Red-Black Trees

Red-Black Properties
Balance Lemma
Rotate Operations
Insert and Delete
A red-black tree is a binary search tree with one extra bit of storage per node: its color, which can be either RED or BLACK. By constraining the node colors on any simple path from the root to a leaf, red-black trees ensure that no such path is more than twice as long as any other, so that the tree is approximately balanced.

Each node of the tree now contains the attributes color, key, left, right, and p. If a child or the parent of a node does not exist, the corresponding pointer attribute of the node contains the value NIL. We shall regard these NILs as being pointers to leaves (external nodes) of the binary search tree and the normal, key-bearing nodes as being internal nodes of the tree.

A red-black tree is a binary tree that satisfies the following red-black properties:

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.
Red-Black Tree Example 2

RB Properties
Example 1
Example 2
Example 3
RB Properties
Rotations
Example Rotation
Insert
Insert 1
Insert 2
Delete 1
Delete 2
Delete 3
Delete 4

CS 3343 Analysis of Algorithms
Red-Black Trees – 4
Red-Black Tree Example 3

RB Properties
Example 1
Example 2
Example 3
RB Properties
Rotations
Example Rotation
Insert
Insert 1
Insert 2
Delete 1
Delete 2
Delete 3
Delete 4

(c)

CS 3343 Analysis of Algorithms
Red-Black Trees – 5
Balance Lemma

We call the number of black nodes on any simple path from, but not including, a node $x$ down to a leaf the **black-height** of the node, denoted $bh(x)$. By property 5, the notion of black-height is well defined, since all descending simple paths from the node have the same number of black nodes. We define the black-height of a red-black tree to be the black-height of its root.

The following lemma shows why red-black trees make good search trees.

**Lemma 13.1**
A red-black tree with $n$ internal nodes has height at most $2 \log(n + 1)$. 
Left-Rotate and Right-Rotate

Left-Rotate\( (T, x) \)

Right-Rotate\( (T, y) \)
Rotation Example

- RB Properties
  - Example 1
  - Example 2
  - Example 3
- Rotations
  - Example
  - Rotation
- Insert
  - Insert 1
  - Insert 2
- Delete
  - Delete 1
  - Delete 2
  - Delete 3
  - Delete 4

**LEFT-ROTATE(T, x)**

Diagram showing the process of a left-rotate operation in a Red-Black tree.
Red-Black Insert

\[
\text{RB-Insert}(T, z)
\]

1. \text{Tree-Insert}(T, z)
2. \( z.color = \text{RED} \)
3. \textbf{while} \( z.p \neq \text{NIL} \) and \( z.p.color == \text{RED} \) and \( z.p.p.left.color == \text{RED} \) and \( z.p.p.right.color == \text{RED} \)
4. \( z.p.p.left.color = \text{BLACK} \)
5. \( z.p.p.right.color = \text{BLACK} \)
6. \( z = z.p.p \)
7. \( z.color = \text{RED} \)
8. \textbf{if} \( z.p \neq \text{NIL} \) and \( z.p.color == \text{RED} \)
9. rotate \( z, z.p, \) and \( z.p.p \) around so median is \text{BLACK} and other two are \text{RED} children
10. \( T.root.color = \text{BLACK} \)
Insert Illustration 1

(a) C
A
α
β
γ
B
z
D
y
δ
ε

(b) C
B
γ
D
y
δ
ε
A
z
α
β

new z
C
A
α
B
β
γ
D
δ
ε

CS 3343 Analysis of Algorithms
Red-Black Trees – 10
Insert Illustration 2

RB Properties
Example 1
Example 2
Example 3
RB Properties
Rotations
Example Rotation
Insert
Insert 1
Insert 2
Delete 1
Delete 2
Delete 3
Delete 4
Red-Black Delete, Part 1

\[
\text{RB-Delete}(T, z) = \begin{cases} 
\text{if } z.\text{left} == \text{NIL} & \text{TRANSPLANT}(T, z, z.\text{right}) \\
\text{if } z.\text{right} == \text{NIL} \text{ and } z.\text{color} == \text{BLACK} & \text{RB-Delete-Fix}(T, \text{NIL}, z.\text{p}, z.'s \text{ sibling}) \\
\text{else } z.\text{right}.\text{color} == \text{BLACK} & \text{else if } z.\text{right} == \text{NIL} \\
\text{TRANSPLANT}(T, z, z.\text{left}) & \text{else}
\end{cases} \\
\text{if } z.\text{left}.\text{color} == \text{BLACK} \\
\text{else}
\text{a} = \text{Tree-Minimum}(z.\text{right}) \\
\text{RB-Delete}(T, a) \\
\text{TRANSPLANT}(T, z, a) \\
a.\text{color} == z.\text{color} \\
a.\text{left} == z.\text{left} \\
a.\text{right} == z.\text{right}
\]

CS 3343 Analysis of Algorithms
Red-Black Delete, Part 2

RB-DELETE-FIX\( (x, y, w) \)

1. while \( x \neq T.\text{root} \) and \( (x == \text{NIL} \text{ or } x.\text{color} == \text{BLACK}) \)
2. \hspace{1em} if \( w.\text{color} == \text{RED} \)
3. \hspace{2em} rotate \( w \) so \( w \) is parent of \( y \)
4. \hspace{2em} \( y = x.\text{p} \)
5. \hspace{2em} \( w = \text{sibling of } x \)
6. \hspace{1em} if \( w \)'s children are both \text{BLACK} (or both \text{NIL})
7. \hspace{2em} \( w.\text{color} = \text{RED} \)
8. \hspace{2em} \( x = y \)
9. \hspace{2em} if \( x = T.\text{root} \) exit loop
10. \hspace{2em} \( y = x.\text{p} \)
11. \hspace{2em} \( w = \text{sibling of } x \)
12. \hspace{1em} else
13. \hspace{2em} use \text{RED} child of \( w \) to fix tree
14. \hspace{2em} exit loop
15. \hspace{1em} \( x.\text{color} = \text{BLACK} \)
Delete Illustration 1

(a) $x \quad A \quad \alpha \quad \beta \\
\quad C \quad \gamma \quad \delta \quad \epsilon \\
\quad E \quad \zeta$

(b) $x \quad A \quad \alpha \quad \beta \\
\quad C \quad \gamma \quad \delta \quad \epsilon \\
\quad D \quad w \quad E \quad \zeta$

Case 1

Case 2

Example Rotation
Insert
Insert 1
Insert 2
Delete 1
Delete 2
Delete 3
Delete 4
Delete Illustration 2

(c)  

```
(c) x A B c D w
  α β γ δ ε ξ
```

Case 3

```
Case 3
```

(d)  

```
(d) x A B c D w
  α β γ δ ε ξ
```

Case 4

```
Case 4
```

Red-Black Trees – 15