# AVL Trees

COS 326

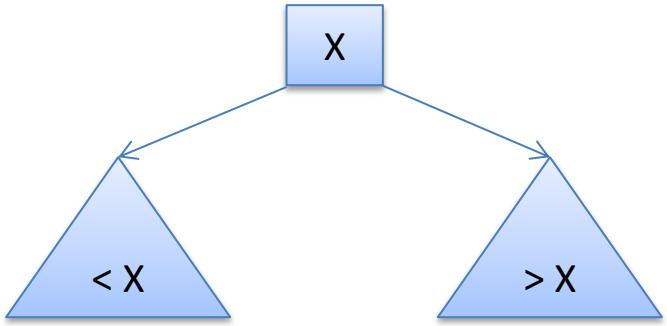
Assignment #6

Princeton University

# AVL Trees

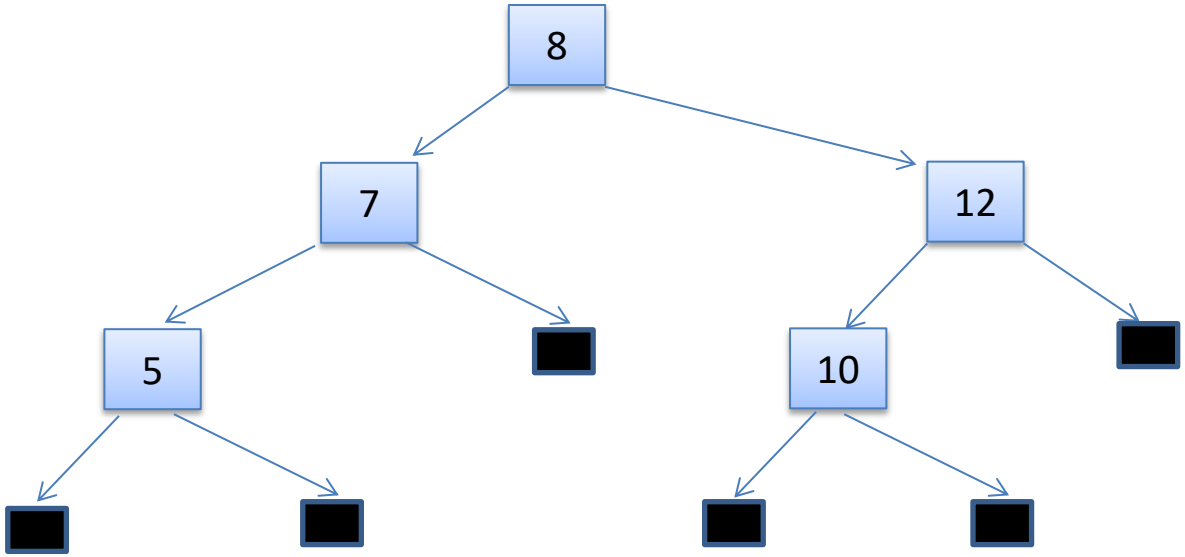
Leaf:

Node:

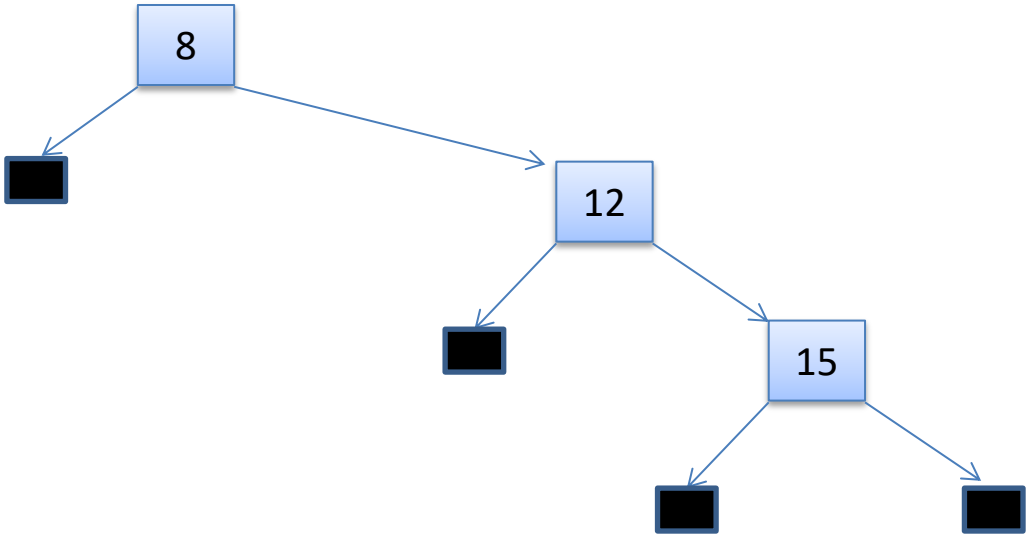



The difference between the height of both subtrees must be at most one. The height of a subtree is the **longest** path from the root to a Leaf.

# AVL Tree Example

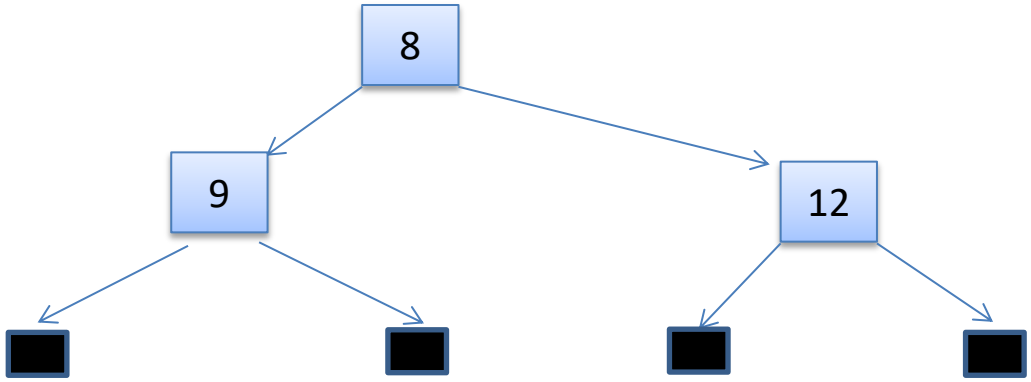


# AVL Tree *Non*-Example




Subtree height difference is too great at node 8!

# AVL Tree *Non-*Example

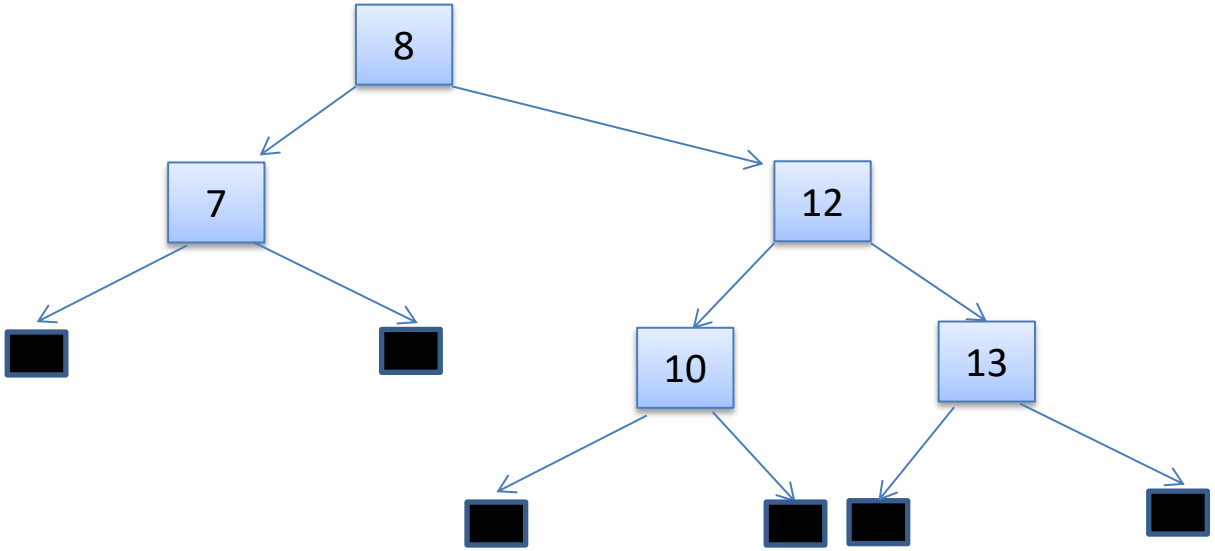



Out of order keys!

# INSERT

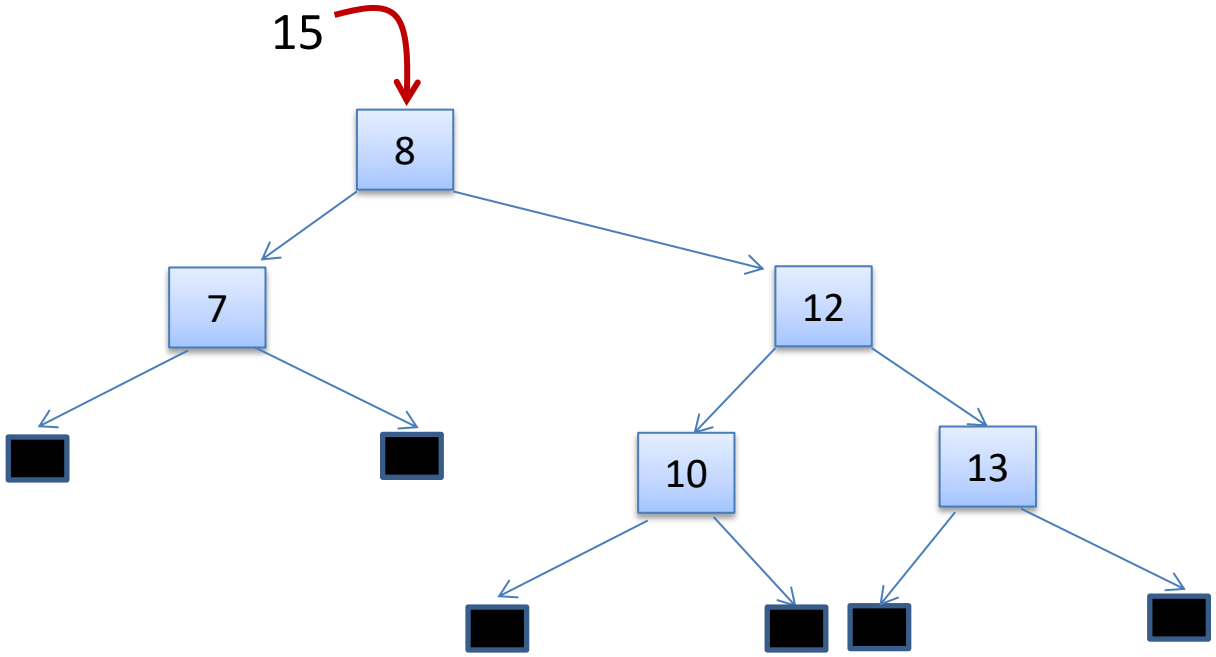
# How to Insert

insert 15 into:



