Data Structures and Algorithms

Red-Black Tree Deletion

Start out by swapping the value to be deleted to the appropriate leaf (unlike most red-black code, leaves in this implementation are normal binary search tree leaves). Call this node \( p \). Pass a pointer to \( p \) to \( \text{deletionFixUp} \). After \( \text{deletionFixUp} \) returns, prune \( p \) from the tree.

```plaintext
function \text{deletionFixUp}(x)
{
    loop
        {           
            if (x is root) exit the loop
            if (x is red) exit the loop
            if (sibling is red)
                {
                    color parent red
                    color sibling black
                    rotate sibling to parent
                    // should have black sibling now
                }
            else if (nephew is red)
                {
                    color sibling the same as parent
                    color parent black
                    color nephew black
                    rotate sibling to parent
                    // subtree and tree is BH balanced
                    exit the loop
                }
            else if (niece is red)
                {
                    // nephew must be black
                    color niece black
                    color sibling red
                    rotate niece to sibling
                    // should have red nephew now
                }
            else // sibling, niece, and nephew must be black
                {
                    color sibling red
                    x = parent
                    // this subtree is BH balanced, but tree is not
                }
    }
    color x black
}
```

Like the \textit{uncle} function, the \textit{nephew}, \textit{niece}, and \textit{sibling} functions handle leftness and rightness issues.