CSCI 241

Lecture 11

Balanced Binary Search Trees

Announcements

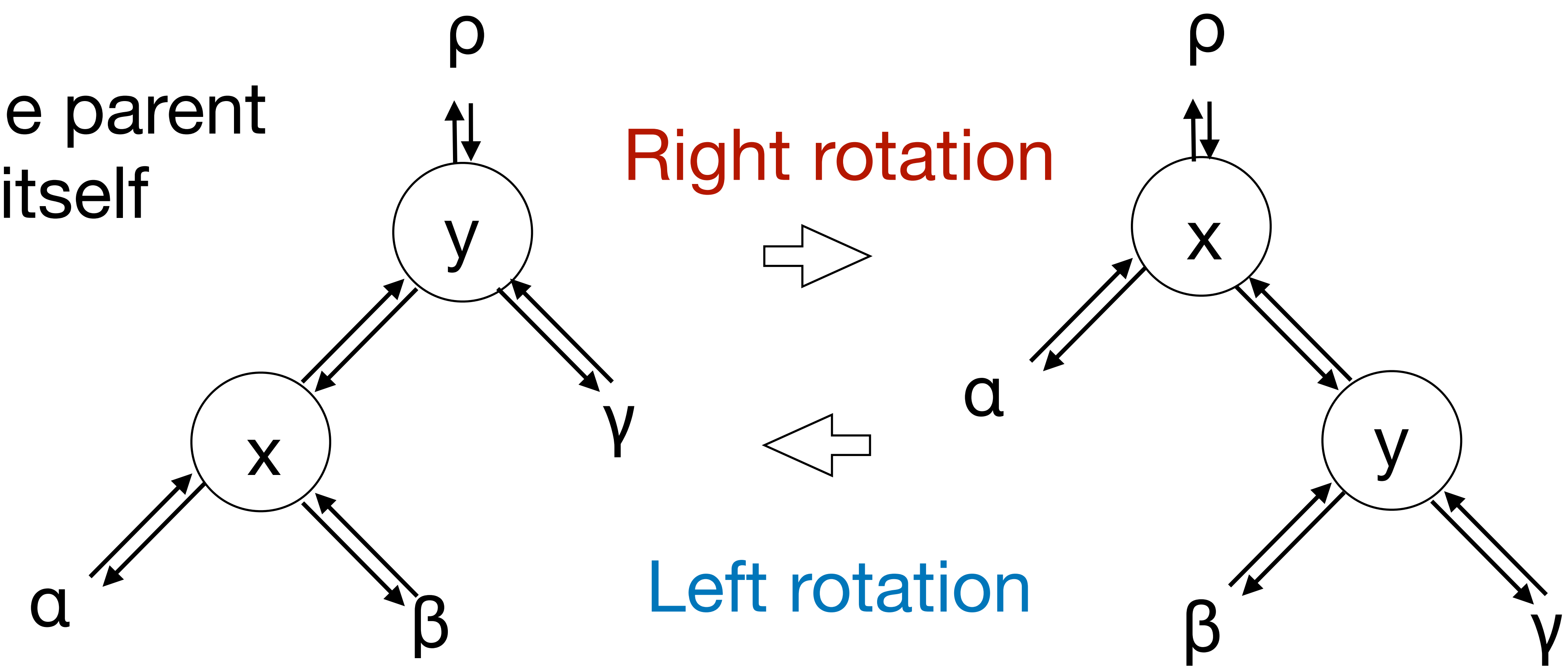
Goals

- Be prepared to implement **rotations** in BSTs
- Be prepared to implement AVL rebalancing.

Tree Rotations

Steps in left rotation (move y up to x's position):

1. Transfer β
2. Transfer the parent
3. Transfer x itself



Right rotation

Left rotation

x.R gets y.L
y.L.p gets x

y.p gets x.p
p.[L/R] gets y

y.L gets x
x.p gets y

Warm-up

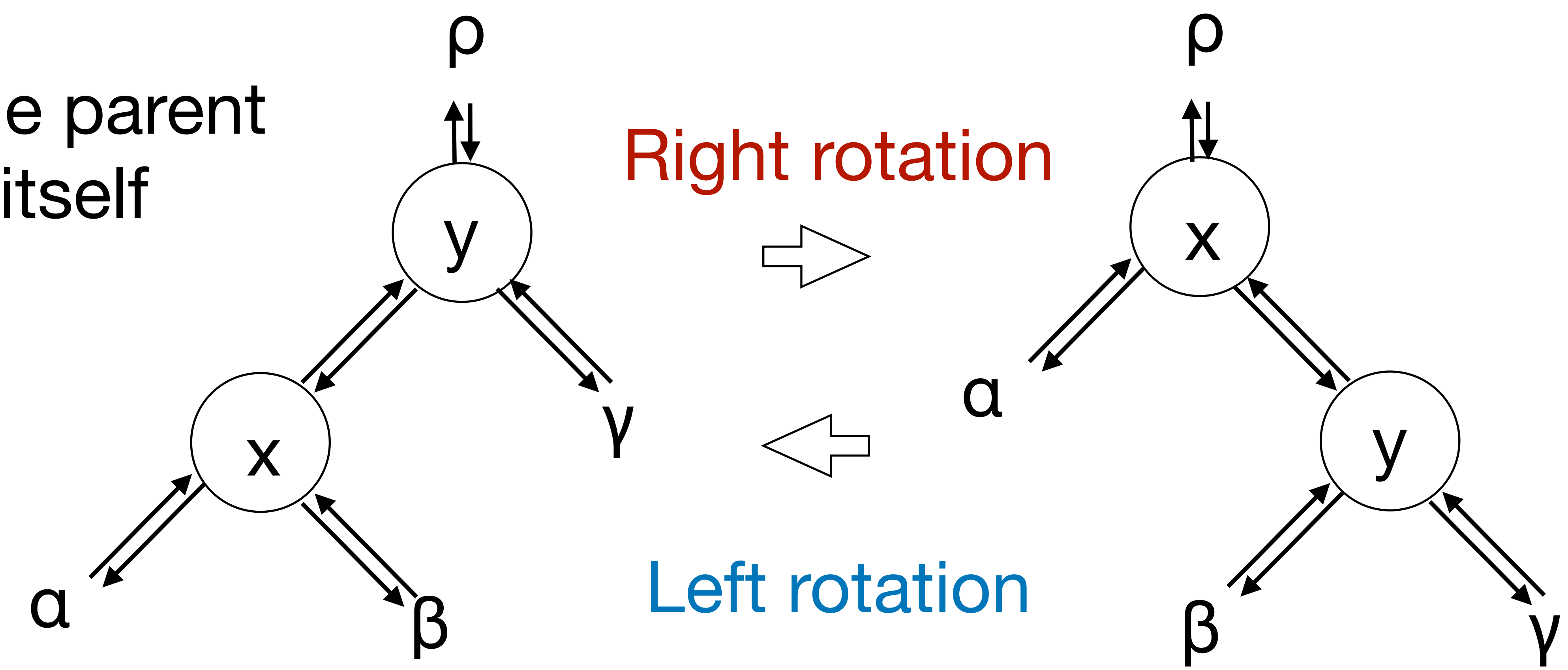
Steps in left rotation (move y up to x's position):

