Binary Trees and Binary Search Trees (BST)

William Fiset

Outline

- Discussion & examples
  - What is a Binary Tree (BT)?
  - What is a Binary Search Tree (BST)?
  - Where are BTs and BSTs used?
- Complexity Analysis

Outline

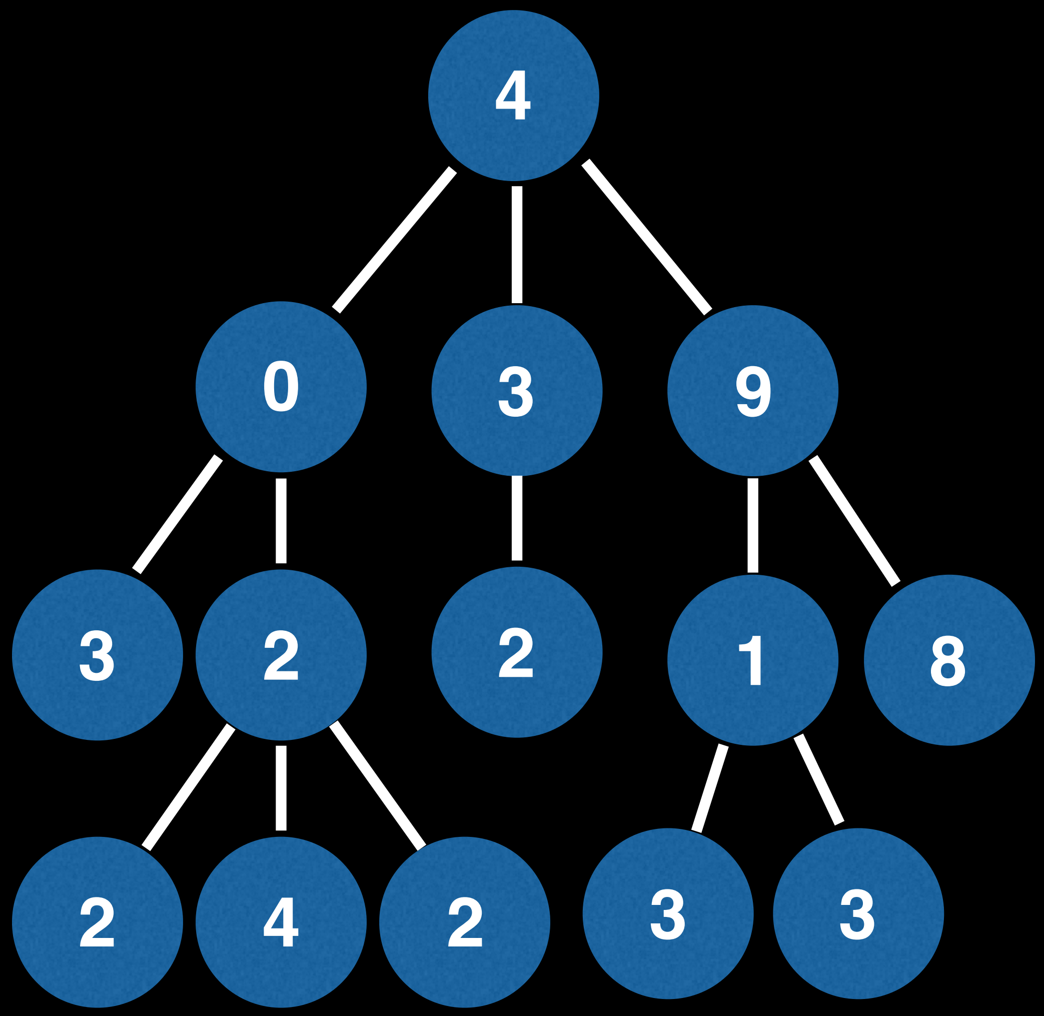
- How to insert nodes into a BST
- How to remove nodes from a BST
- Binary tree traversals
  - preorder, inorder, postorder, and level order traversals
- A glance at some source code :)

Discussion and Examples

Quick terminology crash course

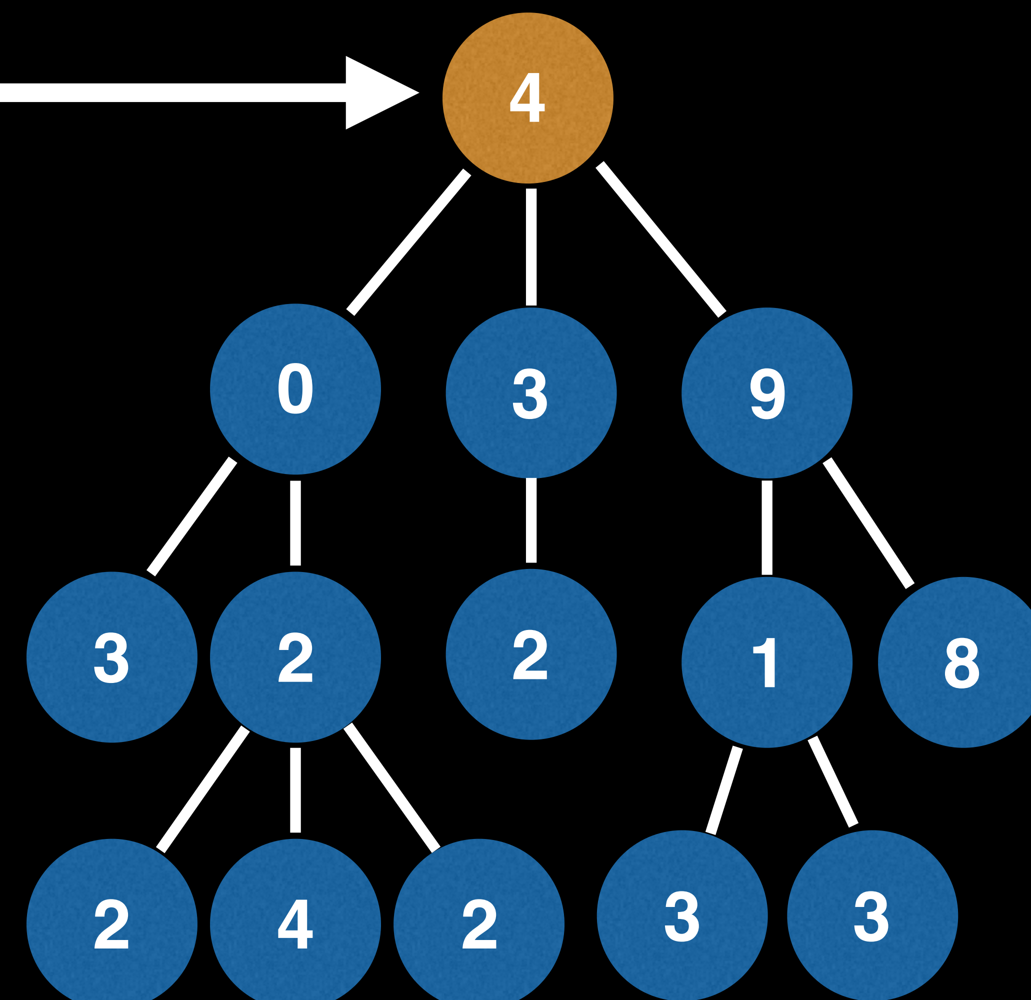
A **tree** is an **undirected graph** which satisfies any of the following definitions:

