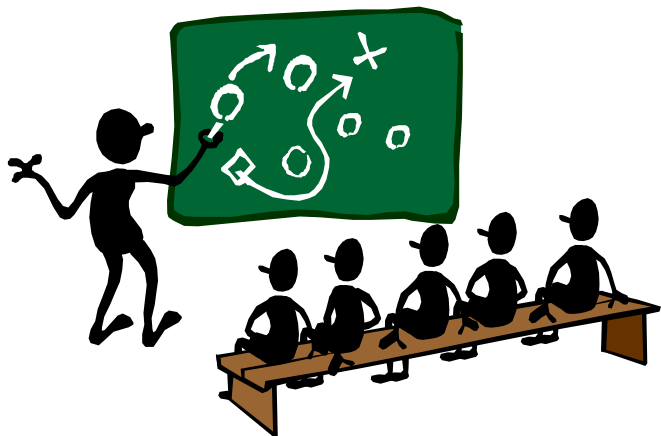# Algorithms – Chapter 13
Red-Black Trees

Juinn-Dar Huang

Professor

jdhuang@mail.nctu.edu.tw

October 2007

Rev. '08, '11, '12, '15, '16, '18, '19, '20, '21

Red-Black Trees

- A red-black tree
  - is a binary tree with colored nodes
  - no such path is more than twice as long as any other from the root to a leaf
 - is approximately balanced

Properties of Red-Black Trees

- Property

    1) every node is either red or black

    2) the root is black

    3) every leaf (NIL) is black

    4) if a node is red, its 2 children are black (if any)

    5) for each node, all paths from the node to descendant leaves contain the same number of black nodes
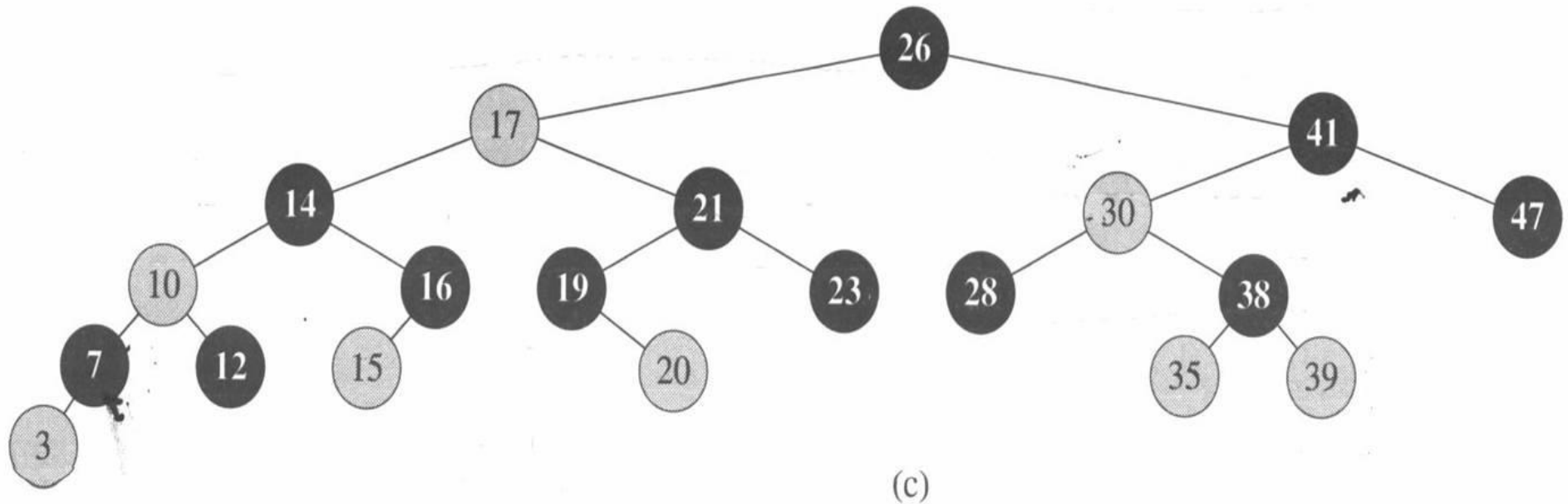
Juinn-Dar Huang   jdhuang@mail.nctu.edu.tw

Red-Black Trees

Example (1/3)

black

red

(b)