1. Transfer β
2. Transfer the parent
3. Transfer x itself

x.R gets y.L
y.L.p gets x

y.p gets x.p
p.[L/R] gets y

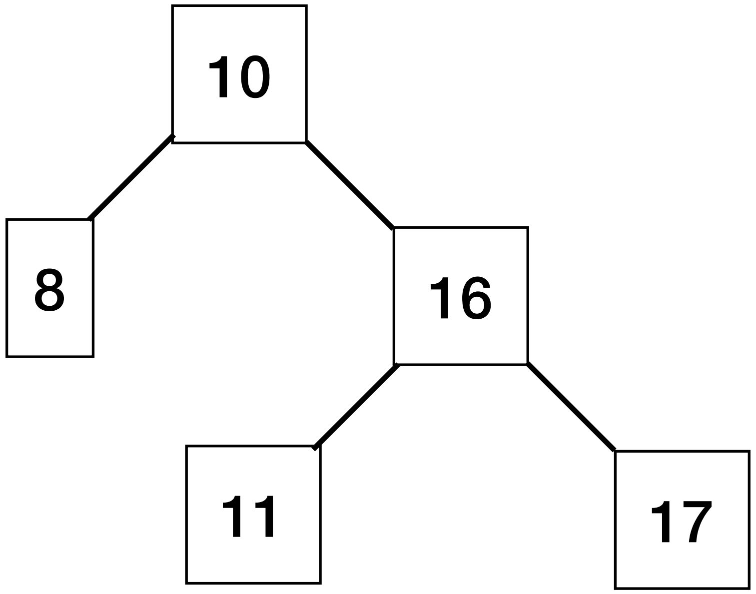
y.L gets x
x.p gets y



Right rotation

Left rotation

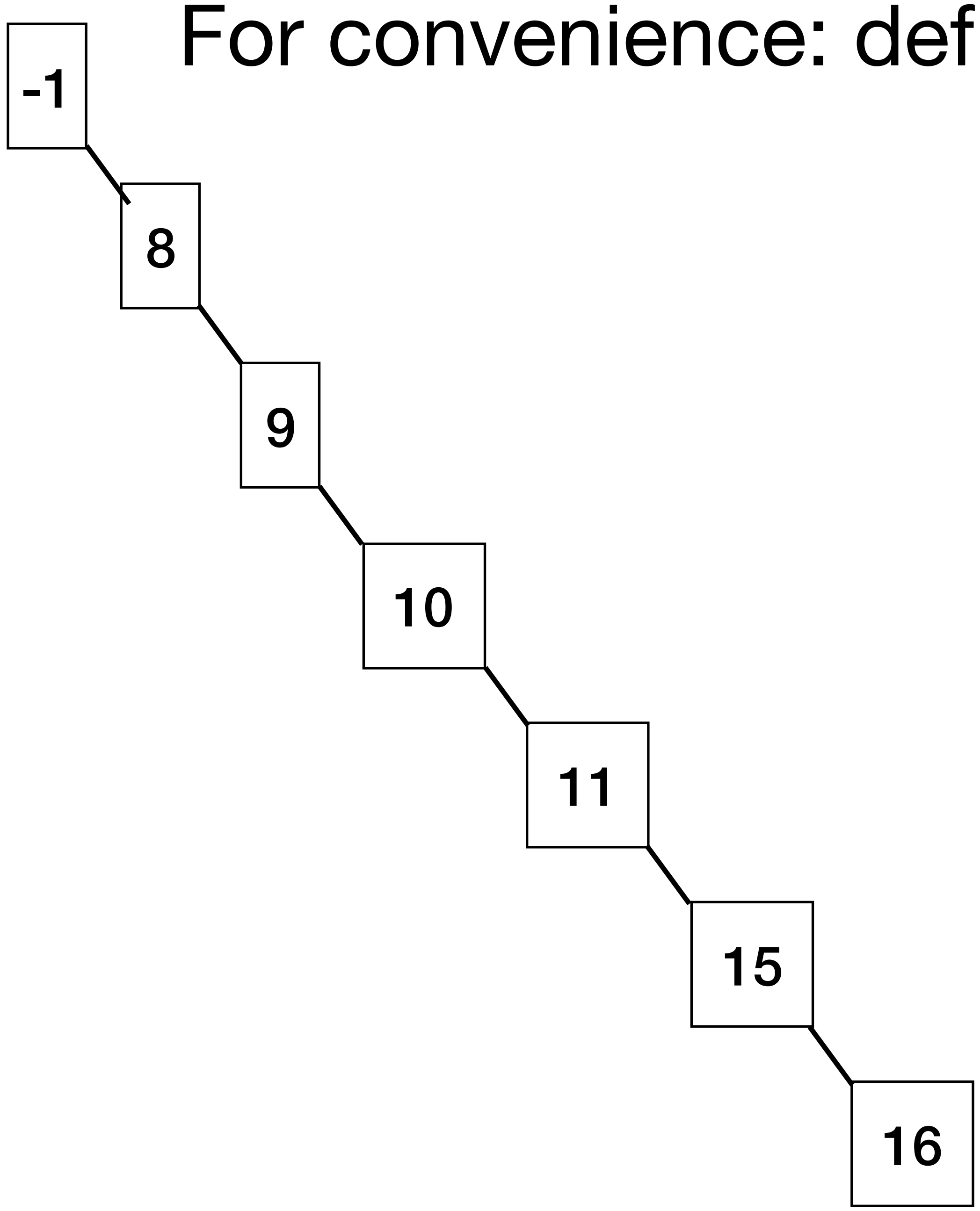
Perform a **left** rotation
on the root of this tree:

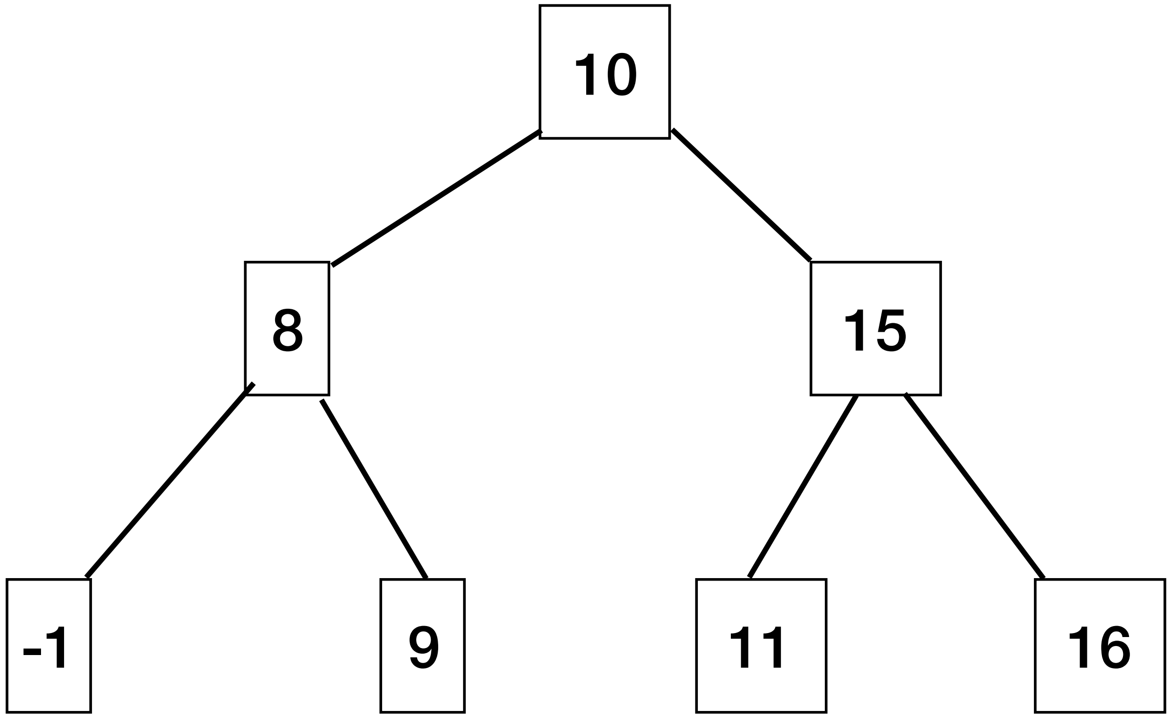


Can we improve balance?

Balace Factor(n) = height(n.right) - height(left)

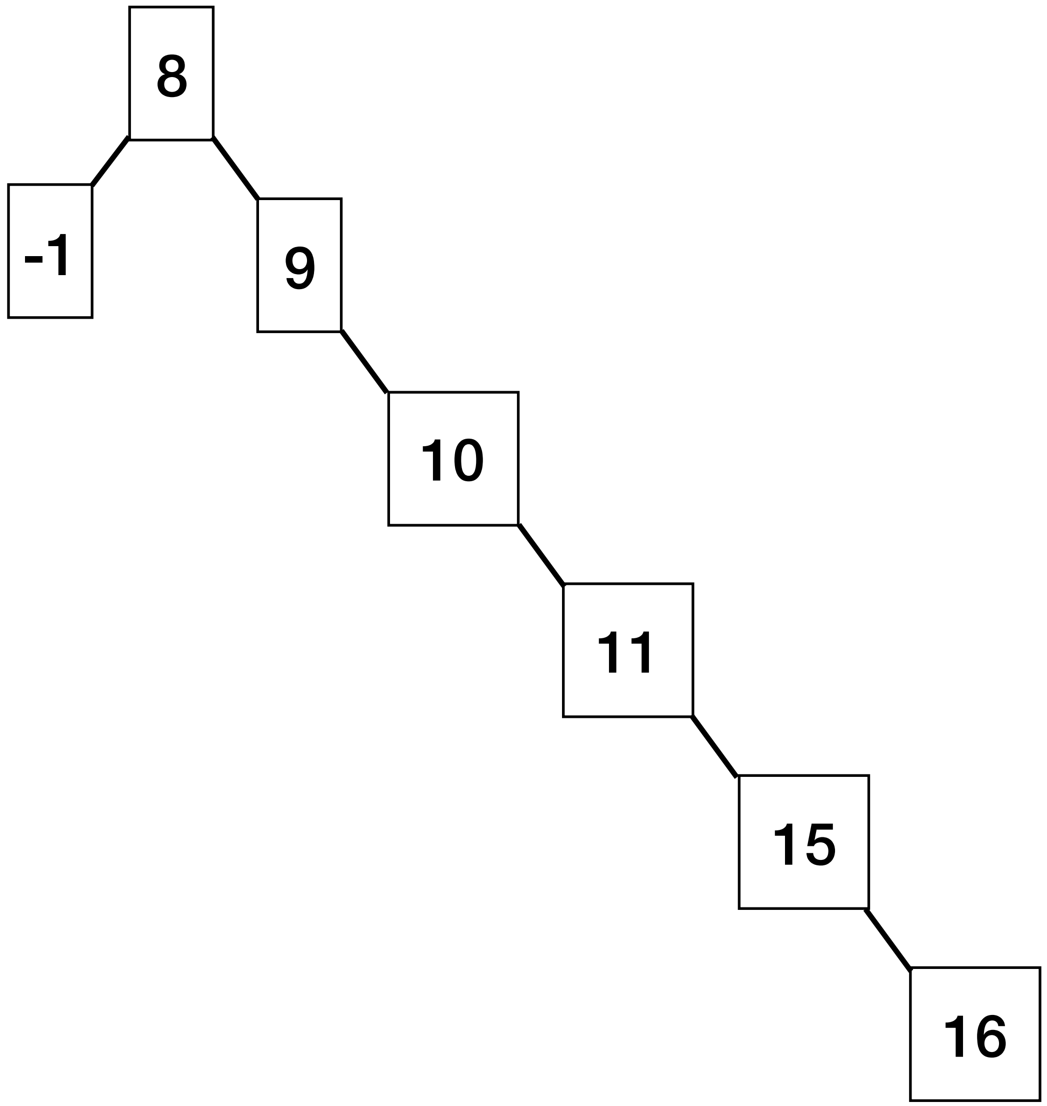
For convenience: define **height(null) = -1**

