Data Structures and Algorithms

Red-Black Tree Insertion

Start out by using a regular binary search tree insertion. Color the newly inserted node red. Call *insertionFixUp*, passing a pointer to the newly inserted node.

```
    function insertionFixUp(x)    // x is the newly inserted node
        {
        loop
            {
            if (x is root) exit the loop
            if (parent is black) exit the loop
            if (uncle is red)
                {
                color parent black
                color uncle black
                color grandparent red
                x = grandparent
                }
            else
                {
                // uncle must be black

                if (x and parent are not linear)
                    {
                    rotate x to parent
                    x = old parent
                    parent = old x
                    }

                color parent black
                color grandparent red
                rotate parent to grandparent
                exit the loop
                }
            }

        color root black
        }
```

Note that in this pseudocode, there are no references to leftness and rightness. This issue is deferred to the helper functions. For example, the *uncle is red* test could be implemented as:

```
   color(uncle(x)) == RED
```

where *uncle* is implemented as:

```
    function uncle(x)
        {
        if (isLeftChild(parent(x)))
            return rightChild(grandparent(x));
        else
            return leftChild(grandparent(x));
        }
```

The *color* function returns the color field of the given node, unless the given node is null, in which case it returns `BLACK`:

```
    function color(x)
        {
        if (isNull(x))
            return BLACK;
        else
            return x.color;
        }
```

The parent of the root node should be null; thus the color of the parent of the root is BLACK.

Next: *Deleting from a red-black trees*