(c)
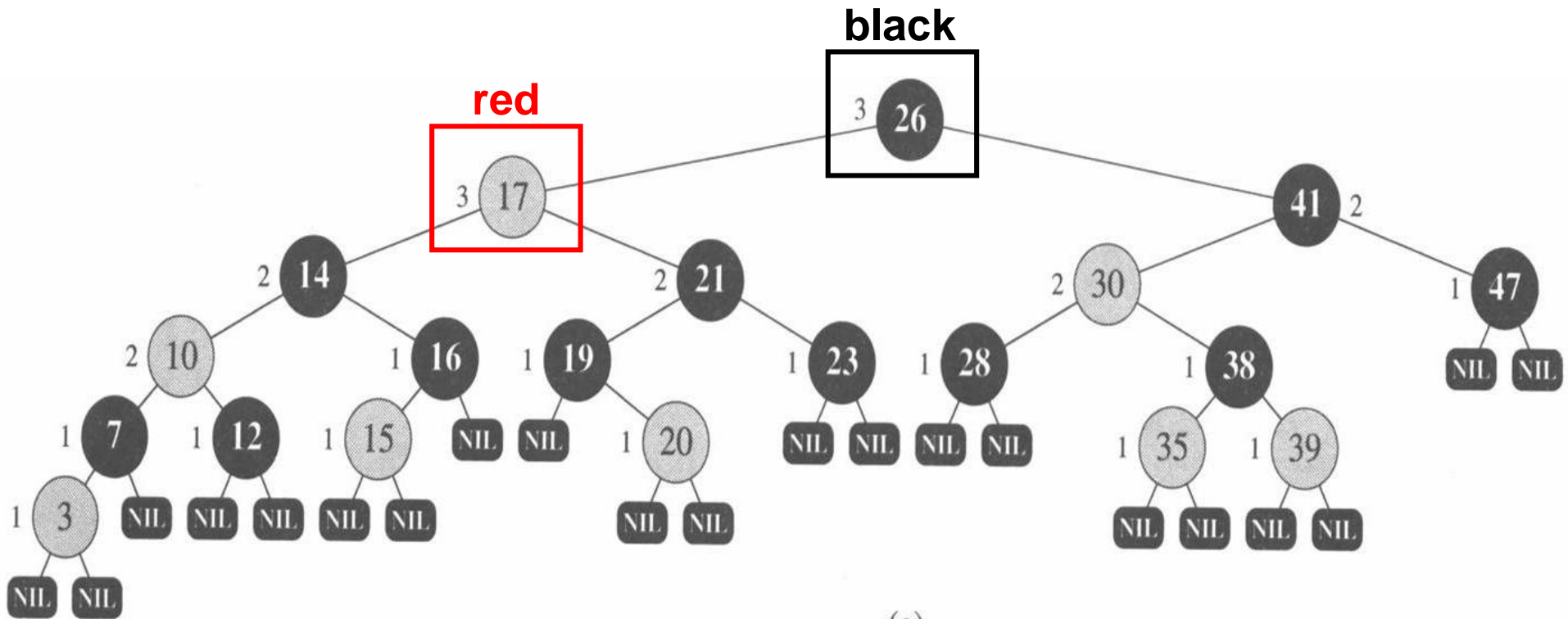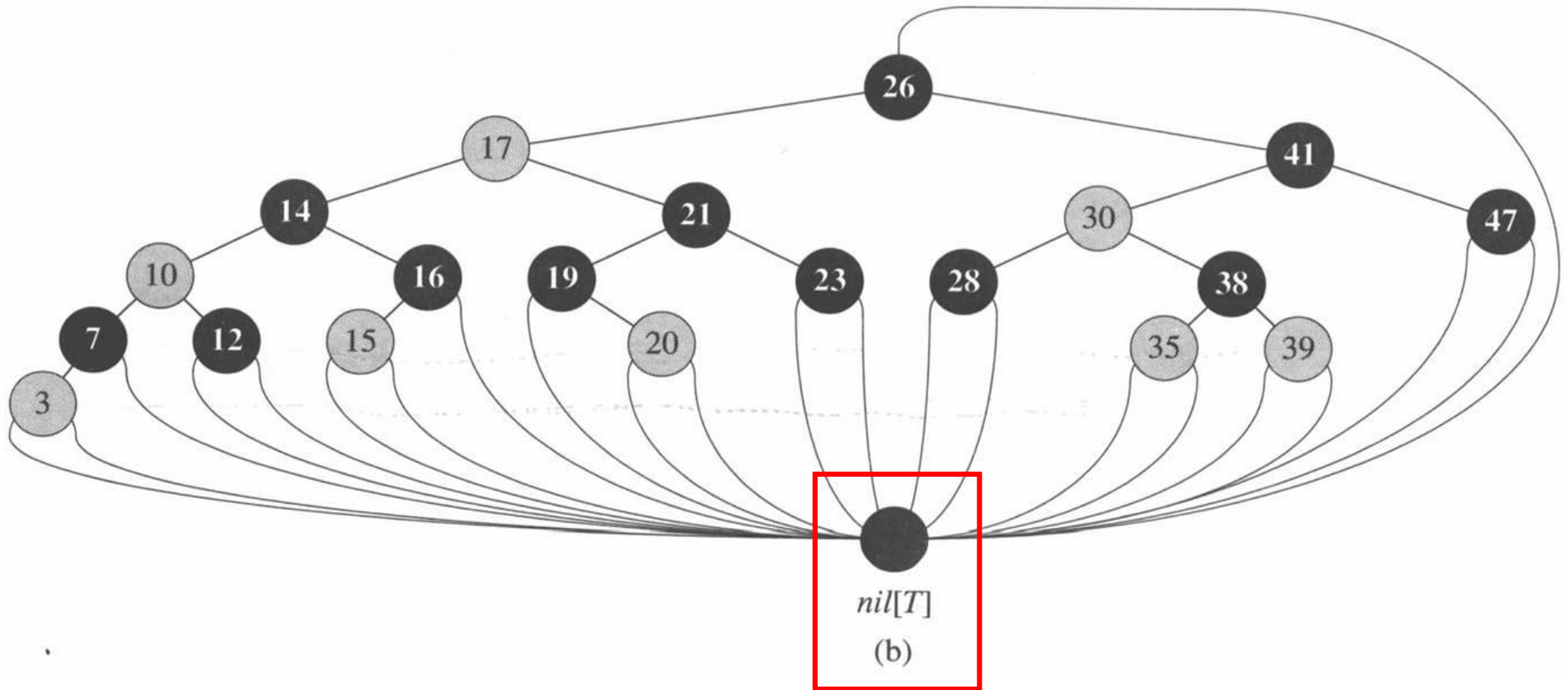
Use the simplified representation later

