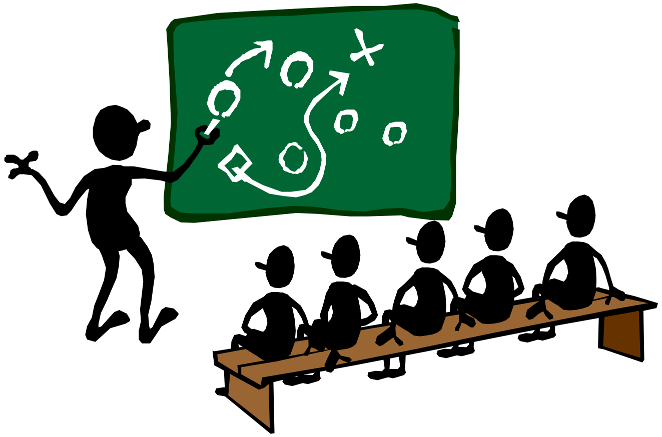
Algorithms – Chapter 13
Red-Black Trees



Juinn-Dar Huang

Professor

jdhuang@mail.nctu.edu.tw

October 2007

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

Red-Black Trees

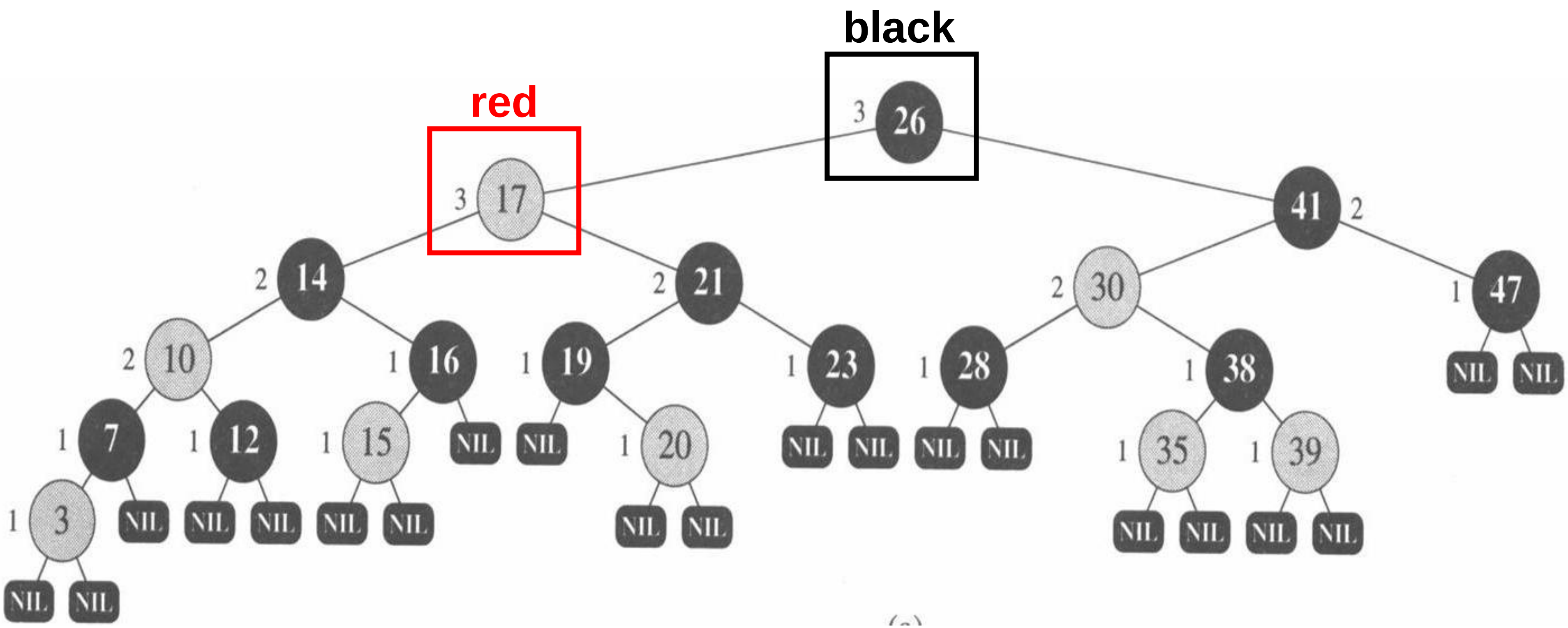
- A red-black tree
  - is a binary tree with colored nodes
  - no such path is more than twice as long as any other from the root to a leaf
  - is approximately balanced

