# CST 405
# Algorithm Analysis & Design

Al Lake
Oregon Institute of Technology

Chapter 13

Red-Black Trees

# Red-Black Trees

- A red-black tree is a binary search tree with a color: RED or BLACK.
- By constraining the way nodes can be colored on any path from the root to a leaf, red-black trees ensure that no such path is more than twice as long as any other, so that the tree is approximately balanced.
- The balanced nature of red-black trees guarantees that basic operations run in O(lg n) worst-case time.

# Red-Black Tree Properties

- Every node is either red or black.
- The root is black.
- Every leaf (NIL) is black.
- If a node is red, then both its children are black.
- For each node, all paths from the node to descendant leaves contain the same number of black nodes.

# Red-Black Tree Structure

- A red-black tree is organized in a binary tree.
- A binary tree can be represented by a linked data structure in which each node is an object.
- A red-black tree contains the following:
  - Left child node
  - Right child node
  - Parent node
  - Key
  - Color (red or black)

# Left-Rotate

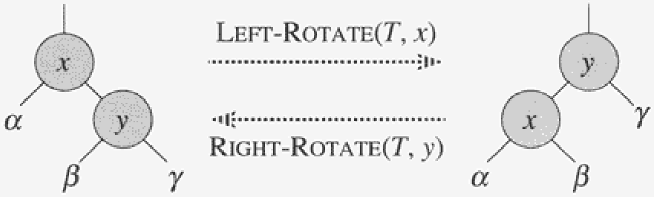


**Figure 13.2** The rotation operations on a binary search tree. The operation LEFT-ROTATE($T$, $x$) transforms the configuration of the two nodes on the left into the configuration on the right by changing a constant number of pointers. The configuration on the right can be transformed into the configuration on the left by the inverse operation RIGHT-ROTATE($T$, $y$). The letters $\alpha$, $\beta$, and $\gamma$ represent arbitrary subtrees. A rotation operation preserves the binary-search-tree property: the keys in $\alpha$ precede *key*[$x$], which precedes the keys in $\beta$, which precede *key*[$y$], which precedes the keys in $\gamma$.