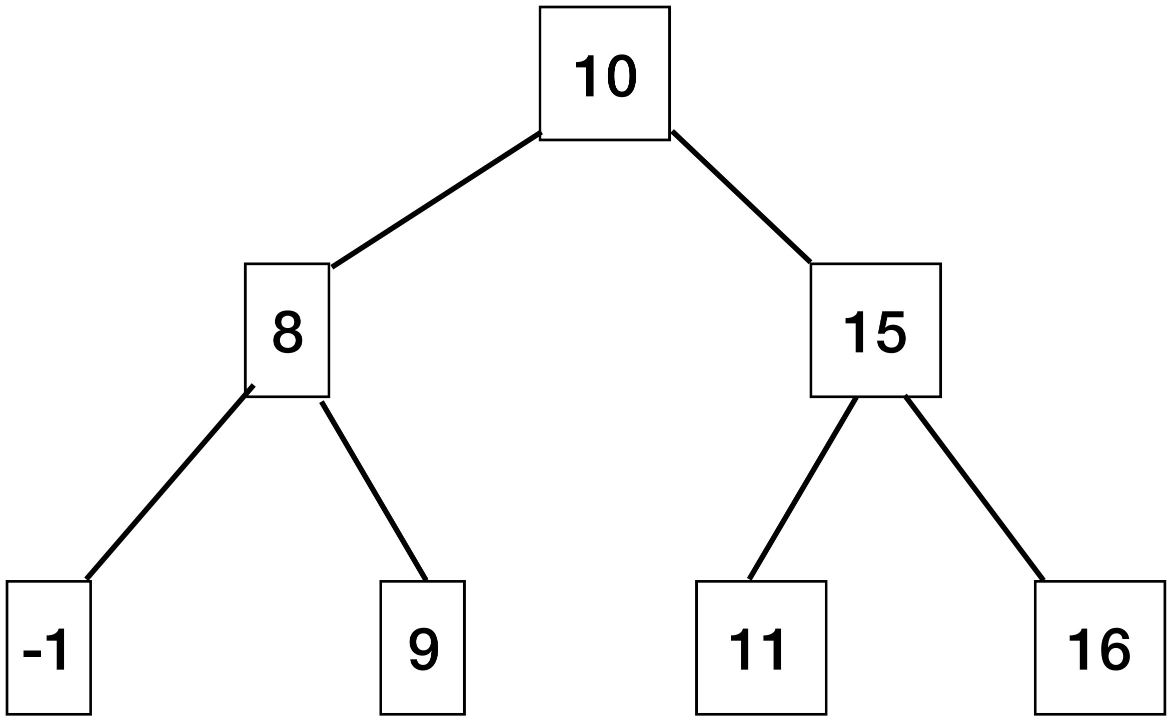




Can we improve balance?

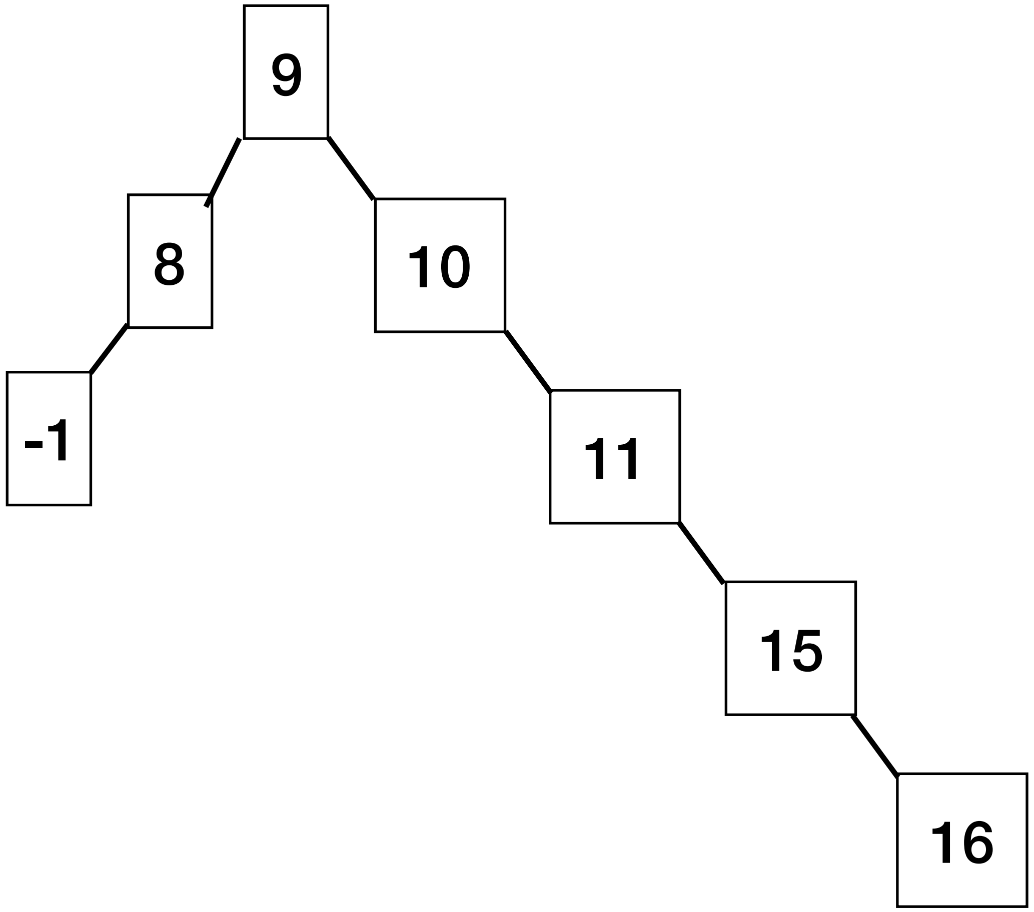
Balace Factor(n) = height(n.right) - height(left)

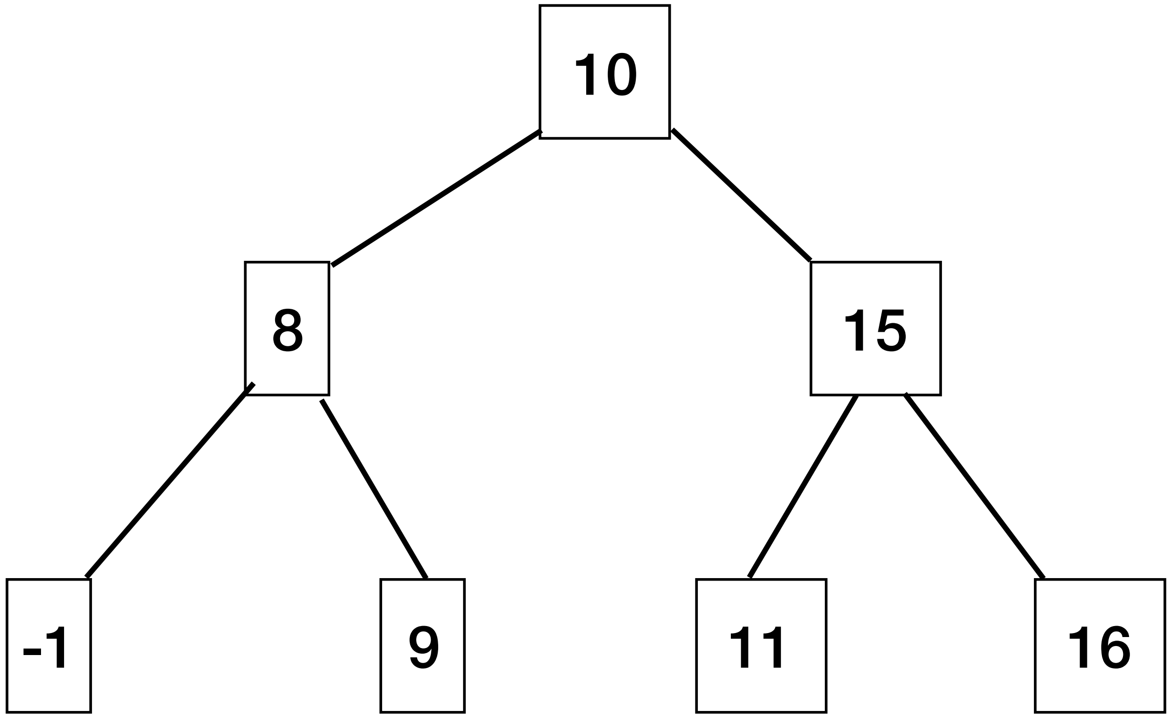




Can we improve balance?