Properties of Red-Black Trees

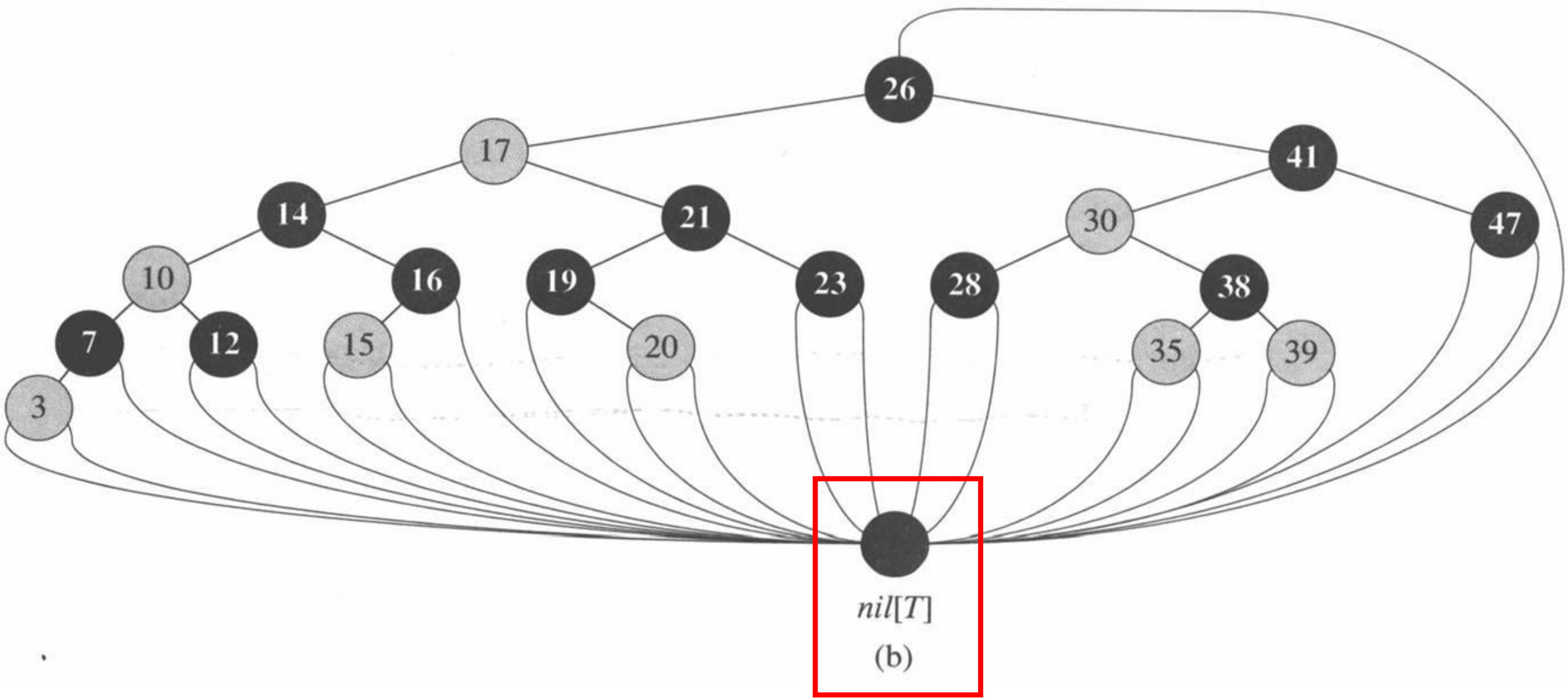
- Property
  1) every node is either red or black
  2) the root is black
  3) every leaf (NIL) is black
  4) if a node is red, its 2 children are black (if any)
  5) for each node, all paths from the node to descendant leaves contain the same number of black nodes

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

Example (1/3)


black
red
3 26
3 17
41 2
2 14
2 21
2 30
1 47
2 10
1 16
1 19
1 23
1 28
1 38
1 7
1 12
1 15
1 20
1 35
1 39
1 3
NIL

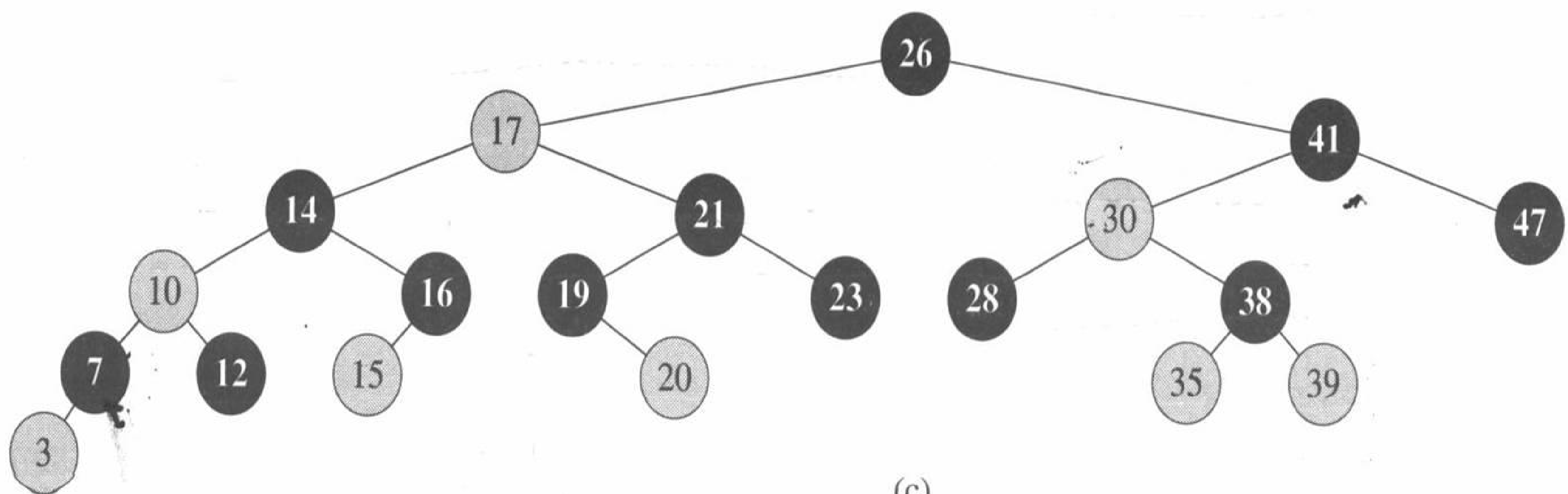
Example (2/3)



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

Red-Black Trees

Example (3/3)



