Chapter 11

Red-Black Trees

(Revision Date: 26 March 1999)

©1999, Thomas A. Anastasio. May not be reproduced in any form without written permission of the author.

11.1 Red-Black Trees

(These notes on red-black trees expound on Chapter 14 of "Introduction to Algorithms," Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, MIT Press, 1992.)

• Red-black trees are binary search trees with a more relaxed notion of balance than AVL trees.

• In a red-black tree, no path from a node N to a leaf is more than twice as long as any other (proved in Theorem 11.3.

• For a red-black tree of n nodes and height h, $h \leq 2\lg(n+1)$ (proved in Theorem 11.4).

◇ **Definition**: A red-black tree is a binary search tree with the following additional properties:

1. Every node is either red or black.
2. Each `NULL` pointer is considered to be a black node.
3. If a node is red, then *both* of its children are black.
4. *Every* path from a node to a leaf contains the same number of black nodes.

Although it is not necessary for the definition, we adopt the further property that the root is black.

◇ **Definition**: The *black height* of a node N in a red-black tree is the number of black nodes on any path to a leaf, not counting N itself.

Figure 11.1 shows a full binary search tree of 15 nodes (`NULL` pointers not shown), then a number of equivalent red-black trees with the `NULL` pointers shown as black nodes.

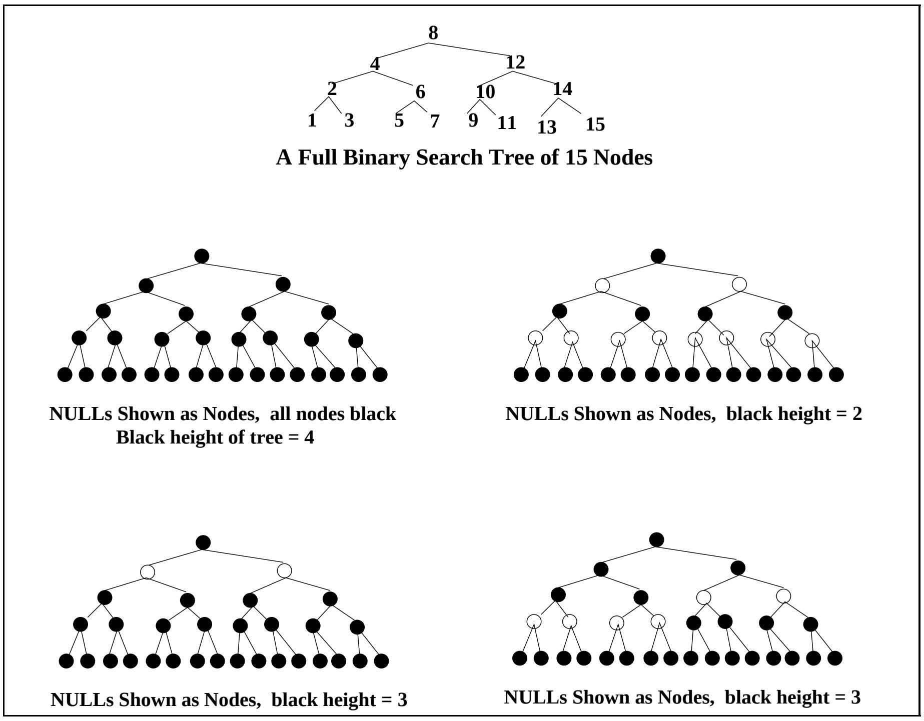

Figure 11.1: Examples of Red-Black Trees Based on Full BST of 15 Nodes

11.1.1 Bottom-Up Insertion In a Red-Black Tree

• The first phase of insertion is done in the usual binary search tree fashion. Recursively descend the tree and insert the new node as a leaf.

• Every node is red upon insertion. This *may* cause violation of red-black properties.

What red-black properties *could* be violated by insertion of a node? The binary search tree property would not be violated since insertion is done as for binary search trees. Looking at the additional properties for red-black trees:

1. Every node is red or black: **not** violated.

2. The leaf nodes are the `NULL` pointers from the nodes: **not** violated.

3. Every leaf (`NULL`) is black: **not** violated.

4. If a node is red, then *both* of its children are black: **not** violated for the inserted node (both `NULL` children are black); but **may be** violated for the parent of the inserted node.

5. *Every* path from a node to a descendent leaf contains the same number of black nodes: **not** violated.
   Proof: Suppose the path to the insertion point contained k black nodes before insertion. Insertion of the red node would reduce the path to $k-1$ black nodes, but the `NULL`s on the inserted node add 1 black. Therefore the number of black nodes on the paths is unchanged.

• The only violation possibility is if the inserted node has a red parent.

• Therefore, the only situation in which adjustment of the tree must be done after insertion is when the parent of `X` is red.

Details of the Insertion Process

Figure 11.2 summarizes the rules for insertion of a node into a red-black tree. At the start, `X` is a pointer to the newly-inserted node.

• Note: the inserted node does not move. The pointer `X` does.

• The color of the "uncle" node (sibling of parent node) is important in determining the actions to be taken.

There are four cases identified in the figure:

Case 0 `X` is the root.

