Recall:

RB tree $T$ with $n$ nodes. Show: $\text{height}(T) = O(\log n)$.

Last time: for any node $p \in T$, subtree rooted at $p$ has $\geq 2^{\text{bh}(p)} - 1$ many nodes. (Induction on height of $p$)

$\text{bh}(p) = \# \text{ black nodes encountered along any path from } p \text{ to a leaf (excluding } p).$

$n = \text{ size of tree rooted at root}[T] \geq 2^{\text{bh}(r)} - 1$

where $r = \text{root}[T].$

Notice: $\text{height}(r) \leq 2 \text{bh}(r).$

Why? Because no two red nodes in a row along any path.

So: $\text{bh}(r) \geq \frac{\text{height}(r)}{2}$
\[ n = \text{size at } r \\
\geq 2^{\frac{\text{height}(r)}{2}} - 1 \\
\geq 2^{\frac{\text{height}(r)}{2}} - 1 \\
n + 1 \geq 2^{\frac{\text{height}(r)}{2}} \\
1 \log(n + 1) \geq \frac{\text{height}(r)}{2} \\
2 \log(n + 1) \geq \text{height}(r) \\
= h \quad (\text{height of } T) \]

Implementing RB trees as a data struct.

3 basic ops:

- \text{Search}(T, x) — find item with key \( x \) in \( T \)
- \text{Insert}(T, x): insert \( x \) into \( T \).
- \text{Delete}(T, x): remove \( x \) from \( T \).

\text{Search}: just as in any BST. \( O(\log n) \) worst-case time.

\text{Insertion}: Insert \( x \) into \( T \) (assuming \( x \) is not in \( T \) already).
\( T \) empty: insert \( x \); color \( [x] := \text{black} \)
Assume $T$ not empty before inserting $x$.

2 phases:
- insert phase
  - as with
- clean-up phase
  - just as with regular BST:
    - $x$ becomes a new “leaf”
    - $\text{color}[x] := \text{red}$

Only possible violation:
$\text{color}[\text{parent}[x]] = \text{red}$.

Clean-up phase:
- if $\text{color}[\text{parent}[x]] = \text{black}$
  - then do nothing & exit.

  // parent($x$) is red

Assume that parent($x$)
- is a left child
  (if parent($x$)
  - is a right child
    - then do all of the following in mirror image):
  - (parent($x$) not root since it is red)
if \( x \) is right child, then rotate left at \( y \).

Digression: Rotations in general:

rotate left at \( p \)

rotate right at \( q \)

Rotation takes \( O(1) \) time.

End-of-digression
(If \( x \) is left child, then go on to next step):

**Either:**

- **\( B R \)**

  \[
  \begin{array}{c}
  x \\
  y \\
  \\
  R \\
  \end{array}
  \]

  * or *

- **\( B R \)**

  \[
  \begin{array}{c}
  x \\
  y \\
  \end{array}
  \]

  treat both the same:

Suppose

- **\( R B \)**

  \[
  \begin{array}{c}
  x \\
  y \\
  \end{array}
  \]

  \( w \) is \( x \)'s sibling

- **\( R R \)**

  \( y \)'s uncle

**If** \( \text{color}[w] = \text{red} \):

- \( \text{color}[x] := \text{black} \)
- \( \text{color}[w] := \text{black} \)
- \( \text{color}[z] := \text{red} \)

**What if parent \( [z] \) is red?**

**Answer:** repeat.

\[
\begin{array}{c}
A \\
R \\
R \\
\end{array}
\]

Do same with \( t \),

*as with \( x, z \) (recursive)*

Recursion ends because root is black.

**Last case to deal with:**

- **\( B R \)**

  \[
  \begin{array}{c}
  x \\
  w \\
  \end{array}
  \]

  \( \text{color}[w] = \text{black} \)

- **\( R R \)**

  \[
  \begin{array}{c}
  x \\
  w \\
  \end{array}
  \]

  \( y \)

**Rotate right at \( z \)**

*adjust colors*
The AVL condition:

A binary tree $T$ satisfies the AVL condition if any two siblings in $T$ differ in height by $\leq 1$.

Prop: If $T$ has size $n$ and has the AVL property then $T$ has height $O(\lg n)$.

Proof: Given some height $h \geq -1$ ($\text{height}(\emptyset) = -1$ by convention), let $m_h = \text{smallest possible}...
The size of any AVL tree with height $h$:

\[ m_{-1} = 0 \]
\[ m_0 = 1 \]

Assume $h > 0$

An AVL tree of height $h$:

\[
\begin{align*}
& h \\
& \left[ \begin{array}{c}
& \left[ \begin{array}{c}
& h-2 \\
& m_{h-2} \\
& m_{h-1}
& \end{array} \right]
& \end{array} \right] \\
& h-1
\end{align*}
\]

So: if $h > 0$

\[ m_h = m_{h-2} + m_{h-1} + 1 \]

\[
\begin{align*}
& m_{-1} = 0 \\
& m_0 = 1 \\
& h > 0: \ m_h = m_{h-2} + m_{h-1} + 1 \\
\end{align*}
\]

Fibonacci sequence:

\[
\begin{align*}
& F_0 = 0 \\
& F_1 = 1 \\
& n > 1: \ F_n = F_{n-2} + F_{n-1}
\end{align*}
\]