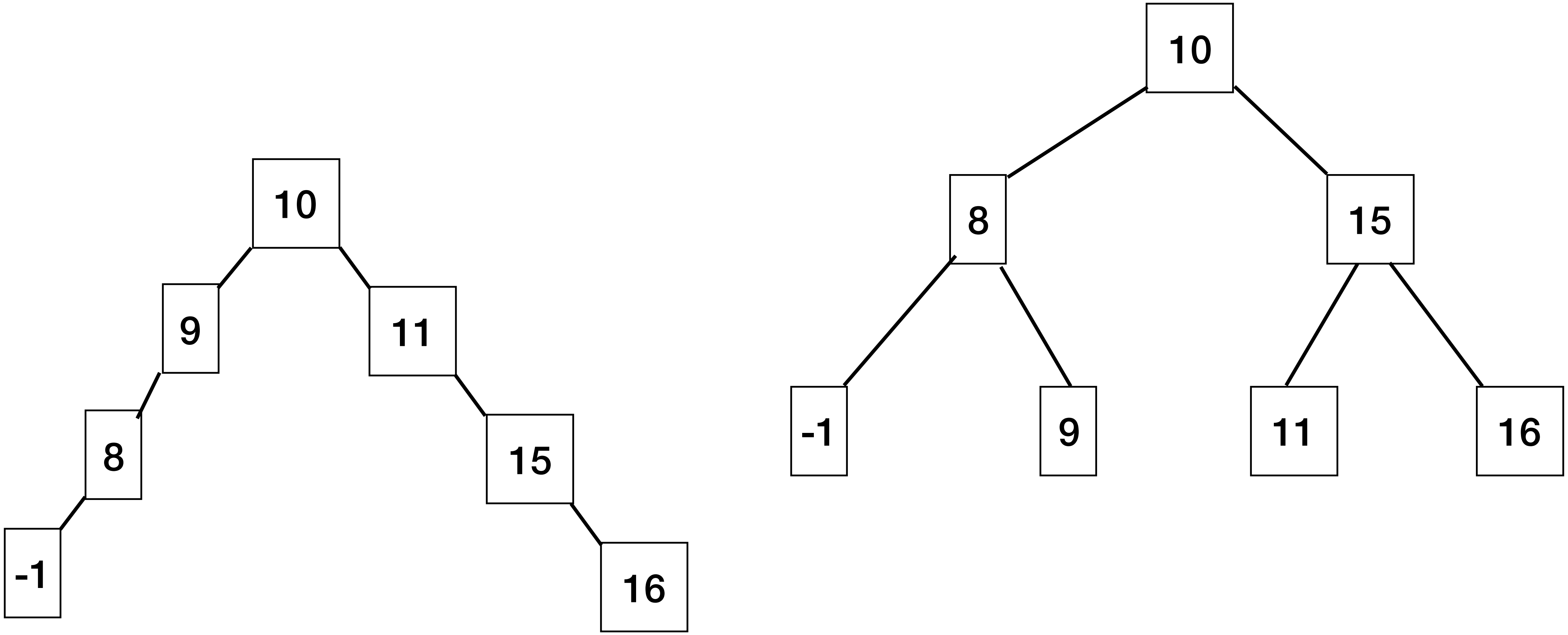
Balace Factor(n) = height(n.right) - height(left)





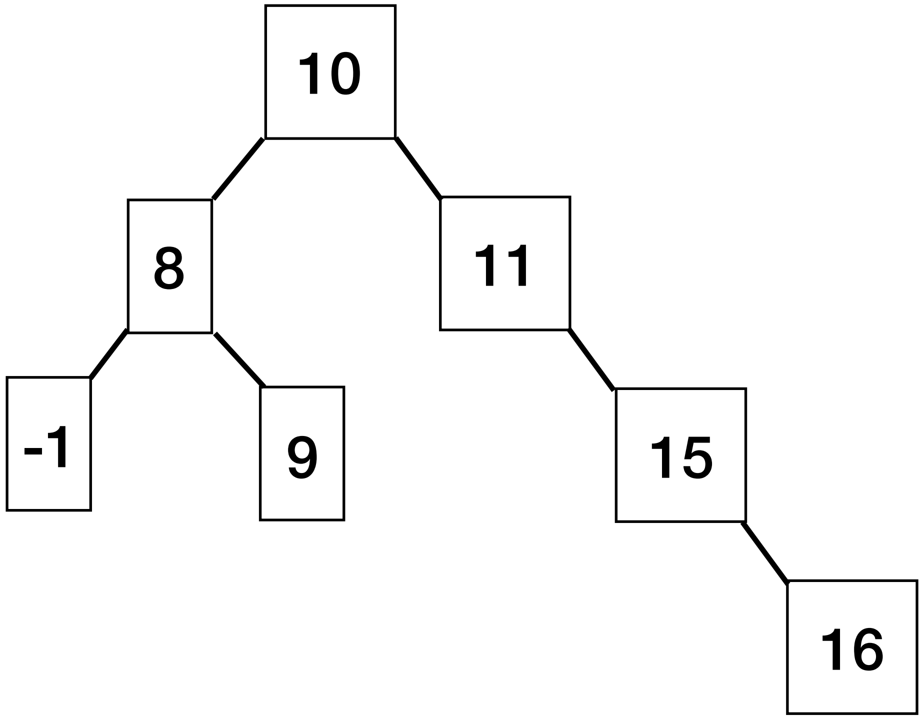
Can we improve balance?

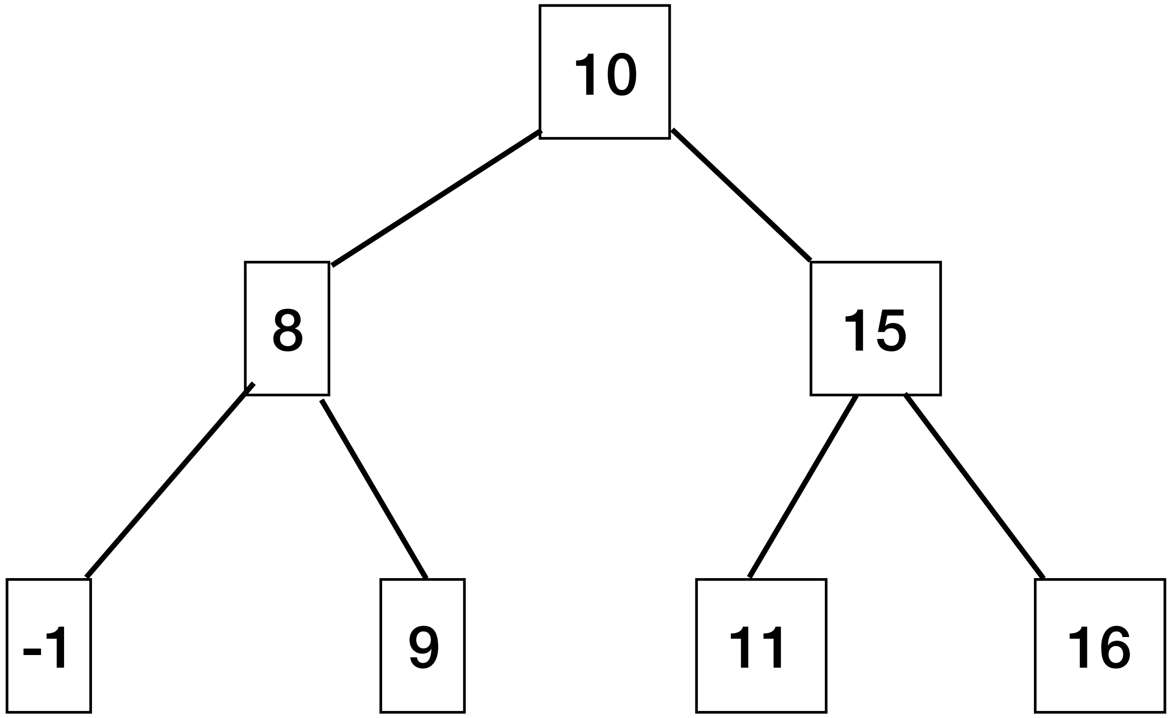
Balace Factor(n) = height(n.right) - height(left)



Can we improve balance?

