

Ch14: Binary Search Tree

305233, 305234

Algorithm Analysis and Design

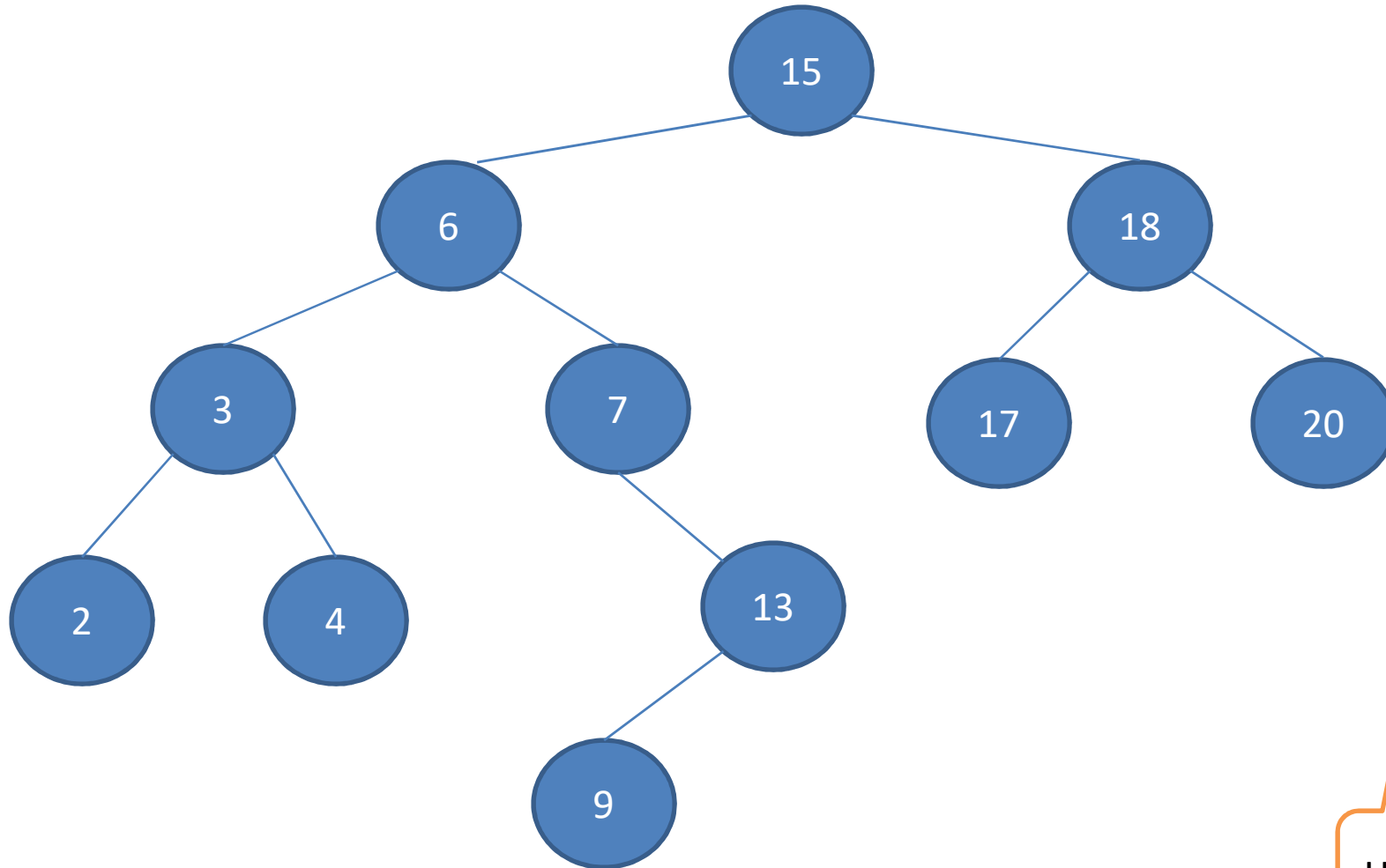
Jiraporn Pooksook

Naresuan University

Binary Search Tree

- A binary search tree is organized in a binary tree where each node contains fields left, right, and p that point to the nodes corresponding to its left child, right child and parent, respectively.
- The **binary search tree property**:
 - Let x be a node in a binary search tree. If y is a node in the left subtree of x , then $\text{key}[y] \leq \text{key}[x]$. If y is a node in the right subtree of x , then $\text{key}[x] \leq \text{key}[y]$.
- Take time proportional to the height of the tree.
- Expected height of a randomly built binary search tree is $O(\lg n)$, so that basic dynamic-set operations on such a tree take $\Theta(\lg n)$ time on average.