Case 1 Both parent and parent's sibling ("uncle") are red.

Case 2 Parent is red but uncle is black. `X` and its parent are both left or both right children.

Case 3 Parent is red but uncle is black. `X` and its parent are opposite type children.

Note: In the Heileman text, Case 1 corresponds to Figure 11.11(a), Case 2 corresponds to Figure 11.11(c), and Case 3 corresponds to Figure 11.11(b).

Example of Insertion into a Red-Black Tree

Figure 11.3 shows how a red-black tree is modified after insertion of the node with value **4**. For clarity, the `NULL` nodes are not explicitly shown.

1. `X` points to the inserted node **4**. Node **4** is red since every node is red upon insertion. Since the parent of node **4** is red, correction is necessary.

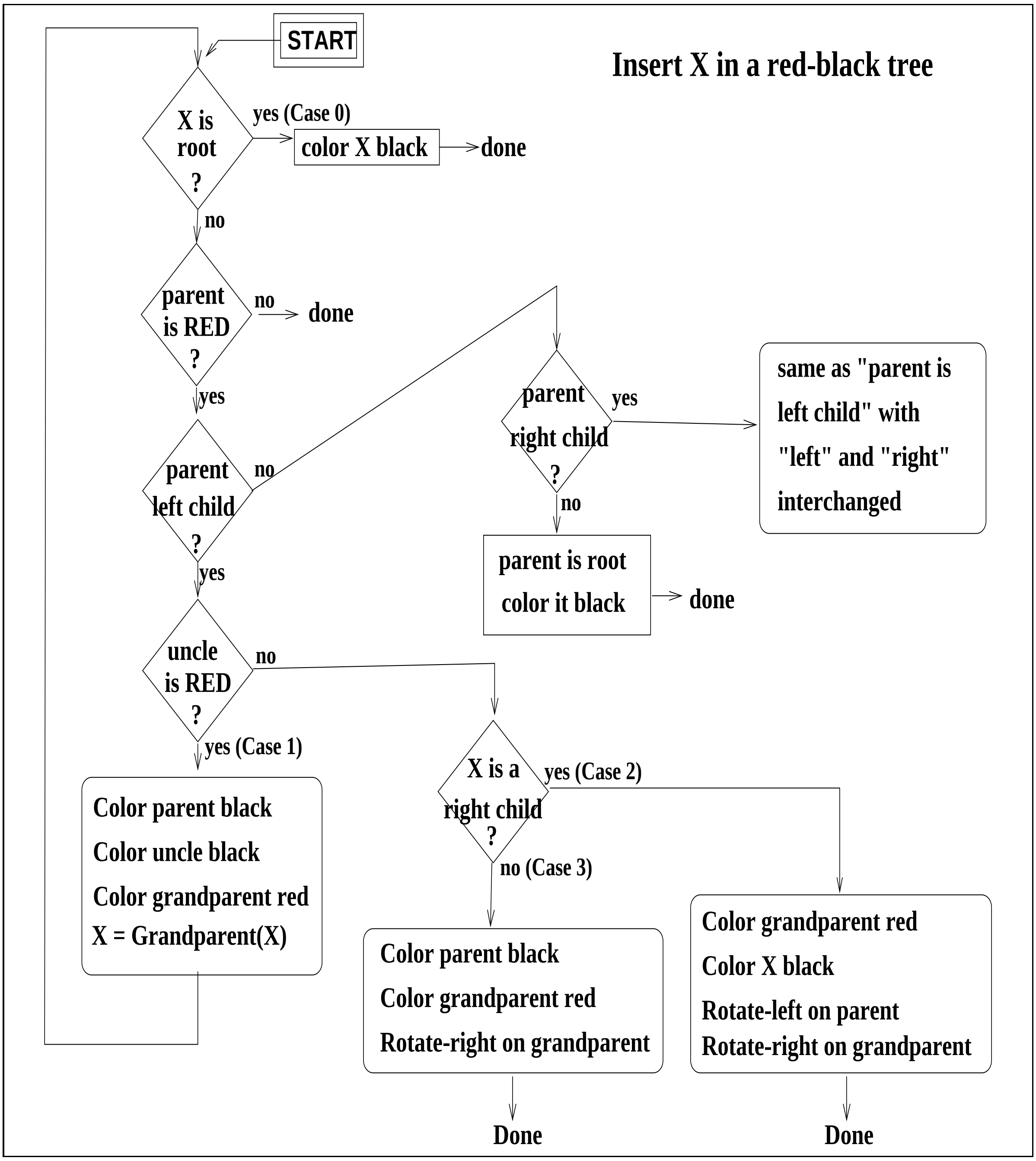

Figure 11.2: Flow Chart for Insertion in Red-Black Trees

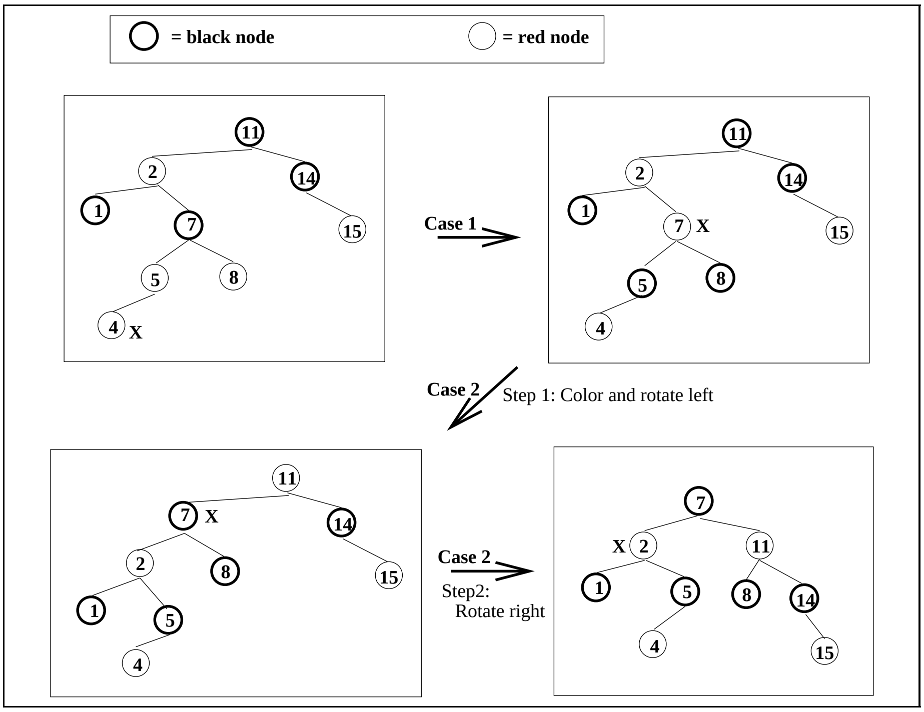

Figure 11.3: Example of Insertion in a Red-Black Tree

Node `4`s uncle (node `8`) is red, so case 1 applies.

(a) Color the parent (node `5`) black.

(b) Color the uncle (node `8`) black.

(c) Color the grandparent (node `7`) red.

(d) Point `X` at the grandparent.

2. Now, `X` points to node `7` which has a red parent (node `2`). The uncle of node `7` (namely node `14`) is black and `X` is a right child. Therefore, case 2 applies.

Step 1

(a) Color the grandparent (namely node `11`) red.

(b) Color `X` (node `7`) black.

(c) Rotate left about `X`'s parent node `2`.

Step 2 Rotate right about the grandparent node `11`.

Theorem 11.1 Any red-black tree, with root x, has at least $n = 2^{\mathrm{bh}(x)} - 1$ internal nodes, where $\mathrm{bh}(x)$ is the black-height of node x.

$\triangleright$ **Proof**: by induction on height of x.
Base: x is a leaf (no internal nodes), height is zero, $\mathrm{bh}(x) = 0$. $2^0 - 1 = 0$. Therefore true for base case.
IH: Let x be an internal node with two children (which may be `NULL`, and assume the theorem holds for all nodes of height less than x.
