<college>
  <student>
    <firstname>Tamanna</firstname>
    <lastname>Bhatia</lastname>
    <contact>09990449935</contact>
    <email>tammanabhatia@abc.com</email>
    <address>
      <city>Ghaziabad</city>
      <state>Uttar Pradesh</state>
      <pin>201007</pin>
    </address>
  </student>
</college>
RED-BLACK trees

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

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

Structure: 1) nodes are colored red or black.
**RED-BLACK**

**trees**

**Structure:**
1) nodes are colored **red** or **black**.
2) root is always **black**.
**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.
**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.
4) Every red node has a black parent.
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 = \( \text{black-height}[x] \)  
   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 = \( \text{black-height}[x] \)

\[ \rightarrow \text{Fails rule 5} \]
4) every red node has a black parent.

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

\[ \Rightarrow \text{fails rule 5} \Rightarrow \text{fix by making some nodes red.} \]
No hope to recolor... too unbalanced
No hope to recolor

... too unbalanced

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

Rule 4
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 $< 2\log n$.

Any root→leaf path of size $k$ must have $\geq \frac{k}{2}$ black nodes. Rule 4

So if any path is $>2$ times longer than another, we can’t make it RB.
Proof that RB trees have height < 2log\(n\)

1) Fact: If a tree is perfectly balanced the height is log\(n\)

\[\text{Any root} \rightarrow \text{leaf path of size } k \geq \frac{k}{2} \text{ black nodes.}\]

So if any path is \(> 2\) times longer than another, we can't make it RB.
Proof that RB trees have height < 2logn

1) Fact: If a tree is perfectly balanced the height is logn

2) If a tree is not perfectly balanced, there is a node, x, at depth d < logn
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 $> 2d$

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.
We have seen that RB trees are reasonably balanced: \( \sim 2\log n \)

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

Next: how to update RB trees (insert, delete)
We have seen that RB trees are reasonably balanced: $\sim 2\log n$

$\Rightarrow$ search, min, max, next, prev : $O(\log n)$ time.

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

$T_1$

$A \circ B$

$X \preceq A \preceq Y \preceq B \preceq Z$
We have seen that RB trees are reasonably balanced: \( n \approx 2 \log n \) search, min, max, next, prev: \( O(\log n) \) time.

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

**ROTATIONS** in arbitrary BSTs

\[ T_1 \quad \text{right-rotate}(T_1, B) \quad T_2 \]

\( X \leq A \leq Y \leq B \leq Z \)
We have seen that RB trees are reasonably balanced: $\sim 2\log n$.

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

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

**ROTATIONS in arbitrary BSTs**

$T_1$ to $T_2$:

- right-rotate$(T_1, B)$
- left-rotate$(T_2, A)$

$X \leq A \leq Y \leq B \leq Z$

$O(1)$ time

$X \leq A \leq Y \leq B \leq Z$
INSERTION IN A R-B TREE
INSERTION
IN A R-B TREE

Regular insertion as RED leaf
INSERTION
IN A R-B TREE

Regular insertion as RED leaf

→ No black height violation introduced.
Regular insertion as RED leaf

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

Loop invariants:
• No black height violation.
• 1 red-red violation, at positions \((p, x)\)
fixing: locally & upward

Loop invariants:
- No black height violation.
- 1 red-red violation, at positions $P_{1x}$.

Loop outcomes:
- Obtain R-B tree
- OR
- Redefine ancestor as $x$
fixing: locally & upward

Loop invariants:
• No black height violation.
• 1 red-red violation, at positions

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

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

Loop outcomes:
- Obtain R-B tree

OR
- Redefine ancestor as $X = \text{fix upward}$
Fixing: locally & upward

Loop invariants:
- No black height violation.
- 1 red-red violation, at positions (P, X)

Loop outcomes:
- Obtain R-B tree
- OR
- Redefine ancestor as $x = \text{fix upward}$
$x \& p$: both red if we needed to continue loop

g: grandparent of $x$

$y$: "uncle" of $x$
x & p: both red if we needed to continue loop

g: grandparent of x

y: "uncle" of x

2 possible shapes...
$x$ & $p$: both red if we needed to continue loop

g: grandparent of $x$

$y$: "uncle" of $x$

2 possible shapes...

...and their mirror images
X & p: both red if we needed to continue loop

g: grandparent of x

y: "uncle" of x

2 possible shapes...