Binary Search Tree



Height = $\lg n$

Inorder-Tree-Walk(x)

```
if x != NIL
  then Inorder-Tree-Walk(left[x])
  print key[x]
  Inorder-Tree-Walk(right[x])
```

It takes $\Theta(n)$ time

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)
```

Recursion from a path downward from the root of the tree , so the running time is $\Theta(h)$

Iterative-Tree-Search(x,k)

```
while x != NIL and k != key[x]
  do if k < key[x]
      then x = left[x]
      else x = right[x]
return x
```

Tree-Minimum(x)

```
while left[x] != NIL
    do x = left[x]
return x
```

Tree-Maximum(x)

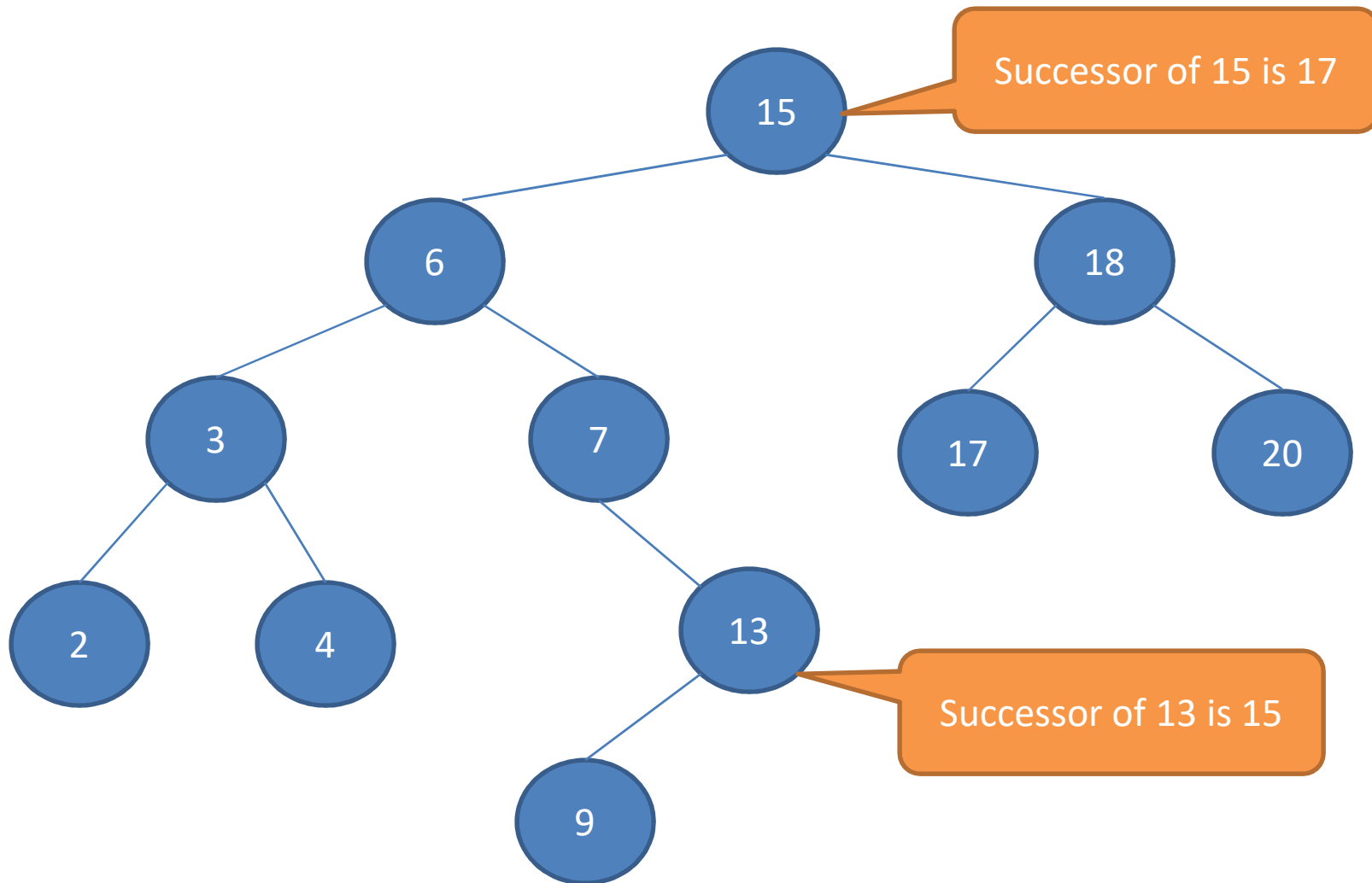
```
while right[x] != NIL  
    do x = right[x]  
return x
```


Tree-Successor(x)