# How to Insert

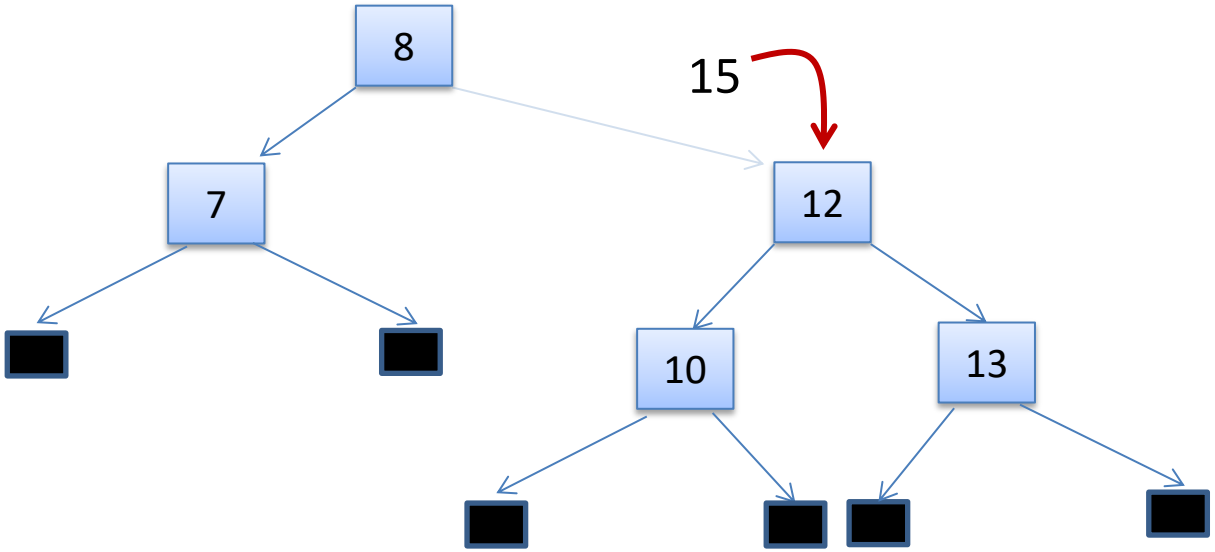
insert 15 into:




Compare 15 to the root node

# How to Insert

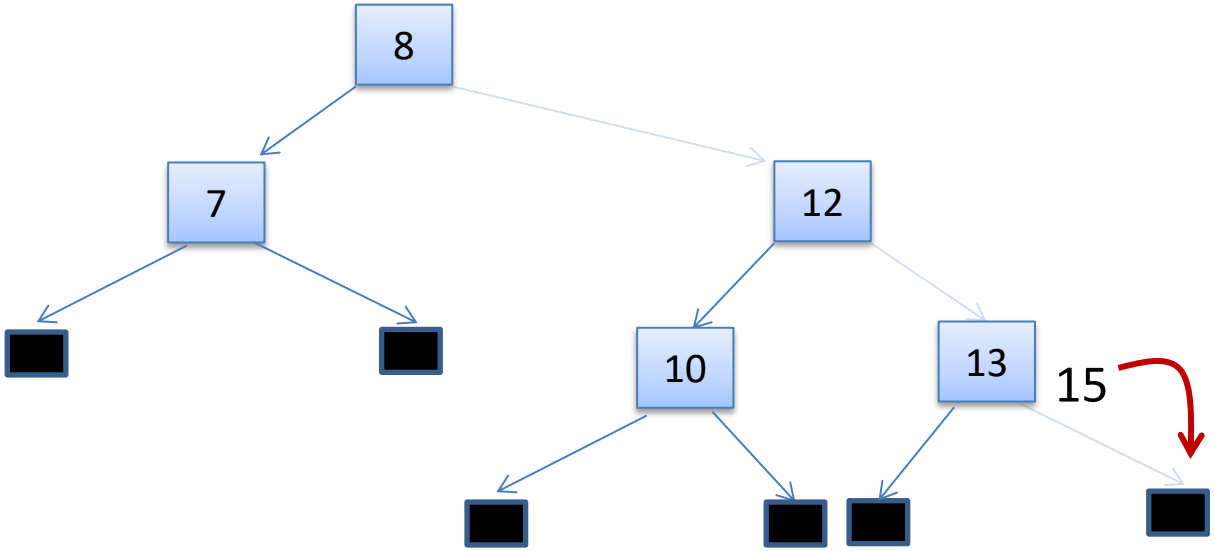
insert 15 into:




Recursively insert into the right subtree

# How to Insert

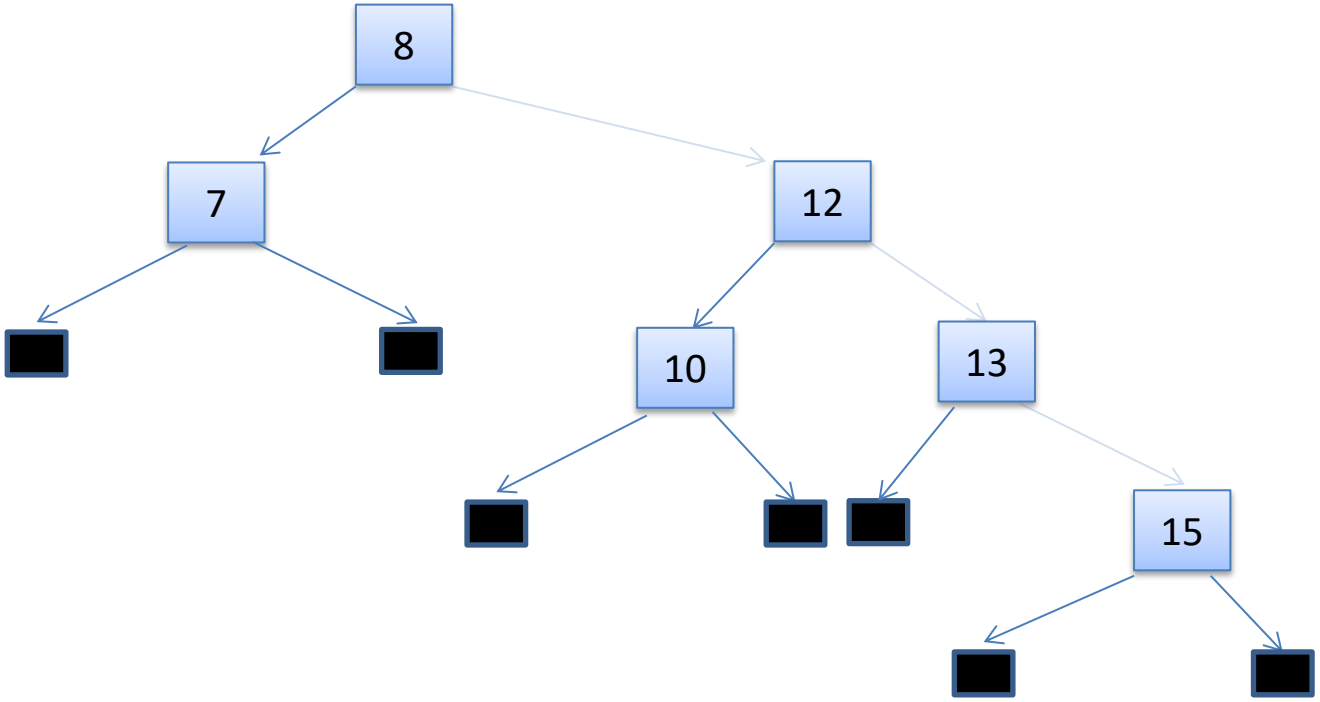
insert 15 into:




Reach a leaf node.