(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 $h \le 2\lg(n+1)$
- Proof

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

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

  assume $2^k - 1 \le n$ 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

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

➔ $bh(root) \geq h/2$

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

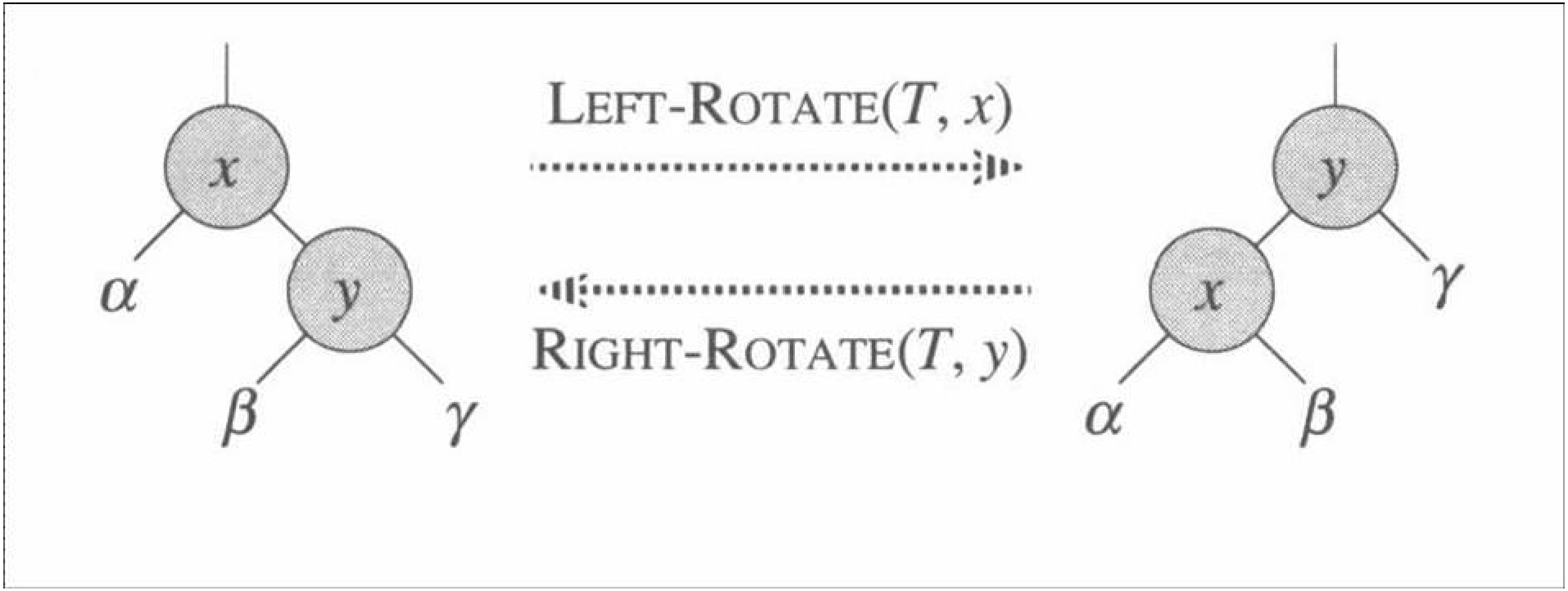
➔ $h \leq 2\lg(n+1)$

- 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

Rotations



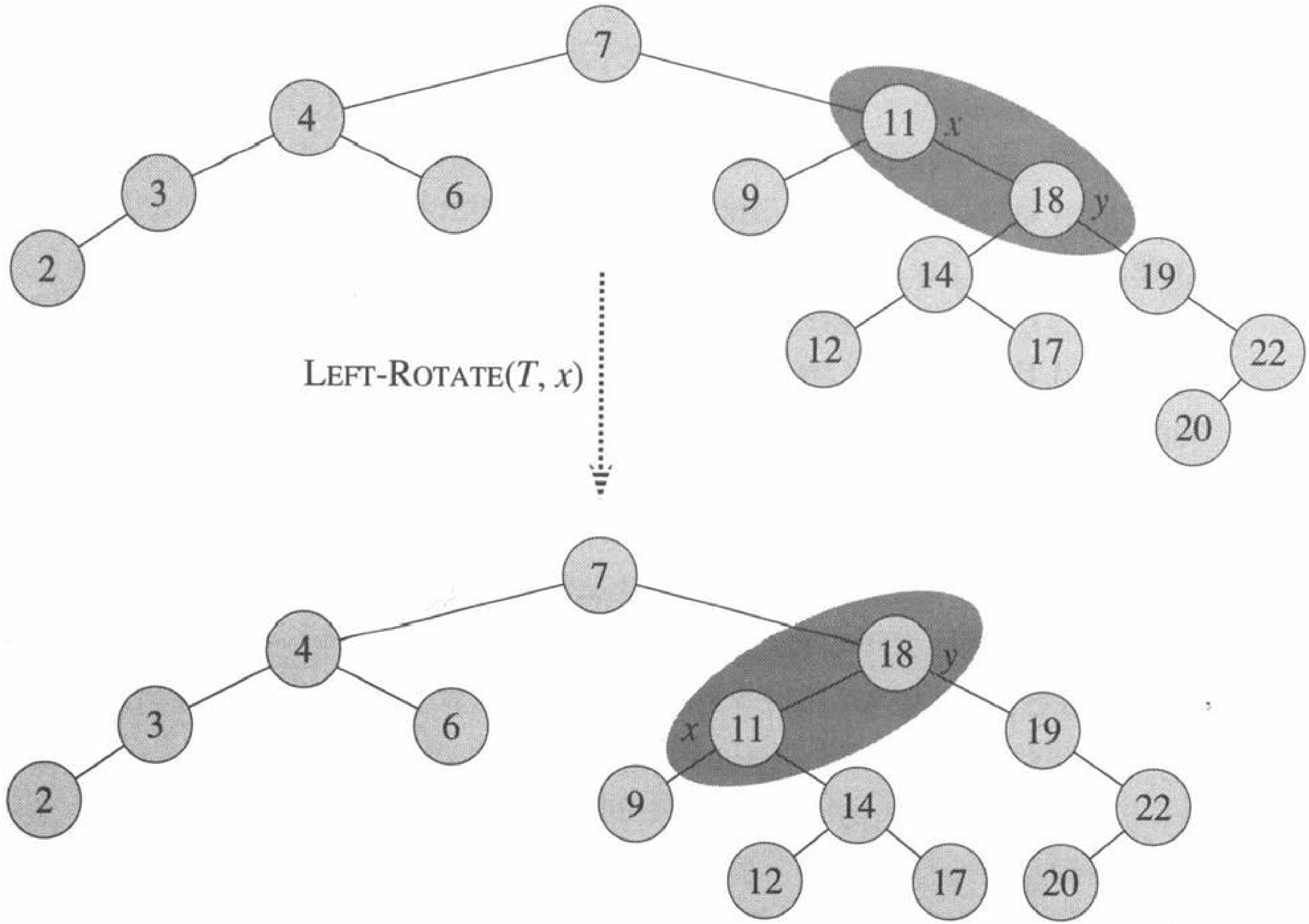
Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Left Rotations

```
LEFT-ROTATE(T, x)
y ← right[x]  // assume y is not nil
right[x] ← left[y]
if left[y] ≠ nil[T]
    then p[left[y]] ← x
p[y] ← p[x]
if p[x] = nil[T]
   then root[T] ← y
   else if x = left[p[x]]
           then left[p[x]] ← y
           else  right[p[x]] ← y
left[y] ← x
p[x] ← y
```