```
if right[x] != NIL
    then return Tree-Minimum(right[x])
Y=p[x]
while y!=NIL and x = right[y]
    do x=y
    y=p[y]
return y
```

We either follow a path up the tree or follow a path down the tree, so the running time is $\Theta(h)$

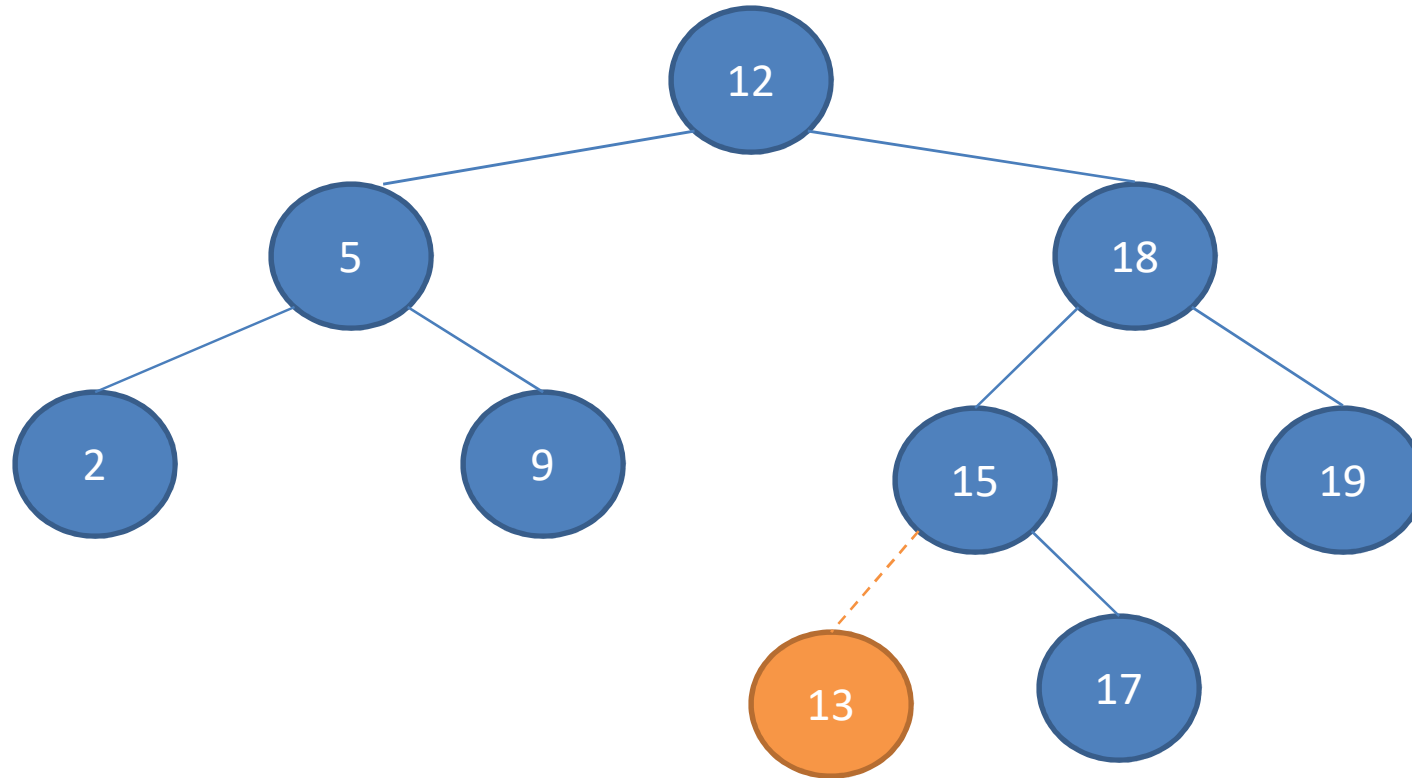
Tree-Successor



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
    else if key[z] < key[y]
        then left[y] = z
        else right[y] = z
```

Example: Tree-Insert(T,13)