# Left-Rotate

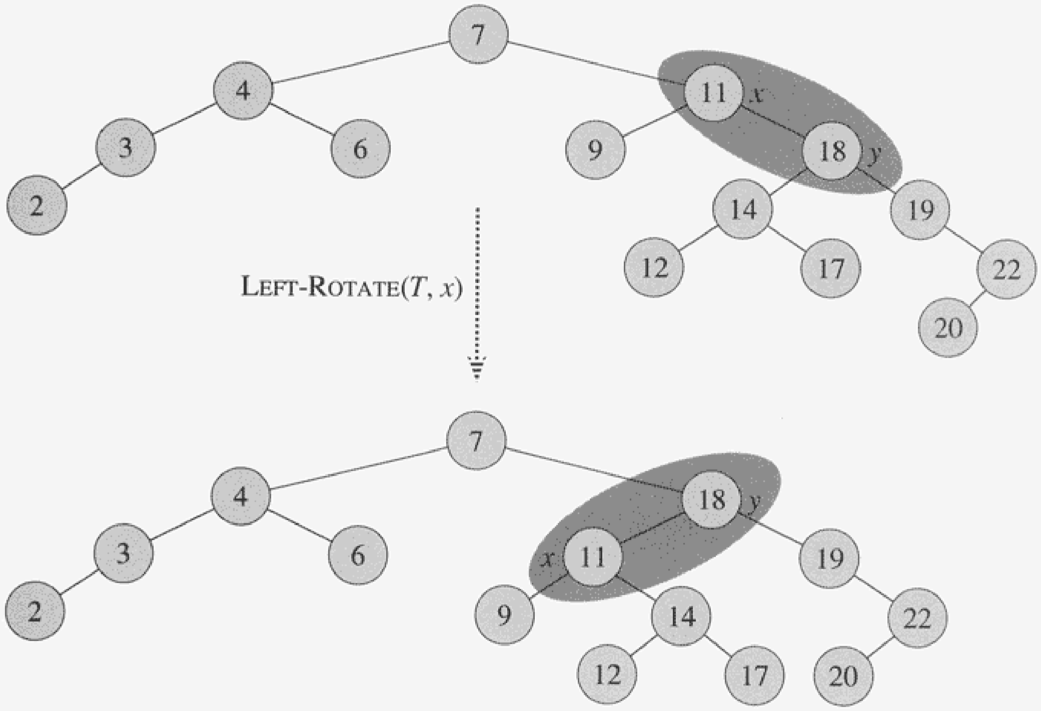
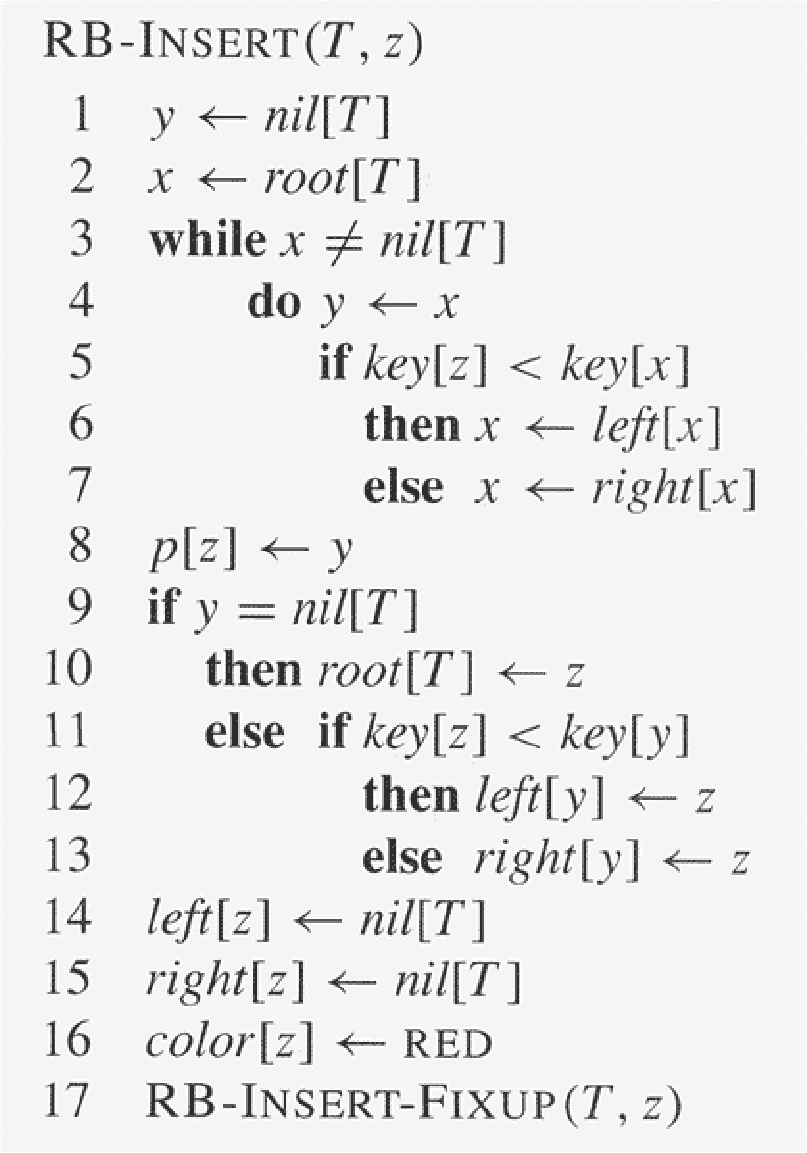