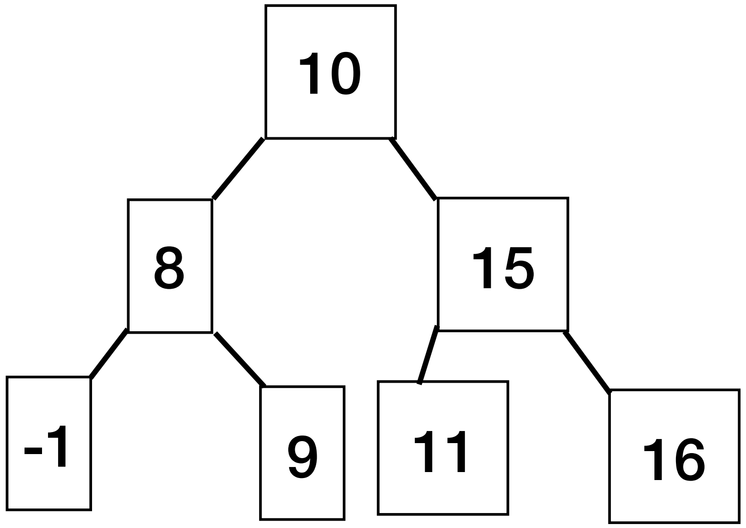
Balace Factor(n) = height(n.right) - height(left)

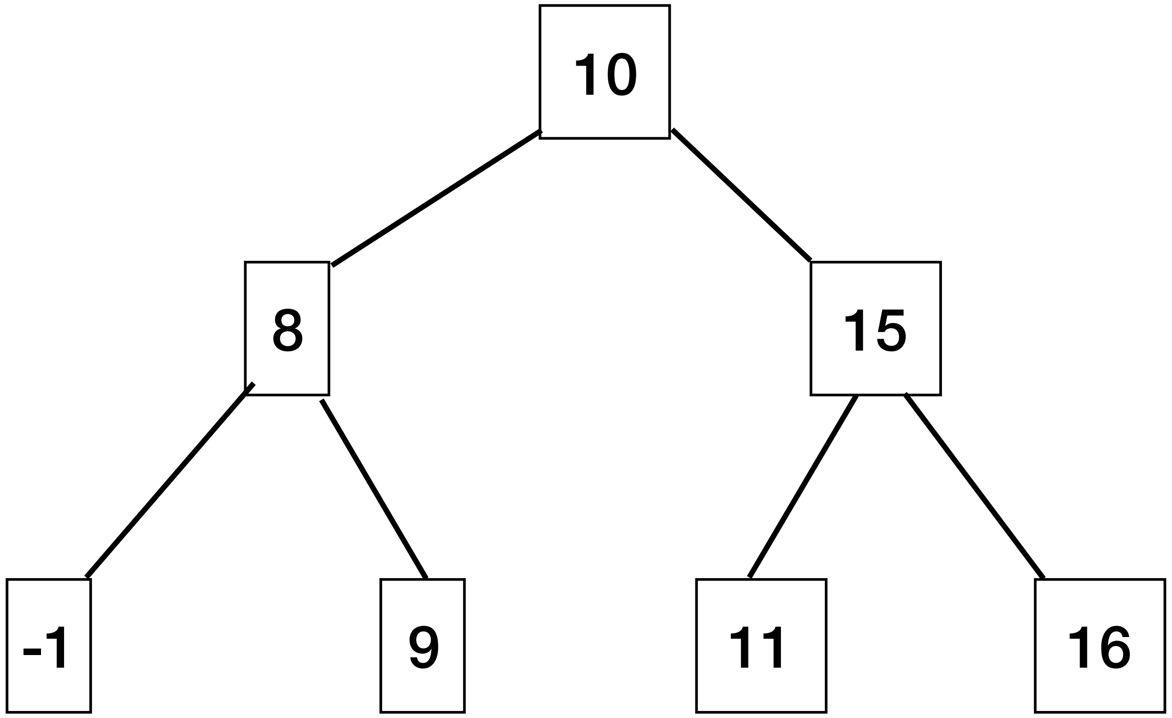




Can we improve balance?

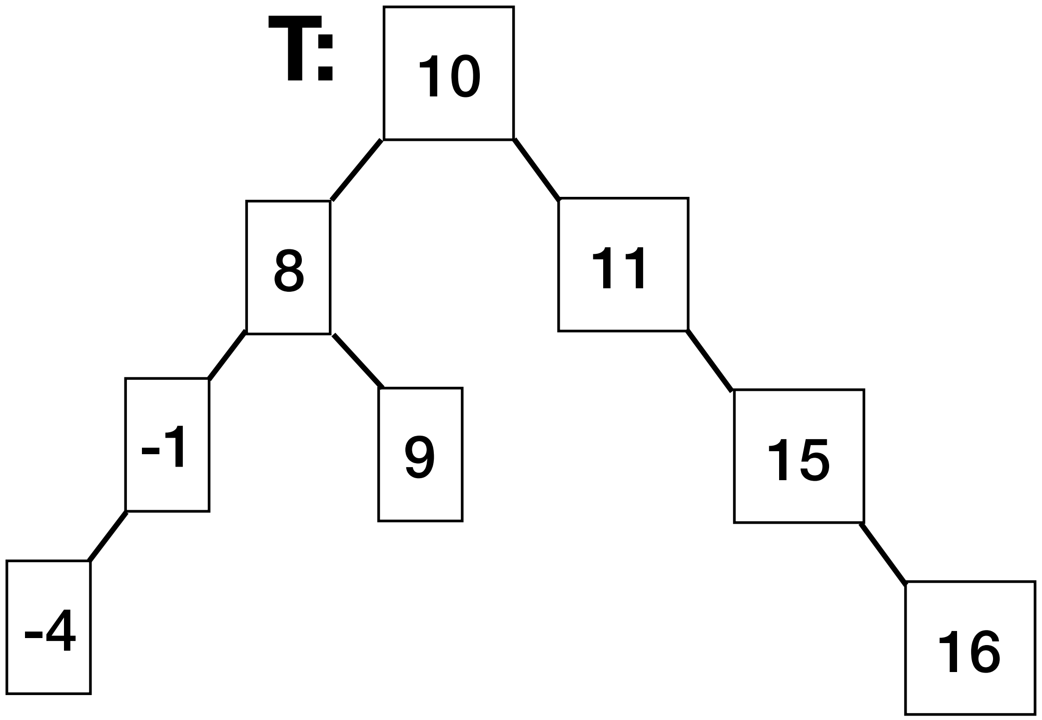
Balace Factor(n) = height(n.right) - height(left)

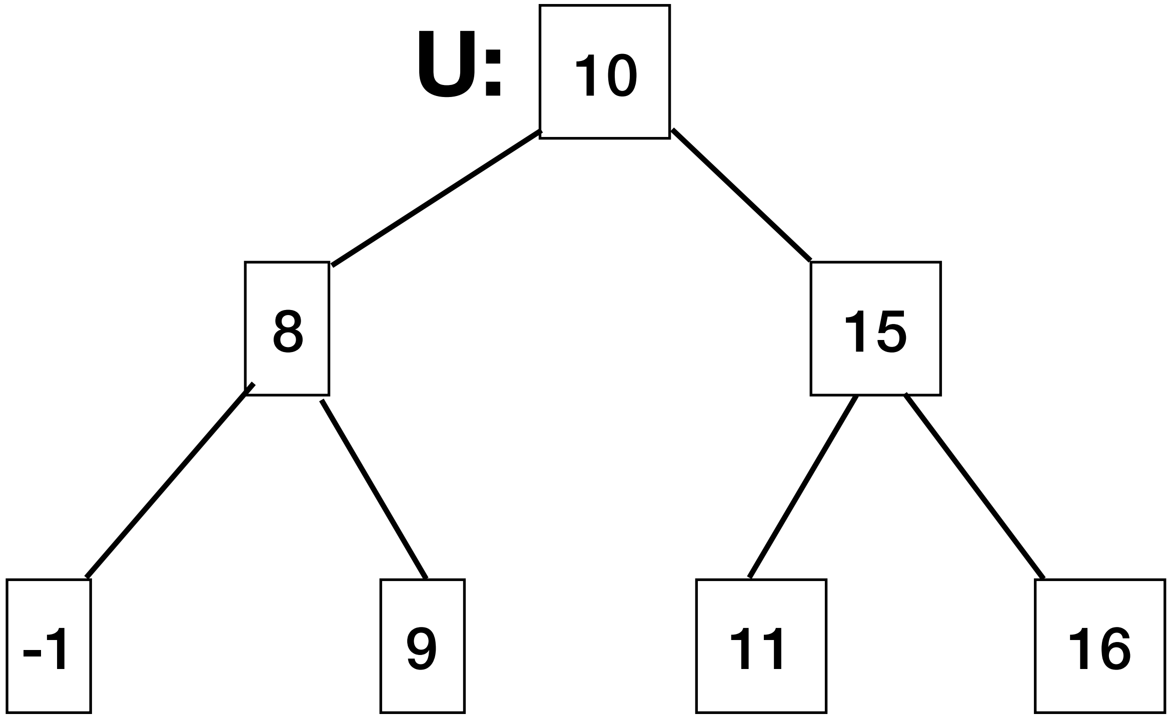




Balance Factor

Balace Factor b(n) = height(n.right) - height(left)





ABCD: What's the largest *absolute* balance factor of any node in each tree?

	T	U
A	0	0
B	2	1
C	2	0
D	1	1

AVL Trees

Balace Factor b(n) = height(n.right) - height(left)

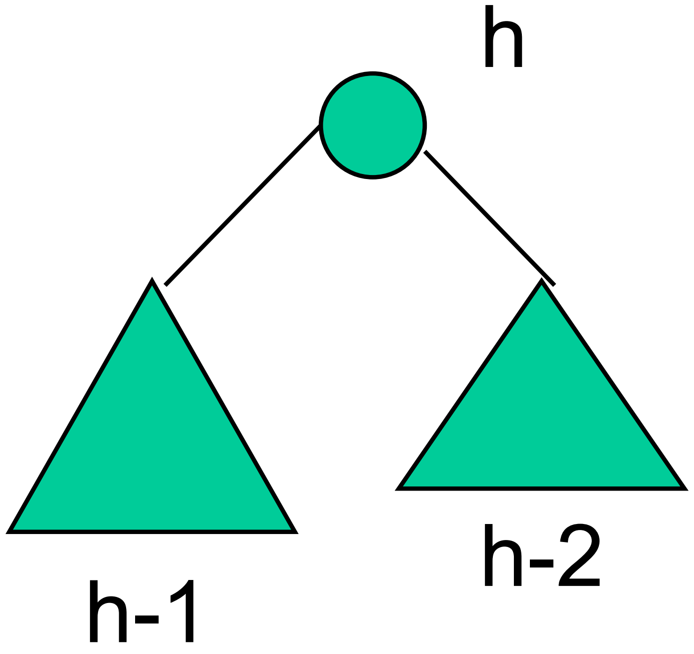
- Devised by **A**delson-**V**elsky and **L**andis
- An AVL tree is a Binary Search Tree in which the following property holds:

AVL property: -1 <= b(n) <= 1 for all nodes n.

