SFU CMPT-307 2008-2 Lecture: Week 9

Ján Maňuch

E-mail: jmanuch@sfu.ca

Lecture on July 8, 2008, 5.30pm-8.20pm

Binary search trees

Support many dynamic-set operations, e.g.

- Search
- Minimum
- Maximum
- Insert
- Delete
- …

Can be used as dictionary, priority queue…

Running time depends on **height** of tree:

- if complete, then $\Theta(\log n)$ in the worst case
- if just one chain, then $\Theta(n)$ in the worst case
- if random, then $\Theta(\log n)$ expected

First of all, **what is a binary search tree**?

- it's a binary tree
- represented using linked data structure
- nodes are objects
- objects store
  - key
  - data
  - pointers to left child, right child, and parent
    (NULL if one is missing)
    root is only node with parent=NULL

Binary-search-tree property

x node in binary search tree

- nodes y in **left** subtree of x have key$[y] \leq$ key$[x]$
- nodes y in **right** subtree of x have key$[y] \geq$ key$[x]$

Note: heaps are different!

Definition: *inorder walk* of binary tree:

for each node x

1. visit left subtree (recursively)
2. print key of x
3. visit right subtree (recursively)

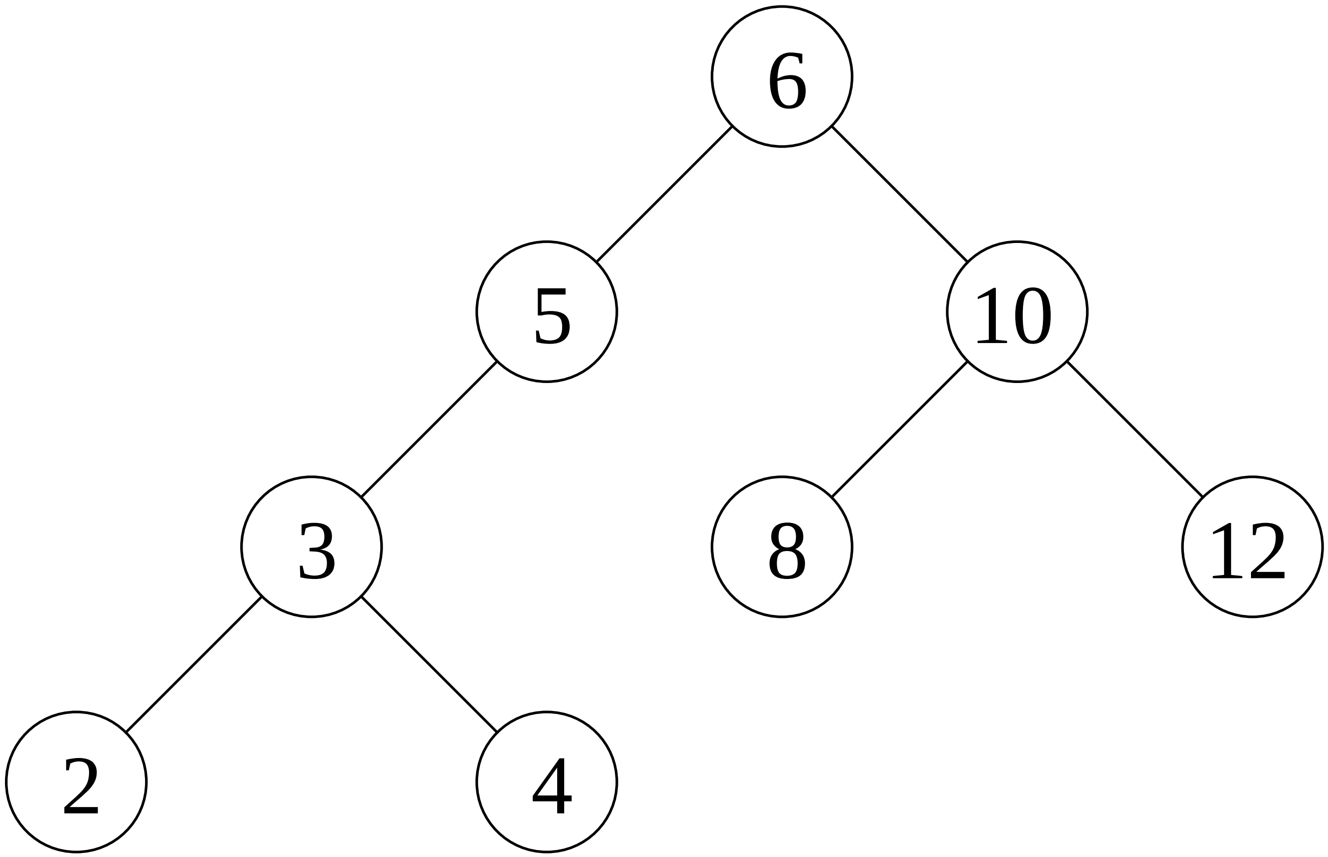
Inorder-Tree-Walk(x)

```
if x ≠ NULL then
    Inorder-Tree-Walk(left(x))
    print key(x)
    Inorder-Tree-Walk(right(x))
end if
```

