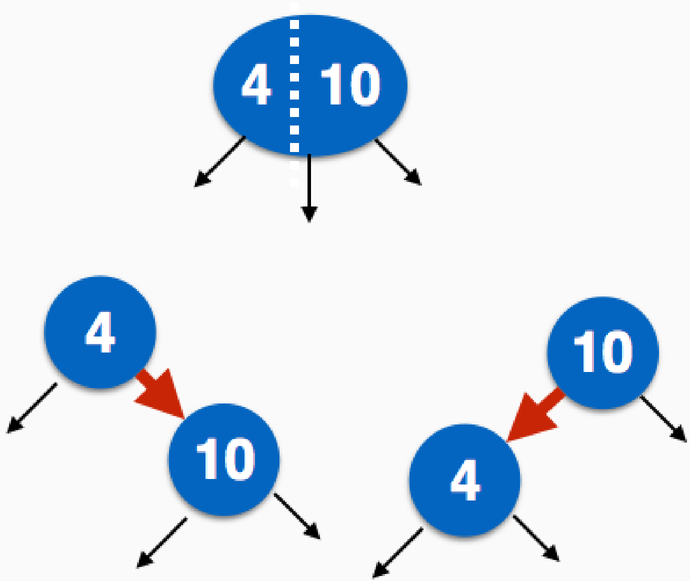
Disadvantages?

Encoding 2-3-4 Trees in Binary

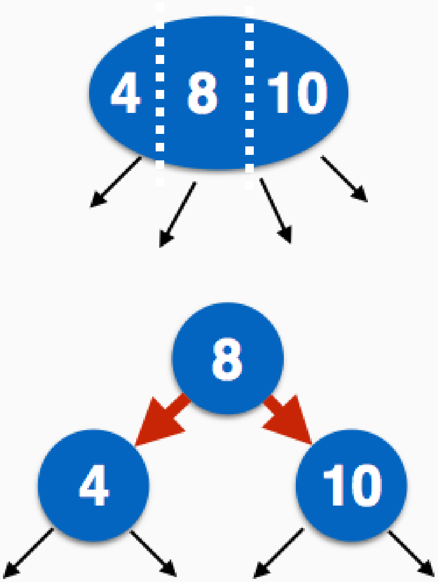
Super Links!



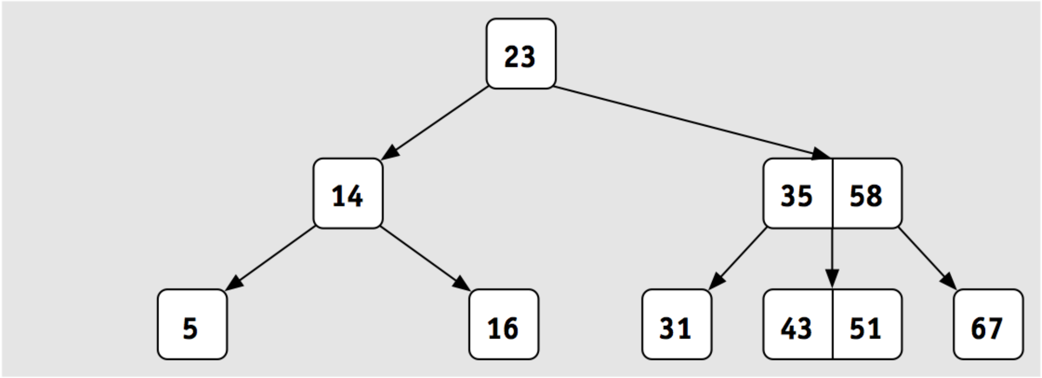
3-Nodes with Super links



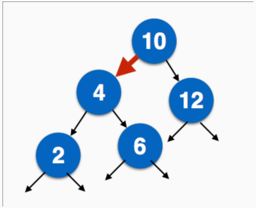
4-nodes with Super links



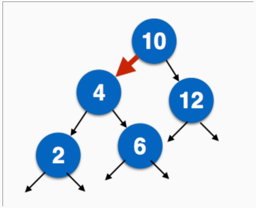
Exercise: Convert to Binary + Superlinks



Coloring *pointers* is weird...