Time Complexity: O(1)

Example


7
4
11
x
3
6
9
18
y
2
14
19
12
17
22
20
LEFT-ROTATE(T, x)
7
4
18
y
3
6
x
11
19
2
9
14
22
12
17
20

Insertions in Red-Black Tree (1/2)

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 $x \leftarrow right[x]$
$p[z] \leftarrow y$

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

Insertions in Red-Black Tree (2/2)

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)

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

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

INSERT-FIXUP (1/2)

RB-INSERT-FIXUP(T, z)

```
while color[p[z]] = RED
   do if p[z] = left[p[p[z]]]
        then y ← right[p[p[z]]]
             if  color[y] = RED
                 then color[p[z]] ←BLACK          Case 1
                      color[y] ← BLACK            Case 1
                      color[p[p[z]]] ← RED        Case 1
                      z ← p[p[z]]                 Case 1
```

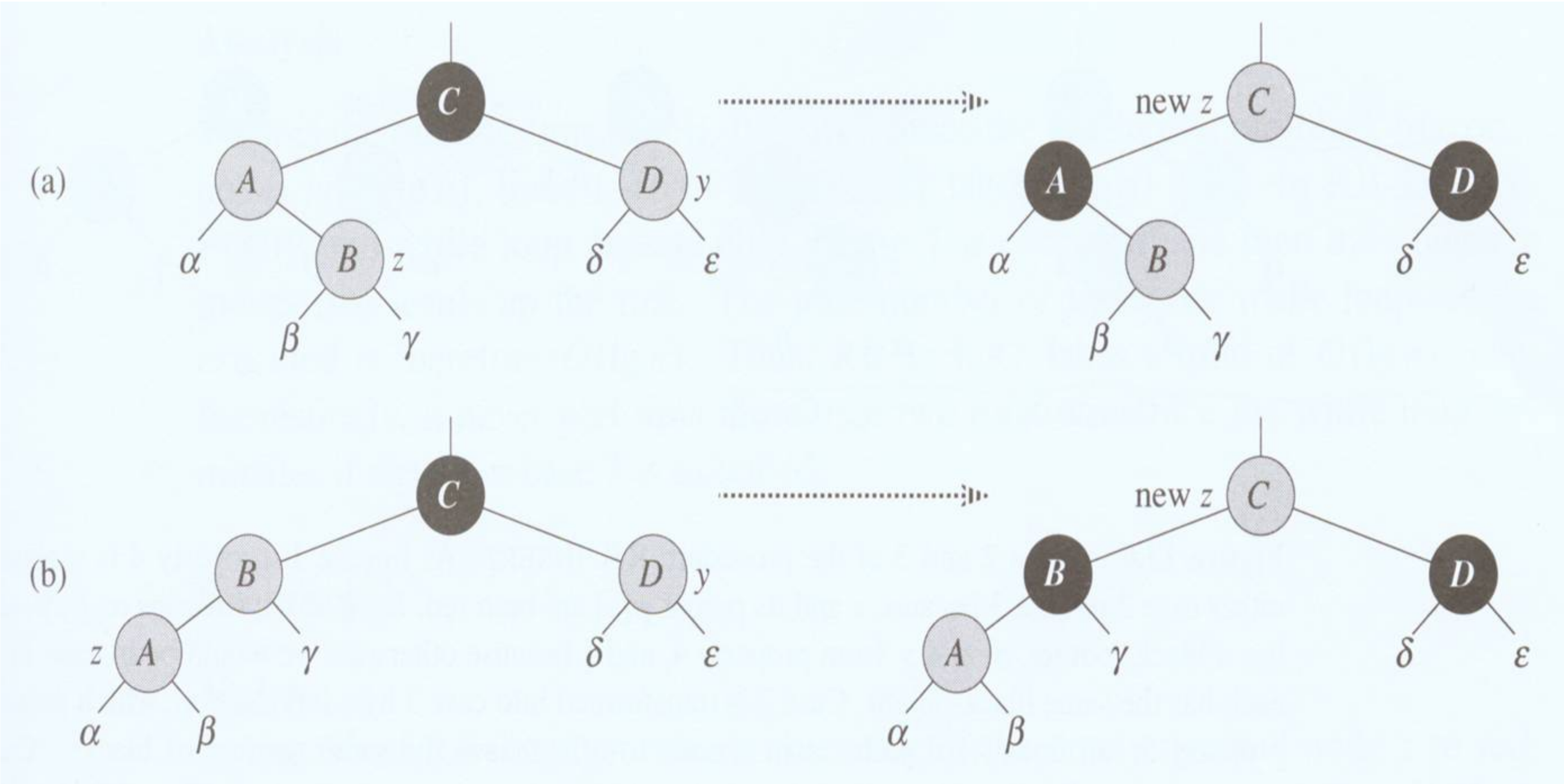
Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