Interesting property of BSTs:

In-order walk of BST prints all keys in *sorted order*

Example for inorder walk:



Result is **2 – 3 – 4 – 5 – 6 – 8 – 10 – 12**

Assignment Problem 9.1. (deadline: July 15, 5:30pm)
Give a nonrecursive algorithm that performs an inorder tree walk using only a constant memory. Your algorithm can test two pointers for equality.

Theorem. If x is root of n-node (sub)tree, then Inorder-Tree-Walk(x) takes $\Theta(n)$ time.

Proof. $T(n)$ time if procedure called on n-node (sub)tree

Clearly $T(0) = c$ for some constant c
(test $x \neq$ NULL)

Otherwise, suppose left subtree has k nodes, right subtree has $n - k - 1$ nodes. Then for $n > 0$

$$T(n) = T(k) + T(n - k - 1) + d$$

with d constant

$$T(n) = T(k) + T(n-k-1) + d$$

We will use substitution method to show $T(n) = (c+d)n + c$

For $n = 0$ we have $(c+d)n + c = c = T(0)$, OK

Induction hypothesis: For every $m < n$, $T(m) = (c+d)m + c$.

We will show that is true also for n:

$$\begin{aligned}
T(n) &= T(k) + T(n-k-1) + d \\
&= [(c+d)k + c] + [(c+d)(n-k-1) + c] + d \\
&= (c+d)k + c + (c+d)n - (c+d)k - \\
&\quad -(c+d) + c + d \\
&= (c+d)n + c - (c+d) + c + d \\
&= (c+d)n + c
\end{aligned}$$

