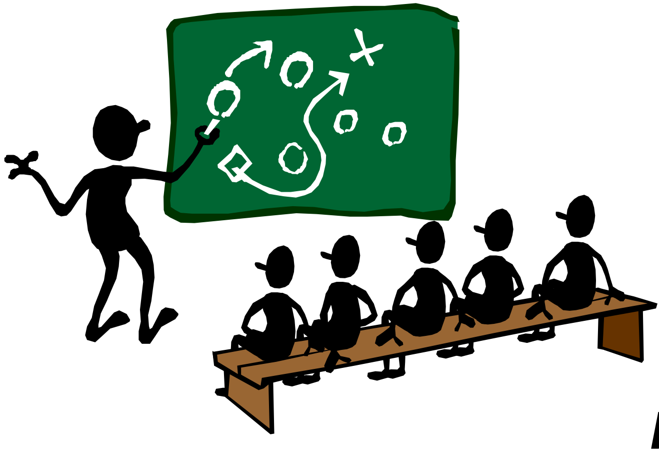
Algorithms – Chapter 12
Binary Search Trees



Juinn-Dar Huang

Professor

jdhuang@mail.nctu.edu.tw

September 2007

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

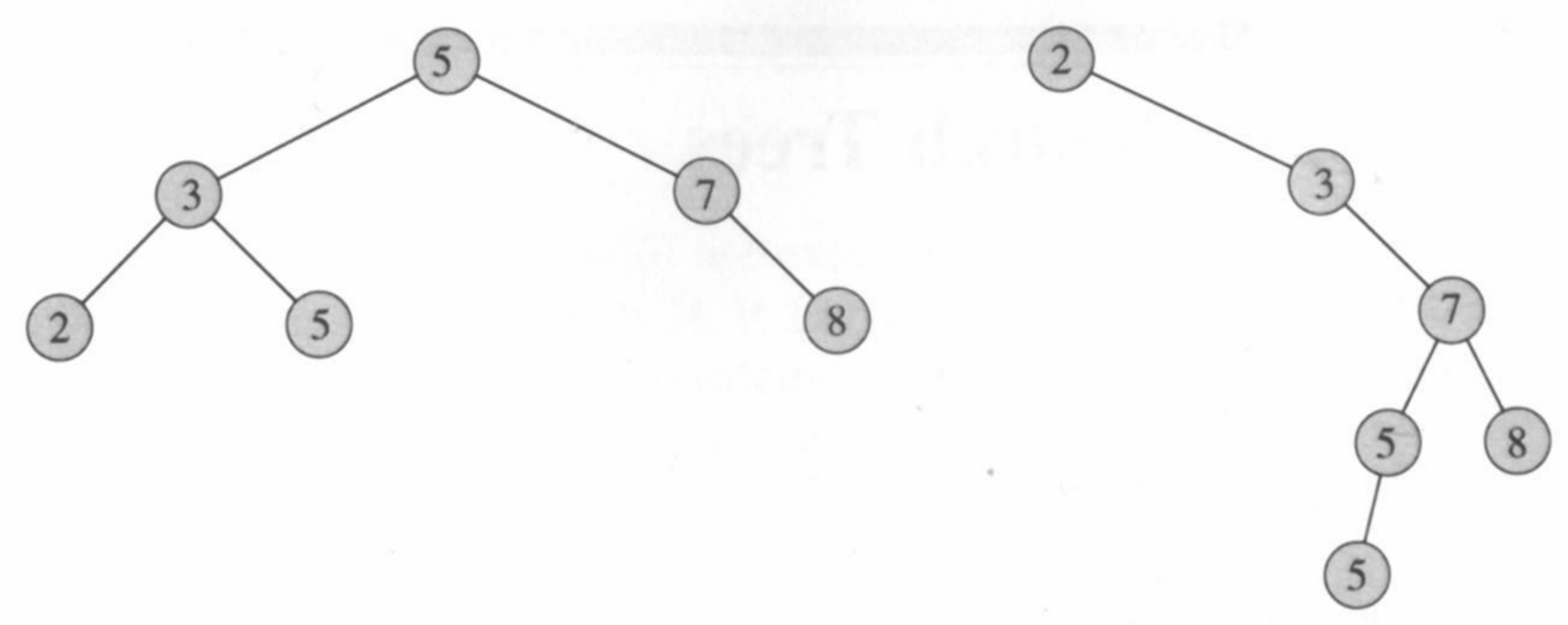
Binary Search Tree (BST)

