12.Binary Search Trees

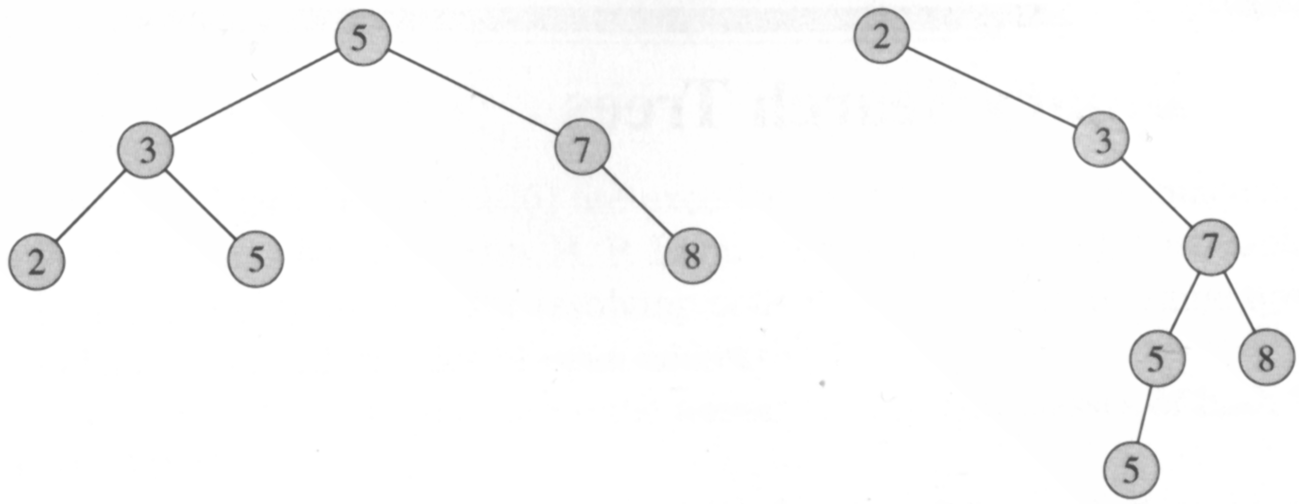
Hsu, Lih-Hsing

12.1 What is a binary search tree?

- ***Binary-search property***:

  Let x be a node in a binary search tree. If y is a node in the left subtree of x, then key[y] $\leq$ key[x]. If y is a node in the right subtree of x, then key[x] $\leq$ key[y].

Binary search Tree



Inorder tree walk

INORDER_TREE_WALK(*x*)

if $x \neq nil$

then INORDER_TREE_WALK(*left*[*x*])

print *key*[*x*]

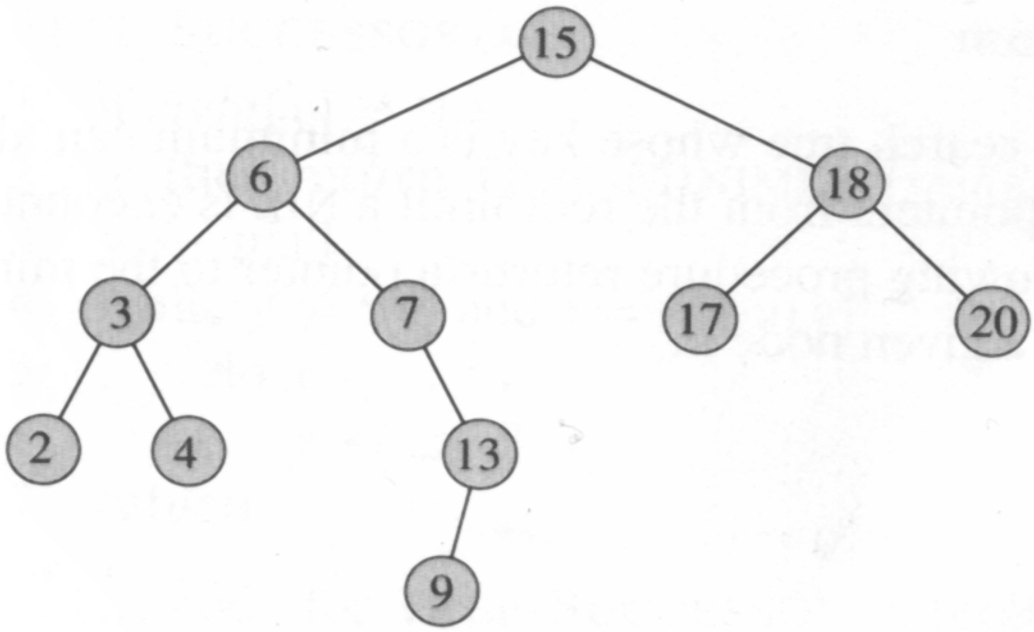
INORDER_TREE_WALK(*right*[*x*])

Theorem 12.1

If x is the root of an n-node subtree, then the call INORDER-TREE-WALK(x) takes $\Theta(n)$ time.