Searching

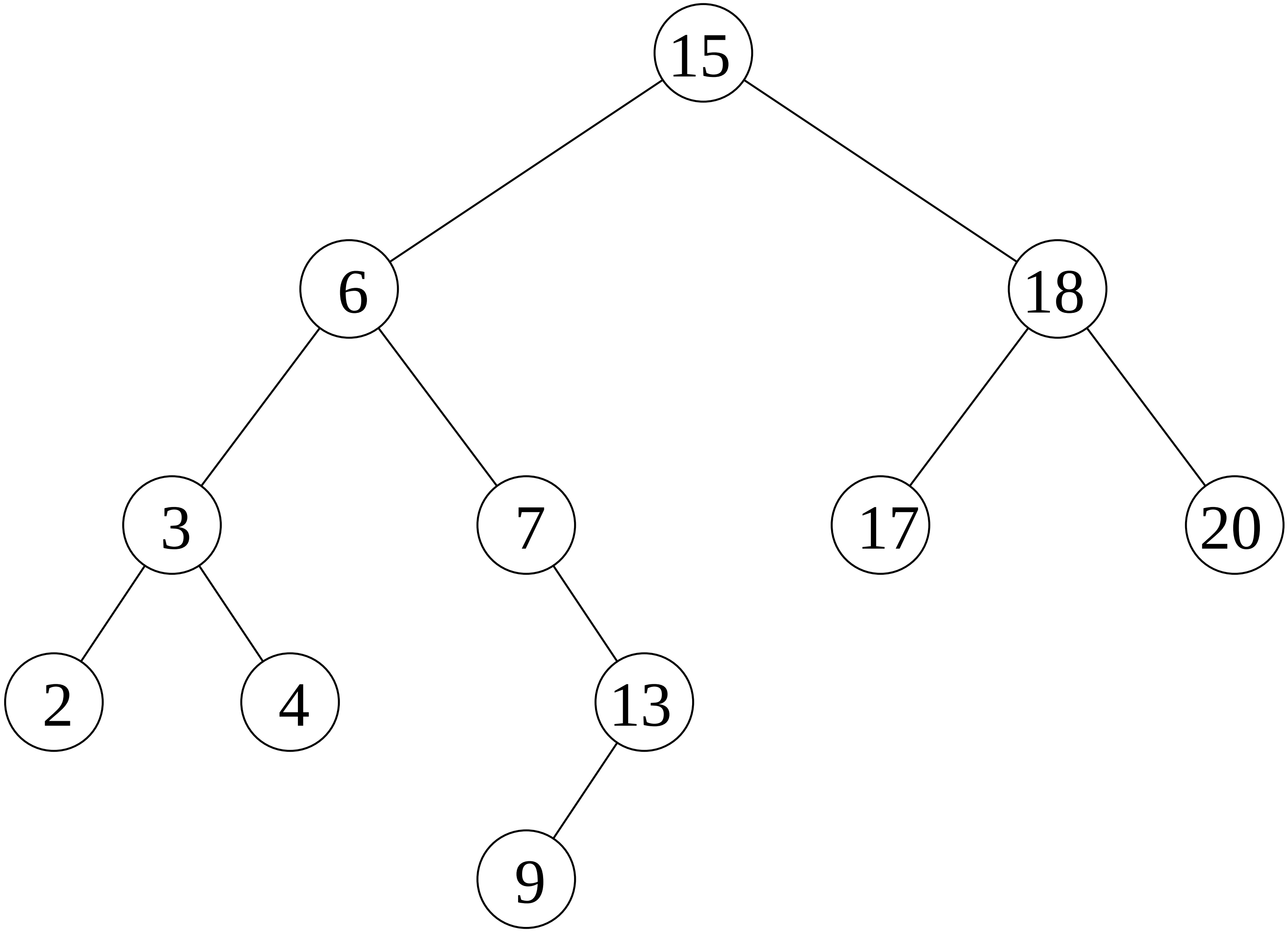
Want to search for a node with given key k

Return pointer to node if exists, otherwise NULL

- begin search at root
- follow path downward
  for node x on path, compare key$[x]$ with k
  - if equal, done
  - if $k <$ key$[x]$, continue in left subtree
    (left subtree contains keys $\leq$ key$[x]$)
  - if $k >$ key$[x]$, continue in right subtree
    (right subtree contains keys $\geq$ key$[x]$)

Tree-Search(x, k)

if $x =$ NULL **or** $k =$ key$[x]$ **then**
    return x
else if $k <$ key$[x]$ **then**
    return Tree-Search(left$[x], k$)
else
    return Tree-Search(right$[x], k$)
end if



Running time is $O(h)$, h height of tree

Minimum/maximum

Minimum of the tree rooted in x can be found by following **left pointers** as long as possible (**not** necessarily to a leaf!)

Tree-Minimum(x)

while left$[x] \neq$ NULL **do**
    $x \leftarrow$ left$[x]$
end while
return x

Maximum of the tree rooted in x can be found by following **right pointers** as long as possible (**not** necessarily to a leaf!)

