Red-Black Trees

Definitions
and
Bottom-Up Insertion
Red-Black Trees

• Definition: A red-black tree is a binary search tree in which:
  – Every node is colored either Red or Black.
  – Each NULL pointer is considered to be a Black “node”.
  – If a node is Red, then both of its children are Black.
  – Every path from a node to a NULL contains the same number of Black nodes.
  – By convention, the root is Black

• Definition: The black-height of a node, X, in a red-black tree is the number of Black nodes on any path to a NULL, not counting X.
A Red-Black Tree with NULLs shown
Black-Height of the tree (the root) = 3
Black-Height of node “X” = 2
A Red-Black Tree with
Black-Height = 3
Black Height of the tree?
Black Height of X?
Theorem 1 – Any red-black tree with root $x$, has $n \geq 2^{bh(x)} - 1$ nodes, where $bh(x)$ is the black height of node $x$.

Proof: by induction on height of $x$. 
Theorem 2 – In a red-black tree, at least half the nodes on any path from the root to a NULL must be Black.

Proof – If there is a Red node on the path, there must be a corresponding Black node.

Algebraically this theorem means

\[ bh( x ) \geq h/2 \]
Theorem 3 – In a red-black tree, no path from any node, X, to a NULL is more than twice as long as any other path from X to any other NULL.

Proof: By definition, every path from a node to any NULL contains the same number of Black nodes. By Theorem 2, at least \( \frac{1}{2} \) the nodes on any such path are Black. Therefore, there can no more than twice as many nodes on any path from X to a NULL as on any other path. Therefore the length of every path is no more than twice as long as any other path.
Theorem 4 –
A red-black tree with \( n \) nodes has height
\[ h \leq 2 \lg(n + 1). \]
Proof: Let \( h \) be the height of the red-black tree with root \( x \). By Theorem 2,
\[ bh(x) \geq h/2 \]
From Theorem 1, \( n \geq 2^{bh(x)} - 1 \)
Therefore \( n \geq 2^{h/2} - 1 \)
\[ n + 1 \geq 2^{h/2} \]
\[ \lg(n + 1) \geq h/2 \]
\[ 2\lg(n + 1) \geq h \]
Bottom –Up Insertion

- Insert node as usual in BST
- Color the node Red
- What Red-Black property may be violated?
  - Every node is Red or Black?
  - NULLs are Black?
  - If node is Red, both children must be Black?
  - Every path from node to descendant NULL must contain the same number of Blacks?
Bottom Up Insertion

• Insert node; Color it Red; X is pointer to it
• Cases
  0: X is the root -- color it Black
  1: Both parent and uncle are Red -- color parent and uncle Black, color grandparent Red. Point X to grandparent and check new situation.
  2 (zig-zag): Parent is Red, but uncle is Black. X and its parent are opposite type children -- color grandparent Red, color X Black, rotate left(right) on parent, rotate right(left) on grandparent
  3 (zig-zig): Parent is Red, but uncle is Black. X and its parent are both left (right) children -- color parent Black, color grandparent Red, rotate right(left) on grandparent
Case 1 – U is Red

Just Recolor and move up

3/20/2006
Case 2 – Zig-Zag

Double Rotate
  X around P; X around G

Recolor G and X
Case 3 – Zig-Zig

Single Rotate P around G

Recolor P and G

3/20/2006
Asymptotic Cost of Insertion

- $O(lg \, n)$ to descend to insertion point
- $O(1)$ to do insertion
- $O(lg \, n)$ to ascend and readjust == worst case only for case 1

- Total: $O(\log \, n)$
Top-Down Insertion

An alternative to this “bottom-up” insertion is “top-down” insertion.

Top-down is iterative. It moves down the tree, “fixing” things as it goes.

What is the objective of top-down’s “fixes”? 
Insert 4 into this R-B Tree
Insertion Practice

Insert the values 2, 1, 4, 5, 9, 3, 6, 7 into an initially empty Red-Black Tree