Tree-Delete(T,z)

case 1: z has no children

set p[z]'s pointer to NIL instead of z

case 2: z has 1 child

set p[z] to point at z's child

case 3: z has 2 children

recursively delete z's successor y and put y in z's place

if left[z]=NIL or right[z]=NIL then y=z

else y=Tree-Successor(z) /* y is the node that will replace z */

if $left[y] \neq NIL$ then x=left[y]

else x=right[y] /* x is a child of y, if y has any */

if $x \neq NIL$ then p[x]=p[y] /* setting up x's new parent */

if p[y]=NIL then root[T]=x

else if y=left[p[y]] then left[p[y]]=x

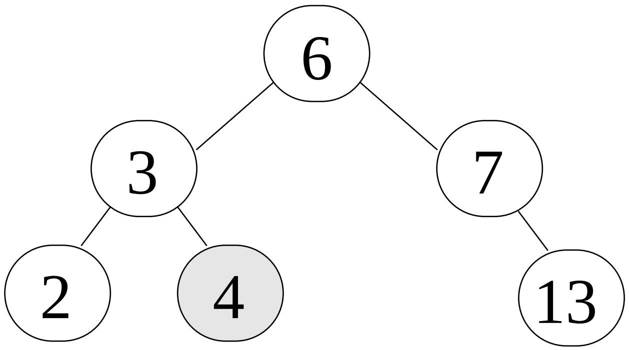
else right[p[y]]=x /* setting x's new parent to point at x */

if $y \neq z$ then key[z]=key[y] /* replacing z with y */

Tree-Delete(T,z)