- ***Preorder tree walk***
- ***Postorder tree walk***

12.2 Querying a binary search tree



TREE_SEARCH(*x*,*k*)

TREE_SEARCH(*x*,*k*)

1 **if** $x = nil$ **or** $k = key[x]$

2 **then return** *x*

3 **if** $k < key[x]$

4 **then return** TREE_SEARCH(left[*x*],*k*)

5 **else return** TREE_SEARCH(right[*x*],*k*)

ITERATIVE_SEARCH(*x*,*k*)

ITERATIVE_SEARCH(*x*,*k*)

1 **While** $x \neq nil$ **or** $k \neq key[x]$

2 **do if** $k < key[x]$

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

4 **then** $x \leftarrow right[x]$

5 **return** *x*

MAXIMUM and MINIMUM

- TREE_MINIMUM(x)

1 **while** $left[x] \neq \mathrm{NIL}$

2 **do** $x \leftarrow left[x]$

3 **return** *x*

- TREE_MAXIMUM(x)

1 **while** $right[x] \neq \mathrm{NIL}$

2 **do** $x \leftarrow right[x]$

3 **return** *x*

SUCCESSOR and PREDECESSOR

TREE_SUCCESSOR

TREE_SUCCESSOR

if $right[x] \neq nil$
then return $\text{TREE\_MINIMUM}(right[x])$
$y \leftarrow p[x]$
while $y \neq nil$ **and** $x = right[y]$
do $x \leftarrow y$
$y \leftarrow p[y]$
return y

Theorem 12.2

- The dynamic-set operations, SEARCH, MINIMUM, MAXIMUM, SUCCESSOR, and PREDECESSOR can be made to run in O(h) time on a binary search tree of height h.