# How to Insert

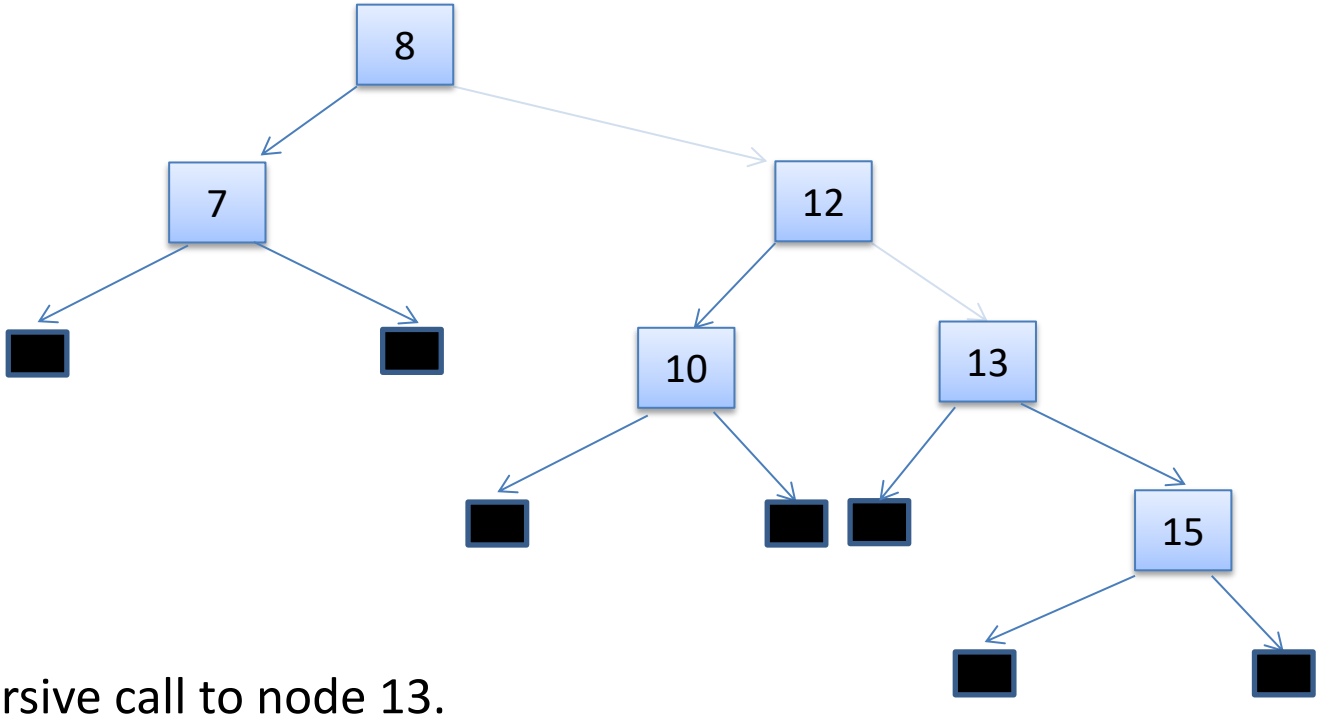
insert 15 into:




Create a new subtree with 15

# How to Insert

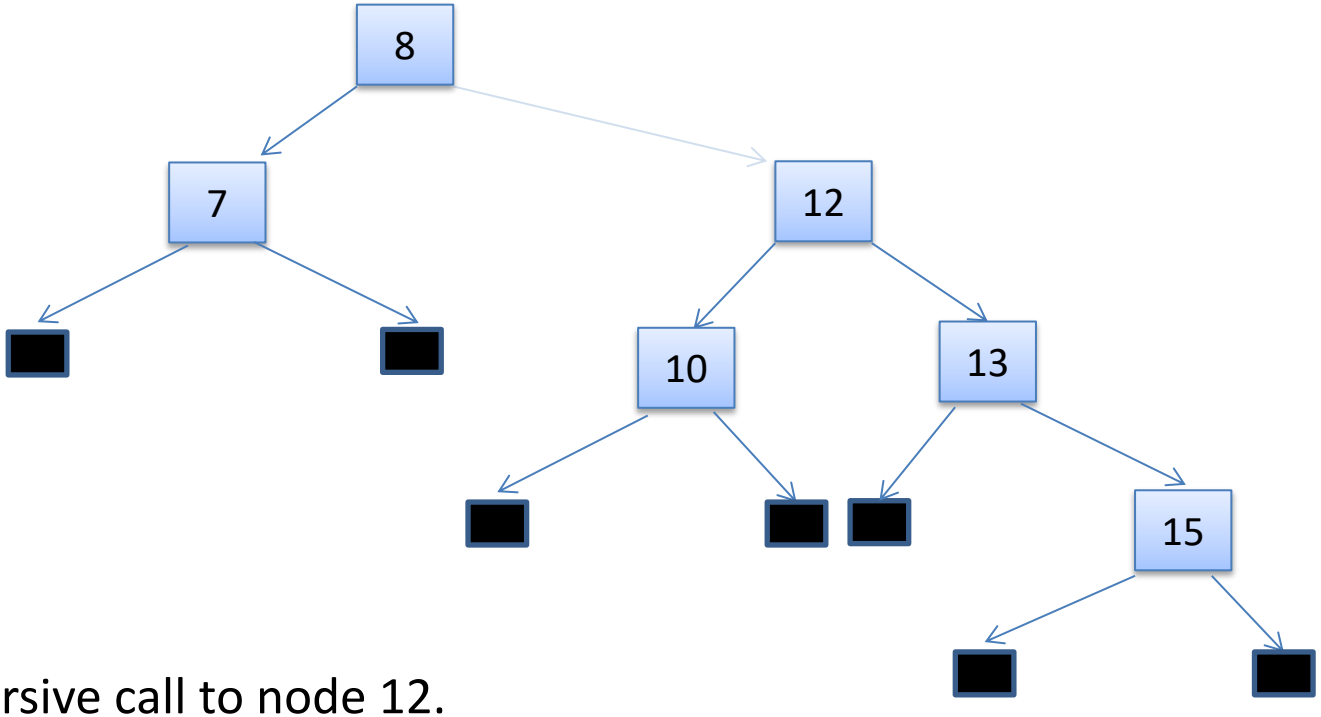
insert 15 into:




Return from recursive call to node 13.
Note that no action is needed as the height of the nodes subtrees are 1 and 0, 1 apart.

# How to Insert

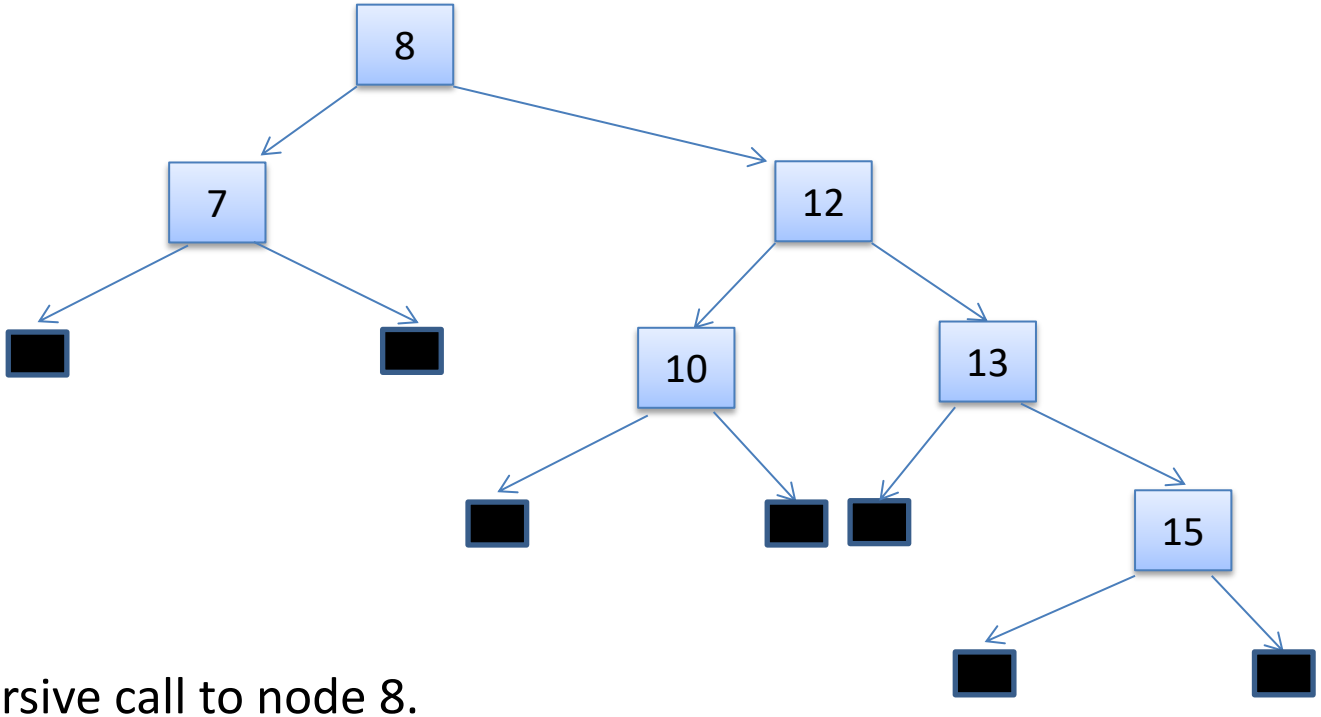
insert 15 into:




Return from recursive call to node 12.
Note that no action is needed as the height of the nodes subtrees are 2 and 1, 1 apart.

# How to Insert

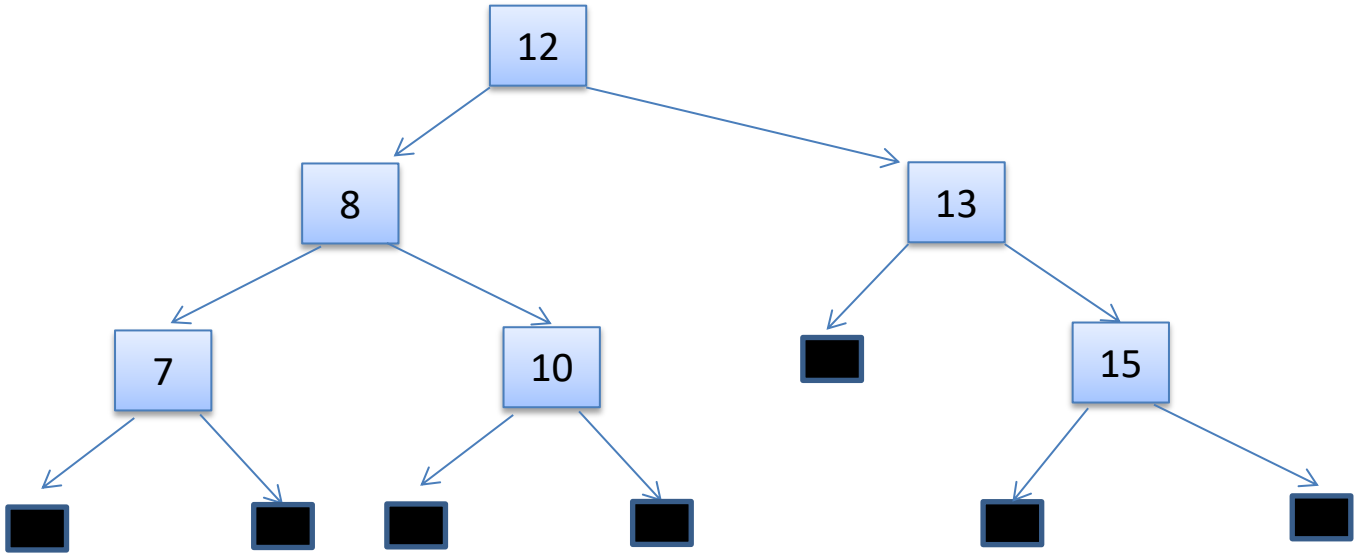
insert 15 into:




Return from recursive call to node 8.
The heights of the subtrees is now 3 and 1.
Action is needed to rebalance the tree!

