Red-Black Trees (Balanced Binary Search Trees)

**Inorder**

```
     i
    / 
   b   s
  /     
 a     o
 /     / 
a n r t
```

**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

**Frontier (instead of null)**

black height of 2
Upper bound height

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

\text{max # nodes on a path from root to a leaf}

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

\begin{align*}
  l=1 & & l=2 & & l=3 \\
  \text{leaf} & & \text{leaf} & & \text{leaf}
\end{align*}

prove by induction
In a red-black tree with black height \( bh \)

\[ n \geq 2^{bh} - 1 \]

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

By No Double Red + Root Black

\[ h \leq 2 \log_2 (n+1) \]
Non-mutating Methods

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

ignore color

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

Insert the new element in standard as a red node.

Maintain Black Balanced throughout (too hard to repair)

Take double red violation & use recoloring & rotations to move violation towards root
Case 1: Uncle red

Only double Red violation is between extraRed and its parent. No meaning to link color.
Case 2: extraRed's uncle is black

2a: extra red is zig-zag relationship with grandparent
Case 26 Zig-Zig

extraRed

lift extra Red's parent

DONE!
$O(\log n)$

- Insert new node red & set extraked to it
- While extraked has red parent
  - Apply case 1 (recolor) if possible (check if uncle is red)
  - Otherwise apply case 2a or 2b (extraked move)
- Color root black

We know we'll exit loop
Deletion (Details in text 288-293)

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

Then remove y.

In other cases (x has 0 or 1 child), let y be the node to delete y.
We need to preserve Black Balanced

double

black

parent

child

null
to delete

If y is red, we're set.