- An acyclic connected graph
- A connected graph with N nodes and N−1 edges.
- An graph in which any two vertices are connected by *exactly* one path.



Quick terminology crash course

Root node



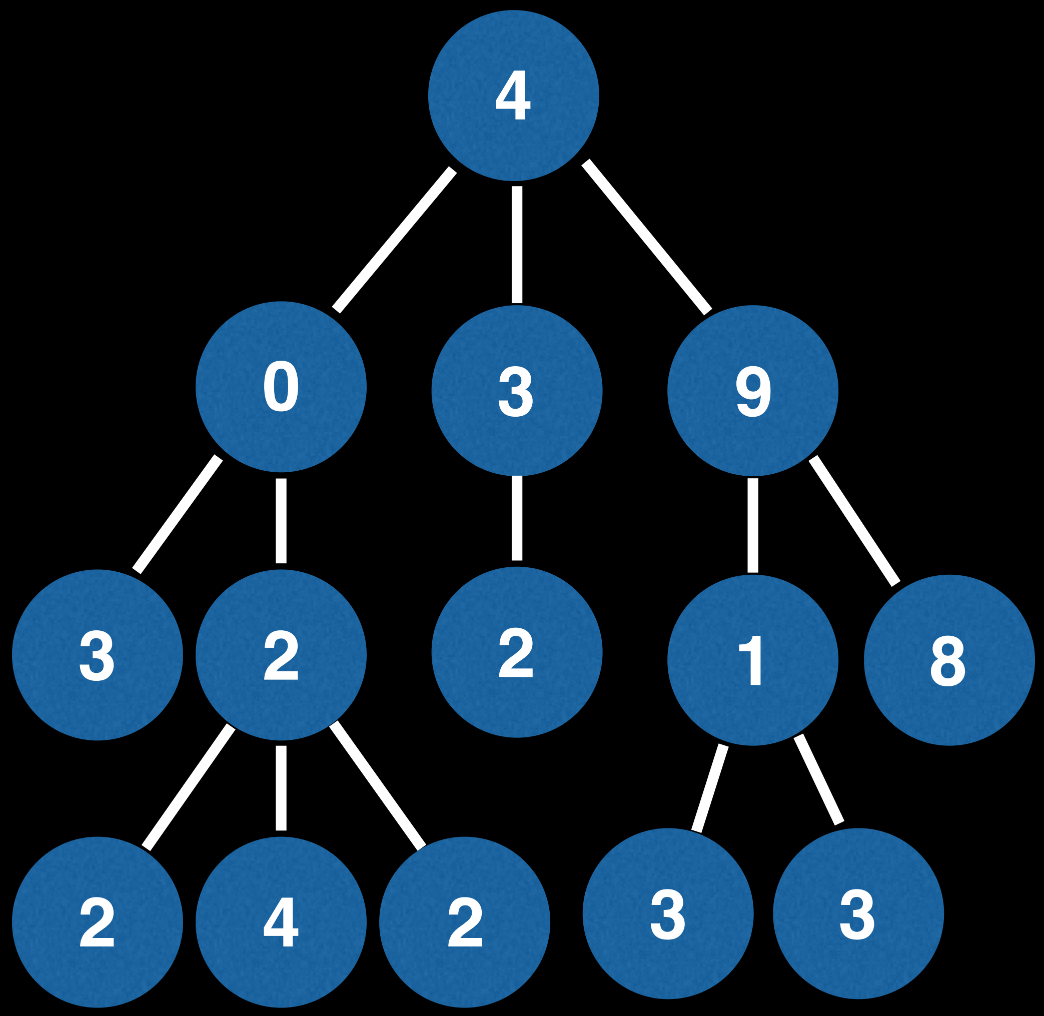
If we have a **rooted tree** then we will want to have a reference to the root node of our tree.

It does not always matter which node is selected to be the root node because any node can root the tree!