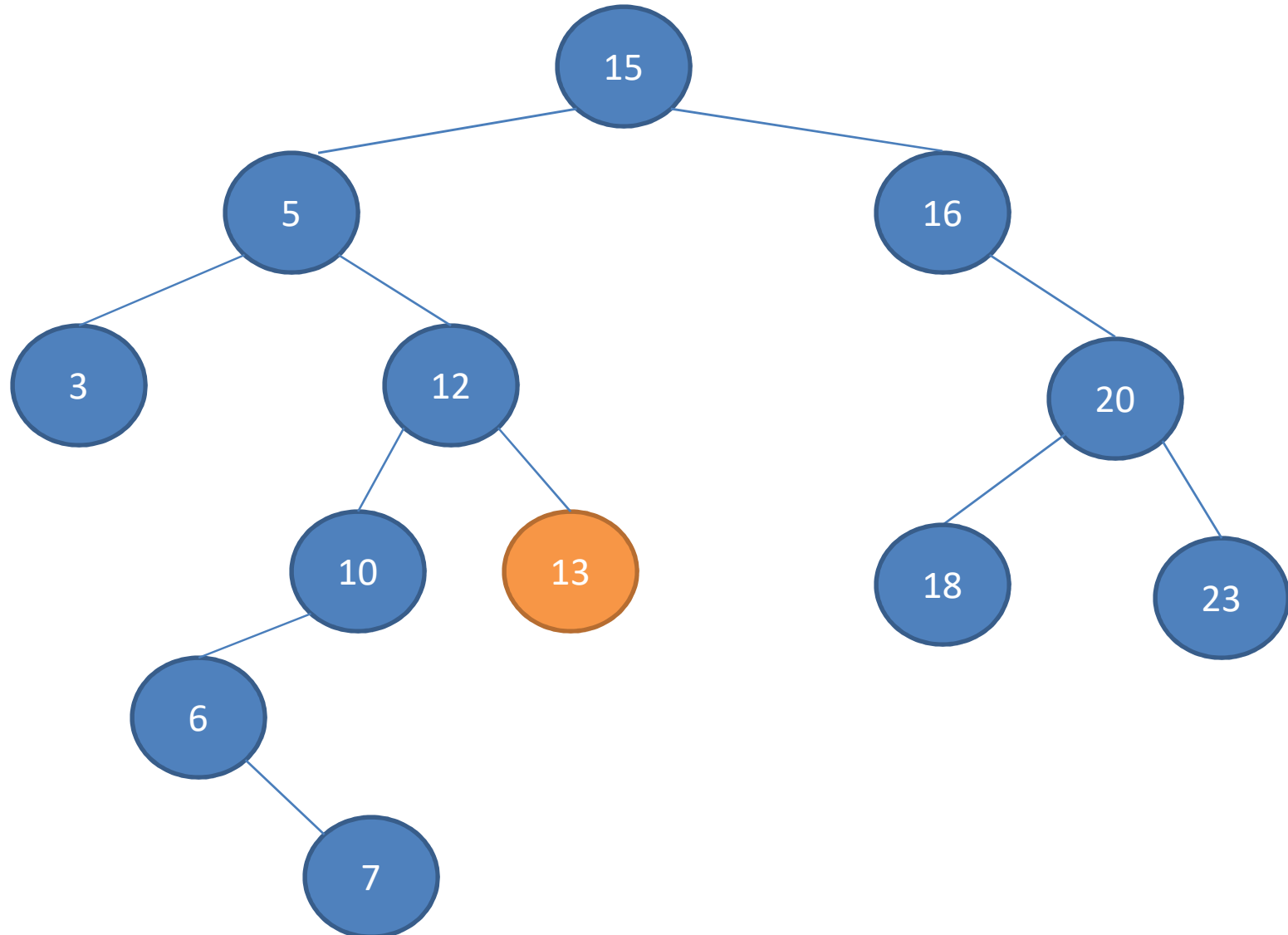


Recursion from a path downward from the root of the tree , so the running time is $\Theta(h)$

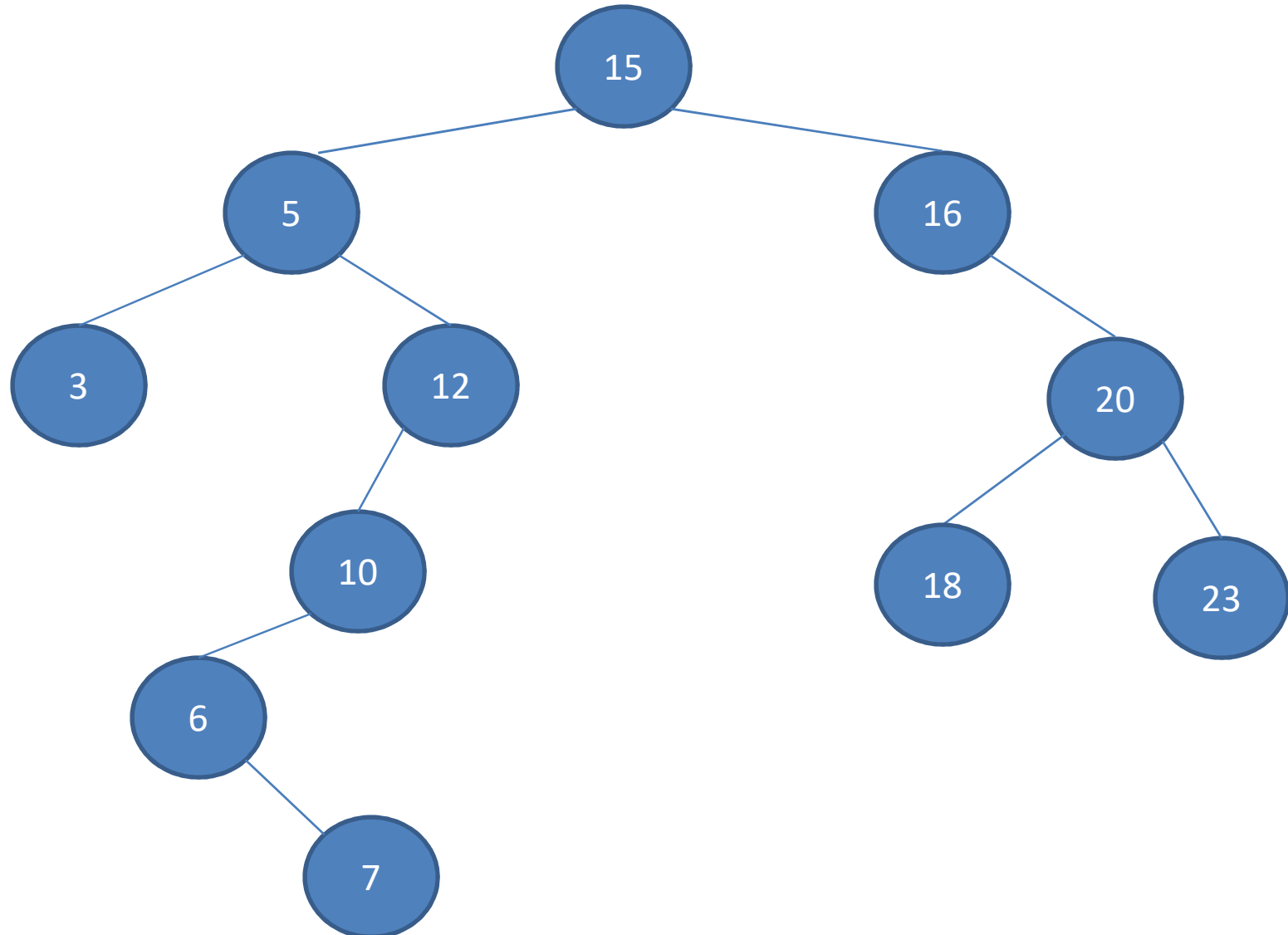
Tree-Delete (T,z)

```
if left[z] = NIL or right[z] = NIL
    then y = z
    else y = Tree-Successor(z)
if left[y] != NIL
    then x = left[y]
    else x = right[y]
if x != NIL
    then p[x] = 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
```

