Dynamic BALANCED SEARCH TREES (Non-Random)

Objectives: search, insert, delete in $O(\log n)$ time

& always maintain $\Theta(\log n)$ height & update in $O(\log n)$ time
RED-BLACK trees

Structure:
1) nodes are colored red or black.
2) root is always black.
3) add black "dummy" leaves so every "real" node has 2 children.

The important rules
4) every red node has a black parent.
5) for any node x: all paths down to leaves contain equal number of black nodes = black-height[x]
   not including x
4) every red node has a black parent.

5) for any node $x$: all paths down to leaves contain equal number of black nodes $= \text{black-height}[x]$

$\Rightarrow$ Fails rule 5
4) every red node has a black parent.

5) for any node $x$: all paths down to leaves contain equal number of black nodes = \text{black-height}[x]

$\rightarrow$ fails rule 5 $\Rightarrow$ fix by making some nodes red.
Any root→leaf path of size $k$ must have $\geq \frac{k}{2}$ black nodes. 

So if any path is $>2$ times longer than another, we can't make it RB.

Proof that RB trees have height $< 2\log n$

1) Fact: If a tree is perfectly balanced the height is $\log n$
2) If a tree is not perfectly balanced, there is a node, $x$, at depth $d < \log n$
3) By our claim above there can’t be any other node at depth $> 2d$
No hope to recolor

... too unbalanced

Any root→leaf path of size $k$ must have $\geq \frac{k}{2}$ black nodes.

So if any path is $\geq 2$ times longer than another, we can't make it RB.

CONTRACTION
Black nodes are fully balanced.
If root\((T)\) has same black-height on all paths then height\((T')\) is perfectly balanced.

\[
\text{#leaves in } T : n+1 = \text{size}(T) + 1
\]
\[
\text{#leaves in } T' : \text{same}
\]
\[
\text{height}(T') \leq \log(n+1)
\]

Every non-leaf (still) has at least 2 children

[higher degree $\Rightarrow$ smaller height ; worst-case : binary]

Re-inserting red nodes : at most doubles height $\rightarrow$ height\((T) \leq 2 \log(n+1)\)
We have seen that RB trees are reasonably balanced: $\sim n \log n$.

- Search, min, max, next, prev: $O(\log n)$ time.

Next: how to update RB trees (insert, delete)

**Rotations** in arbitrary BSTs

$T_1$

- $x \leq a \leq y \leq b \leq z$

$T_2$

- $x \leq a \leq y \leq b \leq z$

**Right-rotate($T_1, B$)**

**Left-rotate($T_2, A$)**

$O(1)$ time
Regular insertion as RED leaf

- No black height violation introduced.
- Might have 1 red-red violation.
- If so, start fixing: locally & upward
fixing: locally & upward

Loop invariants:
• No black height violation.
• 1 red-red violation, at positions $P, I, X$

Loop outcomes:
- Obtain R-B tree
- Redefine ancestor as $x = \text{fix upward}$

Local positions considered in loop
x & p: both red if we needed to continue loop

g: grandparent of x
y: "uncle" of x

---
g must exist: **must be black**
because p can't be root

y must exist
(could be a fake leaf)
\( x \) & \( p \): both red if we needed to continue loop

\( g \): grandparent of \( x \)

\( y \): "uncle" of \( x \)

We will handle the 2 main shapes:

(without loss of generality, via symmetry)

Case 1: \( y \) is red

Cases 2 & 3: \( y \) is black
Case 1: \( y \) is red

\( \rightarrow \) Recolor \( p, g, y \)

- Preserve black-height invariant:
  - (total B nodes on any path from global root down to leaves)
- Eliminate \( P_x \) violation.
- If no new violation, then DONE
  - (note: if \( g = \text{root} \), color \( g \) black)

\[ \text{or} \]

\[ \text{or} \]
Case 1: y is red

\[ \rightarrow \text{Recolor } p, g, y \quad O(1) \]

- Preserve black-height invariant. (total B nodes on any path from global root down to leaves)
- Eliminate \( P^2_X \) violation.
- Might cause violation (otherwise DONE)

fix upward \( ? \leftrightarrow g \)

\(? \rightarrow p \quad g \rightarrow x \)

\(? \rightarrow p \quad g \rightarrow x \)

\(? \rightarrow p \quad g \rightarrow x \)

\(? \rightarrow p \quad g \rightarrow x \)

\(? \rightarrow p \quad g \rightarrow x \)

\(? \rightarrow p \quad g \rightarrow x \)
Cases 2 & 3: $y$ is black

Case 2

Case 3
Cases 2 & 3: \( y \) is black

- Preserve black-height invariant.
- Swap labels \( x \leftrightarrow p \)
- No new violation
Cases 2 & 3: \( y \) is black

- Preserve black-height invariant.

Case 2

\[
\text{Left-rotate}(p) \quad \& \quad \text{swap}(p,x)
\]

Case 3

Handling Case 3:

\[
\text{Right-rotate}(g)
\]
Cases 1 & 3: \( y \) is black

- Preserve black-height invariant.

\[ \checkmark \]

**Case 2**

Left-rotate(p) & swap(p, x)

**Case 3**

Right-rotate(g) Recolor \( p \) & \( g \)

**Handling Case 3:**
Cases 2 & 3: y is black

- Preserve black-height invariant.
- Eliminate $P_{X}$ violation in case 3.
- No new violation

Handling Case 3:

- Left-rotate(p) & swap(p, x)
- Right-rotate(g)
- Recolor p & g

DONE
Each case takes $O(1)$ time

All together $O(\log n)$ time

Possibly lots of recoloring (Case 1)

followed by Case 3 OR Case 2 $\rightarrow 3$

(1 or 2 rotations, total)

* useful property