Quick terminology crash course

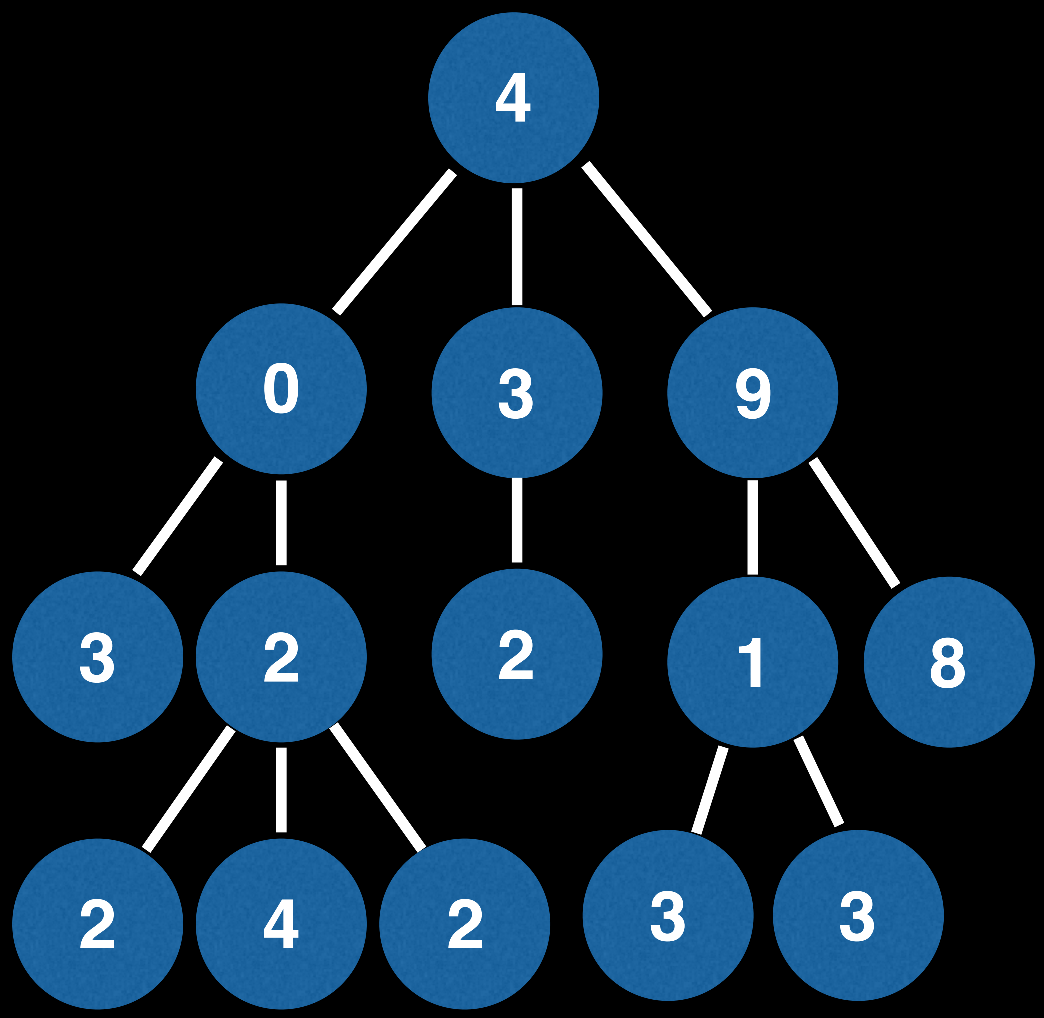
A **child** is a node extending from another node. A **parent** is the inverse of this.

Q: What is the parent of the root node?



Quick terminology crash course

A **child** is a node extending from another node. A **parent** is the inverse of this.



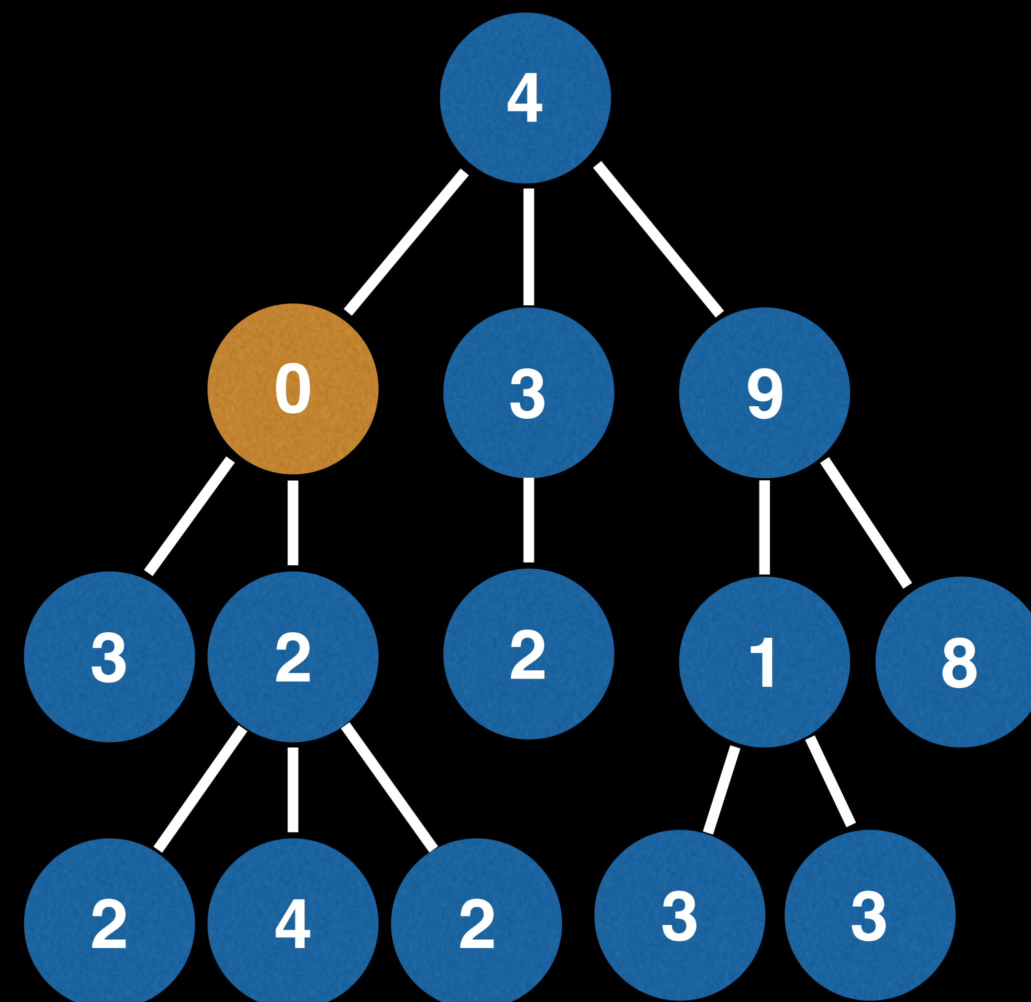
Q: What is the parent of the root node?

A: It has no parent, although it may be useful to assign the parent of the root node to be itself (e.g. filesystem tree).