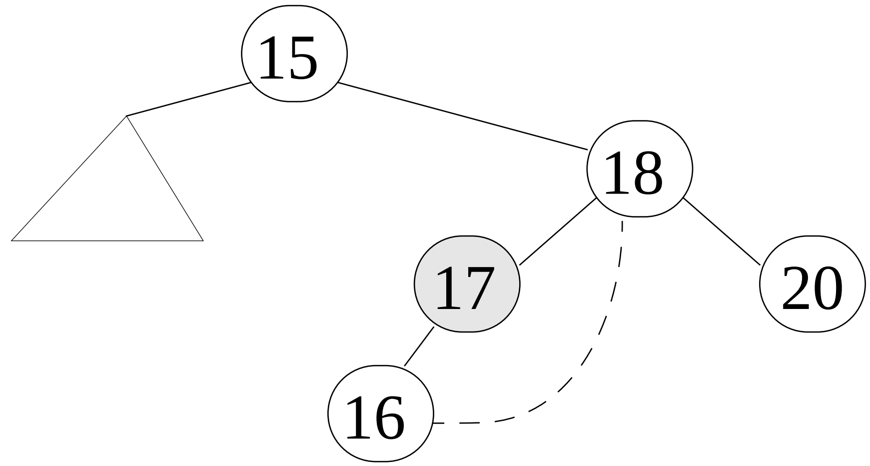
case 1: z has no children

set p[z] to point to NIL



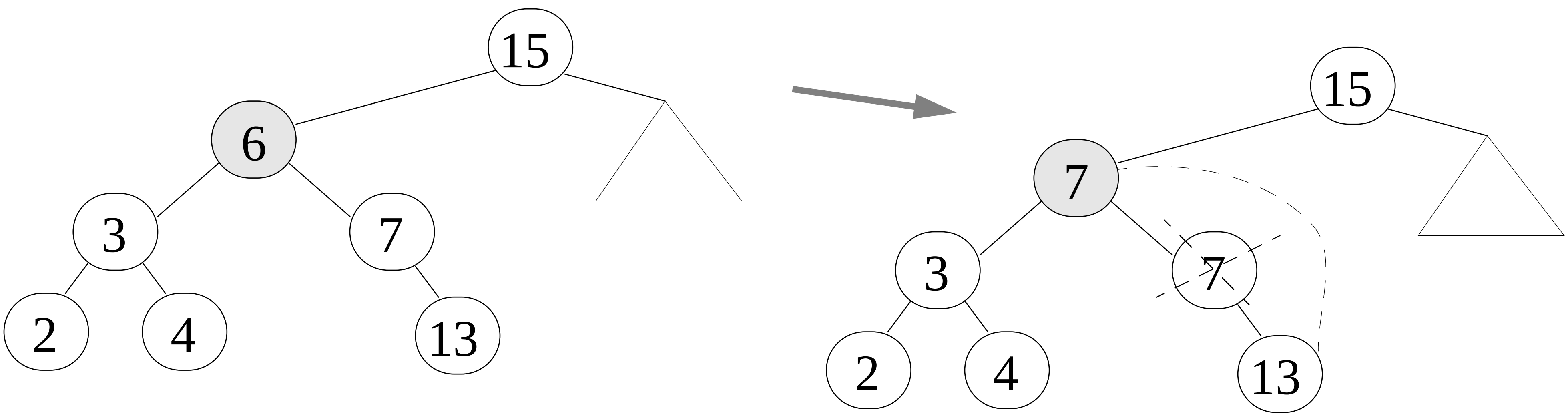

case 2: z has 1 child

set p[z] to point at z's child




case 3: z has 2 children

delete z's successor y and put y in z's place



Tree-Insert(T,z)

y=NIL     x=root(T)

while $x \neq NIL$

y=x

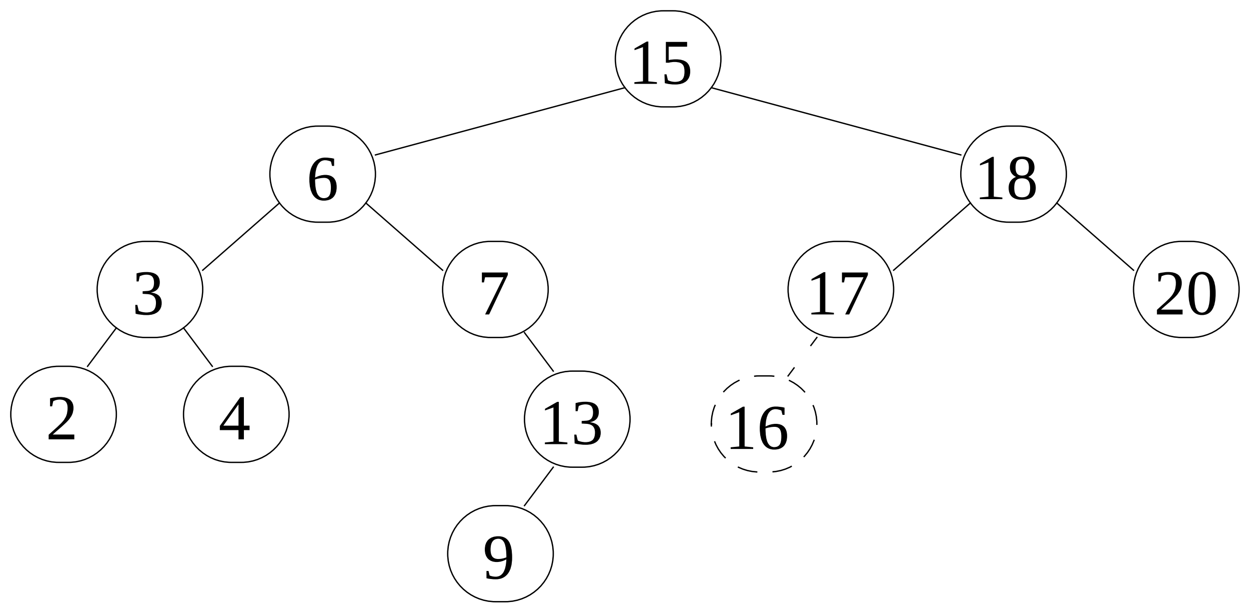
if key[z]<key[x]  then x=left[x]  else x=right[x]

p[z]=y

if y=NIL then root[T]=z

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

else right[y]=z




Running time: O(h)

# Querying a binary search tree

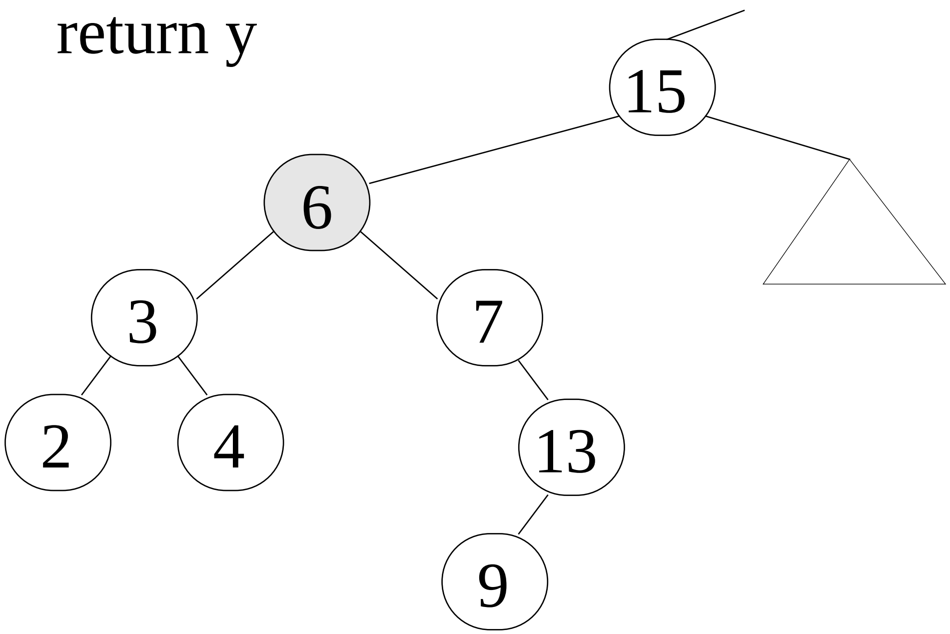
Tree-Successor(x)