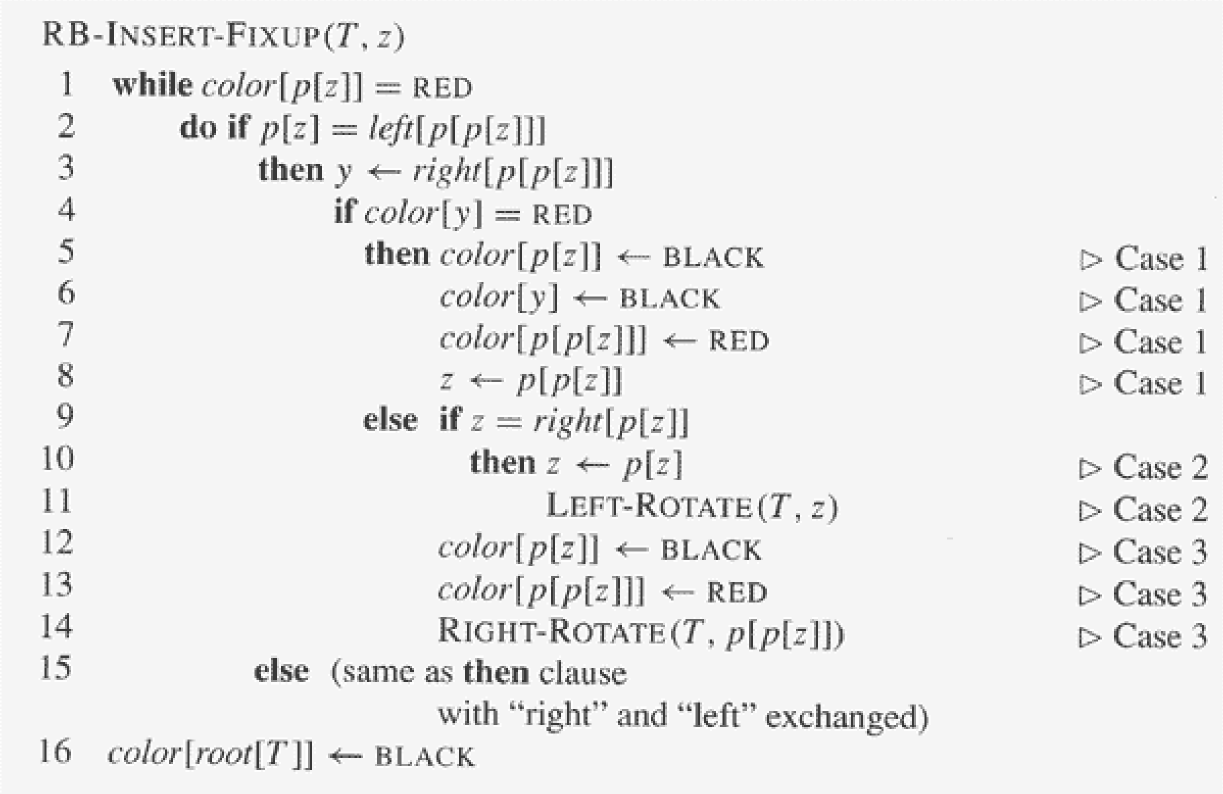
**Figure 13.3** An example of how the procedure LEFT-ROTATE($T$, $x$) modifies a binary search tree. Inorder tree walks of the input tree and the modified tree produce the same listing of key values.

# Insertion



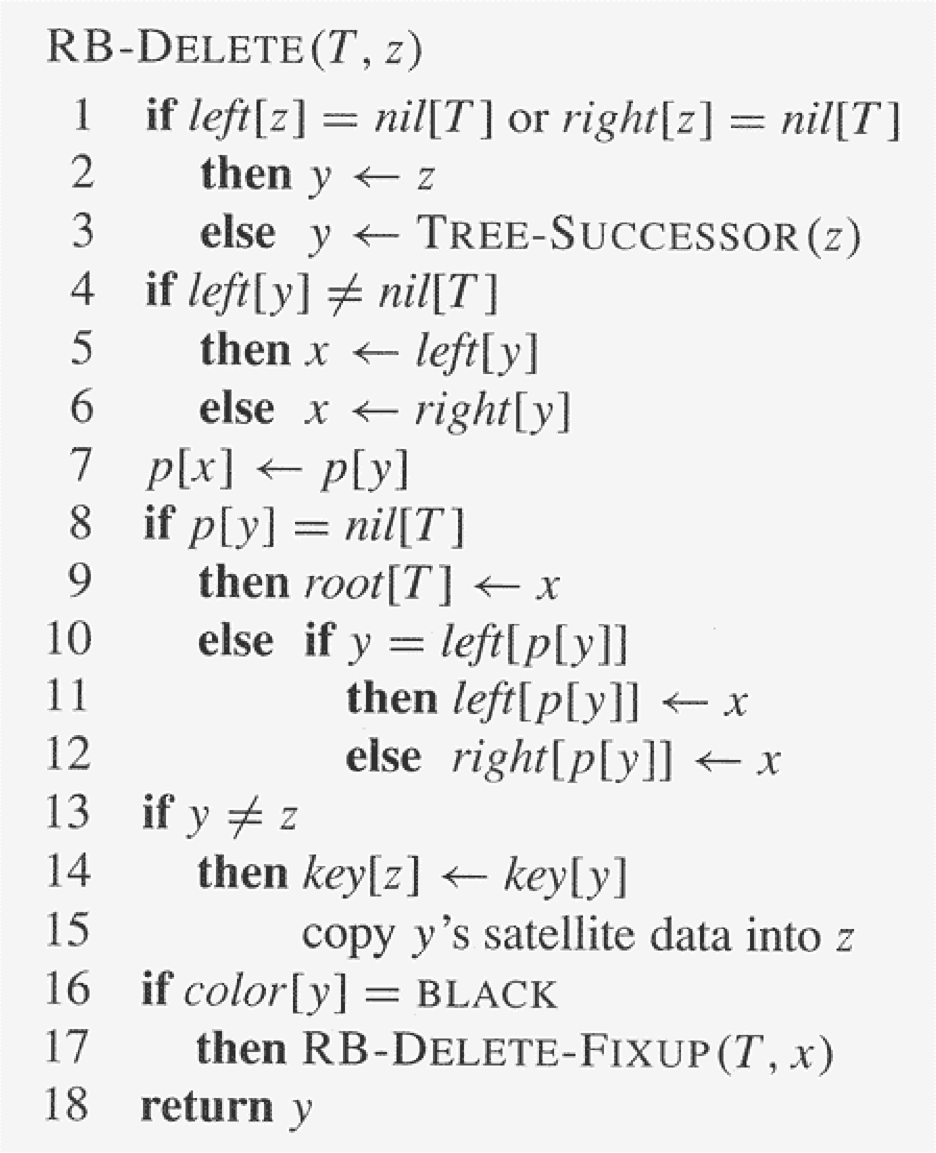
```
RB-INSERT($T, z$)
$y \leftarrow nil[T]$
$x \leftarrow root[T]$
while $x \neq nil[T]$
    do $y \leftarrow x$
       if $key[z] < key[x]$
          then $x \leftarrow left[x]$
          else $x \leftarrow right[x]$
$p[z] \leftarrow y$
if $y = nil[T]$
   then $root[T] \leftarrow z$
   else if $key[z] < key[y]$
           then $left[y] \leftarrow z$
           else $right[y] \leftarrow z$
$left[z] \leftarrow nil[T]$
$right[z] \leftarrow nil[T]$
$color[z] \leftarrow$ RED
RB-INSERT-FIXUP($T, z$)
```