INSERT-FIXUP (2/2)

```
 9          else if z = right[p[z]]
10                  then z ← p[z]                      Case 2
11                       LEFT-ROTATE(T, z)             Case 2
12                  color[p[z]] ← BLACK                Case 3
13                  color[p[p[z]]] ← 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]] ← BLACK
```


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

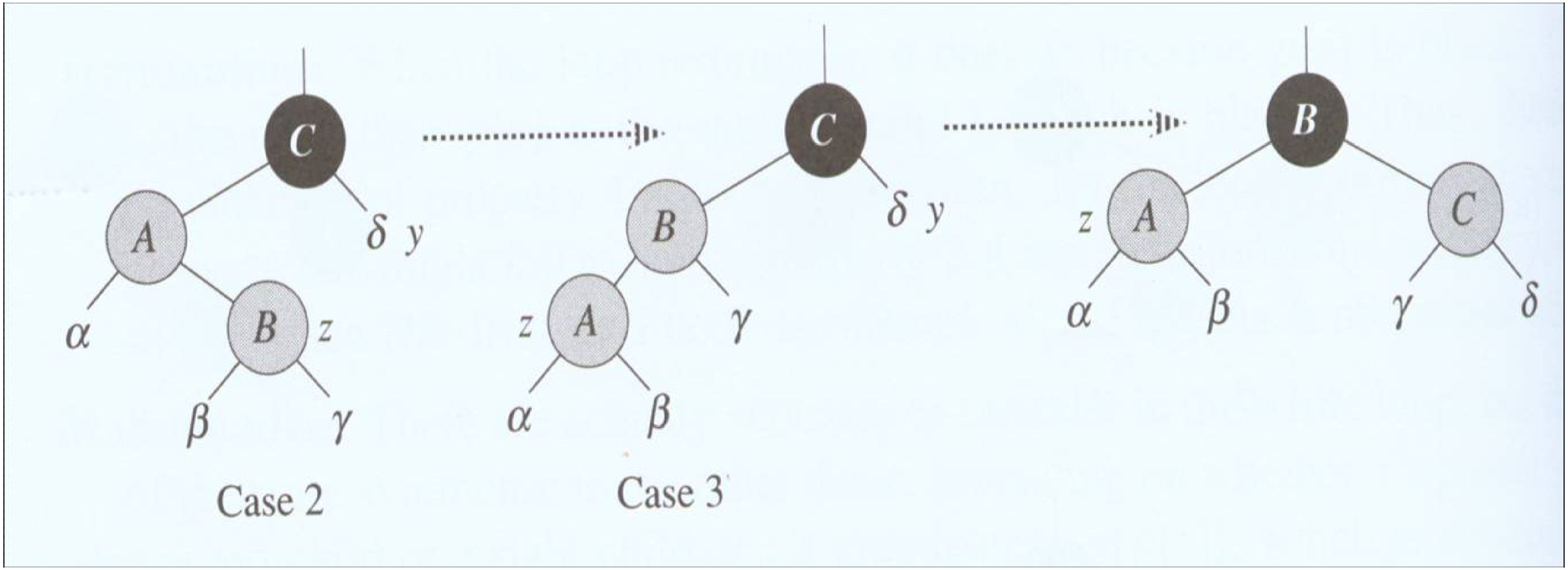
Case 1 in RB-INSERT-FIXUP



Case 1: z's uncle y is red

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

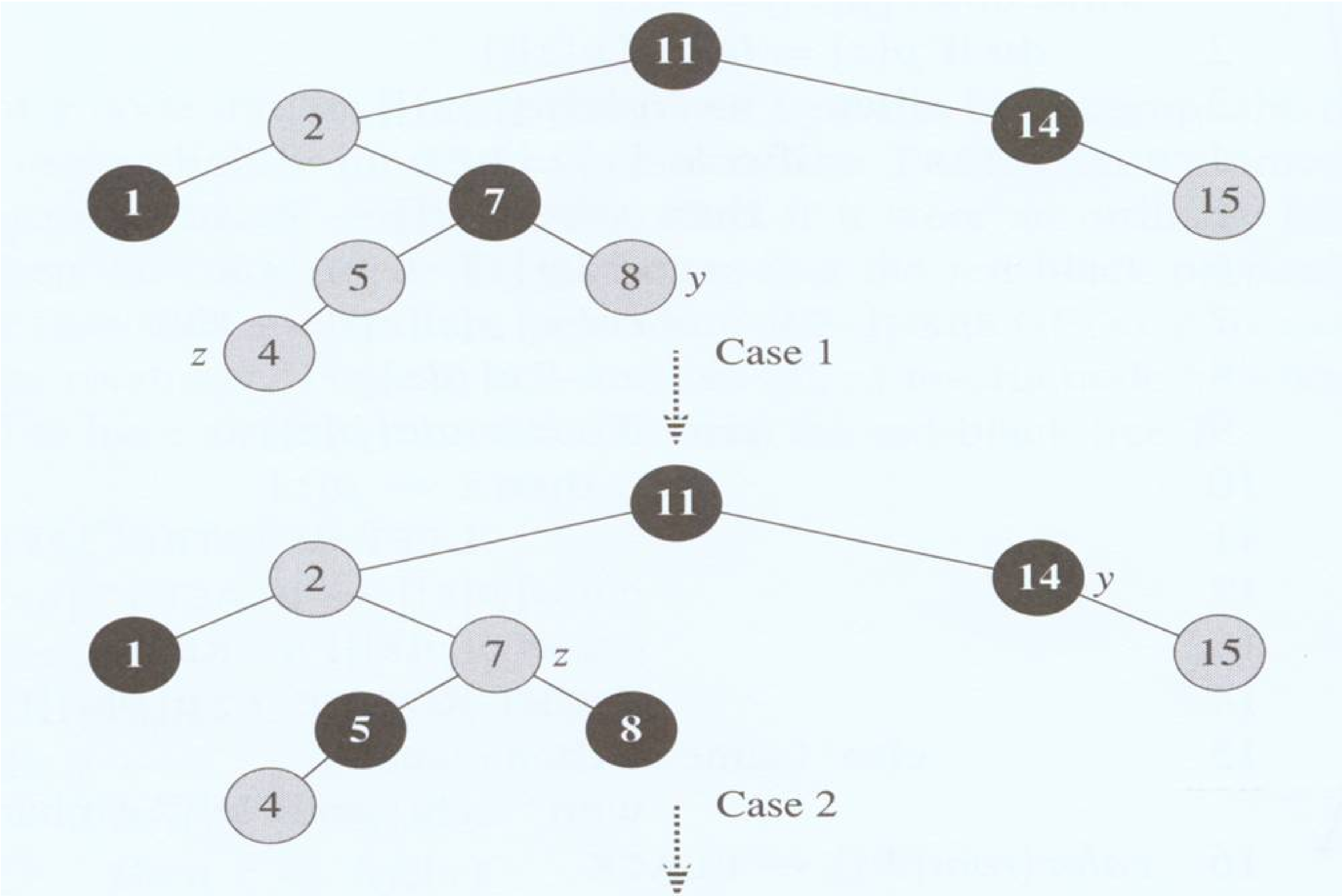
Case 2 & 3 in RB-INSERT-FIXUP



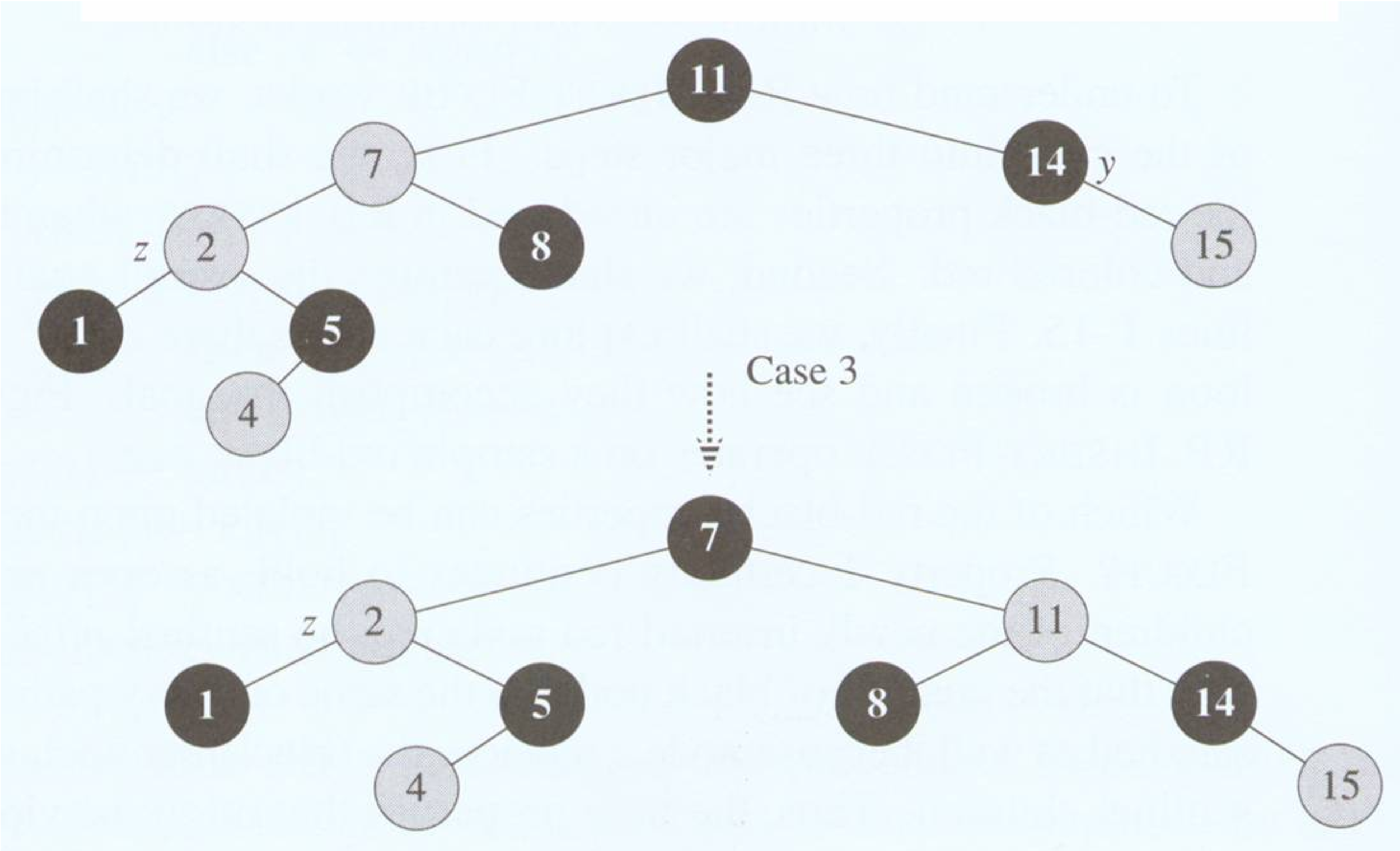
Case 2: z's uncle y is black and z is a right child