Black-Height

- The number of black nodes on any path from, but not including, a node x down to a leaf is defined as the black-height of the node x
  - denoted as bh(x)

Tree Height (1/2)

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

- Proof

  prove $2^{bh(root)} - 1 \leq$ the # of nodes n by induction

  if bh(root) = 0, empty tree ➜ $2^0 - 1 \leq 0$

  assume $2^k - 1 \leq n$ holds for bh(root) = k

  for bh(root) = k+1: bh of 2 children ➜ k+1 or k

  ➜ at least $2^k - 1$ nodes for each subtree

  ➜ at least $2^{k+1} - 1$ nodes in the tree

Juinn-Dar Huang   jdhuang@mail.nctu.edu.tw

Red-Black Trees

Tree Height (2/2)

Let h be the height of the tree

➔ at least half the nodes on any simple path from the root to a leaf, not including the root, must be black

➔ $bh(root) \geq h/2$

➔ $n \geq 2^{bh(root)} - 1 \geq 2^{h/2} - 1$

➔ $h \leq 2\lg(n+1)$

- That is, $h = O(\lg n)$
- How to do insertion and deletion?

Fixes to Preserve the Properties

- Use TREE-INSERT & TREE-DELETE (described in Chap 12) in a red-black tree
  - time complexity: O(lgn)
  - after operations, the tree may violate the properties of red-black tree ➔ some fixes are required
- Fixes
  - do rotations
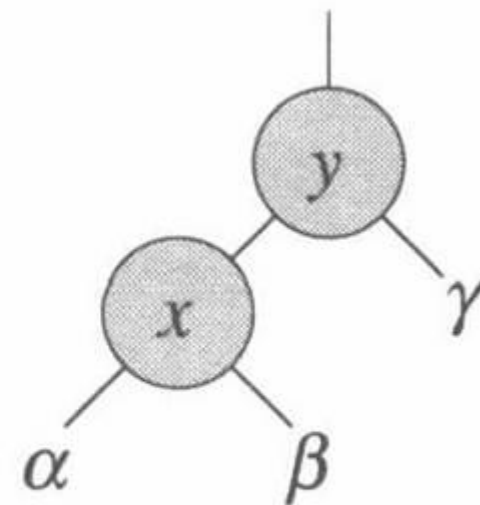 - change the colors of some nodes

Juinn-Dar Huang   jdhuang@mail.nctu.edu.tw

Red-Black Trees
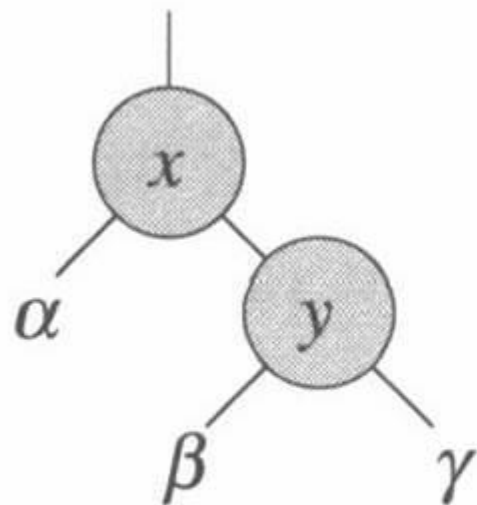
Rotations

LEFT-ROTATE(T, x)

RIGHT-ROTATE(T, y)

Juinn-Dar Huang   jdhuang@mail.nctu.edu.tw

Red-Black Trees

Left Rotations

LEFT-ROTATE(*T*, *x*)

1   *y* ← *right*[*x*] // assume *y* is not *nil*

2   *right*[*x*] ← *left*[*y*]

3   **if** *left*[*y*] ≠ *nil*[*T*]

4   **then** *p*[*left*[*y*]] ← *x*

5   *p*[*y*] ← *p*[*x*]

6   **if** *p*[*x*] = *nil*[*T*]

7   **then** *root*[*T*] ← *y*

8   **else if** *x* = *left*[*p*[*x*]]

9   **then** *left*[*p*[*x*]] ← *y*

10   **else** *right*[*p*[*x*]] ← *y*

11 *left*[*y*] ← *x*

12 *p*[*x*] ← *y*

Time Complexity: O(1)

Example

$\text{LEFT-ROTATE}(T, x)$

RB-INSERT(T, z)

    1 $y \leftarrow nil[T]$

    2 $x \leftarrow root[T]$

    3 **while** $x \neq nil[T]$

    4   **do** $y \leftarrow x$

    5   **if** $key[z] < key[x]$

    6   **then** $x \leftarrow left[x]$

    7   **else** $x \leftarrow right[x]$

    8 $p[z] \leftarrow y$

9 **if** $y = nil[T]$

10   **then** $root[T] \leftarrow z$

11   **else if** $key[z] < key[y]$

12   **then** $left[y] \leftarrow z$

13   **else** $right[y] \leftarrow z$

14 $left[z] \leftarrow nil[T]$

15 $right[z] \leftarrow nil[T]$

16 $color[z] \leftarrow \text{RED}$

17 **RB-INSERT-FIXUP**(T, z)

**Compare with TREE-INSERT (Chap 12, p. 11)**

RB-INSERT-FIXUP(T, z)