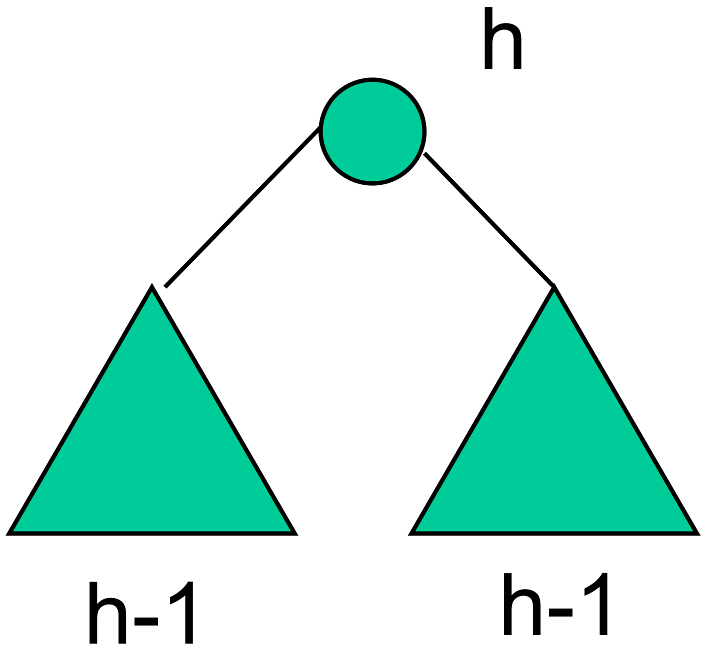
Balance Factor in AVL Trees

AVL property: -1 <= b(n) <= 1 for all nodes n.

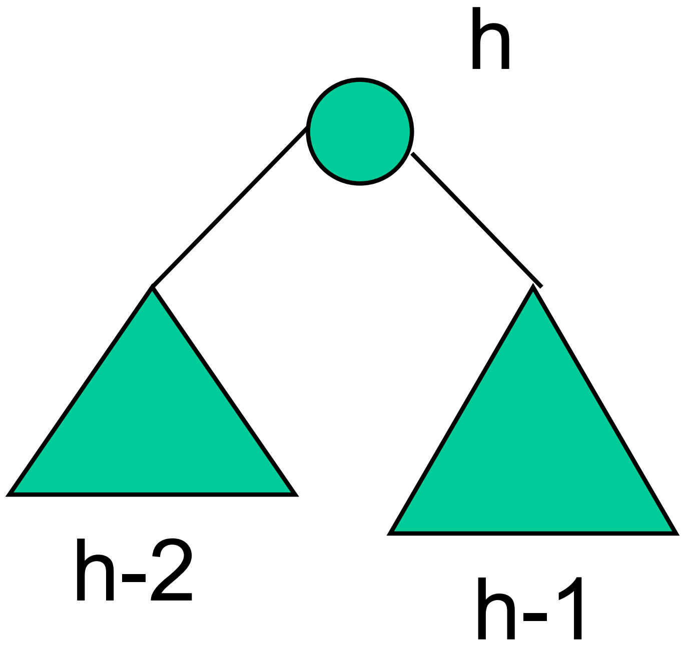
Every subtree in an AVL tree looks like one of these three trees:



(a) Balance factor: 1

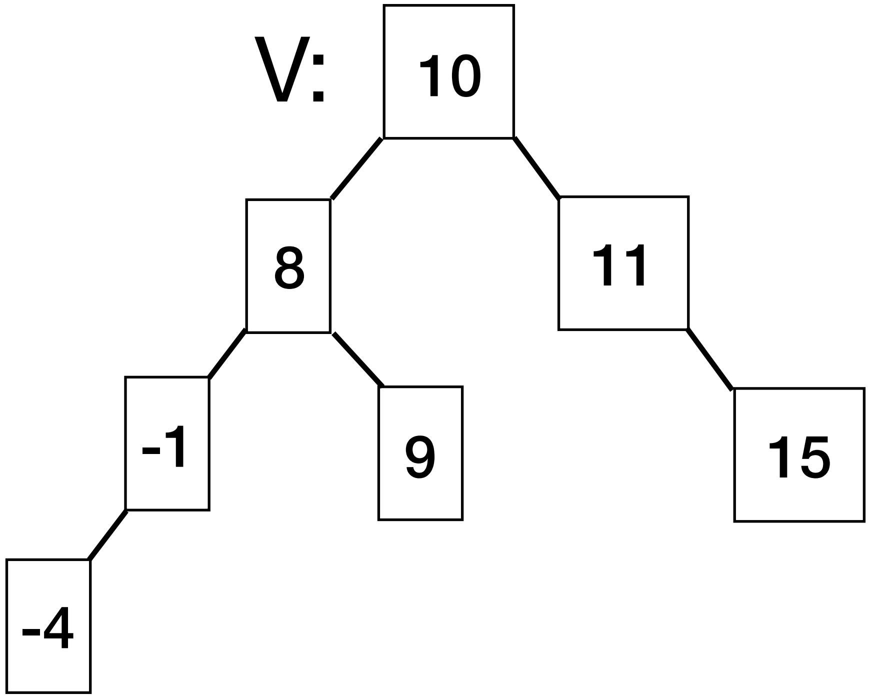


(b) Balance factor: 0

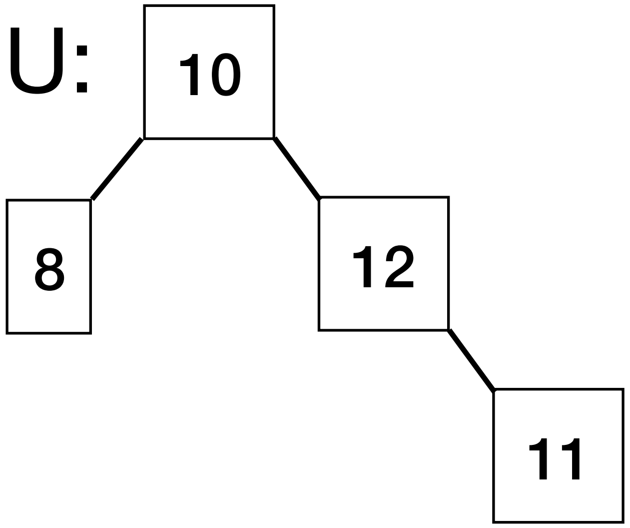


(c) Balance factor: -1

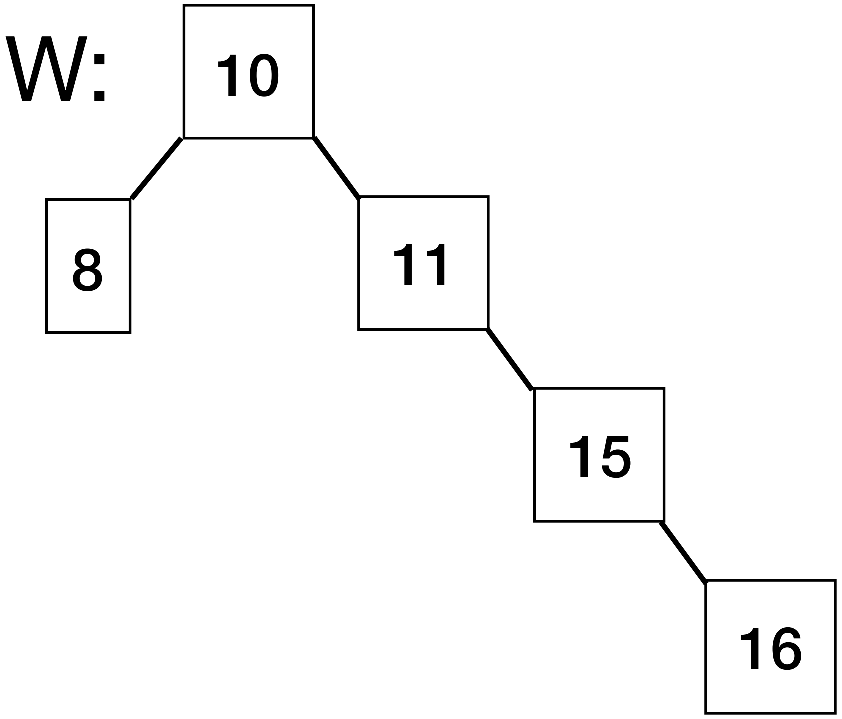
V:



U:



