Red-Black Trees (Balanced Binary Search Trees)

**Representation Properties**

- **Black Balanced** - the number of black nodes on any path from root to a leaf is the same (black height)
- **No Double Reds**
  - No red node has a red child
- **Root Black**
  - Root is black

**Inorder**

Frontier (instead of null)

black height of 2
**Upper bound on Height**

Show height \( \leq 2 \log_2 (n+1) \)

A complete binary tree of \( l \) levels has \( 2^l - 1 \) nodes

- \( l=1 \)
  - \( 2^1 - 1 = 1 \)
- \( l=2 \)
  - \( 2^2 - 1 = 3 \)
- \( l=3 \)
  - \( 2^3 - 1 = 7 \)

\( \max \) \# nodes on a path from root to a leaf

Prove above claim by induction
In a red-black tree with black height $bh$

$$
\text{# nodes} \rightarrow n \geq 2^{bh} - 1
$$

By algebra $bh \leq \log_2(n+1)$

By No Double Reds & Root Black at least half of nodes on any path from root to a leaf are black $\Rightarrow h \leq 2 \log_2(n+1) = O(\log n)$
Non-mutating methods

min, max, search, pred, successor
in-order traversal

Just ignore the color!
It's a valid binary search tree

Cost \( O(h) = O(\log n) \)

In a binary search tree, time complexity to return all elements in range \([b, e]\) is \( O(h + k) \) where \( h \) is the height of tree and \( k \) is the number of elements in given range.
Insertion

Insert the new element in standard way as a red node.

Black Balanced \{ key properties to worry about \}

NoDoubleReds \{ too expensive to fix if violated. Keep it preserved throughout \}

Take possible NoDoubleReds violation & use recoloring & rotations to move violation up towards root until fixed or reaches root.
Case 1: extraRed’s uncle is red (extraRed could be any of y’s four grandchildren)

Recolor propagating extra red up the tree

possible violation with its parent
(no other violation of noDoubleReds + BlackBalance is preserved)
Case 2: extraRed's uncle is black
(could be mirror image)

Case 2a: extraRed
opposite child as its parent (zig-zag)

Lift the extraRed

Still have violation
(at same level
of tree) but
now zig-zig

Only violation of NoDoubleRed
Case 2: extraRed's uncle is black

Case 2b: extraRed same child as its parent (zig-zig)

Lift the extraRed's parent (after which there is no extraRed remaining)

DONE!
Example, insert 0 into below

Case 2a

Case <

Extra red

Only nodes are colored
Time complexity for insert binary search tree insertion

\[ O(h) = O(\log n) \]

insert fix up

\[ \leq h/2 \text{ case 1s (constant time each) } \leq \text{ once for case 2a+2b} \]
Deletion (details on pages 523-528 G&G 298-293 CLRS)

If node \( x \) to delete has 2 children, replace \( x \) by its successor \( y \) (could use predecessor) where \( y \) is given color of \( x \)

Then remove \( y \).

How to do this!

In other cases (\( x \) has 0 or 1 child) let \( y \) be the node to delete.
If y was red
we're done.

If y is black
we make its
child "double black"
treat it as 2 in black height