1   **while** $color[p[z]] = \text{RED}$

2   **do if** $p[z] = left[p[p[z]]]$

3   **then** $y \leftarrow right[p[p[z]]]$

4   **if** $color[y] = \text{RED}$

5   **then** $color[p[z]] \leftarrow \text{BLACK}$ **Case 1**

6                    $color[y] \leftarrow \text{BLACK}$ **Case 1**

7                    $color[p[p[z]]] \leftarrow \text{RED}$ **Case 1**

8                    $z \leftarrow p[p[z]]$ **Case 1**

9   **else if** $z = right[p[z]]$

10   **then** $z \leftarrow p[z]$ **Case 2**

11   LEFT-ROTATE(T, z) **Case 2**

12               $color[p[z]] \leftarrow$ BLACK   **Case 3**

13              $color[p[p[z]]] \leftarrow$ RED   **Case 3**

14   RIGHT-ROTATE$(T, p[p[z]])$ **Case 3**

15   **else** (same as **then** (**Line 3**) clause

                      with "right" and "left" exchanged)

16   $color[root[T]] \leftarrow$ BLACK
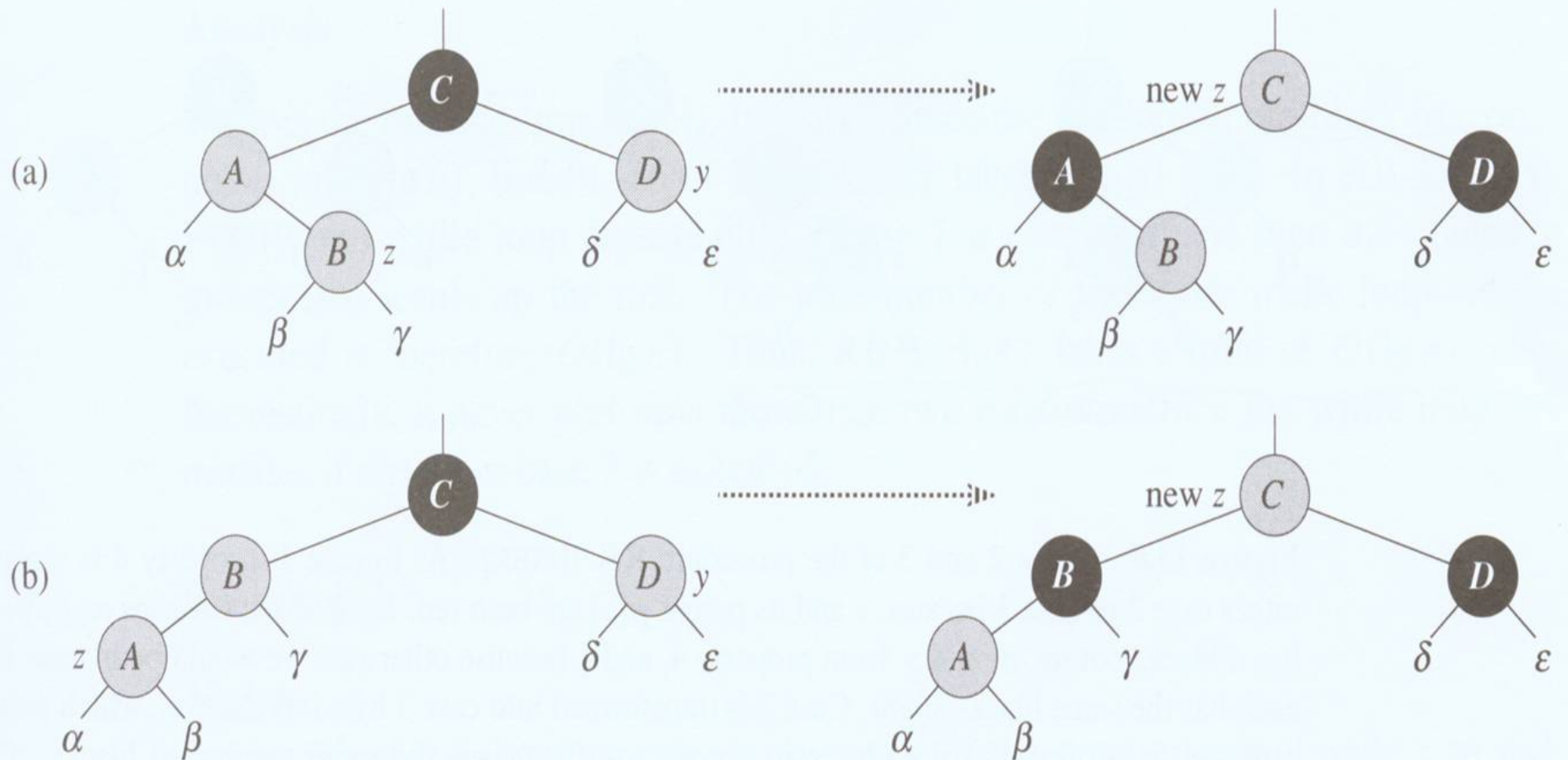
Case 1: z's uncle y is red