...and their mirror images

<table>
<thead>
<tr>
<th>g must exist</th>
<th>because p can't be root</th>
</tr>
</thead>
<tbody>
<tr>
<td>y must exist</td>
<td>(could be a fake leaf)</td>
</tr>
</tbody>
</table>
$x \land p$: 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:

2 possible shapes...

...and their mirror images

The others can be done by symmetry:
\(x \land p\): both red if we needed to continue loop

g: grandparent of \(x\)

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

We will handle the 2 main shapes:

**Case 1**: \(y\) is red

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

![Diagram showing case 1 with two possible configurations involving nodes labeled \( g \) and \( y \).]
Case 1: \( y \) is red

\[ \Rightarrow \text{Recolor } p, g, y \]
Case 1: y is red

\[ \rightarrow \text{Recolor p, g, y} \]

• Preserve black-height invariant.
Case 1:  \( y \) is red

- Recolor \( p, g, y \)
- Preserve black-height invariant.
  (total \( B \) nodes on any path from global root down to leaves)
- Eliminate \( P_X \) violation.
Case 1: $y$ is red

1. 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 (if $g = \text{root}$) then DONE (color $g$ 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 \( px \) violation.

- Might cause violation \( ?g \)

otherwise DONE
Case 1: \( y \) is red

1. Recolor \( p, g, y \)

• Preserve black-height invariant.
  (total \( B \) nodes on any path from global root down to leaves)

• Eliminate \( P \times \) violation.

• Might cause violation \( \frac{?}{g} \)

  otherwise DONE

fix upward

\( \leftarrow \frac{?}{g} \) becomes \( p \)
\( g \) becomes \( x \)

(Just changing labels)
Case 1: \( y \) is red

4. Recolor \( p, g, y \)  \( O(1) \)

- Preserve black-height invariant.
  - (total \( B \) nodes on any path from global root down to leaves)
- Eliminate \( \frac{p}{x} \) violation.
- Might cause violation \( \frac{g}{?} \)
  - otherwise DONE

\[ \text{fix upward} \]

\[ \Rightarrow \text{\( ? \) becomes \( p \) \( \Rightarrow \text{Invariants restored} \) \( g \) becomes \( x \) \]
Cases 2 & 3: \( y \) is black
Cases 2 & 3: \( y \) is black

Case 2

Case 3
Cases 2 & 3: y is black

Case 2:

\[\text{Left-rotate}(p)\]
Cases 2 & 3: \( y \) is black

Case 2: \( \text{Left-rotate}(p) \)
Cases 2 & 3: $y$ is black

- Preserve black-height invariant.

Case 2

```
          g
         /  \
        p    y
       /    /
      A    X  
     /   /   \
    B   C    y
```

Left-rotate($p$)

```
          g
         /  \
        p    y
       /    /
      A    X  
     /   /   \
    B   C    y
```

```
          g
         /  \
        X    p
       /    /
      A    C  
     /   /   \
    B   B    y
```
Cases 2 & 3: $y$ is black

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

Case 2

Case 3
Cases 2 & 3: $y$ is black

Case 2

Case 3

Handling Case 3:
Cases 1 & 3:  $y$ is black

Case 2

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

Case 3

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

### Case 2
- Left-rotate(p)
- & swap(p, x)

### Case 3

Handling Case 3:
- Right-rotate(g)
Cases 1 & 3: \( y \) is black

- Preserve black-height invariant.

Case 2

\[
\begin{align*}
&\text{Left-rotate}(p) \\
&\text{& swap}(p, x)
\end{align*}
\]

Case 3

Handling Case 3:

\[
\begin{align*}
&\text{Right-rotate}(g)
\end{align*}
\]
Cases 2 & 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 $P_X^P$ violation in case 3.

Handling Case 3:

Case 2

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

Case 3

Right-rotate($g$) Re-color $p$ & $g$
Cases 2 & 3: $y$ is black

- Preserve black-height invariant.
- Eliminate $P_x$ violation in case 3.
- No new violation

Handling Case 3:

Case 2

- Left-rotate($p$)
- $\text{swap}(p, x)$

Case 3

- Right-rotate($g$)
- Recolor $p$ & $g$
Cases 1 & 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
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
Case 1

Diagram:

- Insert 15

Tree:

- 7
- 3
- 18
- 10
- 22
- 8
- 11
- 26
- 15

Case 2 and Case 3 are not shown in this diagram.
Case 1

Case 2

insert 15
Case 1

Case 2

rotate-right(18)

Case 3
Case 1

insert 15

Case 2

rotate-right(18)

Case 3

rotate-left(7)

Done!