12.3 Insertion and deletion

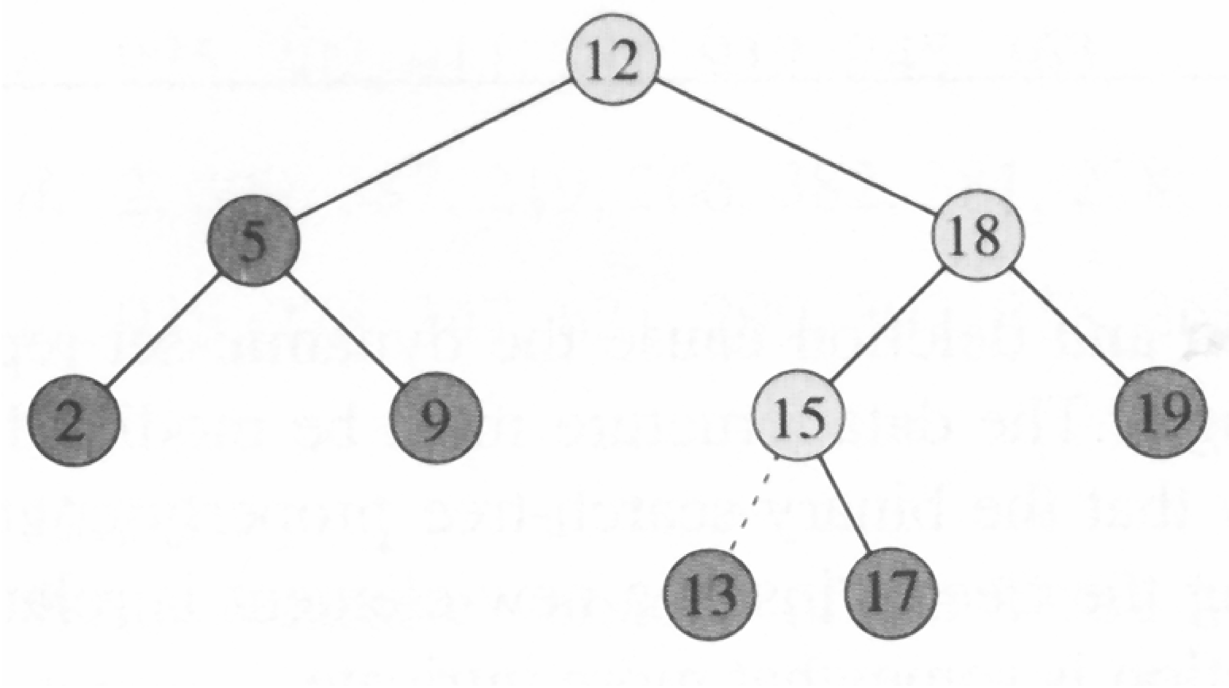
Insertion

Tree-Insert(T,z)

$y \leftarrow$ NIL
$x \leftarrow root[T]$
while $x \neq$ NIL
  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
  then $root[T] \leftarrow z$ ▶ tree T was empty
  else if $key[z] < key[y]$
    then $left[y] \leftarrow z$
    else $right[y] \leftarrow z$

Inserting an item with key 13 into a binary search tree



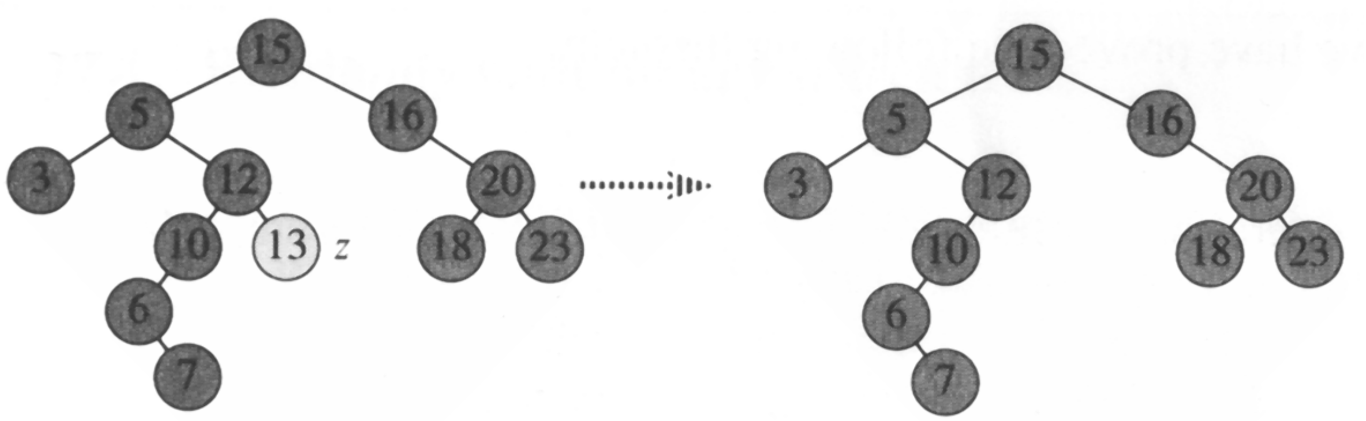
Deletion

Tree-Delete(T,z)

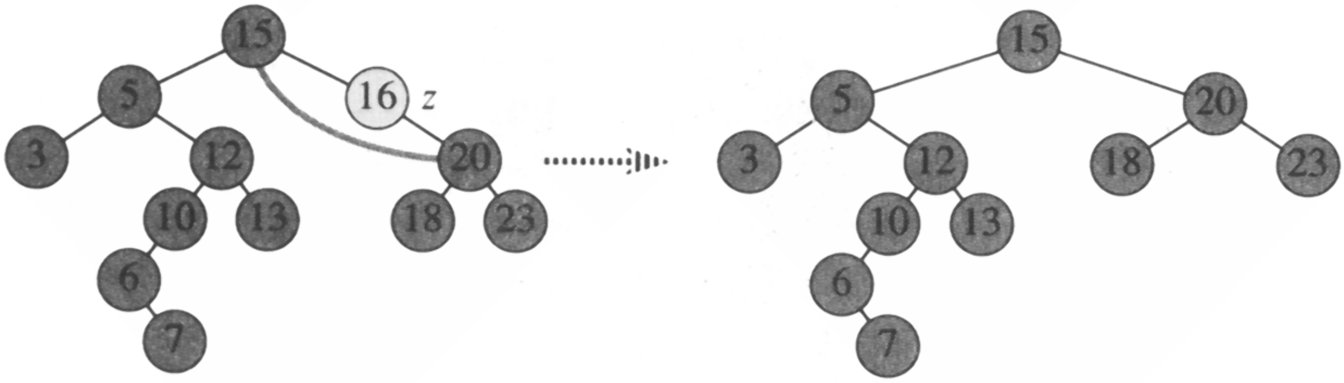
if *left*[*z*] = NIL **or** *right*[*z*] = NIL
    then $y \leftarrow z$
    else $y \leftarrow$ Tree-Successor(z)
if *left*[*y*] $\neq$ NIL
    then $x \leftarrow$ *left*[*y*]
    else $x \leftarrow$ *right*[*y*]
if $x \neq$ NIL
    then $p[x] \leftarrow p[y]$

```
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
```