Case 2
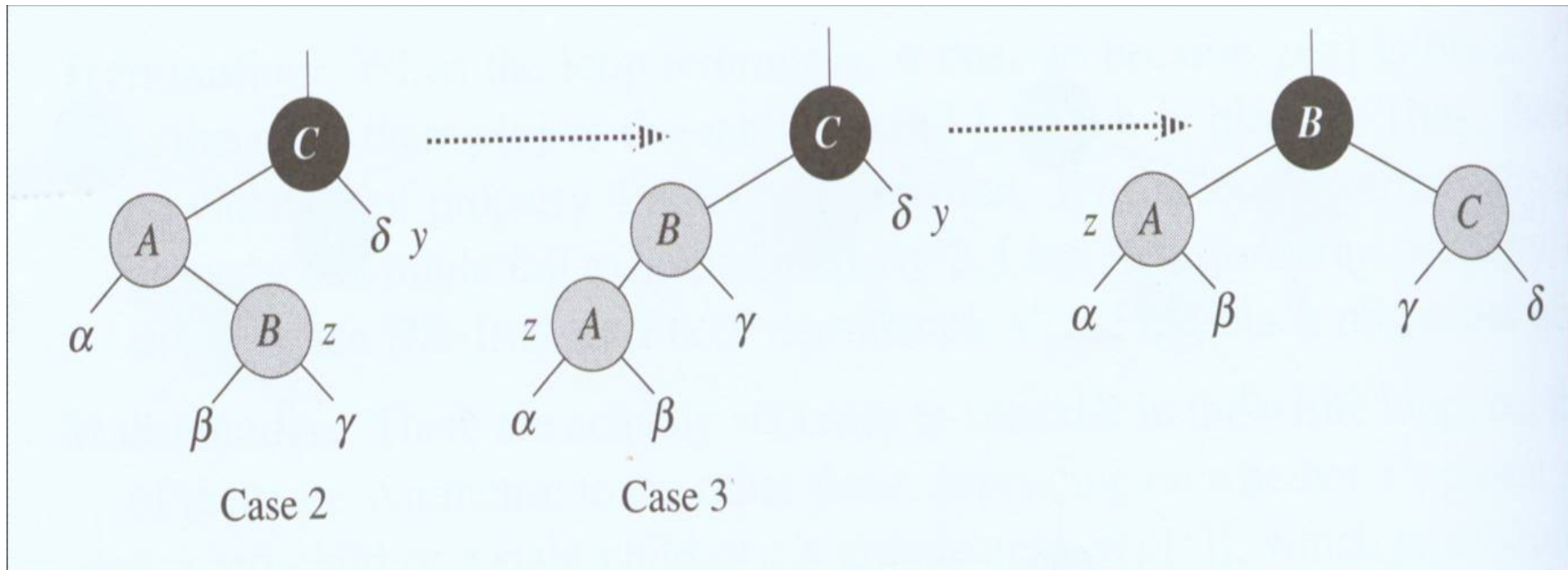
Case 3

Case 2: z's uncle y is black and z is a right child

Case 3: z's uncle y is black and z is a left child
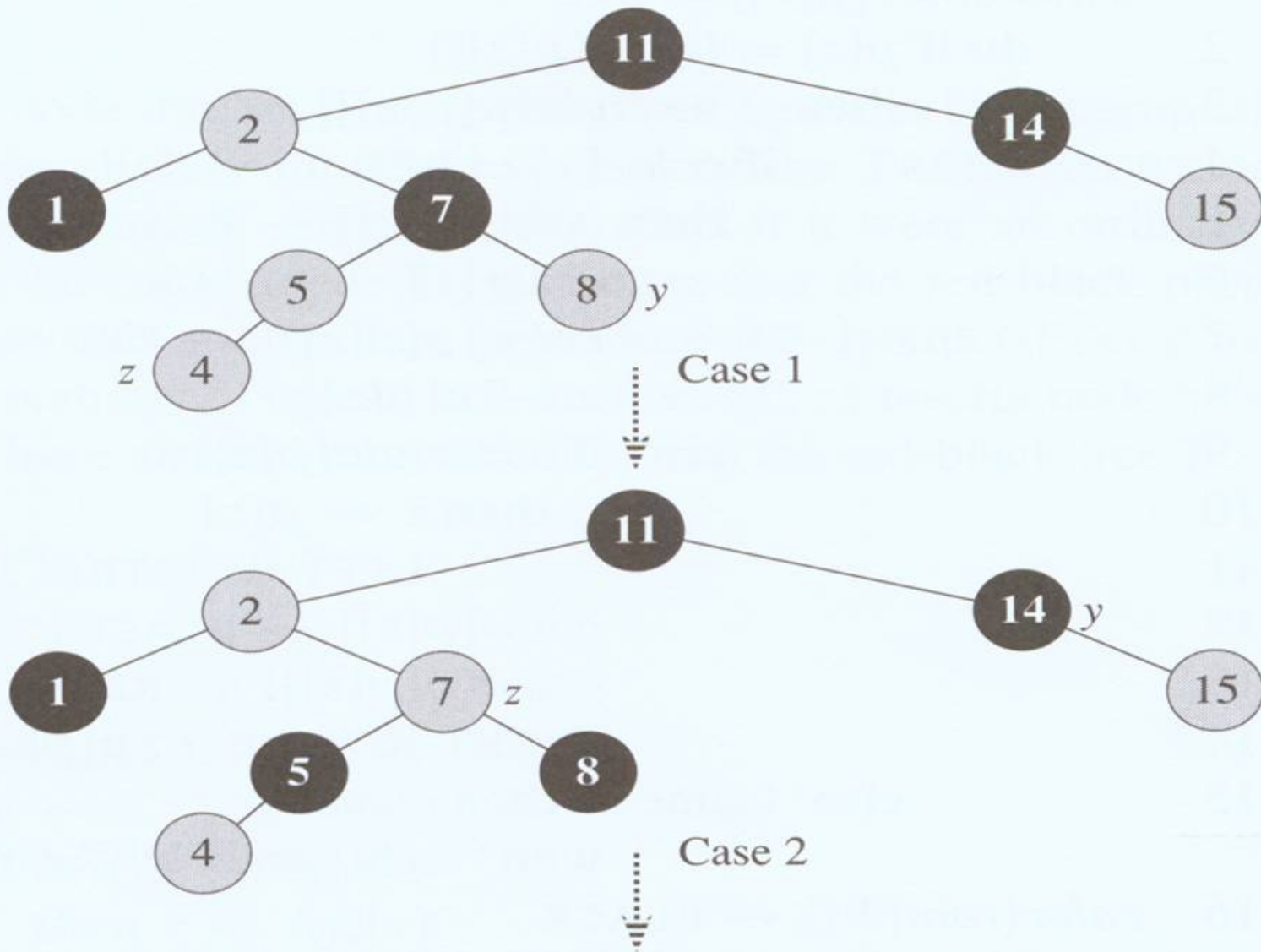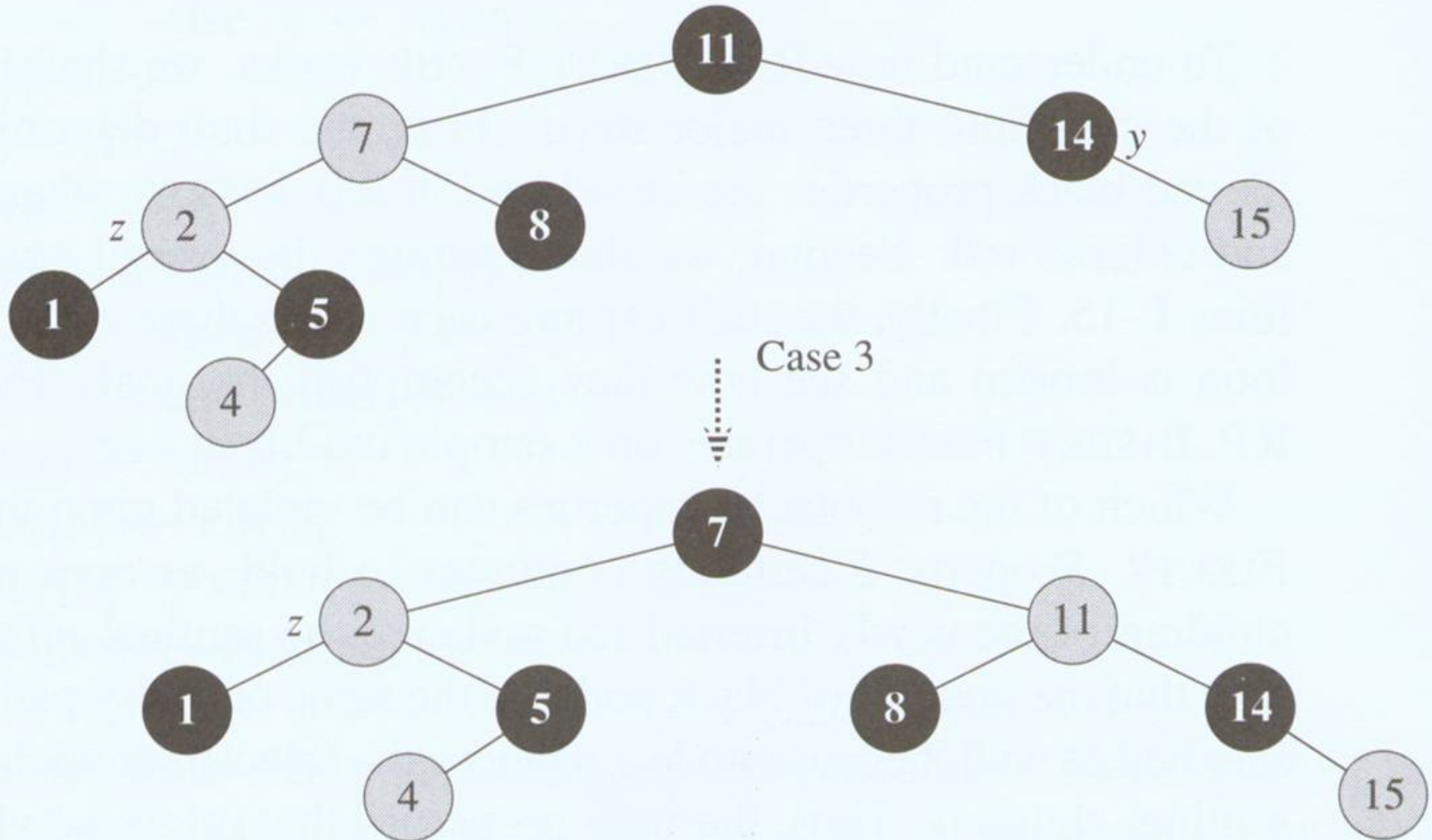
Case 3

Time Complexity

- RB-INSERT-FIXUP takes $O(\lg n)$

- RB-INSERT-FIXUP never performs more than two rotations, since the while loop terminates if Case 2 or Case 3 is executed

- The overall time for RB-INSERT is $O(\lg n)$

Deletions in Red-Black Tree (1/2)

RB-DELETE(T, z)

   1 **if** $left[z] = nil[T]$ or $right[z] = nil[T]$

   2   **then** $y \leftarrow z$

   3   **else** $y \leftarrow$ TREE-SUCCESSOR(z)

   4 **if** $left[y] \neq nil[T]$

   5   **then** $x \leftarrow left[y]$

   6   **else** $x \leftarrow right[y]$

   7 $p[x] \leftarrow p[y]$

8   **if** $p[y] = nil[T]$

9   **then** $root[T] \leftarrow x$

10   **else if** $y = left[p[y]]$

11   **then** $left[p[y]] \leftarrow x$

12   **else** $right[p[y]] \leftarrow x$

13 **if** $y \neq z$

14   **then** $key[z] \leftarrow key[y]$

15   copy y's satellite data into z

16 **if** $color[y] = \text{BLACK}$

17   **then** RB-DELETE-FIXUP(T, x)

18 **return** y

Compare with TREE-DELETE (Chap 12, p. 13)

Juinn-Dar Huang   jdhuang@mail.nctu.edu.tw

Red-Black Trees

Color of Deleted Node (1/2)

- The deleted node y is red
  - no black-height in the tree have changed
  - no red nodes have been made adjacent
  - the deleted node cannot be the root
    ➔ the root remains black

  - so, no fix-up is required!