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

Fixup Example (1/2)



Fixup Example (2/2)



Time Complexity

- RB-INSERT-FIXUP takes O(lgn)
- 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(lgn)

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

Deletions in Red-Black Tree (1/2)

RB-DELETE(*T*, *z*)

```
if left[z] = nil[T] or right[z] = nil[T]
   then y ← z
   else y ← TREE-SUCCESSOR(z)
if left[y] ≠ nil[T]
    then x ← left[y]
    else x ← right[y]
p[x] ← p[y]
```

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

Deletions in Red-Black Tree (2/2)

```
if  p[y] = nil[T]
    then root[T] ← x
    else if y = left[p[y]]
            then left[p[y]] ← x
            else  right[p[y]] ← x
if  y ≠ z
    then key[z] ← key[y]
         copy y's satellite data into z
if  color[y] = BLACK
    then RB-DELETE-FIXUP(T, x)
return y
```

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

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!

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

DELETE-FIXUP (1/3)

- 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 **Case1**
                  $color[p[x]]$ = RED **Case1**
                  LEFT-ROTATE(T, $p[x]$) **Case1**
                  $w \leftarrow right[p[x]]$ **Case1**

DELETE-FIXUP (2/3)

```
if color[left[w]] = BLACK and
   color[right[w]] = BLACK
   then color[w] ← RED                          Case2
        x ← p[x]                                Case2
   else if color[right[w]] = BLACK
        then color[left[w]] ← BLACK             Case3
             color[w] ← RED                     Case3
             RIGHT-ROTATE(T, w)                 Case3
             w ← right[p[x]]                    Case3
```

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

DELETE-FIXUP (3/3)

17 $\quad color[w] \leftarrow color[p[x]]$ **Case4**

18 $\quad color[p[x]] \leftarrow$ BLACK **Case4**

19 $\quad color[right[w]] \leftarrow$ BLACK **Case4**

20 $\quad$ LEFT-ROTATE(T, $p[x]$) **Case4**

21 $\quad x \leftarrow root[T]$ **Case4**

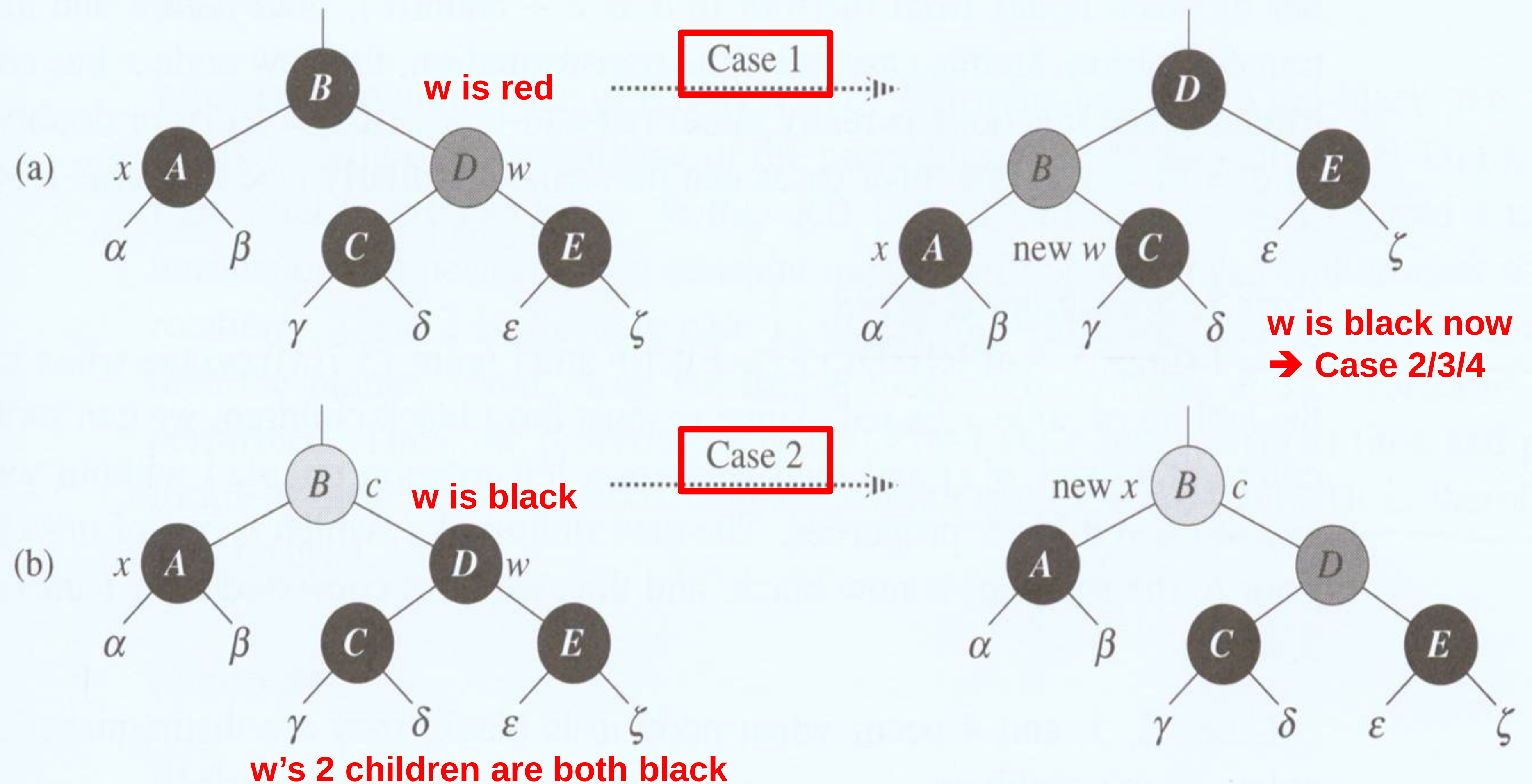
22 **else** (same as **then (Line 3)** clause with “right” and “left” exchanged)