Tree-Maximum(x)

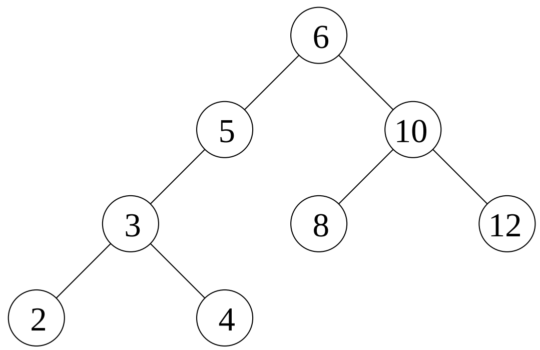
while right$[x] \neq$ NULL **do**
    $x \leftarrow$ right$[x]$
end while
return x

Both have running time $O(h)$, h height of tree

Successor/predecessor

Definition: successor/predecessor in **sorted order given by inorder walk**

For instance, if values $x_1 < x_2 < \cdots < x_n$ are stored in a tree, then the successor of x_i is x_{i+1}.



Idea of an algorithm for finding successor:

- If **right subtree** of x is **nonempty**, then successor of x is **leftmost** node in **right** subtree ("the smallest among the larger")
  Found by calling Tree-Minimum on right subtree
- Otherwise (right subtree **is** empty and x **has** a successor), then this is the **lowest ancestor** of x whose **left child** is **also** ancestor of x
  (A node is ancestor of itself)

Tree-Successor(x)

if right$[x] \neq$ NULL **then**
  return Tree-Minimum(right$[x]$)
end if
$y \leftarrow$ parent$[x]$
while $y \neq$ NULL **and** $x =$ right$[y]$ **do**
  $x \leftarrow y$
  $y \leftarrow$ parent$[y]$
end while
return y

Running time clearly $O(h)$, h height of tree

Tree-Predecessor symmetric

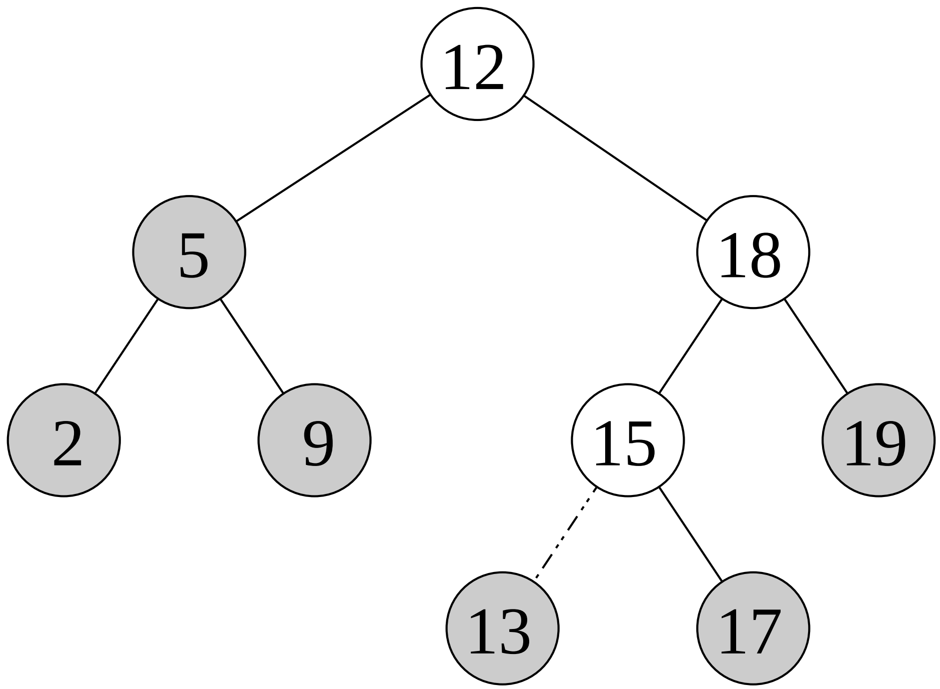
Theorem. Operations Search, Minimum, Maximum, Successor, Predecessor run in time $O(h)$ in BST of height h

Assignment Problem 9.2. (deadline: July 15, 5:30pm)
Consider a binary tree T whose keys are distinct. Show that if the right subtree of a node x in T is empty and x has a successor y, then y is the lowest ancestor of x whose left child is also an ancestor of x. (Recall that every node is its own ancestor.)

