CSE 2320 Notes 12: Red-Black Trees

(CLRS 13.1, 13.3)

12.A. Structural Properties

A red-black tree is a binary search tree whose height is $O(\log n)$ in the number of keys ($n$) stored.

1. Every node is colored red or black. (Colors are only examined during insertion and deletion)

2. Every “leaf” (the sentinel) is colored black.

3. Both children of a red node are black.

4. Every simple path from a child of node X to a leaf has the same number of black nodes.

This number is known as the black-height of X (bh(X)). These are not stored, but appear below nodes in some diagrams.

Example (http://ranger.uta.edu/~weems/NOTES2320/REDBLACKC/notes12.page1.dat):
Observations:

1. A red-black tree with \( n \) internal nodes ("keys") has height at most \( 2 \lg(n+1) \).
2. If a node \( X \) is not a leaf and its sibling is a leaf, then \( X \) must be red.
3. There may be many ways to color a binary search tree to make it a red-black tree.
4. If the root is colored red, then it may be switched to black without violating structural properties. (Implementations usually color root as black.)

12.B. INSERTION

1. Start with unbalanced insert of a "data leaf" (both children are the sentinel).
2. Color of new node is _________.
3. May violate structural property 3. Leads to three cases, along with symmetric versions.

The \( x \) pointer points at a red node whose parent might also be red.

Case 1:
Case 2:

= red
= black

Case 3:

= red
= black

Example 1 (http://ranger.uta.edu/~weems/NOTES2320/REDBLACKC/notes12.ex1.dat):

= red
= black
Insert 15

Insert 13

Insert 75
Example 2 (http://ranger.uta.edu/~weems/NOTES2320/REDBLACKC/notes12.ex2.dat):
Insert 75

```
10

20

40

60

100

140

160

170

20

30

50

70

80

90

100

110

120

130

150

75

x

75

1
```

1
link RBinsert(link h, Item item, int sw, int siblingRed, link hParent)
// CLRS, 3rd ed., RB tree insertion done recursively without parent pointers.
// Also includes tracing. See 2320 notes. BPW
// h is present node in search down tree.
// Returns root of modified subtree.
// item is the Item to be inserted.
// sw == 1 <=> h is to the right of its parent.
// siblingRed has color of h's sibling.
// hParent has h's parent to facilitate case 1 color flips.
{
    Key v = key(item);
    link before; // Used to trigger printing of an intermediate tree
    tracePrint("Down", h);
    if (h == z)
        return NEW(item, z, z, 1); // Attach red leaf
    if (eq(v, h->item))
        return h;
    else if (less(v, h->item)) {
        tracePrint("Insert left", h);
        before=h->l;
        h->l = RBinsert(h->l, item, 0, h->r->red, h); // Insert in left subtree
        if (trace==2 && before!=h->l) // Has a rotation occurred?
            STprintTree();
        if (h->l->red) {
            if (h->red)
                if (sw)
                    if (siblingRed) {
                        tracePrint("Case ~1l", hParent);
                        hParent->red = 1;
                        hParent->l->red = 0;
                        hParent->r->red = 0;
                        if (trace==2)
                            STprintTree();
                        }
                    else { // Future case 3
                        tracePrint("Case -2", h);
                        h = rotR(h); // Set up case ~3 after return
                    }
            else { // Future case 3
                tracePrint("Case -2", h);
                h = rotR(h); // Set up case ~3 after return
            }
        }
        else if (siblingRed) {
            tracePrint("Case 1l", hParent);
            hParent->red = 1;
            hParent->l->red = 0;
            hParent->r->red = 0;
            if (trace==2)
                STprintTree();
        } // Future case 3
    }
    else if (!h->r->red && h->l->l->red) {
        tracePrint("Case 3", h);
        h = rotR(h);
        h->red = 0;
        h->r->red = 1;
    }
}
else {
    tracePrint("Insert right", h);
    before = h->r;
    h->r = RBinsert(h->r, item, 1, h->l->red, h); // Insert in right subtree
    if (trace==2 && before!=h->r)  // Has a rotation occurred?
        STprintTree();
    if (h->r->red) {
        if (h->red)
            if (!sw)
                if (siblingRed) {
                    tracePrint("Case 1r", hParent);
                    hParent->red = 1;
                    hParent->l->red = 0;
                    hParent->r->red = 0;
                    if (trace==2)
                        STprintTree();
                } else {
                    tracePrint("Case 2", h);
                    h = rotL(h);  // Set up case 3 after return
                }
        else if (siblingRed) {
            tracePrint("Case ~1r", hParent);
            hParent->red = 1;
            hParent->l->red = 0;
            hParent->r->red = 0;
            if (trace==2)
                STprintTree();
        } else {
            ;  // Future case ~3
        } else if (!h->l->red && h->r->r->red) {
            tracePrint("Case ~3", h);
            h = rotL(h);
            h->red = 0;
            h->l->red = 1;
        }
    }
}

fixN(h);
tracePrint("Up", h);
return h;