Color of Deleted Node (2/2)

- The deleted node y is black
  - if the deleted node is the root and its red child becomes the new root ➔ violate Property 2
  - if both x and p[y] are red ➔ violate Property 4
  - the black-height decreases 1 for paths passing through the deleted node y ➔ violate Property 5

  - so, fix-ups are definitely required!

- RB-DELETE-FIXUP(T, x)

1   **while** $x \neq root[T]$ and $color[x] = $ BLACK

2   **do if** $x = left[p[x]]$

3   **then** $w \leftarrow right[p[x]]$

4   **if** $color[w] = $ RED

5   **then** $color[w] \leftarrow $ BLACK   **Case1**

6                       $color[p[x]] = $ RED   **Case1**

7   LEFT-ROTATE(T, $p[x]$) **Case1**

8                       $w \leftarrow right[p[x]]$ **Case1**

9	**if** $color[left[w]]$ = BLACK and	
	$color[right[w]]$ = BLACK	
10	**then** $color[w] \leftarrow$ RED	**Case2**
11	$x \leftarrow p[x]$	**Case2**
12	**else if** $color[right[w]]$ = BLACK	
13	**then** $color[left[w]] \leftarrow$ BLACK	**Case3**
14	$color[w] \leftarrow$ RED	**Case3**
15	RIGHT-ROTATE(T, w)	**Case3**
16	$w \leftarrow right[p[x]]$	**Case3**

17	$color[w] \leftarrow color[p[x]]$	**Case4**
18	$color[p[x]] \leftarrow$ BLACK	**Case4**
19	$color[right[w]] \leftarrow$ BLACK	**Case4**
20	LEFT-ROTATE$(T, p[x])$	**Case4**
21	$x \leftarrow root[T]$	**Case4**
22	**else** (same as **then (Line 3)** clause with "right" and "left" exchanged)	

23 $color[x] \leftarrow$ BLACK

x is y's single black child initially; x has one "extra black"
w is x's sibling and must exist

Case 1

w is red

w is black now
➔ **Case 2/3/4**

Case 2

w is black

w's 2 children are both black

In Case 2/3/4, w must have two children

Juinn-Dar Huang   jdhuang@mail.nctu.edu.tw

Red-Black Trees
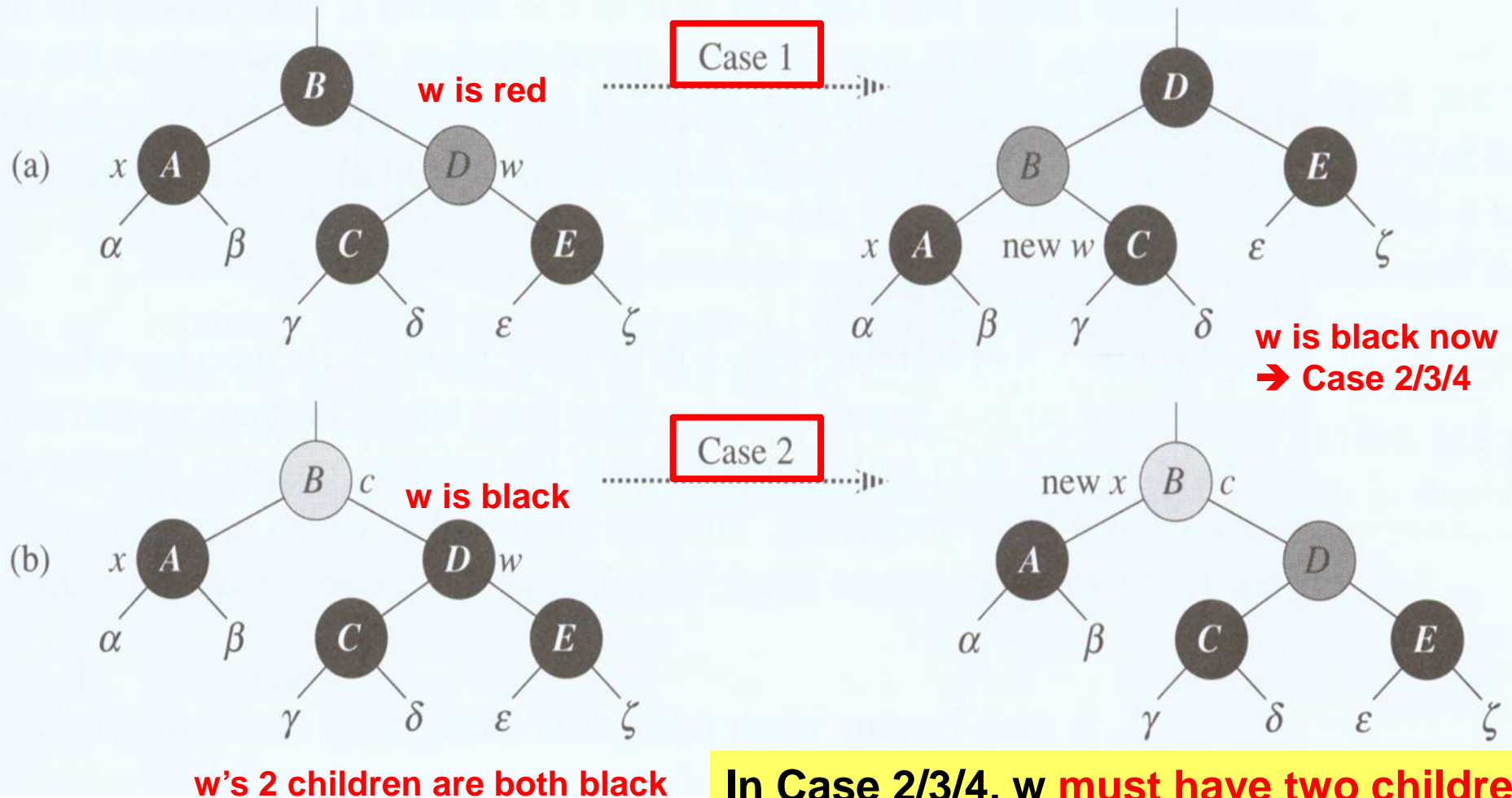
Case 3 & 4 in RB-DELETE-FIXUP