# How to Insert

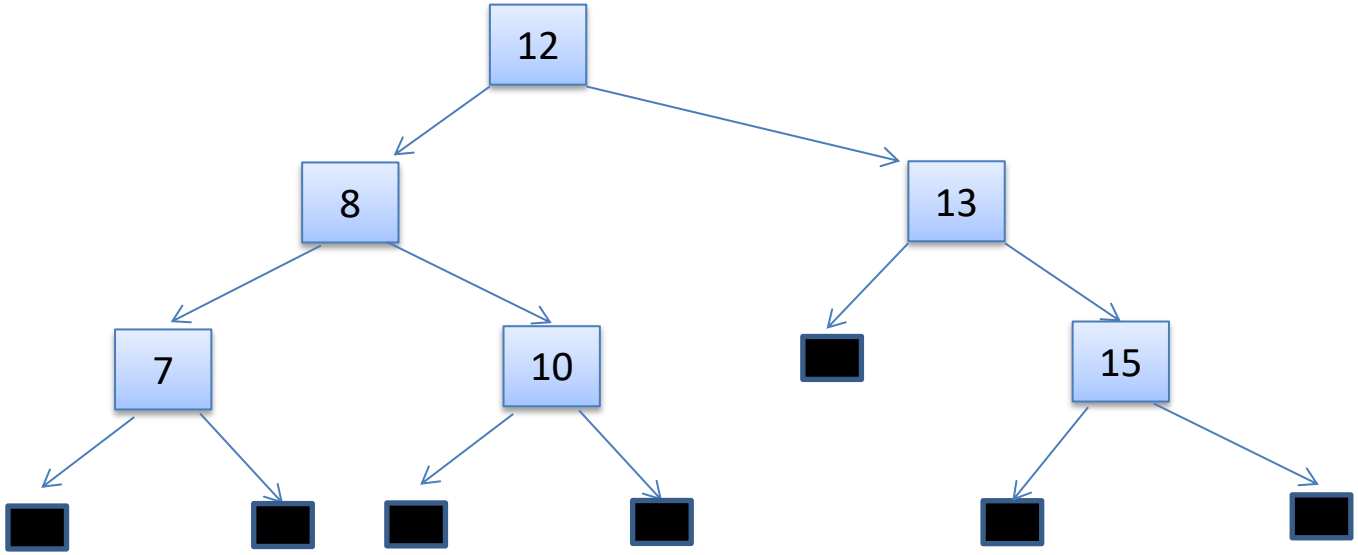
insert 15 into:




Solution: Rotate the tree to the left! 12 becomes the new root and 8 becomes its left subtree. The tree is now balanced!

# How to Insert

insert 15 into:

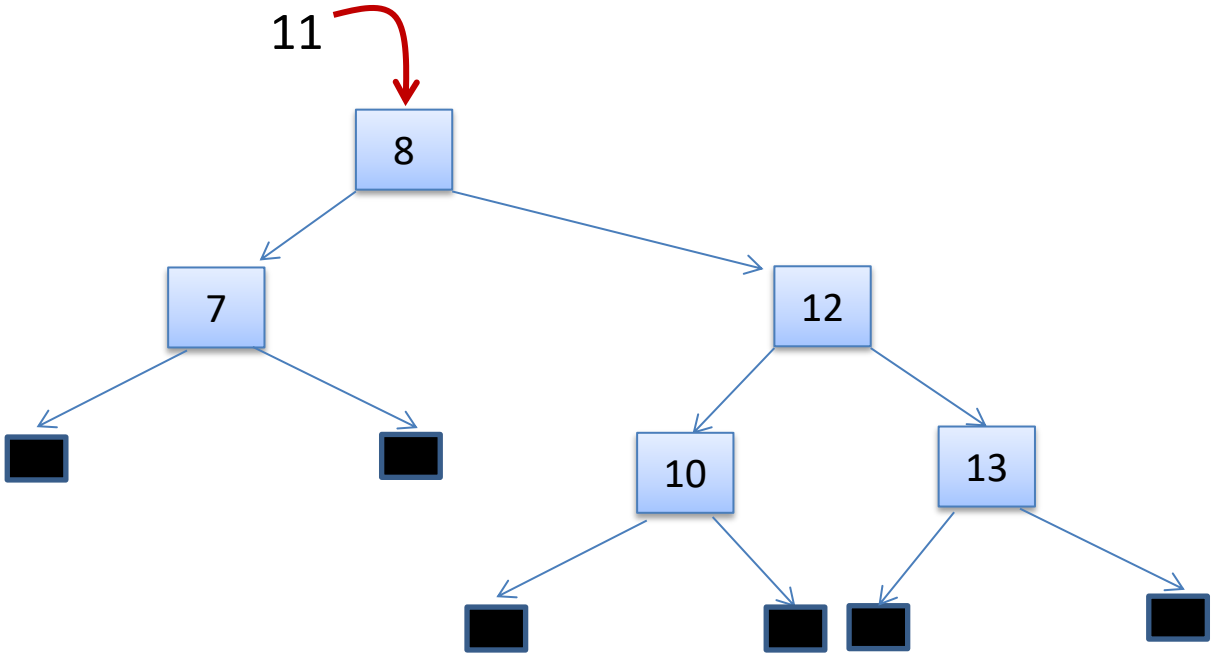



We are done!

Key idea: When returning from a call to insert, compare the heights of the subtrees, rotate left or right to rebalance.

# How to Insert

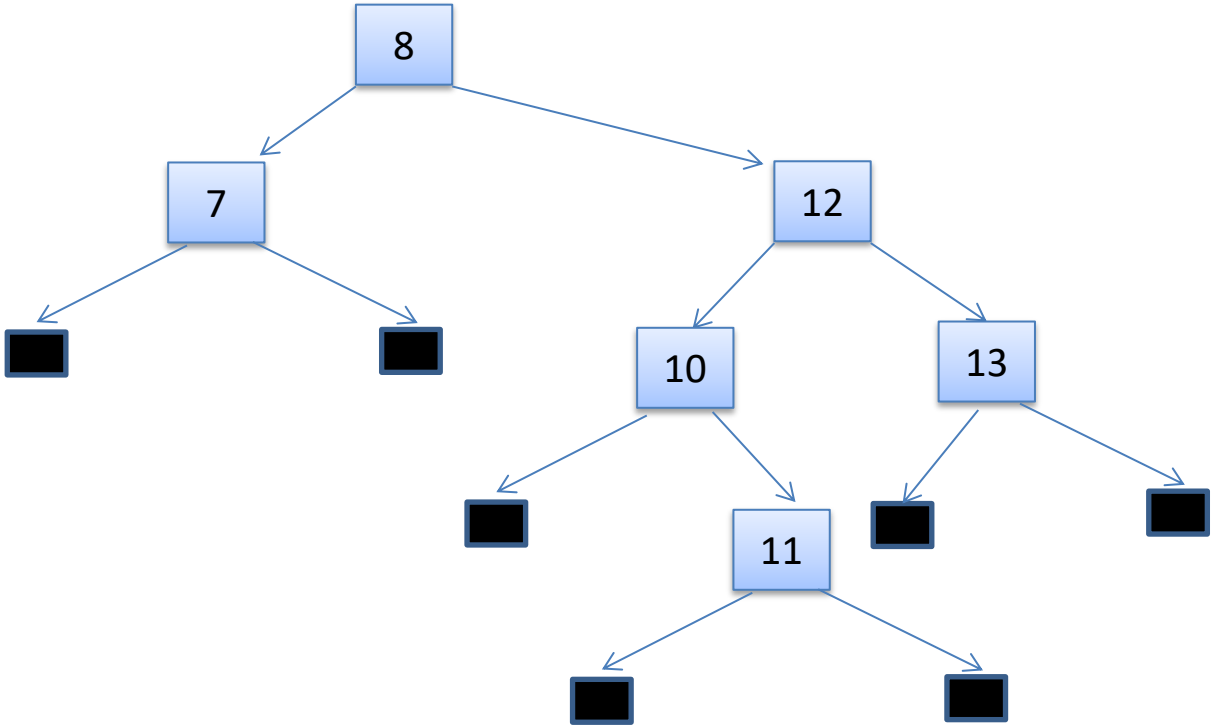
insert 11 into:




What happens when 11 is inserted?