Insertion

Now we're talking about **dynamic** sets

Insertion of new element easy. From root, walk down tree according to value of new key and open new leaf



Running time again $O(h)$

Want to insert new value v

Given node z with key$[z] = v$, left$[z] =$ right$[z] =$ NULL

Tree-Insert(T, z)

$y \leftarrow$ NULL
$x \leftarrow$ root$[T]$
while $x \neq$ NULL **do**
    $y \leftarrow x$
    if key$[z] <$ key$[x]$ **then**
        $x \leftarrow$ left$[x]$
    else
        $x \leftarrow$ right$[x]$
    end if
end while
parent$[z] \leftarrow y$
if $y =$ NULL **then**
    root$[T] \leftarrow z$ /* *T was empty* */
else if key$[z] <$ key$[y]$ **then**
    left$[y] \leftarrow z$
else
    right$[y] \leftarrow z$
end if

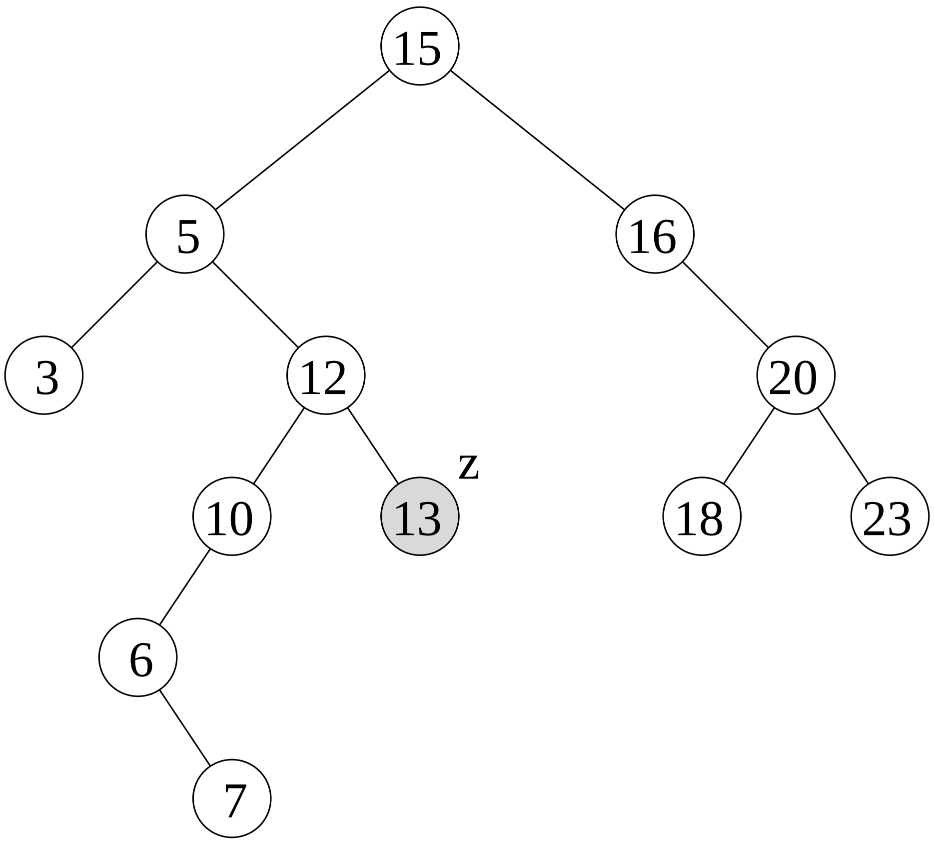
Deletion

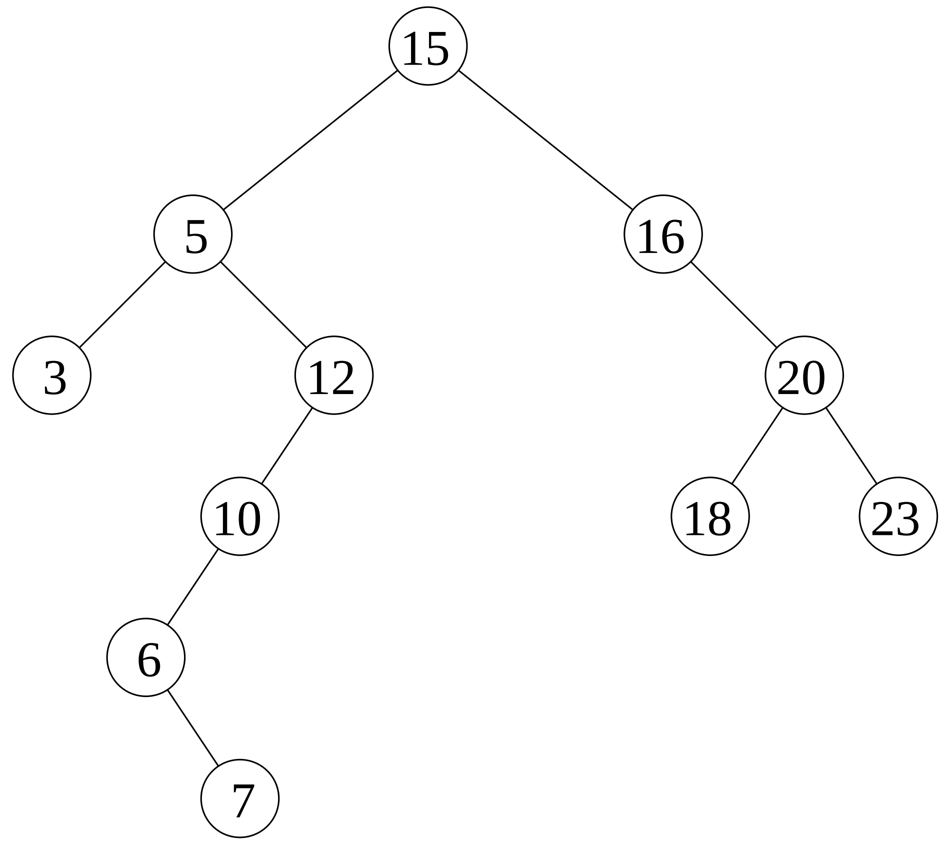
Given pointer to some node z. Three cases.

1. z has no children
   At z's parent parent$[z]$, just replace link to z with NULL
2. z has one child
   splice out z, make new link between its parent and its child
3. z has two children
   splice out z's successor y (which has no left child, as seen from Homework 9.3), and replace z's key and data with y's key and data