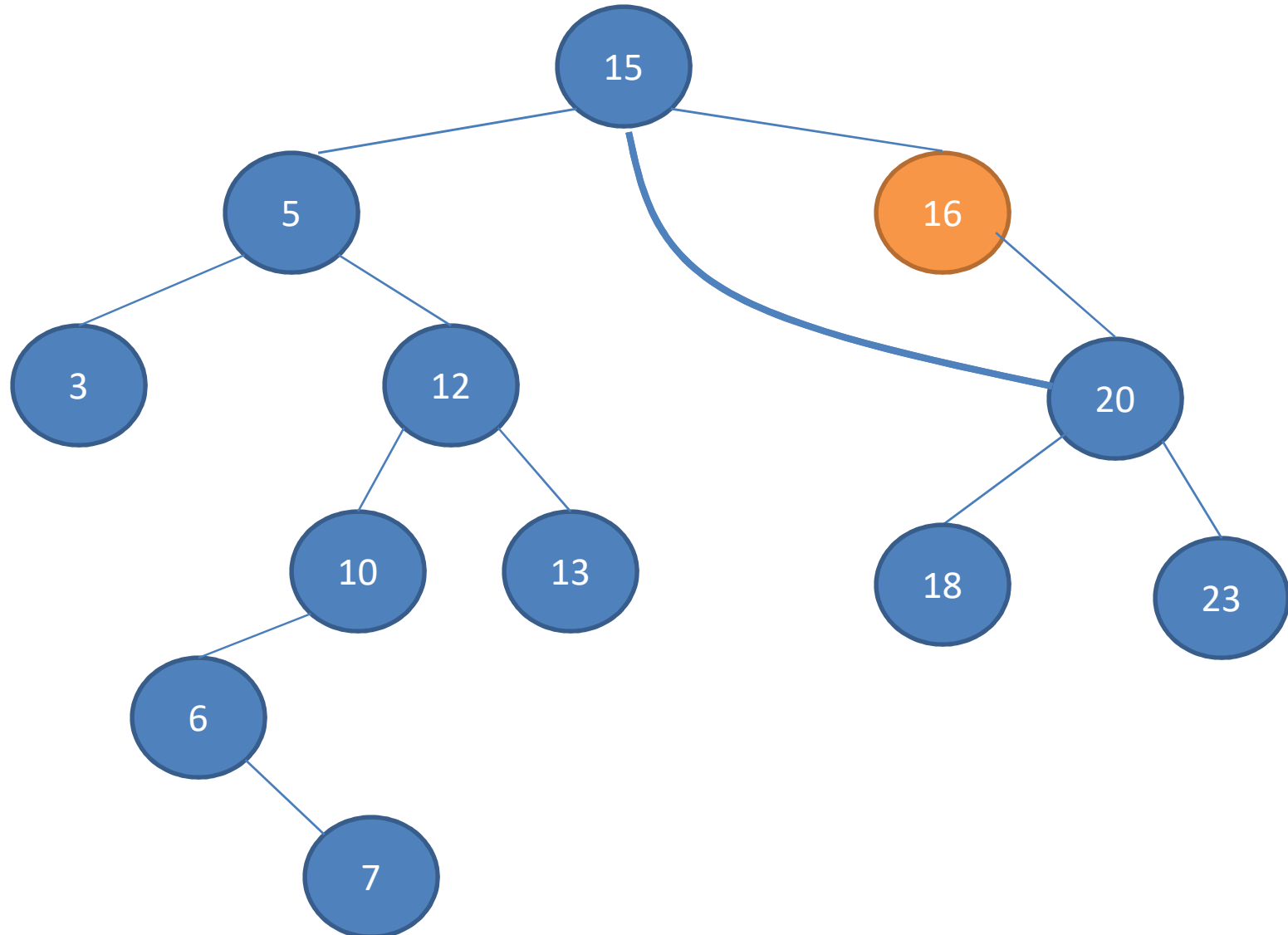
Example: Tree-Delete(T,13)