Quick terminology crash course

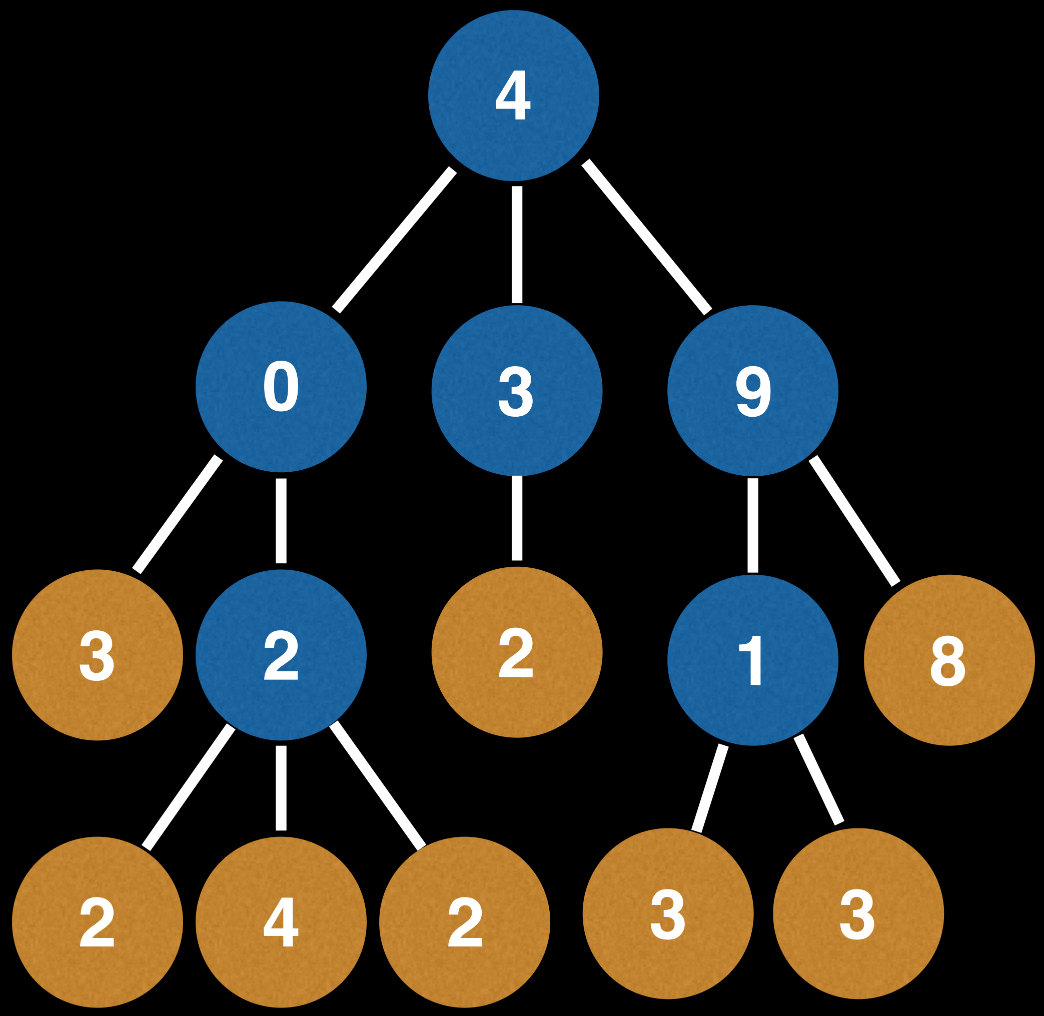
A **child** is a node extending from another node. A **parent** is the inverse of this.

0 has two children (3 and 2) and a parent (4)



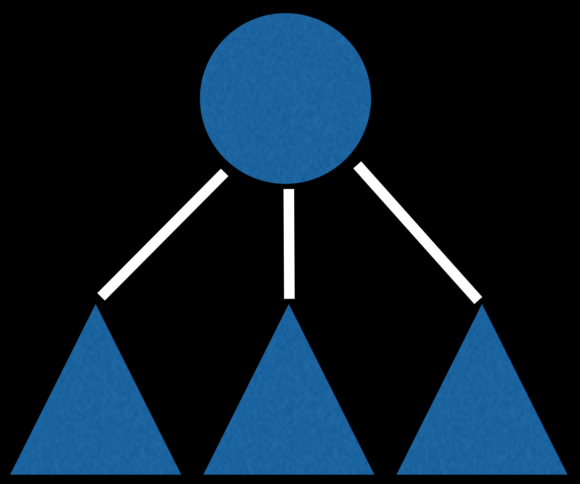
Quick terminology crash course

A **leaf node** is a node with no children. These have been highlighted in orange.



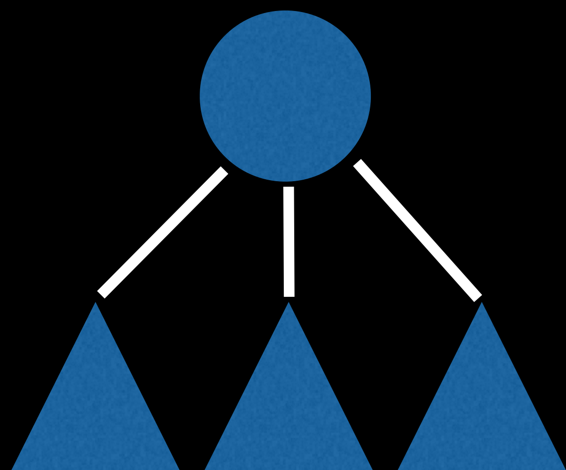
Quick terminology crash course

A **subtree** is a tree entirely contained within another. They are usually denoted using triangles.

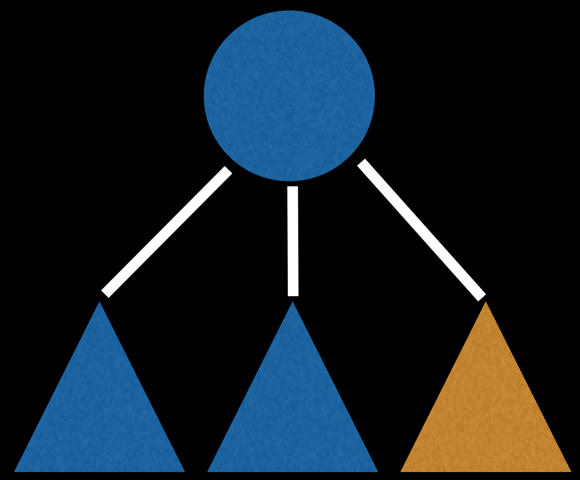


Note: Subtrees may consist of a single node!