Case 3

w is black

w's left child is red   w's right child is black

w's right child is red

Case 4

w is black

w's right child is red

new x = root[T]
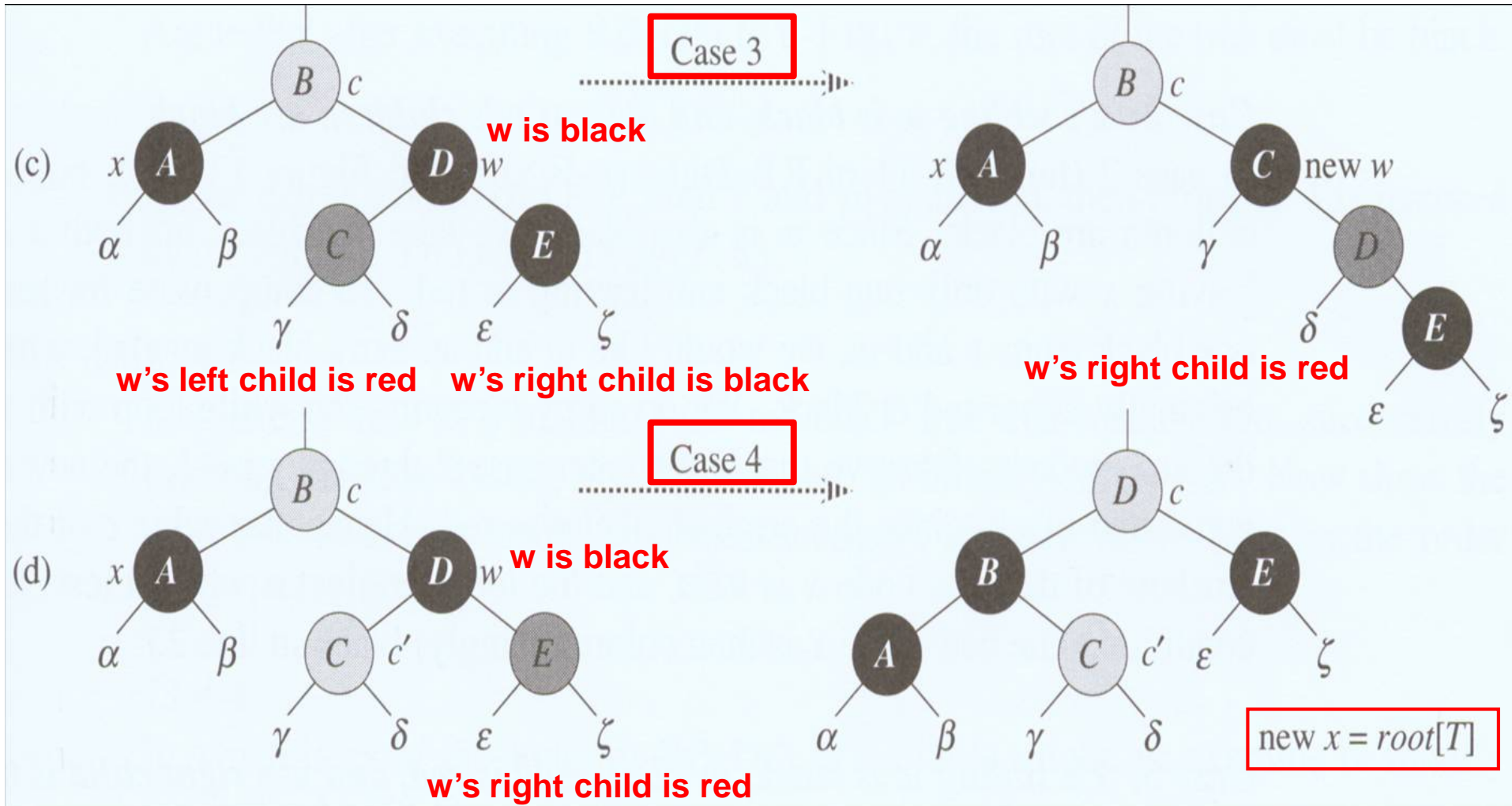
Time Complexity

- RB-DELETE-FIXUP takes O(lgn) time and performs at most three rotations
- The overall time for RB-DELETE is O(lgn)