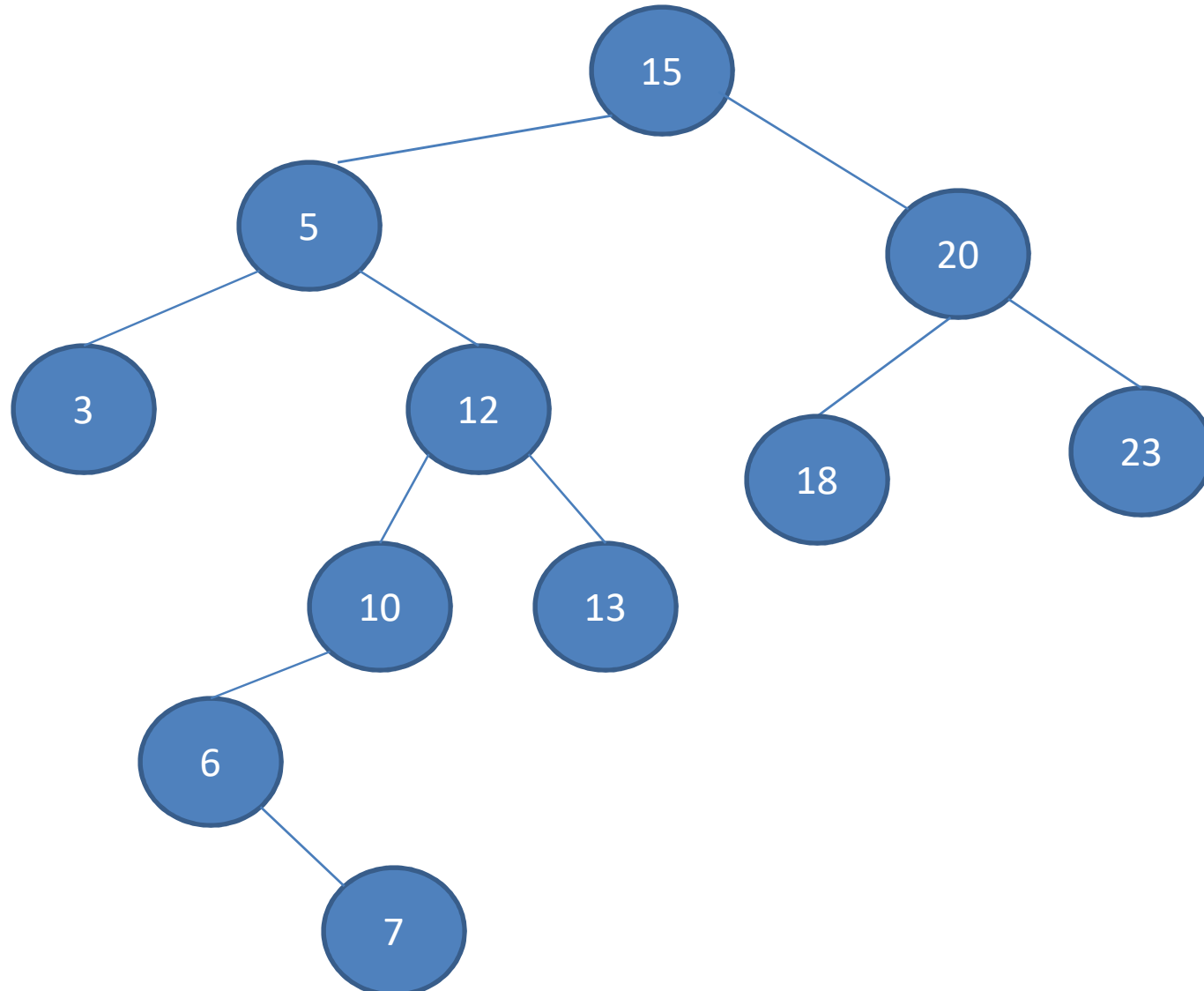
Example: Tree-Delete(T,13)