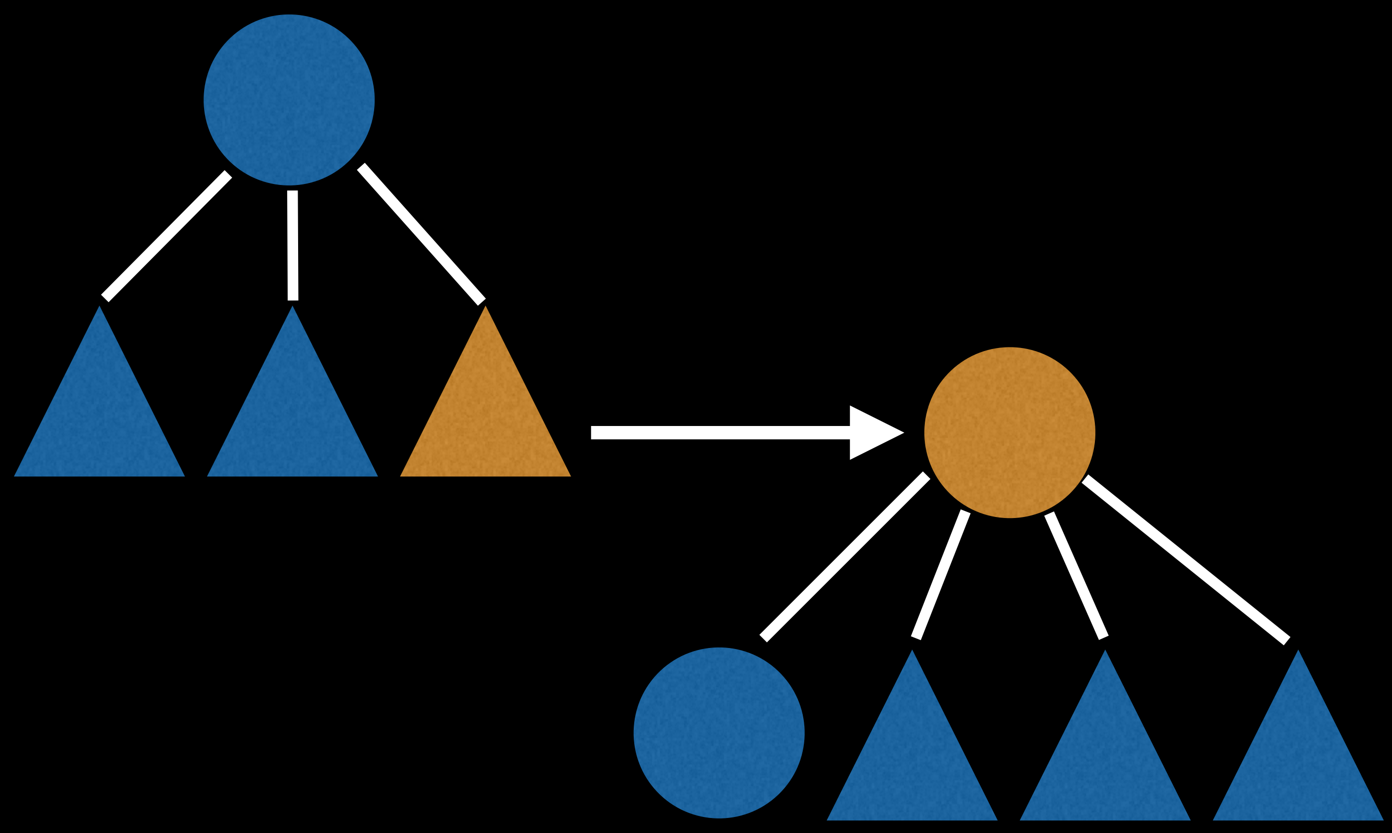
Quick terminology crash course



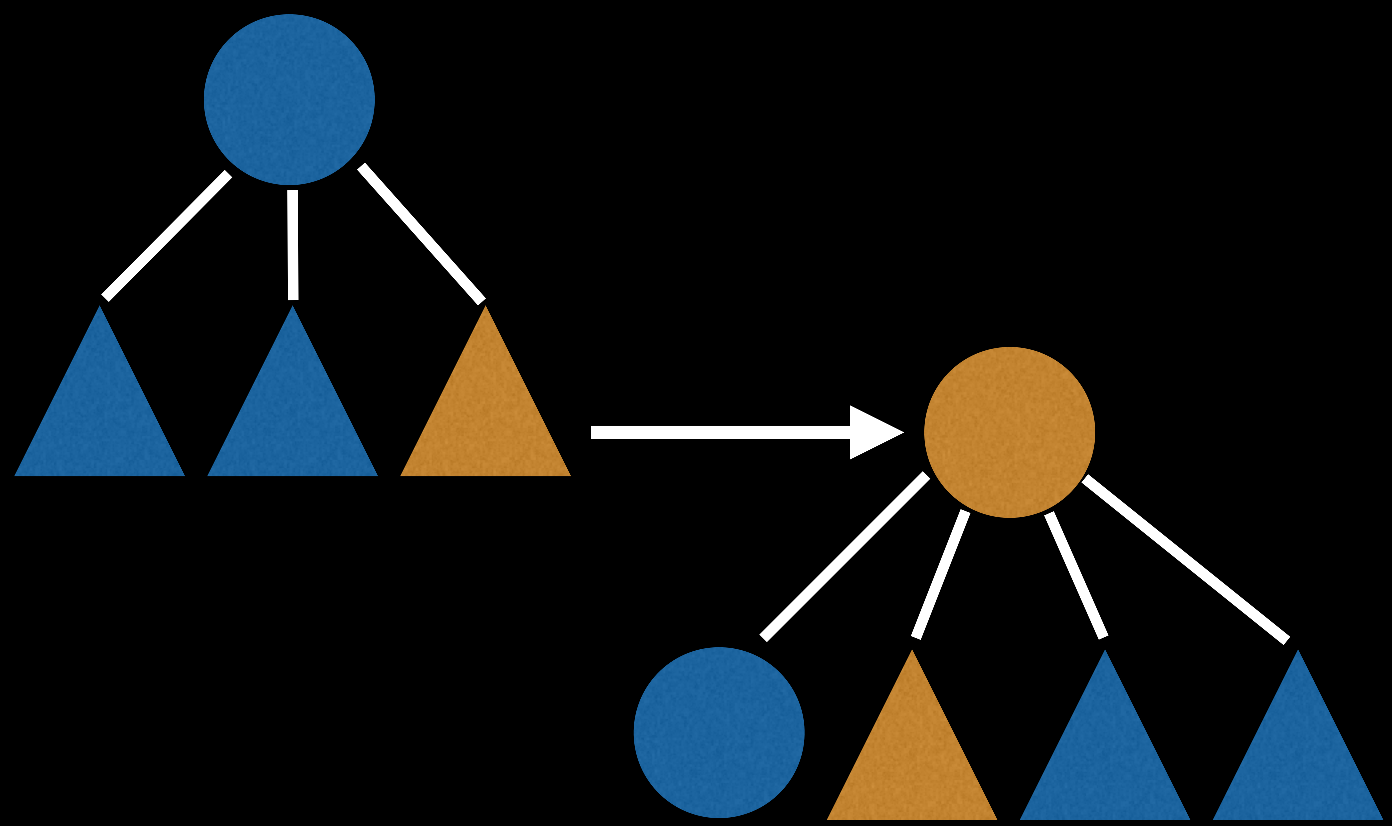
Quick terminology crash course