# Red-Black Insert Fixup



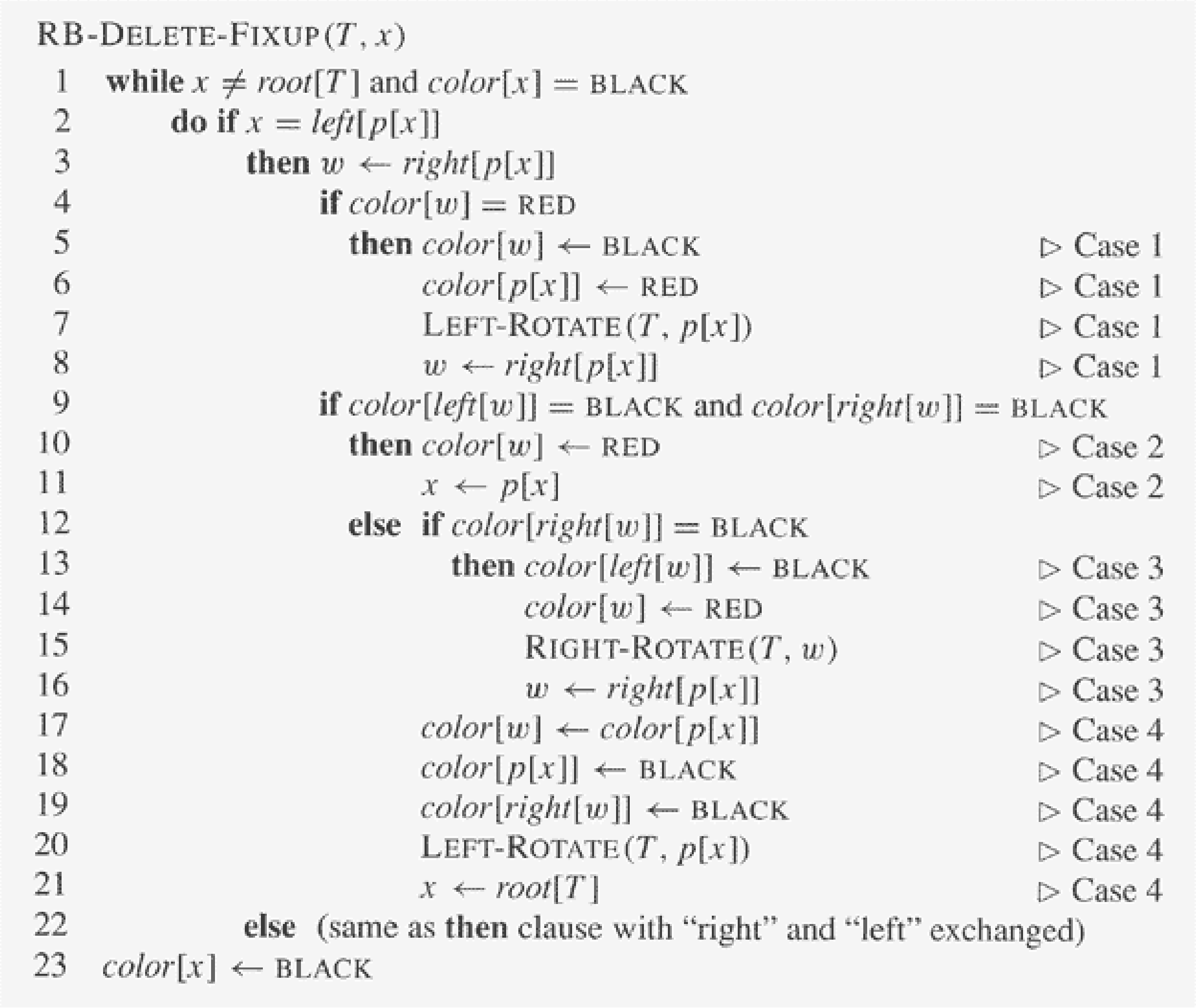
```
RB-INSERT-FIXUP($T, z$)
while $color[p[z]] = $ RED
     do if $p[z] = left[p[p[z]]]$
           then $y \leftarrow right[p[p[z]]]$
                if $color[y] = $ RED
                  then $color[p[z]] \leftarrow$ BLACK                    ▷ Case 1
                       $color[y] \leftarrow$ BLACK                       ▷ Case 1
                       $color[p[p[z]]] \leftarrow$ RED                   ▷ Case 1
                       $z \leftarrow p[p[z]]$                            ▷ Case 1
                  else if $z = right[p[z]]$
                          then $z \leftarrow p[z]$                       ▷ Case 2
                               LEFT-ROTATE($T, z$)                       ▷ Case 2
                       $color[p[z]] \leftarrow$ BLACK                    ▷ Case 3
                       $color[p[p[z]]] \leftarrow$ RED                   ▷ Case 3
                       RIGHT-ROTATE($T, p[p[z]]$)                        ▷ Case 3
           else (same as then clause
                       with “right” and “left” exchanged)
$color[root[T]] \leftarrow$ BLACK
```