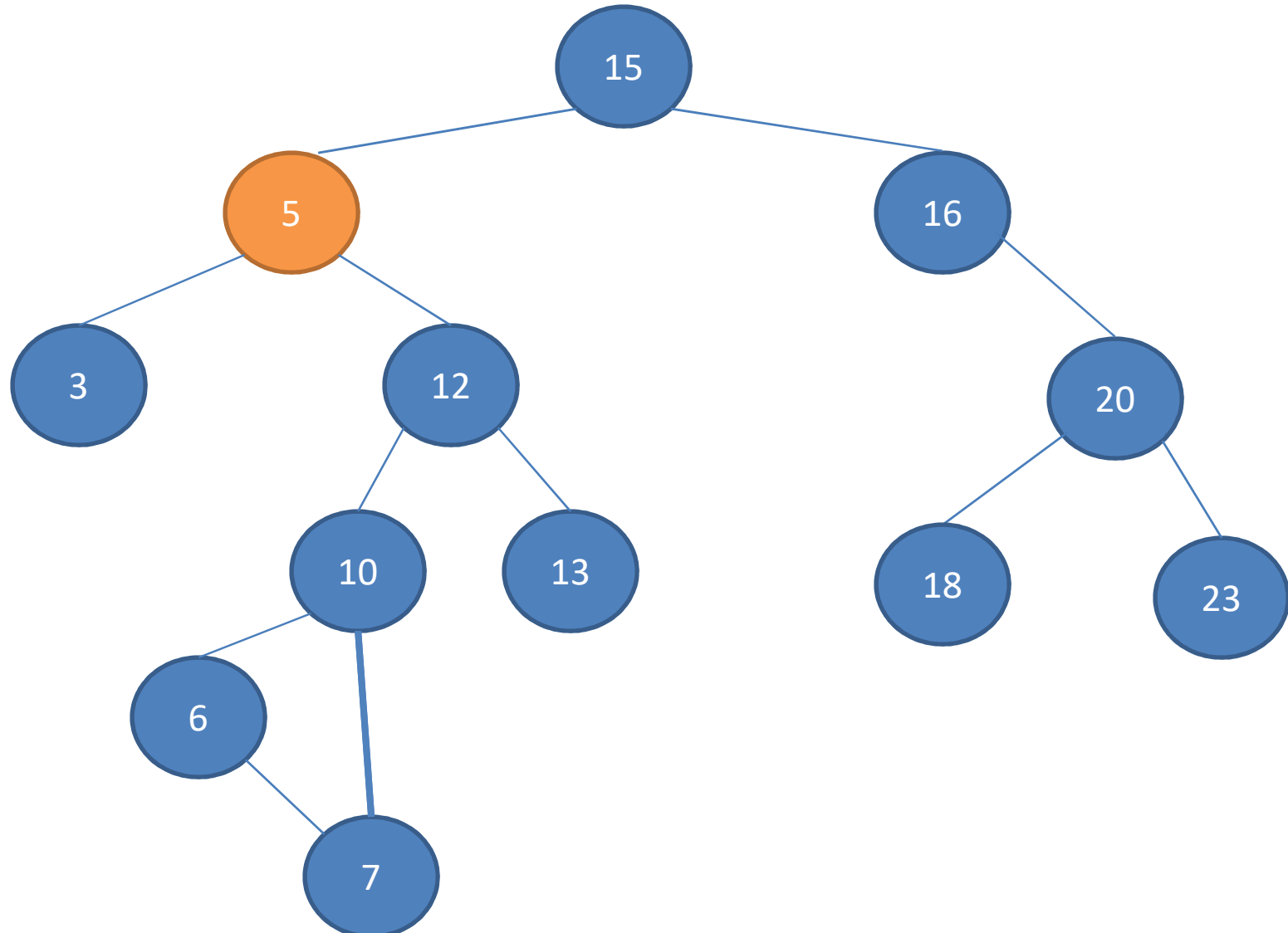
Example: Tree-Delete(T,16)



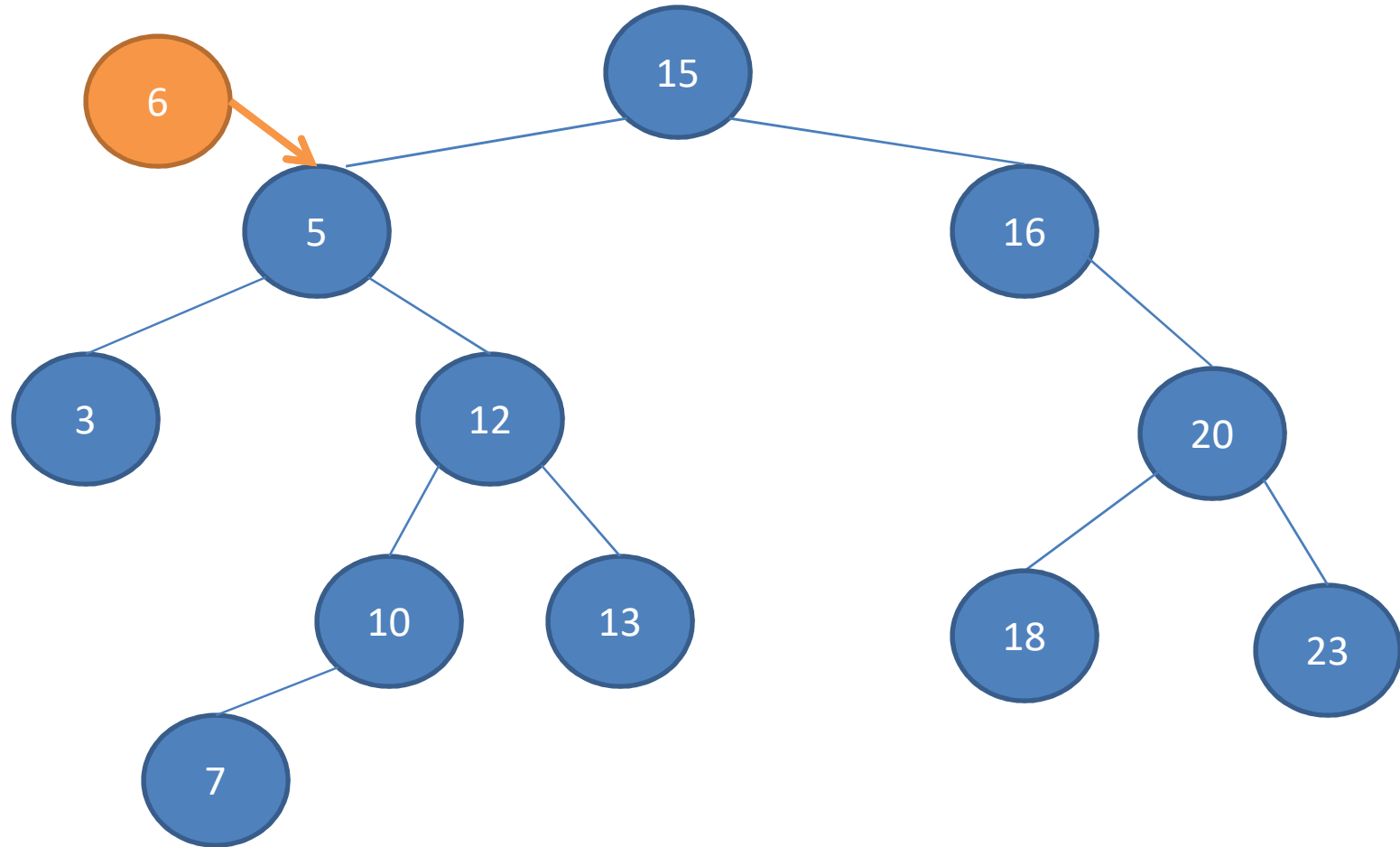
Example: Tree-Delete(T,16)