# How to Insert

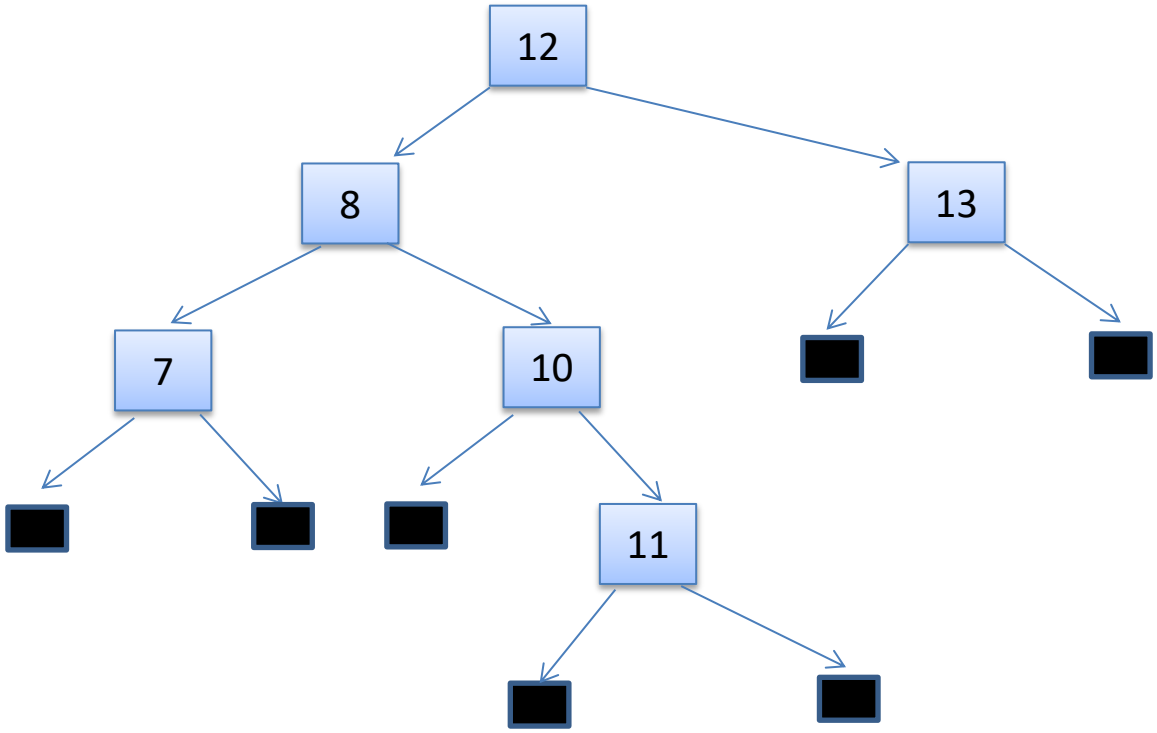
insert 11 into:




What happens when 11 is inserted?

# How to Insert

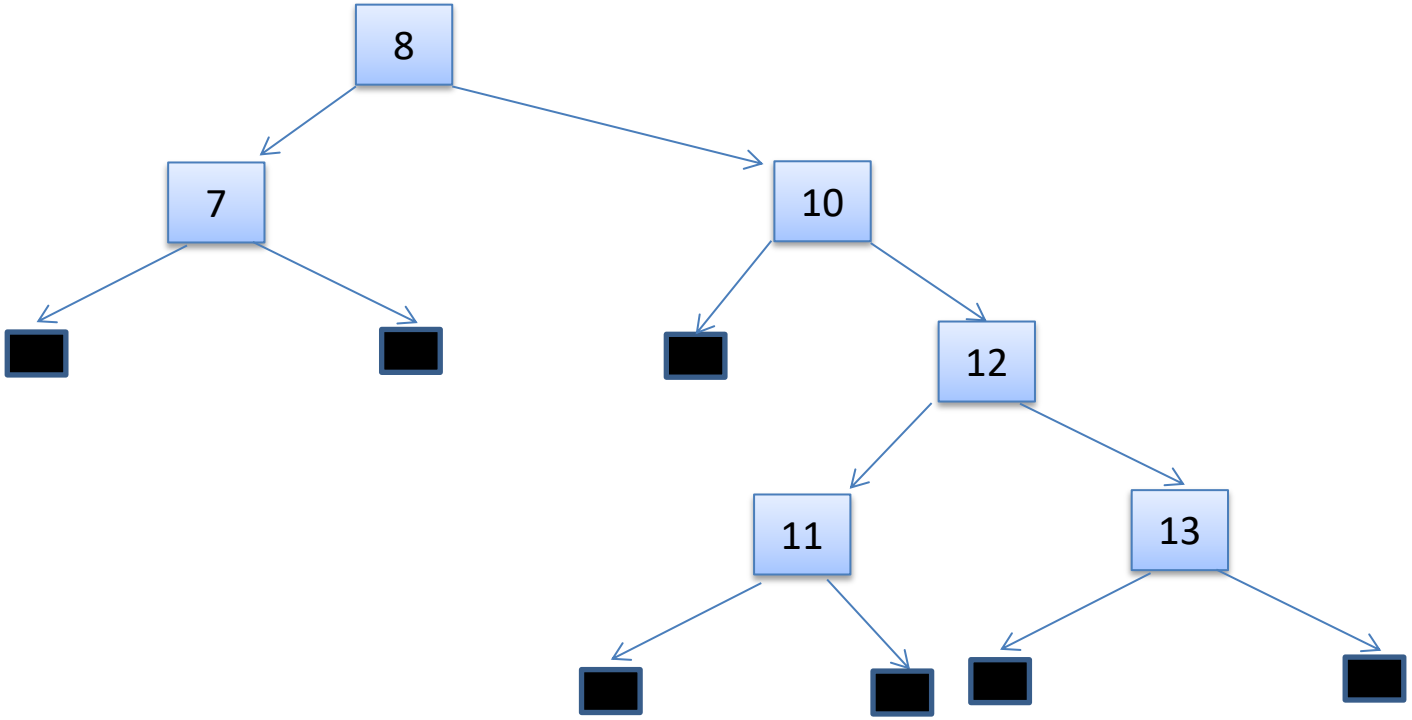
insert 11 into:




Rotate Left does not work!
The tree is still unbalanced!

# How to Insert

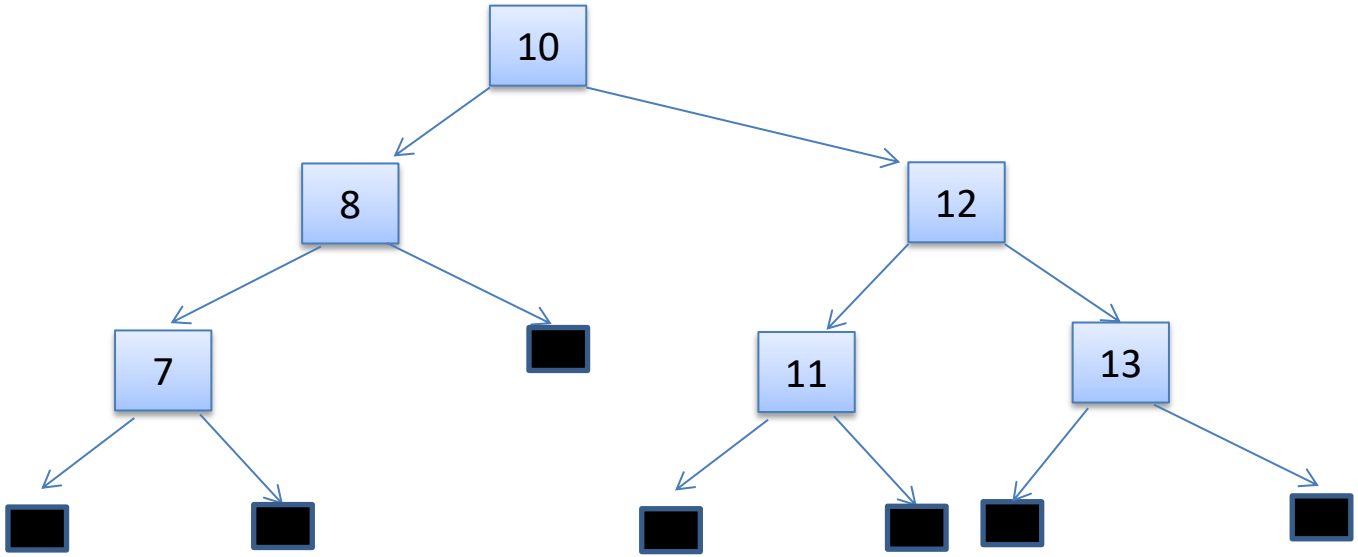
insert 11 into:




Rotate the subtree first in the opposite direction!

# How to Insert

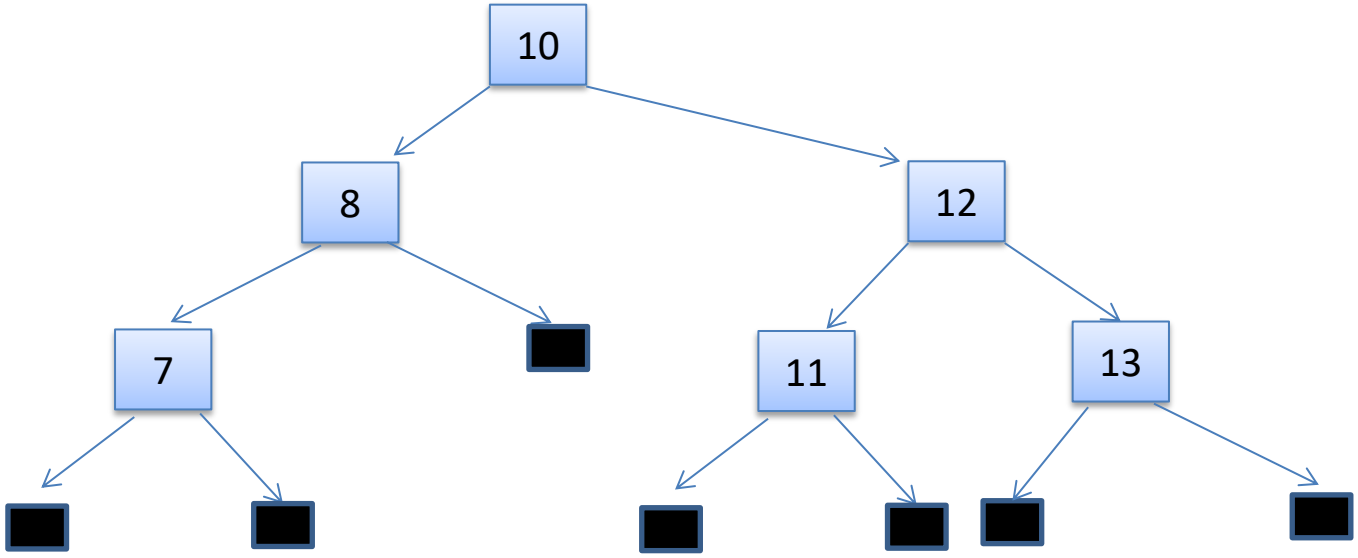
insert 11 into:




Now rotate left!