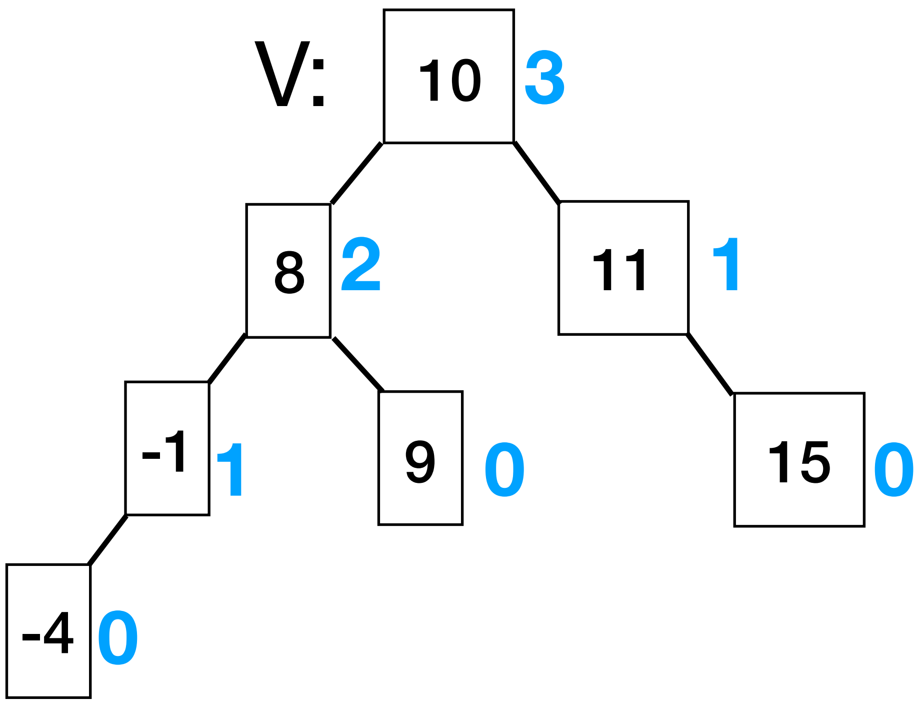
W:



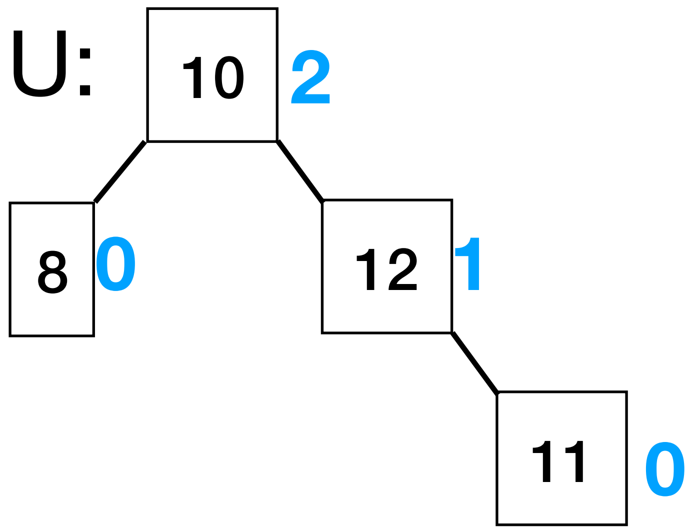
ABCD: Which of these is/are **not** AVL trees?

A. U
B. W
C. V and W
D. U and W

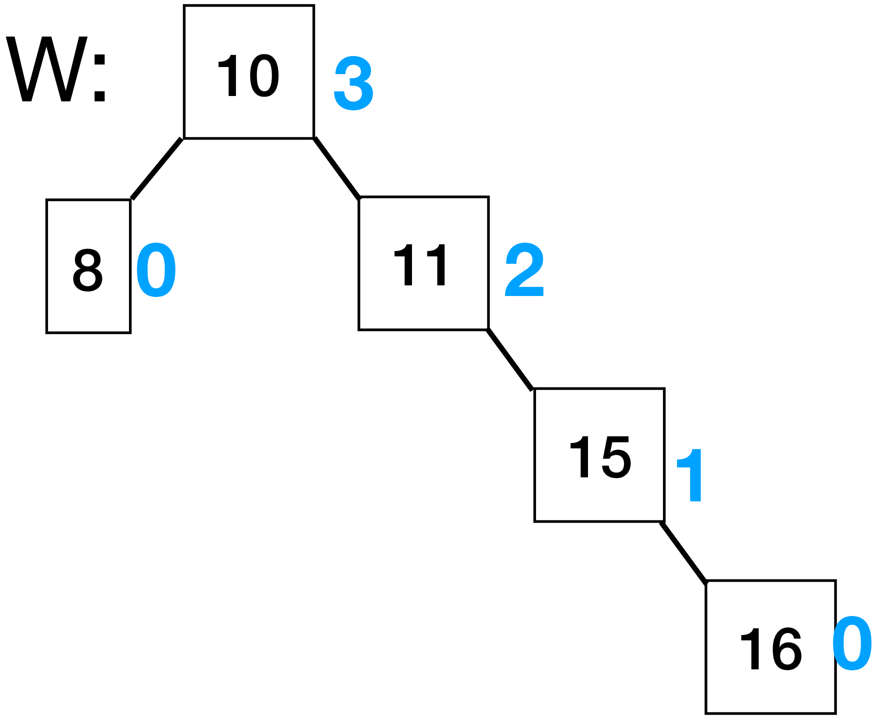
V:



U:



W:



ABCD: Which of these is/are **not** AVL trees?