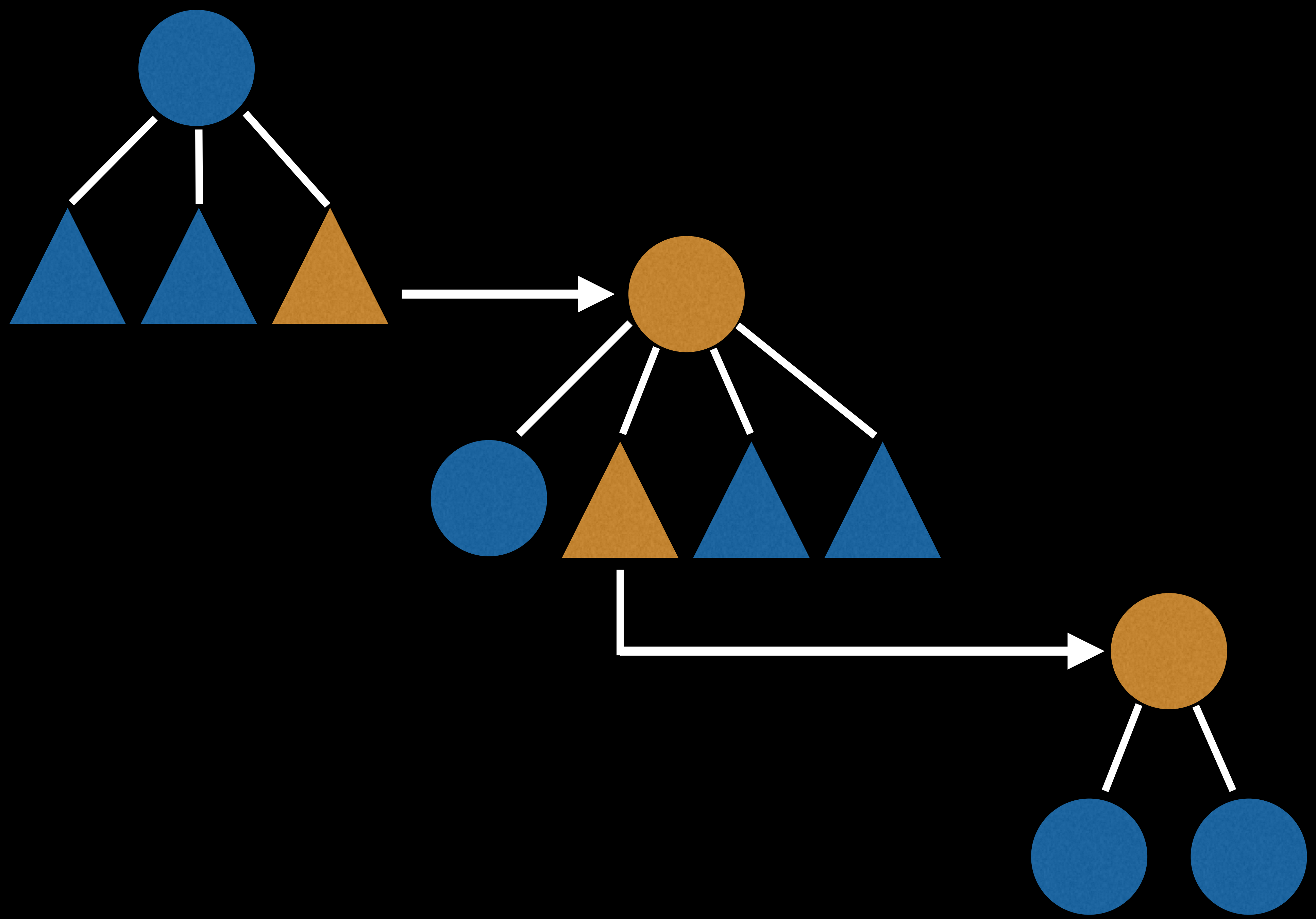
Quick terminology crash course



Quick terminology crash course

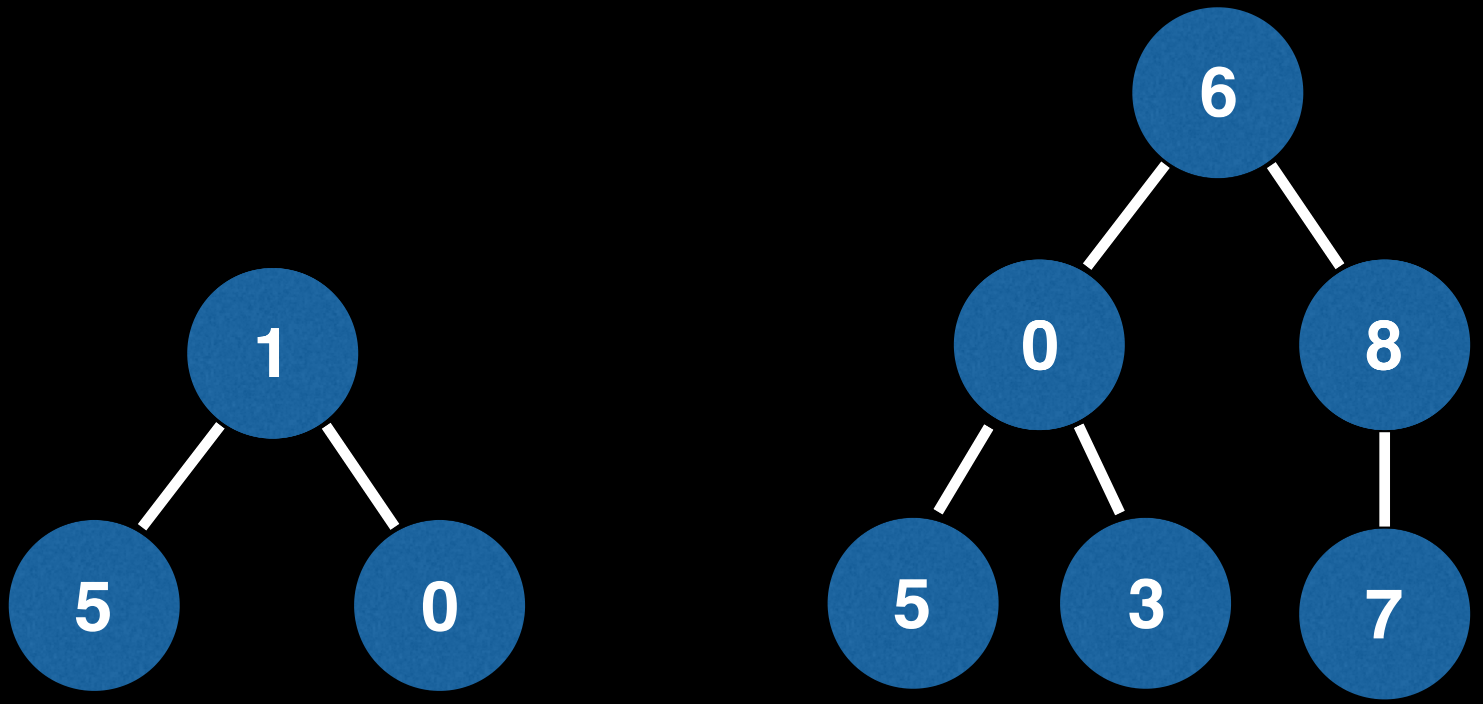


Quick terminology crash course