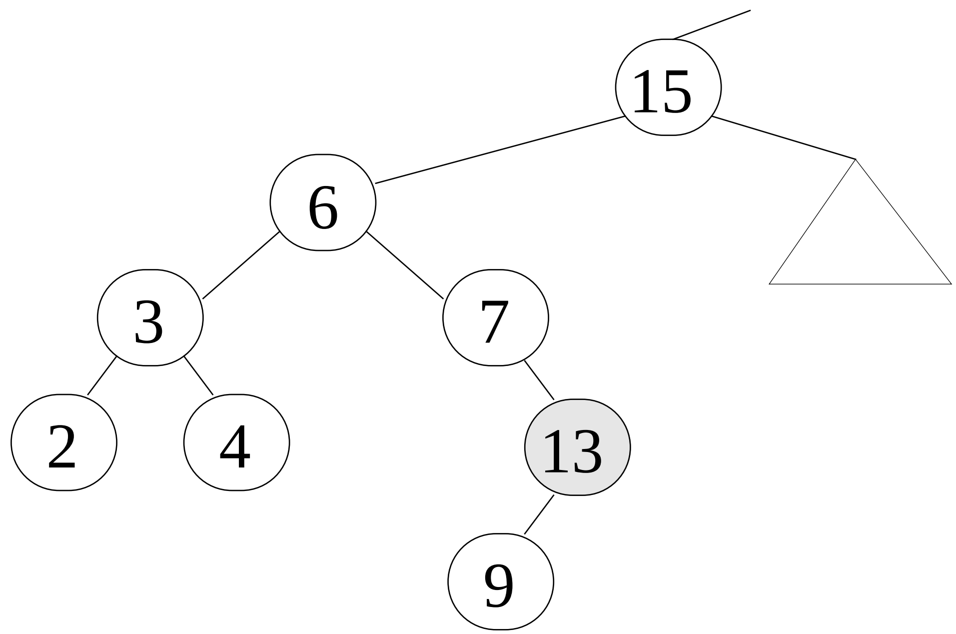
if $right[x] \neq NIL$ then return Tree-Minimum(right[x])

y=p[x]

while $y \neq NIL$ and $x = right[y]$ do x=y; y=p[y]

return y




Running time: either we do Tree-Minimum: O(h) time

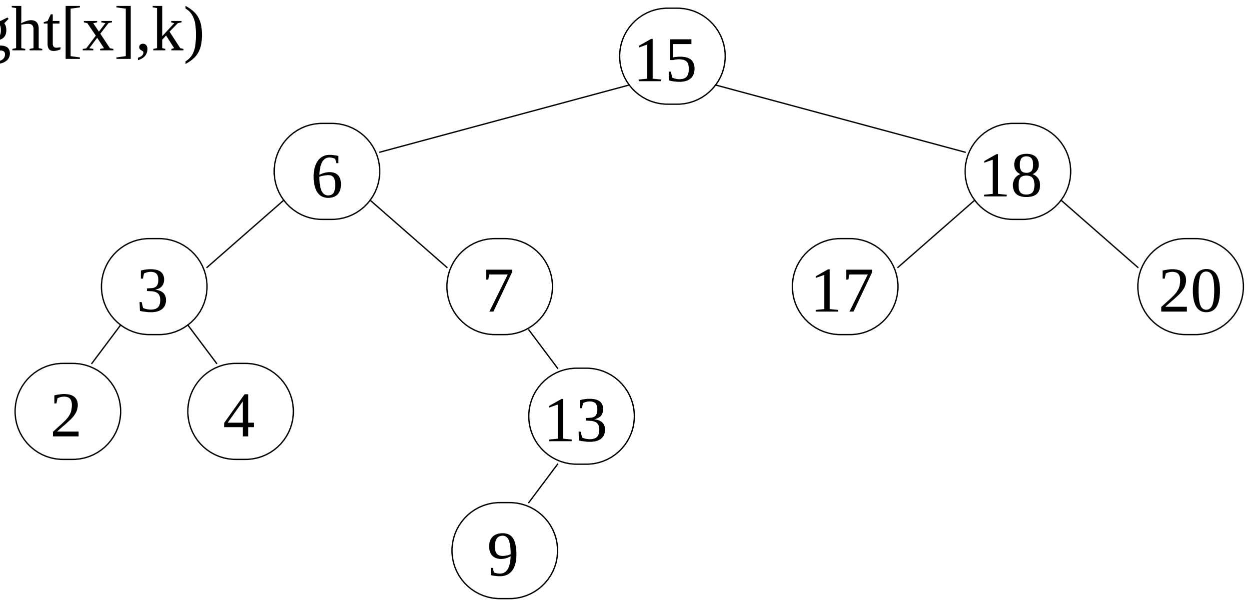
or we traverse a path from node up to (at most) root: O(h) time