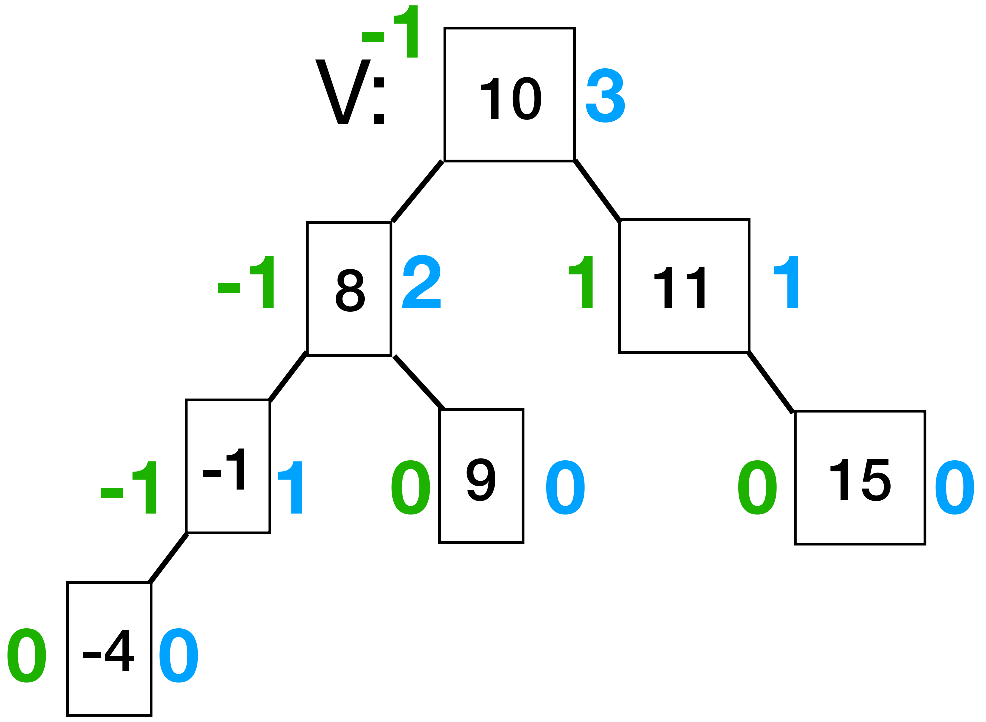
A. U

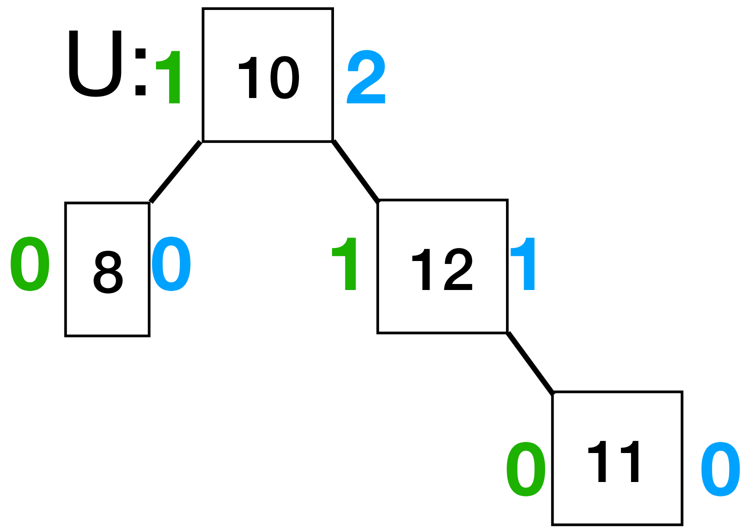
B. W

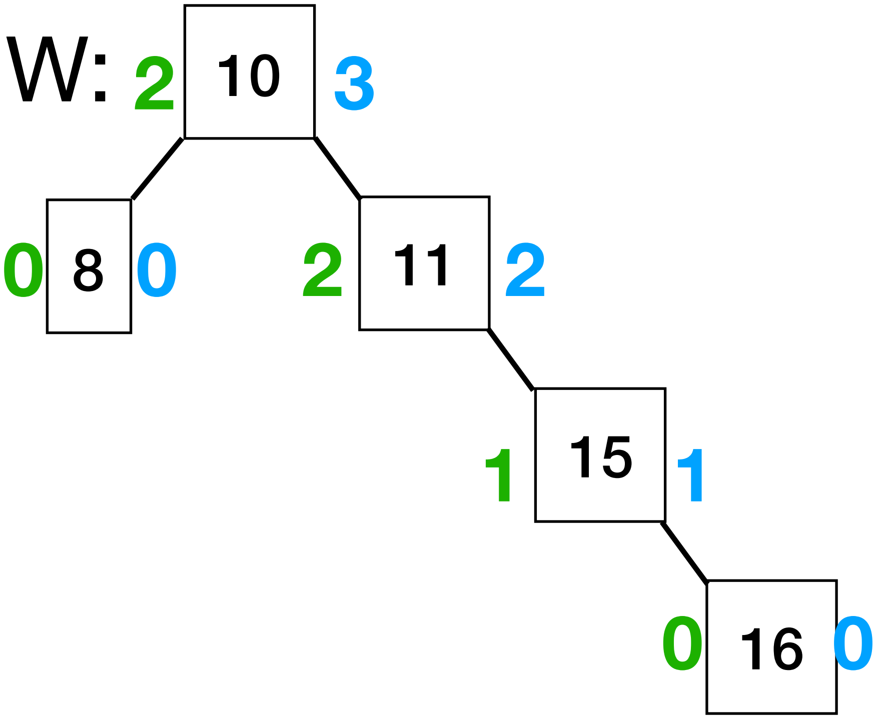
C. V and W

D. U and W

Heights in blue.







ABCD: Which of these is/are **not** AVL trees?

A. U
B. W
C. V and W
D. U and W

Heights in blue.

Balance factors in green.

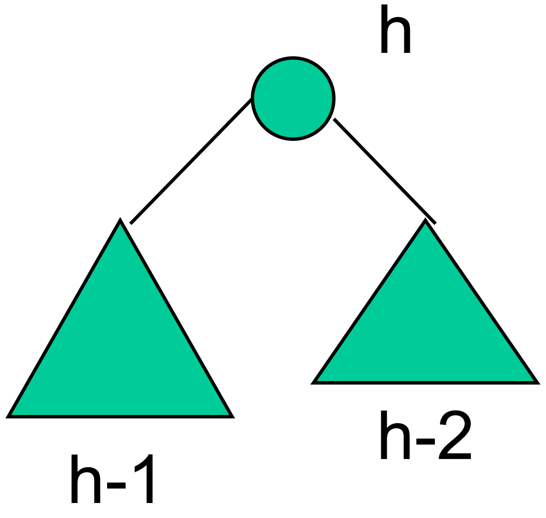
AVL Trees: Insertion

- An AVL tree is a Binary Search Tree in which the following property holds:

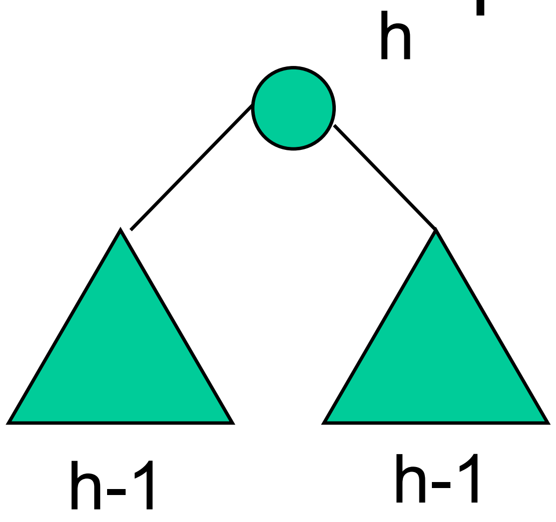
AVL property: -1 <= b(n) <= 1 for all nodes n.

To insert into an AVL tree:

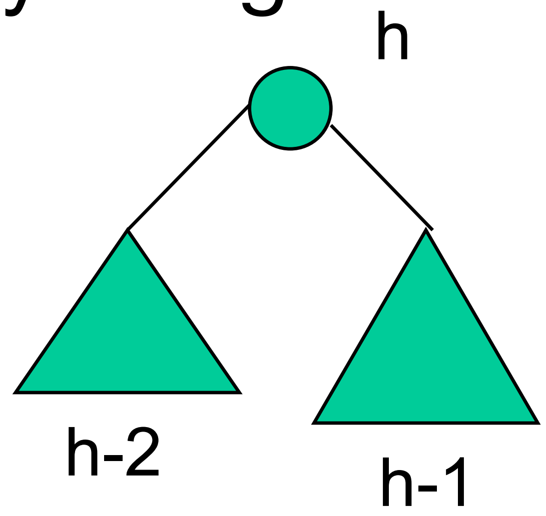
1. Do a normal BST insertion
2. Fix any violations of the AVL property using rotations.



(a) Balance factor: 1



(b) Balance factor: 0



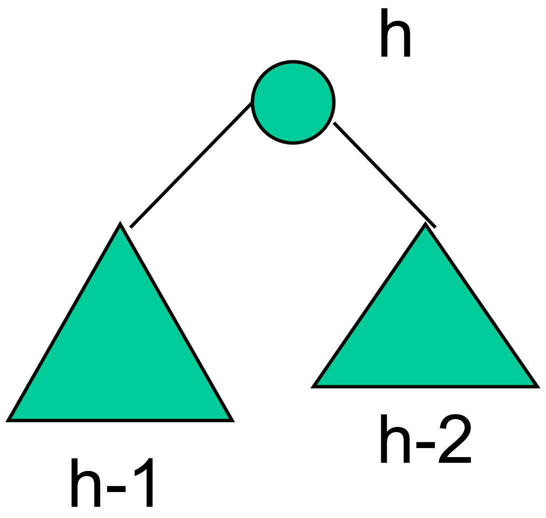
(c) Balance factor: -1

AVL Trees: Insertion

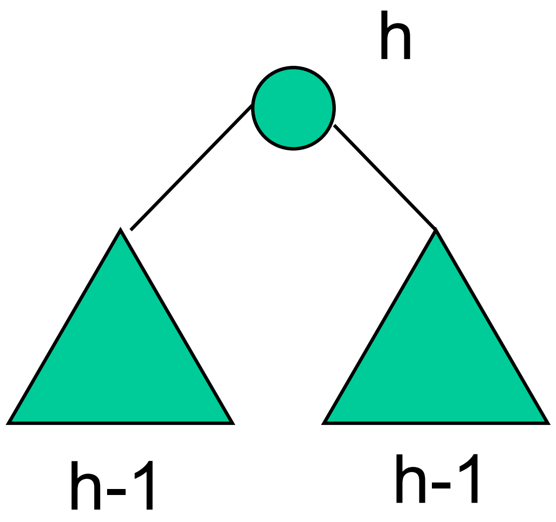
AVL property: -1 <= b(n) <= 1 for all nodes n.

To insert into an AVL tree:

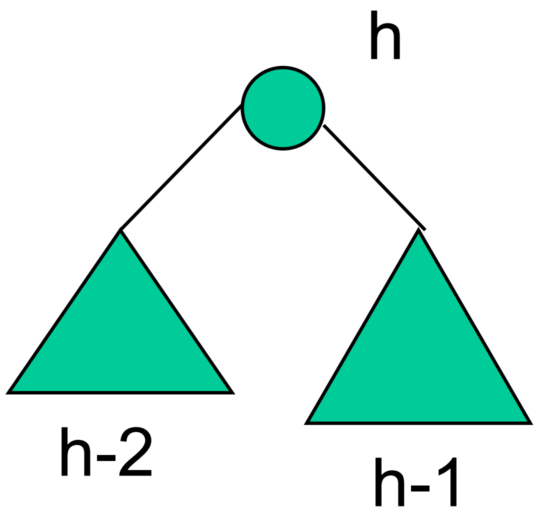
1. Do a normal BST insertion
2. Fix any violations of the AVL property using rotations.



(a) Balance factor: 1



(b) Balance factor: 0



(c) Balance factor: -1

Refresher: BST Insertion

```
/* insert a node with value v into the
 * tree rooted at n. pre: n is not null. */
insert(Node n, int v):
  if n.value == v: return // (duplicate)
  if v < n.value:
    if n has left:
      insert(n.left, v)
    else:
      // attach new node w/ value v to n.left
  else: // v > n.value
    if n has right:
      insert(n.right, v)
    else:
      // attach new node w/ value v to n.right
```

AVL Insertion

```
/* insert a node with value v into the
 * tree rooted at n. pre: n is not null. */
insert(Node n, int v):
  if n.value == v: return // (duplicate)
  if v < n.value:
    if n has left:
      insert(n.left, v)
    else:
      // attach new node w/ value v to n.left
  else: // v > n.value
    if n has right:
      insert(n.right, v)
    else:
      // attach new node w/ value v to n.right
  rebalance(n);  ←
```