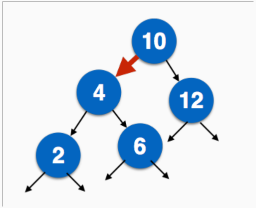


Coloring *pointers* is weird. . .

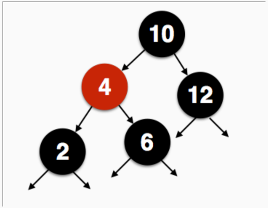


Idea: color the node, not the (incoming) edge!

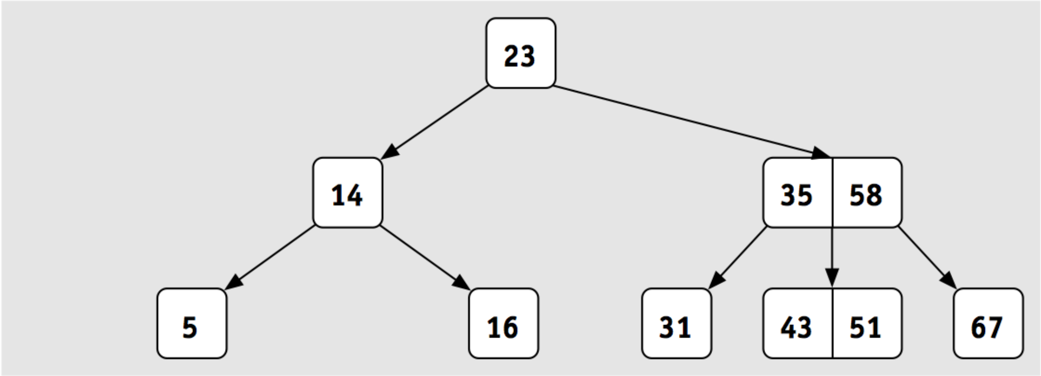
Coloring *pointers* is weird...



Idea: color the node, not the (incoming) edge!



Exercise: Convert to Binary Tree



Word of Warning

- Red-Black Trees are efficient and commonly used (e.g., `std::set`)
- But it's easy to write convoluted, messy, confusing, scary Red-Black Tree implementations!
- If you ever implement a red-black tree, refer back to the 2-3-4 implementation.

Advantages & Disadvantages

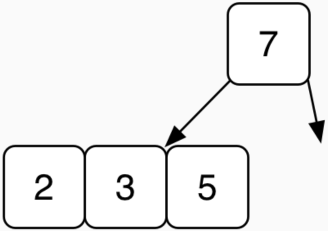
Advantages of Red-Black Trees

- All the balance properties of 2-3-4 Trees
- Only one extra bit of color per node needed over a standard BST
- Only one node type, so easier to implement than 2-3-4 Trees

Disadvantages of Red-Black Trees

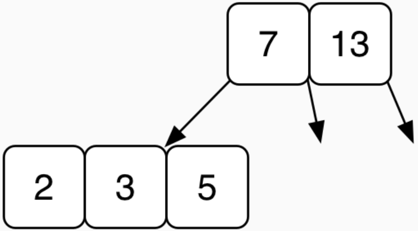
- Can be messy if you don't draw diagrams and think about the 2-3-4 equivalents during implementation.
- Rotations trickier conceptually than 2-3-4 tree operations.

Promote the 3



1. Draw the Red-Black equivalent before promotion
2. Do the promotion on the original 2-3-4 tree
3. Draw the equivalent Red-Black tree after the promotion

Different scenario: Promote the 3



1. Draw the Red-Black equivalent before promotion
2. Do the promotion on the original 2-3-4 tree
3. Draw the equivalent Red-Black tree after the promotion

Learning Targets

1. I can explain the fundamental idea behind Red-Black trees.
2. I can explain the fundamental idea behind 2-3-4 trees.
3. I can explain the fundamental idea behind Red-Black trees.