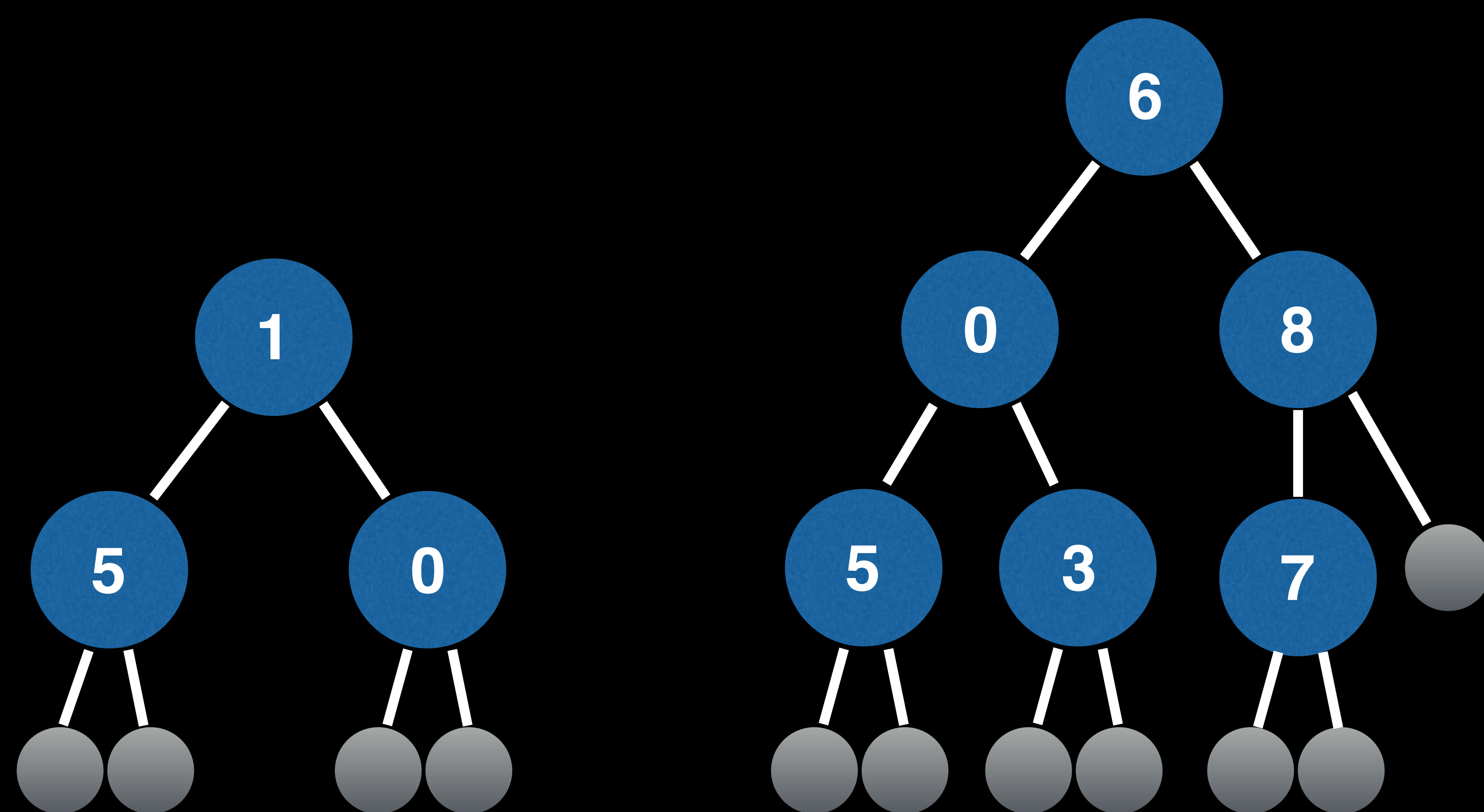
What is a Binary Tree (BT)?

A **binary tree** is a tree for which every node has at most two child nodes.