# How to Insert

insert 11 into:



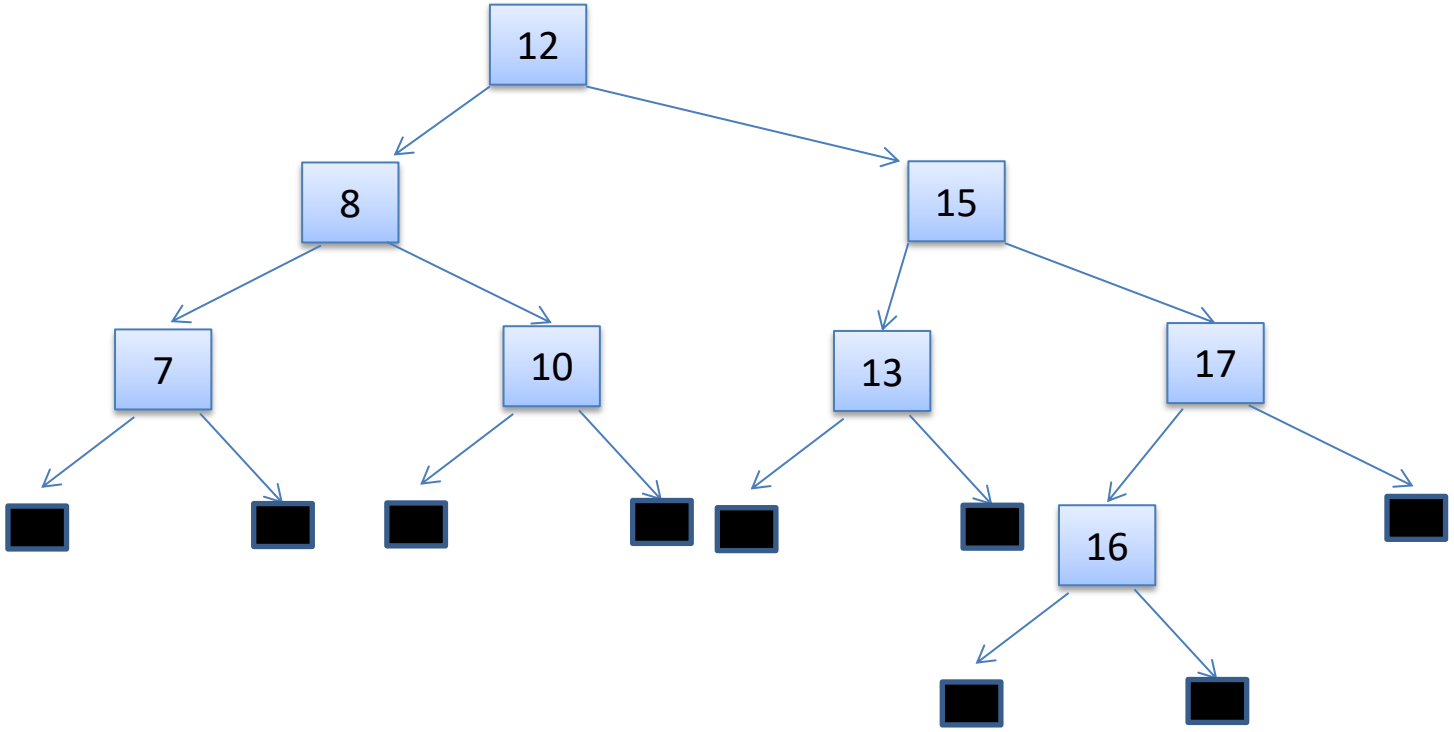

We are done!

Key idea: If we put the node in the left subtree of the right child of the root, or the right subtree of the left child of the root, we must rotate the subtree in the opposite direction as the tree as a whole.

# DELETE

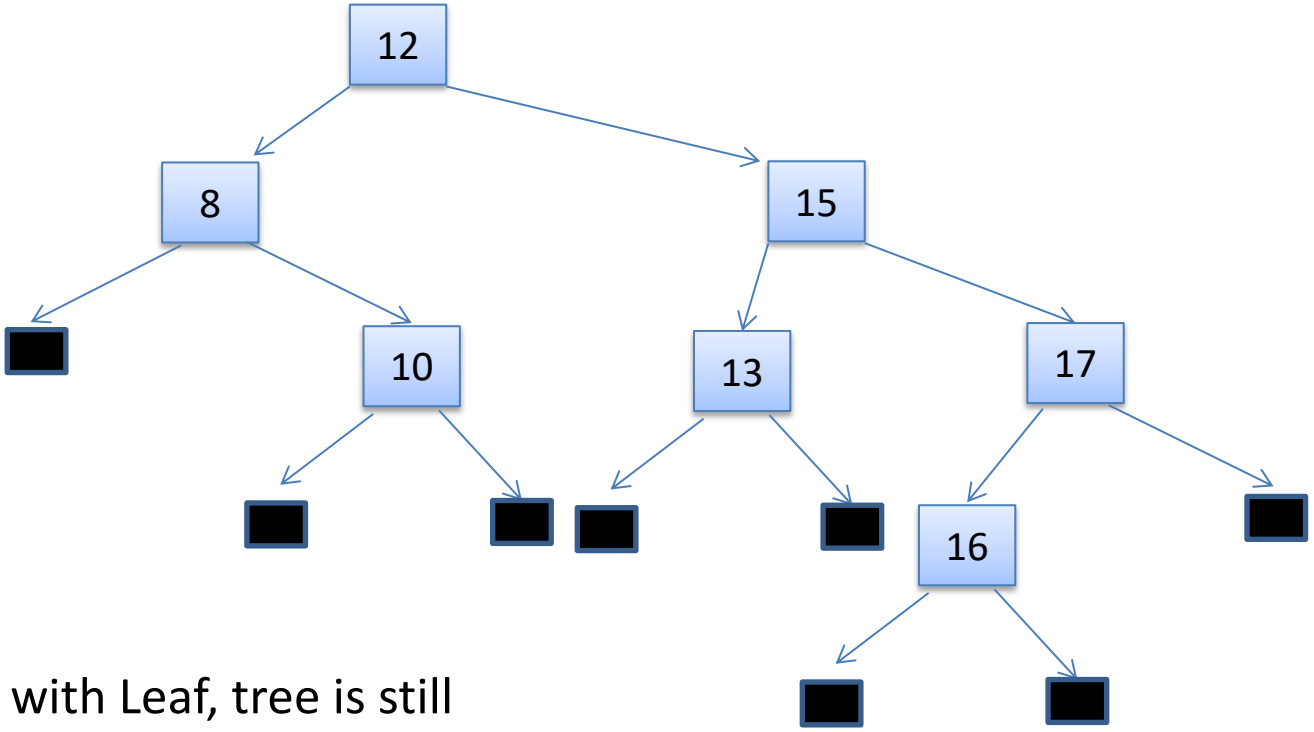
# How to Delete

Delete 7 from:



# How to Delete

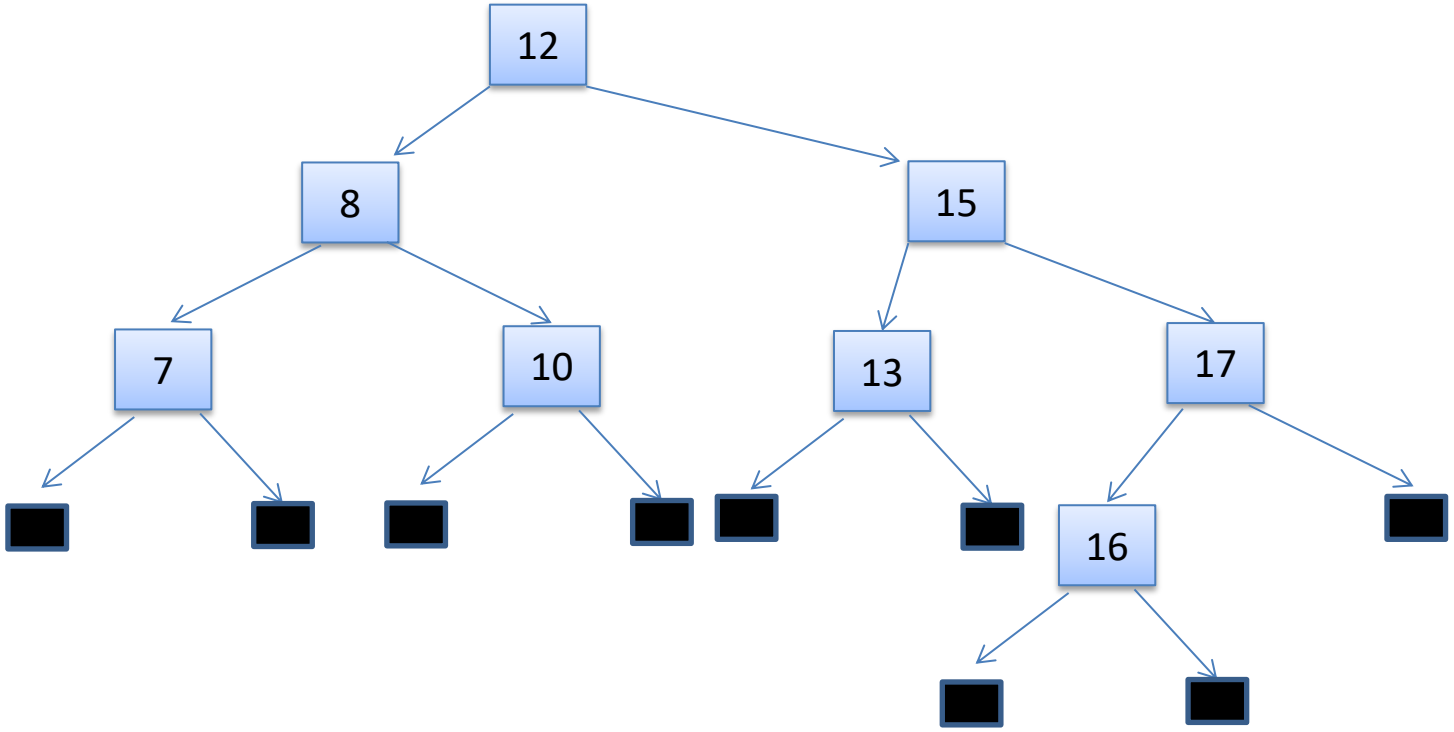
Delete 7 from:




Solution: Replace with Leaf, tree is still balanced.