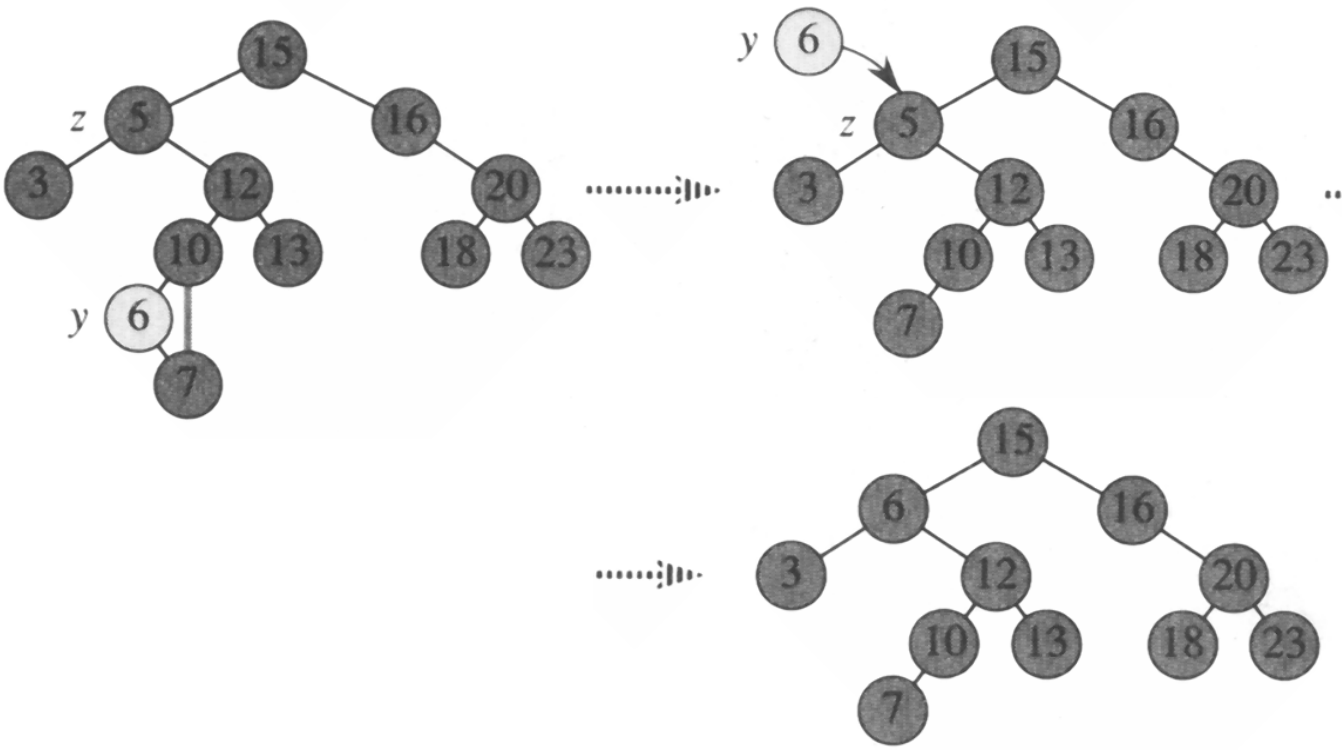

z has no children



z has only one child



z has two children



Theorem 12.3

- The dynamic-set operations, INSERT and DELETE can be made to run in O(h) time on a binary search tree of height h.