# Delete



```
RB-DELETE$(T, z)$
if $left[z] = nil[T]$ or $right[z] = nil[T]$
    then $y \leftarrow z$
    else $y \leftarrow$ TREE-SUCCESSOR$(z)$
if $left[y] \neq nil[T]$
    then $x \leftarrow left[y]$
    else $x \leftarrow right[y]$
$p[x] \leftarrow p[y]$
if $p[y] = nil[T]$
    then $root[T] \leftarrow x$
    else if $y = left[p[y]]$
            then $left[p[y]] \leftarrow x$
            else $right[p[y]] \leftarrow x$
if $y \neq z$
    then $key[z] \leftarrow key[y]$
         copy $y$'s satellite data into $z$
if $color[y] =$ BLACK
    then RB-DELETE-FIXUP$(T, x)$
return $y$
```

# Delete Fixup



```
RB-DELETE-FIXUP(T, x)
while $x \neq root[T]$ and $color[x] =$ BLACK
    do if $x = left[p[x]]$
          then $w \leftarrow right[p[x]]$
               if $color[w] =$ RED
                 then $color[w] \leftarrow$ BLACK                          ▷ Case 1
                      $color[p[x]] \leftarrow$ RED                         ▷ Case 1
                      LEFT-ROTATE($T$, $p[x]$)                             ▷ Case 1
                      $w \leftarrow right[p[x]]$                           ▷ Case 1
               if $color[left[w]] =$ BLACK and $color[right[w]] =$ BLACK
                 then $color[w] \leftarrow$ RED                            ▷ Case 2
                      $x \leftarrow p[x]$                                  ▷ Case 2
                 else if $color[right[w]] =$ BLACK
                         then $color[left[w]] \leftarrow$ BLACK            ▷ Case 3
                              $color[w] \leftarrow$ RED                    ▷ Case 3
                              RIGHT-ROTATE($T$, $w$)                       ▷ Case 3
                              $w \leftarrow right[p[x]]$                   ▷ Case 3
                      $color[w] \leftarrow color[p[x]]$                    ▷ Case 4
                      $color[p[x]] \leftarrow$ BLACK                       ▷ Case 4
                      $color[right[w]] \leftarrow$ BLACK                   ▷ Case 4
                      LEFT-ROTATE($T$, $p[x]$)                             ▷ Case 4
                      $x \leftarrow root[T]$                               ▷ Case 4
          else (same as then clause with "right" and "left" exchanged)
$color[x] \leftarrow$ BLACK
```