CS 260
Red Black Trees

Balanced vs Unbalanced Trees

When working with a binary search tree, the big O time for adding, finding, and removing are all bounded by the height of the tree. For a completely balanced tree, where the difference between the shortest and longest paths from the root to a leaf is one, the height is $\log_2 N$ where there are N nodes in the tree. Thus the performance is O(Log N). In the worst case, for example if values were sorted in ascending order before being added to the tree, each value would be added to the right of the preceding node and the tree appears to be a linked list starting at the root. In this case, the height of the tree is equal to the number of nodes and the performance has decreased to being O(N).

There are multiple ways to keep binary search trees balanced. This document presents one of them, creating a red-black tree.

Red-Black Trees

Red-black trees are binary search trees that have additional properties and requirements. By following these requirements, the trees are automatically kept in a balanced state. To better understand the following description, there is a simulation of a red-black tree available at https://www.cs.usfca.edu/~galles/visualization/RedBlack.html.

Red-black trees add a color variable to each node. The color can either be red or black. There are four rules that the tree must obey:

- Every node is either red or black
- Root is always black
- If a node is red, its children must be black
- Every path from root to leaf must have the same black node count

When adding a new node, the following algorithm is used:

- Color flips as needed on the way down
    - When a black node with two red children is encountered
        - Change node to red
        - Change children to black

- Insert the new node
    - Inserted nodes are always red (unless the first node added to a tree)

- Finally do rotations as needed
    - If parent of new node is black, no problem
    - If parent is red
        - If new node is on the outside
            - Change color of grandparent
            - Change color of parent
            - Rotate to make parent top

        - If new node is on the inside
            - Rotate new node to top, parent down
            - Then parent is on outside, so use previous algorithm
    - Note that the above step only occurs when new node is an only child
        - Either the parent was black and no problem
        - Or sibling triggered the fix
    - Rotations on the way down the tree
        - If color flips cause parent and child to be both red
        - Same flips as above