- BSTs support
  - search, sort
  - minimum, maximum
  - predecessor, successor
  - insert, delete
- Basic operations on BSTs
  - time is proportional to the height of tree
- For a binary tree with n nodes
  - minimum height: $\Theta(\lg n)$ (e.g., complete binary tree)
  - maximum height: $\Theta(n)$ (e.g., skewed binary tree)

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

Property of BST

- Property
  - Let x be a node in a binary search tree
  - If y is a node in the left subtree of x ➔ key[y] $\leq$ key[x]
  - If y is a node in the right subtree of x ➔ key[x] $\leq$ key[y]



Tree Walks

- 3 types of tree walks
  - inorder: LVR
  - preorder: VLR
  - postorder: LRV
- Inorder walk on a BST produces a sorted list

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

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

Inorder Tree Walks

INORDER-TREE-WALK(*x*)
if $x \neq nil$
    then INORDER-TREE-WALK(*left*[*x*])
        print *key*[*x*]
        INORDER-TREE-WALK(*right*[*x*])

- If x is the root of n-node tree, then the call INORDER-TREE-WALK(x) takes Θ(n) time
  - because 2 calls for every node in the tree

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

Recursive Search on BSTs

TREE-SEARCH(x, k)

```
if x = nil or k = key[x]
    then return x
if k < key[x]
    then return TREE-SEARCH(left[x], k)
    else return TREE-SEARCH(right[x], k)
```

Time complexity: O(h)