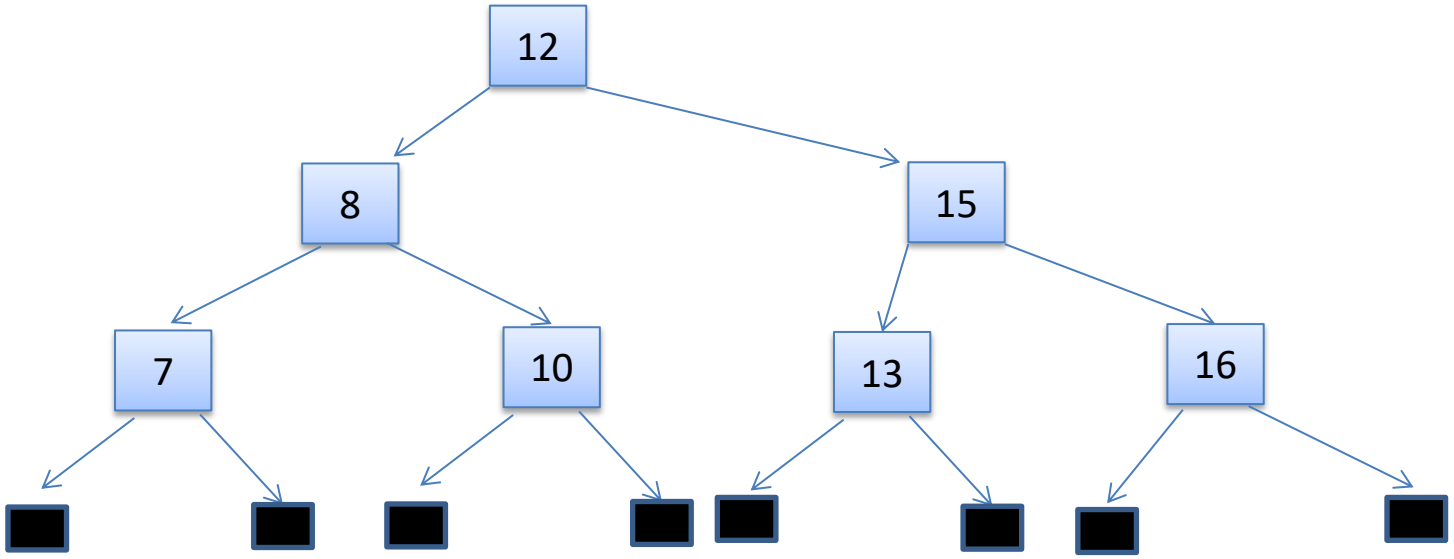
# How to Delete

Delete 17 from:



# How to Delete

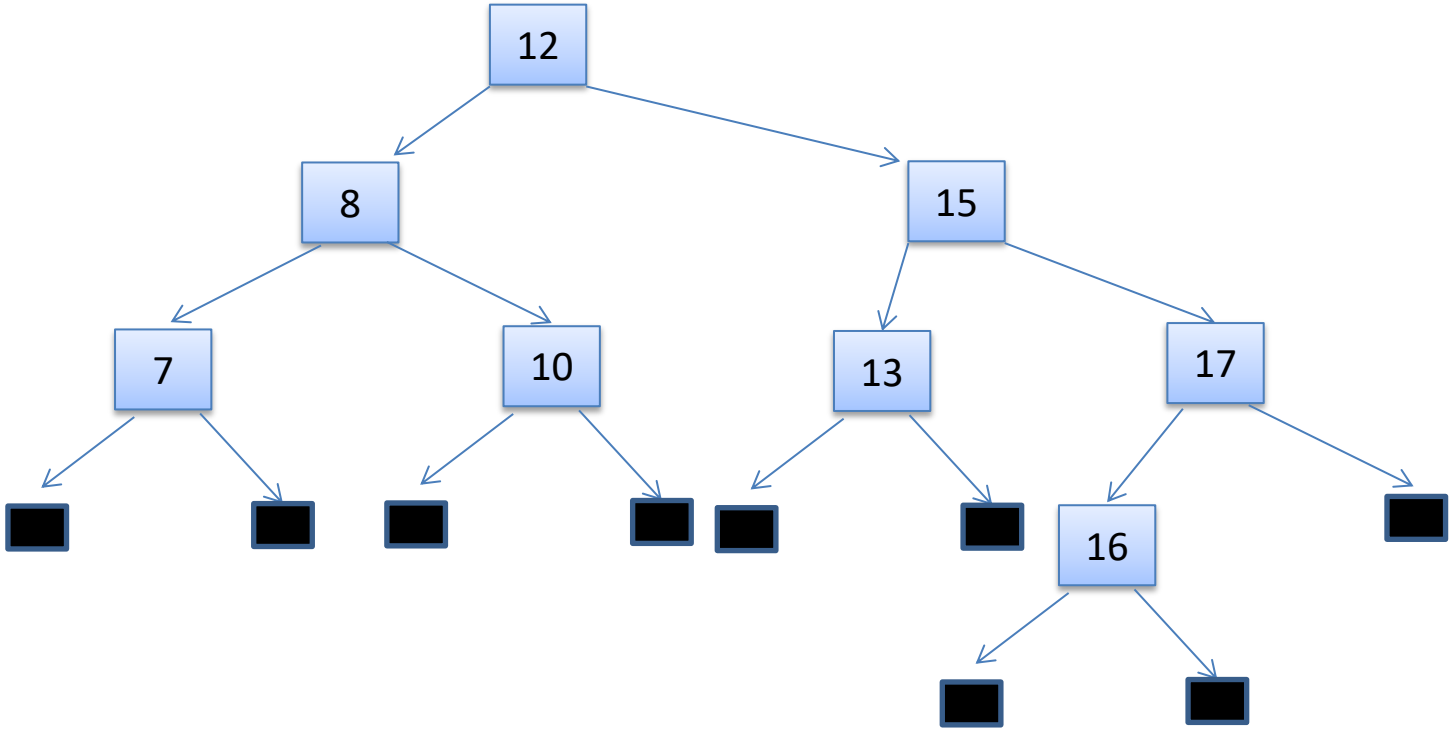
Delete 17 from:




Solution: Replace 17 with its left child, the tree is still balanced.

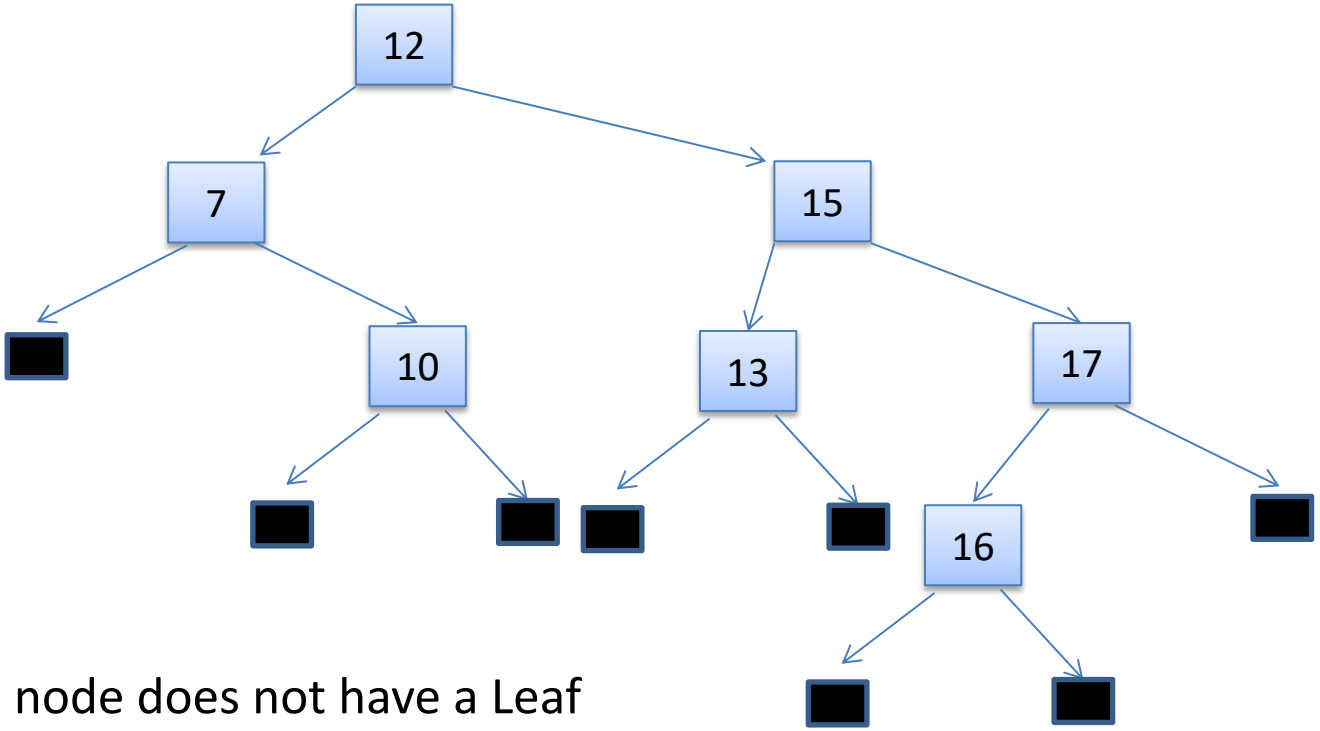
# How to Delete

Delete 8 from:



# How to Delete

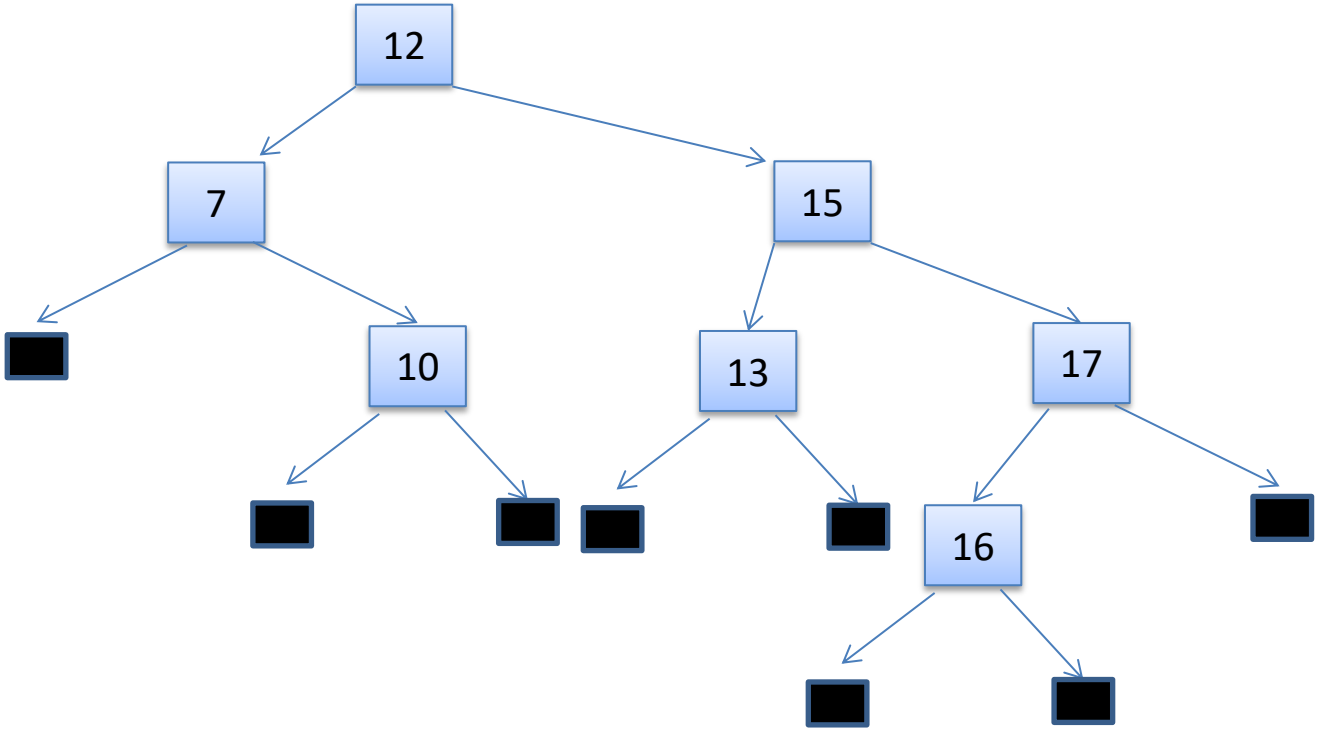
Delete 8 from:




Solution: When a node does not have a Leaf as a child, compare the heights of the two child nodes and replace with the taller child. In this case the two are equal.

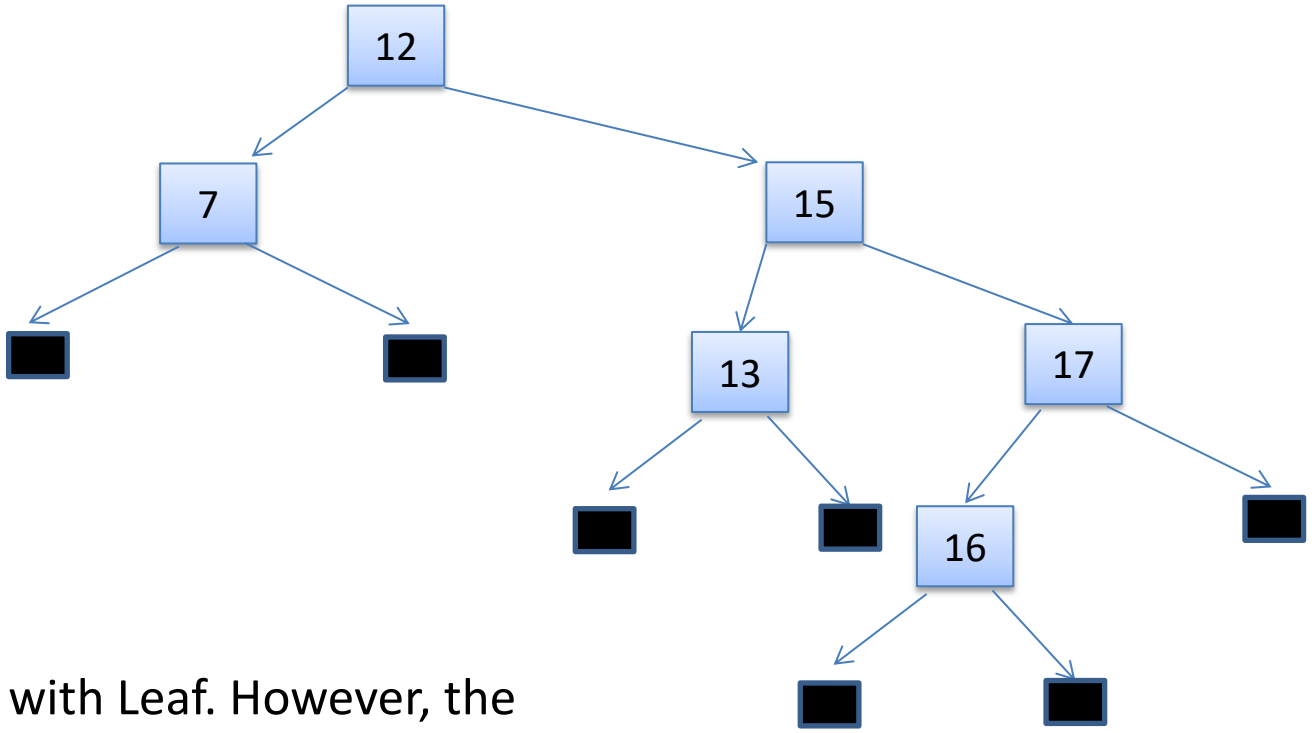
# How to Delete

Now delete 10 from the new tree.
Delete 10 from:



# How to Delete

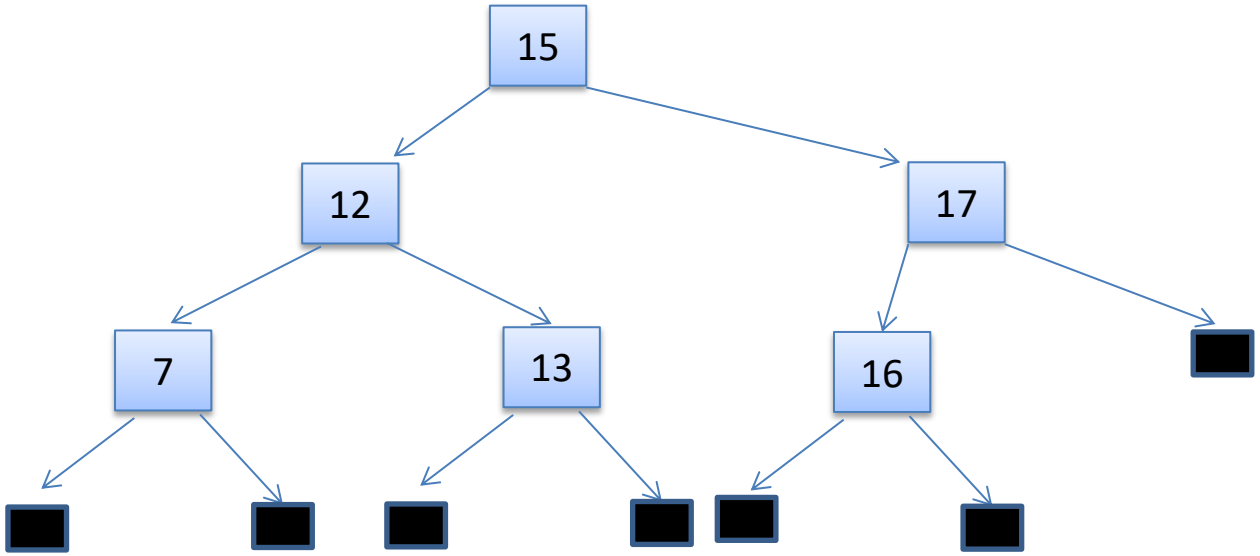
Delete 10 from:




Solution: Replace with Leaf. However, the heights are now unbalanced!

# How to Delete

Delete 17 from:




Solution: Rebalance in the same way as we did for insert, except rotate in the opposite direction as we are now making the tree smaller.

# OCAML IMPLEMENTATION

# OCaml AVL Trees

```
type pair = key * value

type dict =
  | Leaf
  | Node of dict * int * pair * dict
```

Valid AVL trees must be:

- in order
- balanced (subtree height difference less than 2)

You will write an invariant function to check that the trees produced by your functions are valid AVL trees.

This is going to help you debug your routines a lot.

# The OCaml Insert and Remove Functions

insert: dict -> key -> value -> dict

Key Property:

If d is a valid AVL tree and insert_to_tree d k v = d’ then

- d’ is a valid AVL tree
- d’ contains all of the elements of d as well as (k,v)

remove: dict -> key -> dict

Key Property:

If d is a valid AVL tree and remove_from_tree d k = d' then

- d' is a valid AVL tree
- d' contains all of the elements of d except the one for k