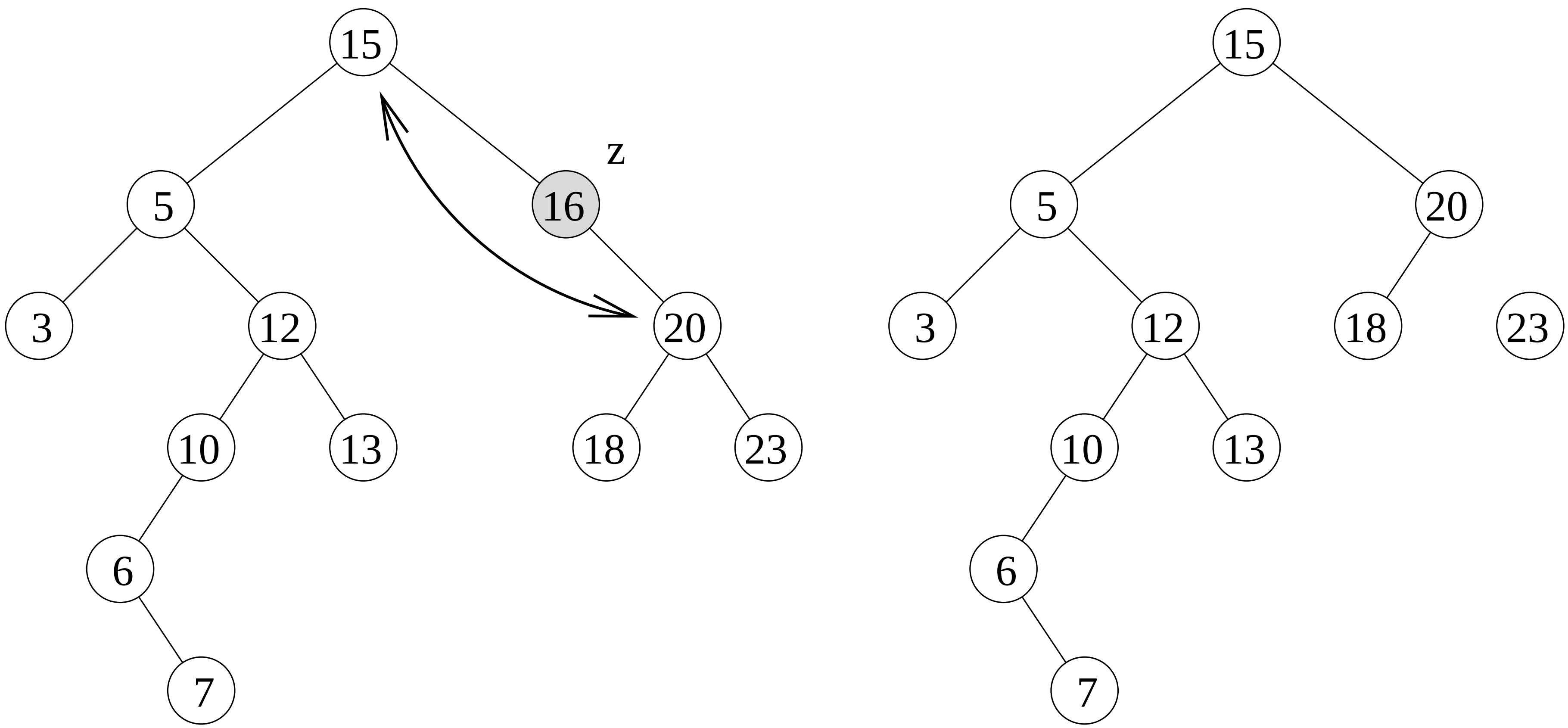
Assignment Problem 9.3. (deadline: July 15, 5:30pm)
Show that if a node in a binary search tree has two children, then its successor has no left child and its predecessor has no right child.