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

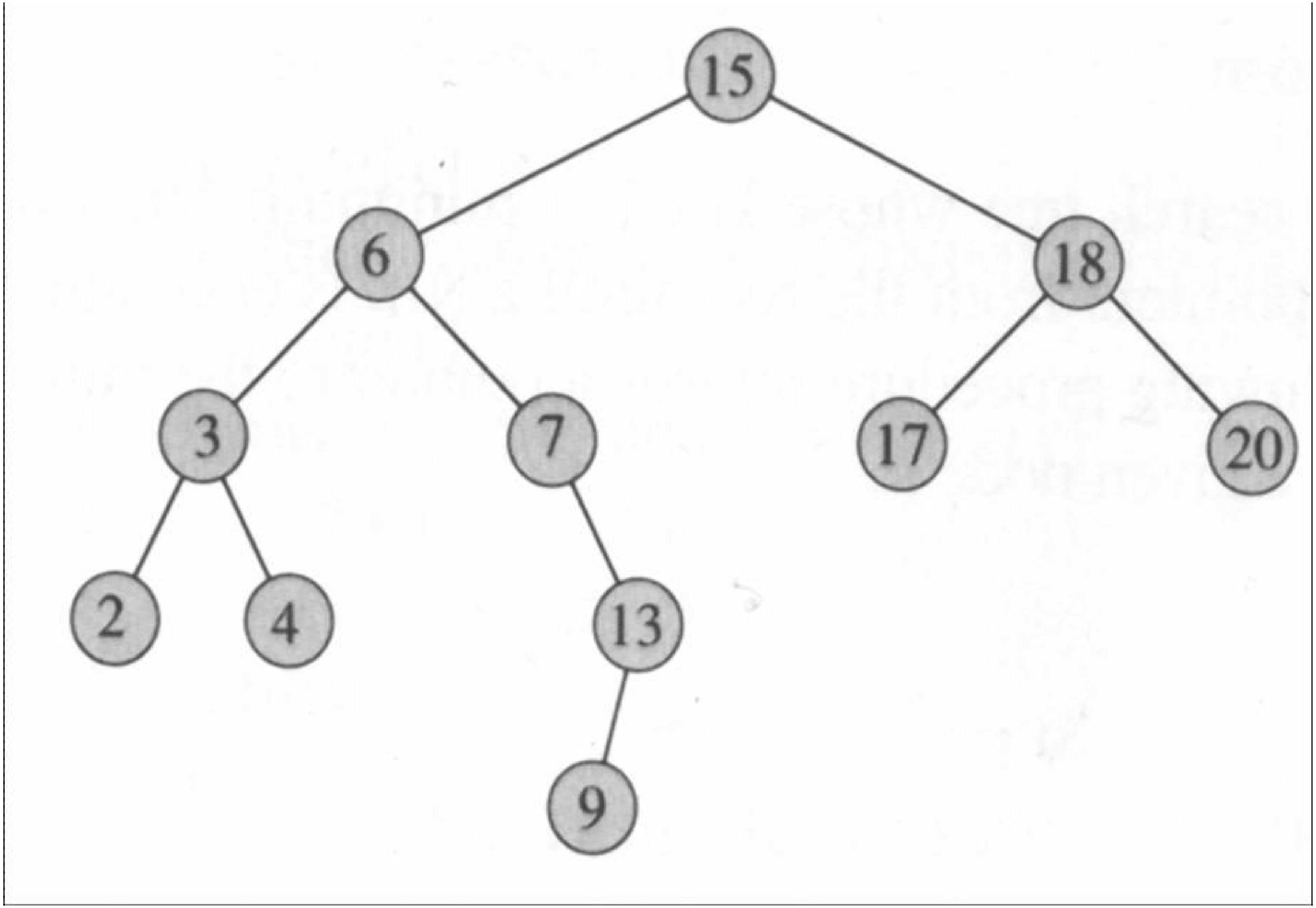
Iterative Search on BSTs

ITERATIVE-TREE-SEARCH(x,k)

While $x \neq nil$ **and** $k \neq key[x]$
    do if $k < key[x]$
        then $x \leftarrow left[x]$
        else $x \leftarrow right[x]$
return x

Time complexity: O(h)

BST Example



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

Minimum and Maximum on BSTs

TREE-MINIMUM(x)

while *left*[*x*] ≠ NIL
  do *x* ← *left*[*x*]
return *x*

leftmost element

TREE-MAXIMUM(x)

while *right*[*x*] ≠ NIL
  do *x* ← *right*[*x*]
return *x*

rightmost element

Time complexity: O(h)

Successor on BSTs (1/2)

- Successor of x
  - case 1: if x has a right subtree ➔ the minimum element in the right subtree
  - case 2: if x has no right subtree ➔ the lowest ancestor of x whose left child is x or an ancestor of x
  - case 3: x is the maximum element ➔ no successor

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

Successor on BSTs (2/2)

TREE-SUCCESSOR(x)

if $right[x] \neq nil$
    then return TREE-MINIMUM($right[x]$)
$y \leftarrow p[x]$
while $y \neq nil$ **and** $x = right[y]$
    do $x \leftarrow y$
        $y \leftarrow p[y]$
return y

Time complexity: O(h)

Try to develop TREE-PREDECESSOR(x)