If a child of x is red, its black height is $\mathrm{bh}(x)$. If the child is black, its black height is $\mathrm{bh}(x)-1$. In any event, the IH holds for the children since their height is less than that of x. Therefore, each child subtree has at least $2^{\mathrm{bh}(x)-1} - 1$ internal nodes. Therefore, the tree rooted at x has at least

$$2(2^{\mathrm{bh}(x)-1} - 1) + 1 = 2^{\mathrm{bh}(x)} - 1$$

internal nodes. (The $+1$ counts node x itself.) $\triangleleft$

Theorem 11.2 In a red-black tree, at least half of the nodes on any path from root to a leaf must be black.

$\triangleright$ **Proof**: If there is a red node, there must also be a black node. $\triangleleft$

Theorem 11.3 In a red-black tree, no path from any node N to a leaf is more than twice as long as any other path from N to any other leaf.

$\triangleright$ **Proof**: By definition, every path from a node to a leaf contains the same number of black nodes. By Theorem 11.2, at least half of the nodes on any such path must be black. Therefore, there can be no more than twice as many red nodes on any path from a node to a leaf. Therefore the length of every path is no more than twice as long as that of any other path. $\triangleleft$

Theorem 11.4 A red-black tree with n internal nodes has height $h \leq 2\lg(n+1)$

$\triangleright$ **Proof**: Let h be the height of the red-black tree. By theorem 11.2, $\mathrm{bh}(x) \geq h/2$. Therefore,

$$n \geq 2^{\frac{h}{2}} - 1$$

$$n - 1 \geq 2^{\frac{h}{2}}$$

$$\lg(n-1) \geq \frac{h}{2}$$

$$2\lg(n-1) \geq h$$

$\triangleleft$

Asymptotic Cost of Insertion

- Theorem 11.4 states that the height of a red-black tree is $O(\lg n)$

  Therefore, cost of insertion is:

  - $O(\lg n)$ to descend to the insertion point.
  - $O(1)$ to do the insertion.
  - $O(\lg n)$ to ascend. Note that if case 2 or case 3 are done, the loop is terminated – these cases are done only once).

  Therefore, insertion is in $O(\lg n)$.