no children

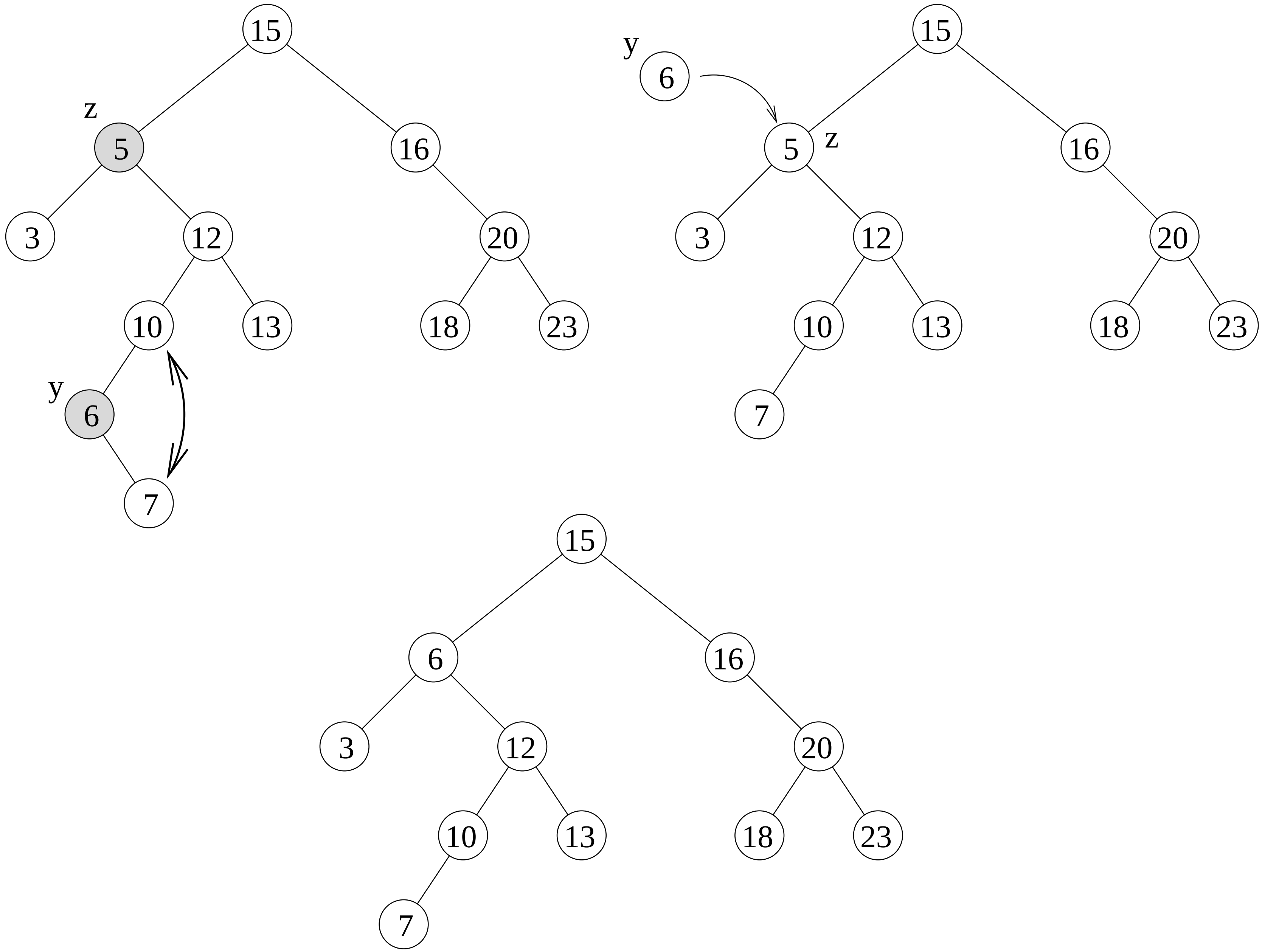




one child



two children



Tree-Delete(T, z)

if left$[z]$ = NULL **or** right$[z]$ = NULL **then**
    $y \leftarrow z$
else
    $y \leftarrow$ Tree-Successor(z)
end if
if left$[y] \neq$ NULL **then**
    $x \leftarrow$ left$[y]$
else
    $x \leftarrow$ right$[y]$
end if
if $x \neq$ NULL **then**
    parent$[x] \leftarrow$ parent$[y]$
end if
if parent$[y]$ = NULL **then**
    root$[T] \leftarrow x$
else if y = left[parent$[y]$] **then**
    left[parent$[y]$] $\leftarrow x$
else
    right[parent$[y]$] $\leftarrow x$
end if
if $y \neq z$ **then**
    key$[z] \leftarrow$ key$[y]$
    copy y's data into z
end if
return y