Insertion on BSTs

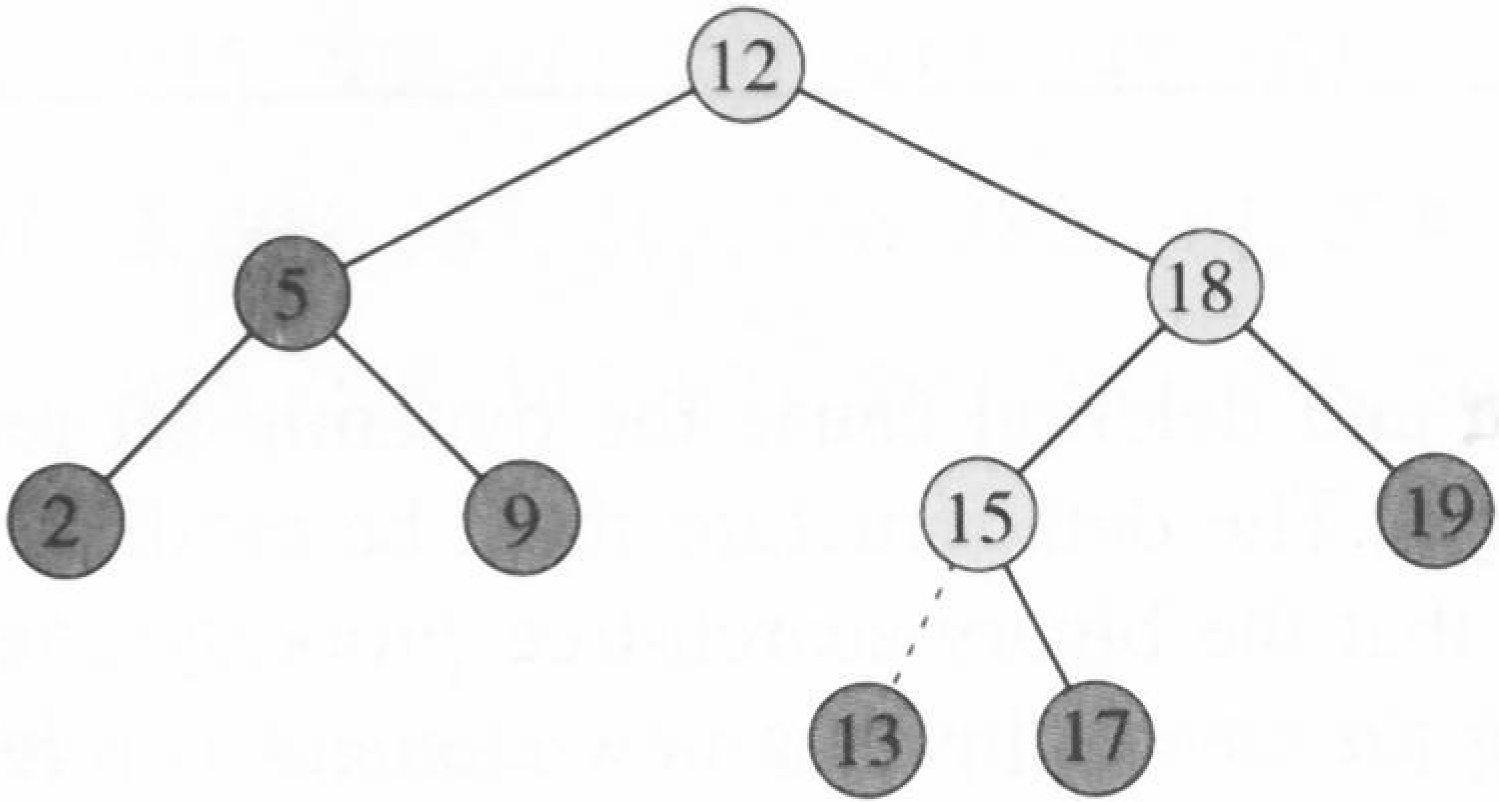
Time complexity: O(h)

TREE-INSERT(T, z)

```
y ← NIL
x ← root[T]
while x ≠ NIL
      do y ← x
            if key[z] < key[x]
               then x ← left[x]
               else  x ← right[x]
p[z] ← y
if  y = NIL
    then root[T] ← z        ▶ tree T was empty
    else if key[z] < key[y]
              then left[y] ← z
              else  right[y] ← z
```

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

Example



Deletion on BSTs (1/2)

TREE-DELETE(T, z)

```
if left[z] = NIL or right[z] = NIL
    then y ← z   ▶ z has at most one child
    else  y ← TREE-SUCCESSOR(z)  ▶ 2 children
if left[y] ≠ NIL
    then x ← left[y]
    else x ← right[y]
if x ≠ NIL
    then p[x] ← p[y]
```