23 $color[x] \leftarrow$ BLACK

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

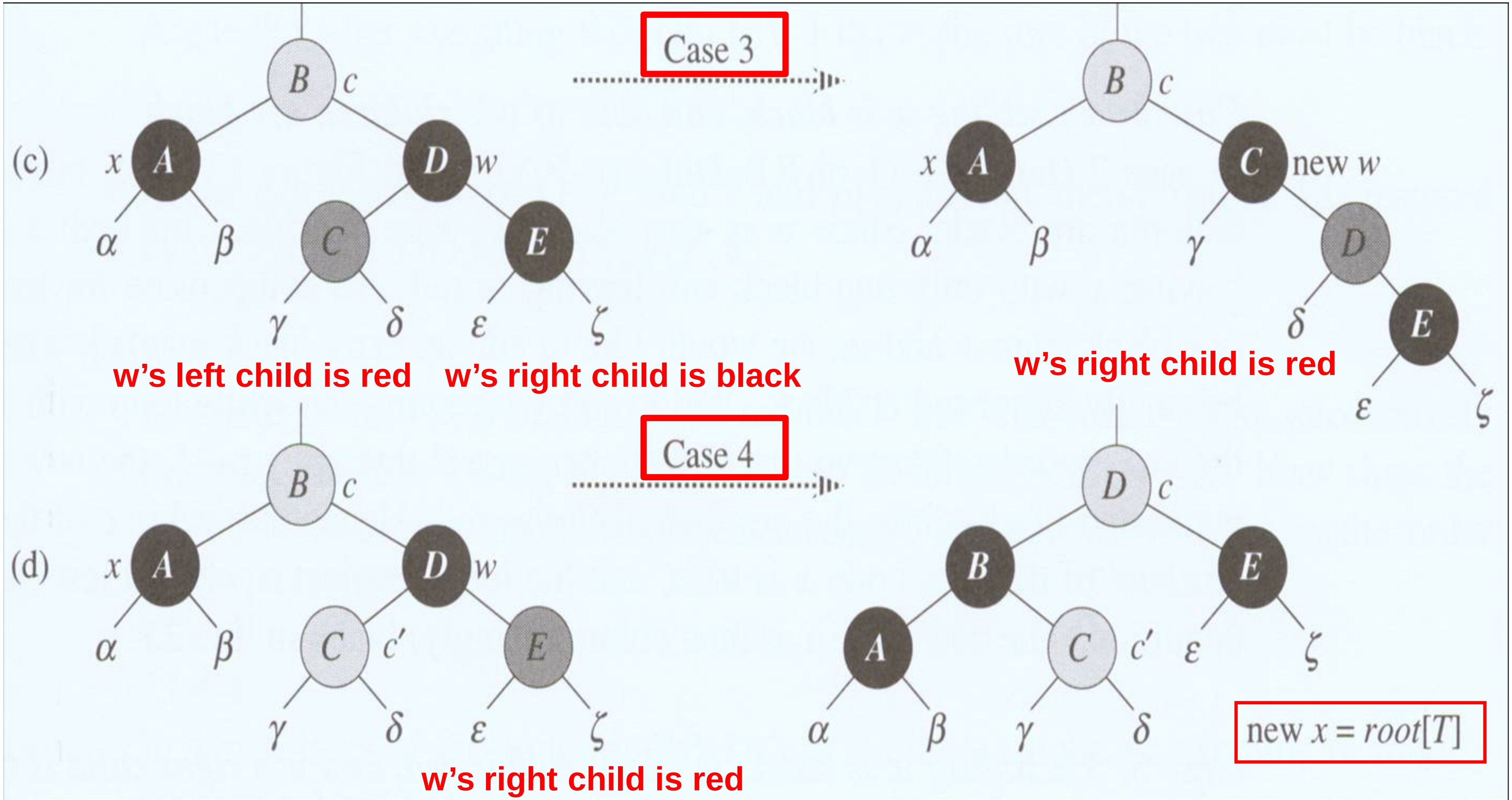
Case 1 & 2 in RB-DELETE-FIXUP

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



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

Case 3 & 4 in RB-DELETE-FIXUP



Time Complexity

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