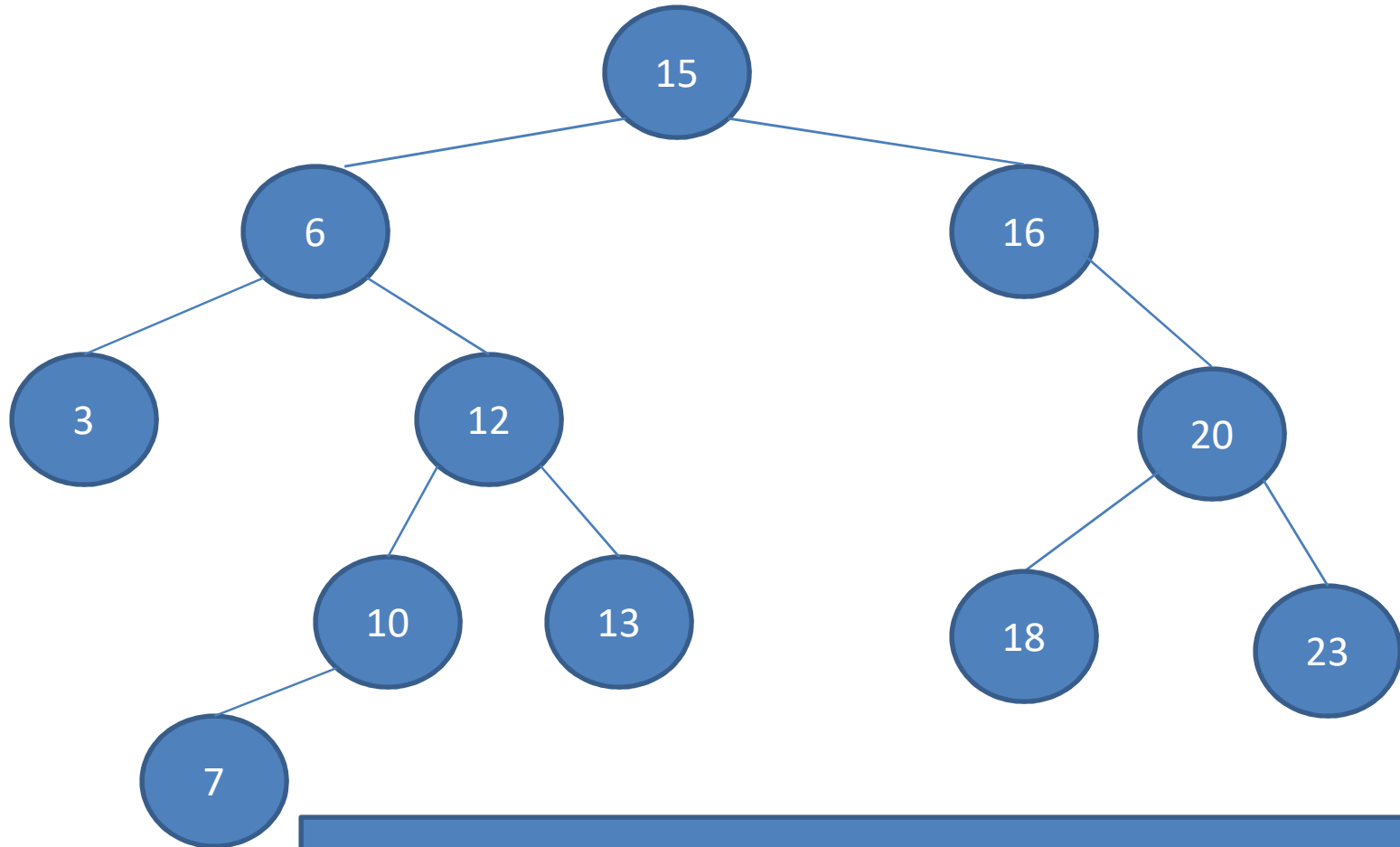
Example: Tree-Delete(T,5)



Example: Tree-Delete(T,5)



Example: Tree-Delete(T,5)



Recursion from a path downward from the root of the tree , so the running time is $\Theta(h)$