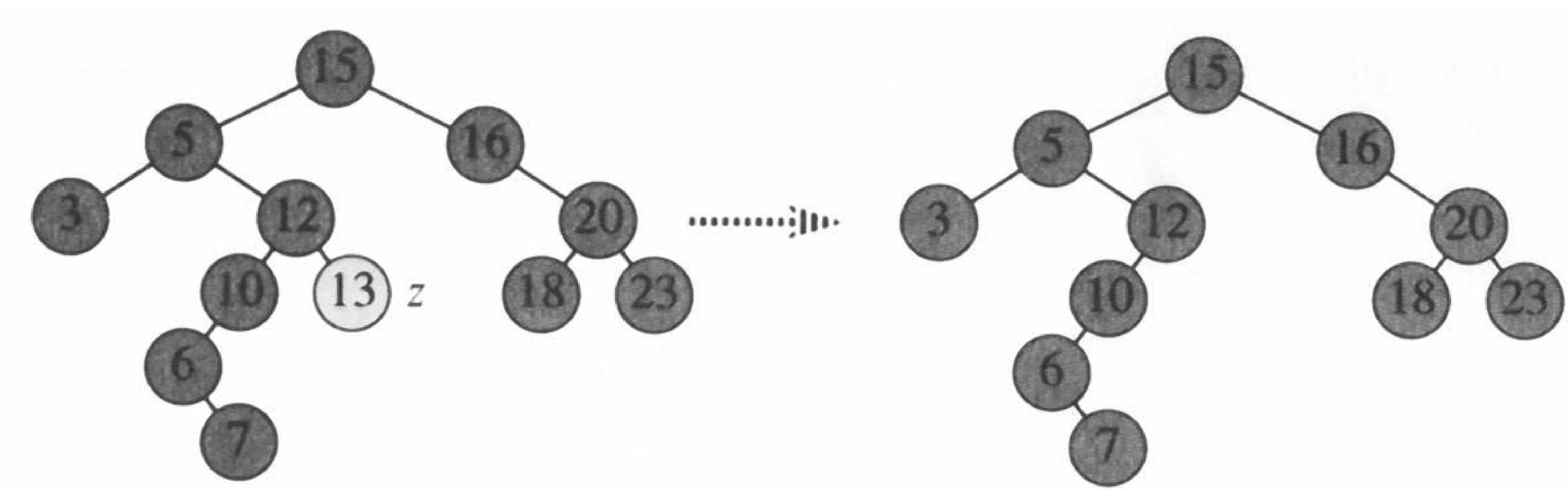
Juinn-Dar Huang
jdhuang@mail.nctu.edu.tw

Deletion on BSTs (2/2)

```
if  p[y] = NIL
        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
return y
```

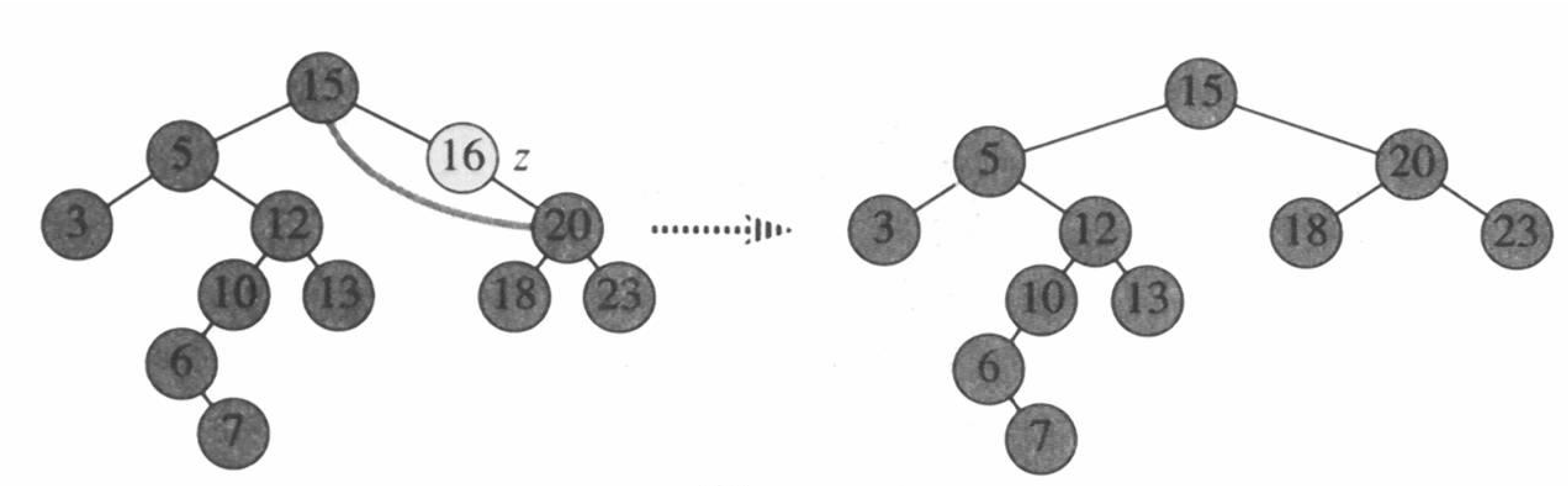

Juinn-Dar Huang jdhuang@mail.nctu.edu.tw

Case 1: z has no children



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

Case 2: z has one child



Case 3: z has 2 children