Red-Black Trees

Red-Black Properties
Balance Lemma
Rotate Operations
Insert and Delete

Red-Black Properties

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 1
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
Red-Black Insert

RB-INSERT($T, z$)
1. TREE-INSERT($T, z$)
2. $z$.color = RED
3. while $z.p \neq NIL$ and $z.p$.color == RED and $z.p$.left.color == RED and $z.p$.right.color == RED
4. $z.p$.left.color = BLACK
5. $z.p$.right.color = BLACK
6. $z = z.p$
7. $z$.color = RED
8. if $z.p \neq NIL$ and $z.p$.color == RED
9. rotate $z$, $z.p$, and $z.p.p$ around so median is BLACK and other two are RED children
10. $T$.root.color = BLACK
Red-Black Delete, Part 1

**RB-Delete**($T, z$)
1. if $z.left \equiv \text{NIL}$
2. Transplant($T, z, z.right$)
3. if $z.right \equiv \text{NIL}$ and $z.color \equiv \text{BLACK}$
4. RB-Delete-Fix($T, \text{NIL}, z.p, z$'s sibling)
5. else $z.right.color \equiv \text{BLACK}$
6. else if $z.right \equiv \text{NIL}$
7. Transplant($T, z, z.left$)
8. $z.left.color \equiv \text{BLACK}$
9. else
10. $a \equiv \text{Tree-Minimum}(z.right)$
11. RB-Delete($T, a$)
12. Transplant($T, z, a$)
13. $a.color \equiv z.color$
14. $a.left \equiv z.left$
15. $a.right \equiv z.right$

Delete Illustration 1

Red-Black Delete, Part 2

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

Delete Illustration 2