T(n) = O(h)

Tree-Predecessor(x) O(h)

# Querying a binary search tree

```
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)
```



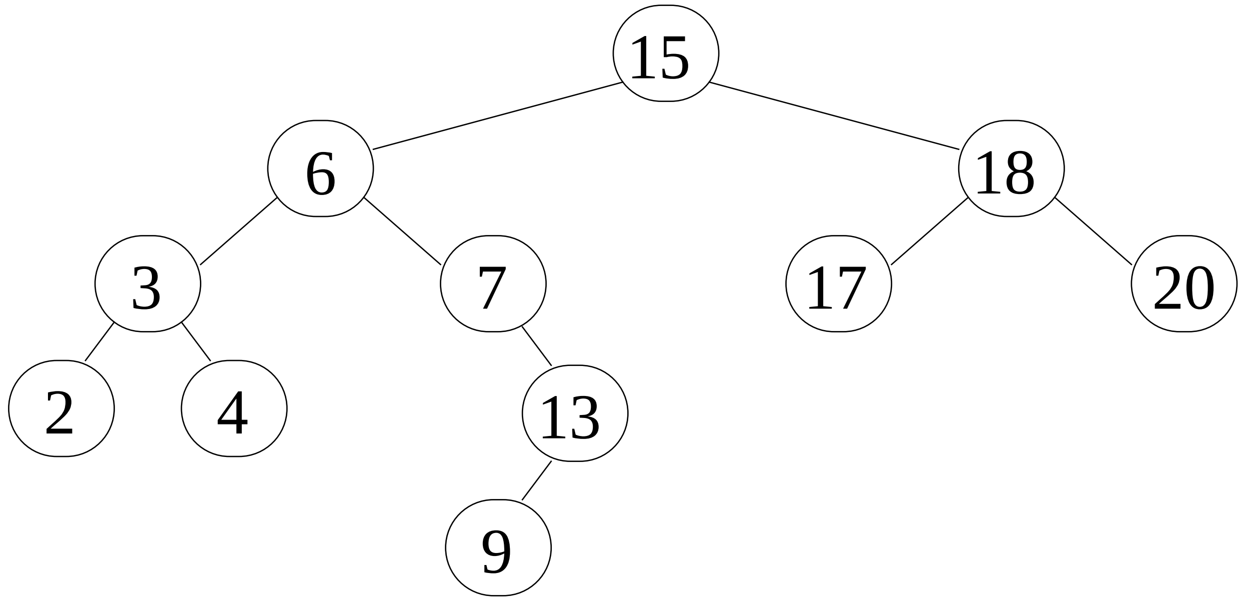

Running time: we traverse a path from the root to (at most) a leaf
at each node we spend $\Theta(1)$ time
let h=height of tree=height of root
$T(n) = O(h)$

Tree-Minimum(x)   $O(h)$

Tree-Maximum(x)   $O(h)$

# Lecture 9: Binary Search Trees

Property: if node y is in the left subtree of node x and node z is in the right subtree of x then $key[y] \le key[x] \le key[z]$




Inorder-Tree-Walk(x)
  if $x \ne NIL$ then
    Inorder-Tree-Walk(left[x])
    print key[x]
    Inorder-Tree-Walk(right[x])

Running time: T(n) = T(q)+Θ(1)+T(n-q) = Θ(n)
  or observe that at each node we spend Θ(1) time, for a total of Θ(n)

Inorder-Tree-Walk(root) prints out all the tree's keys in a sorted order