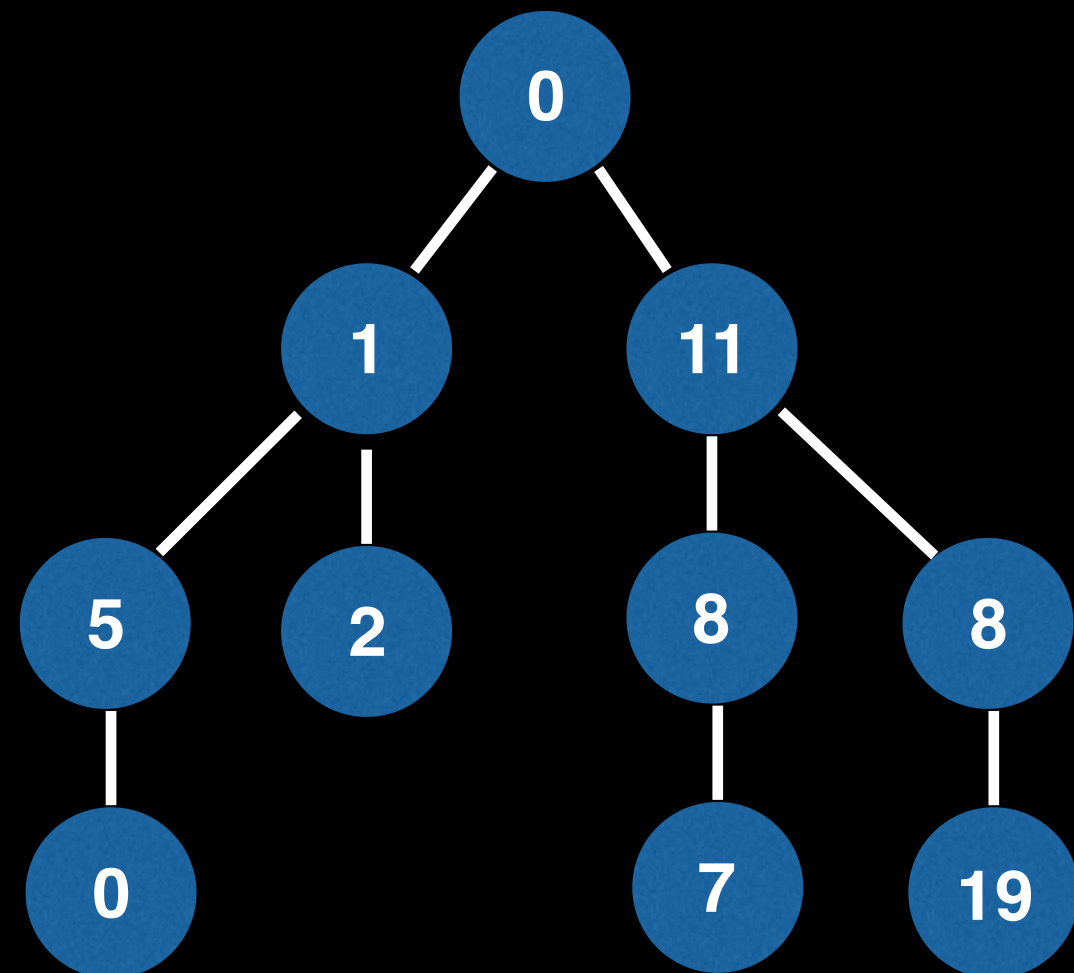


What is a Binary Tree (BT)?

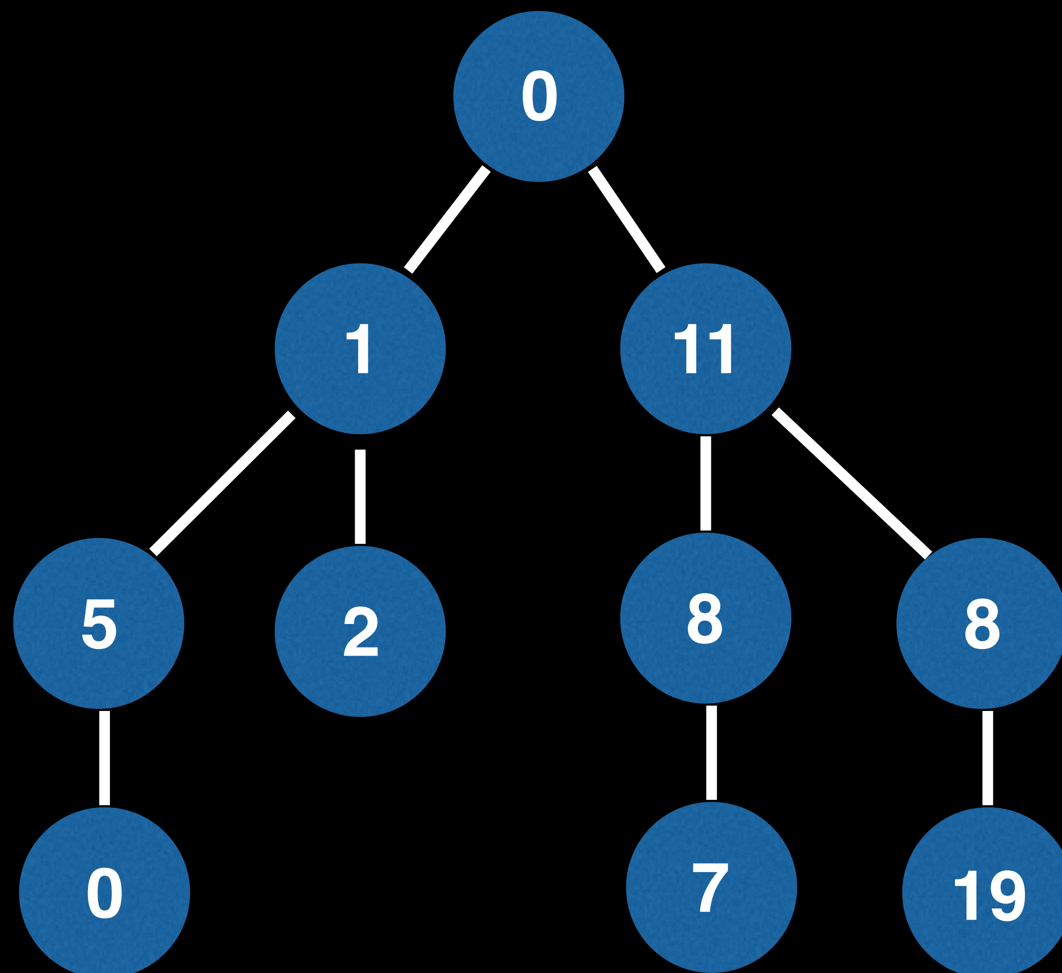
A **binary tree** is a tree for which every node has at most two child nodes.



Is this a BT?

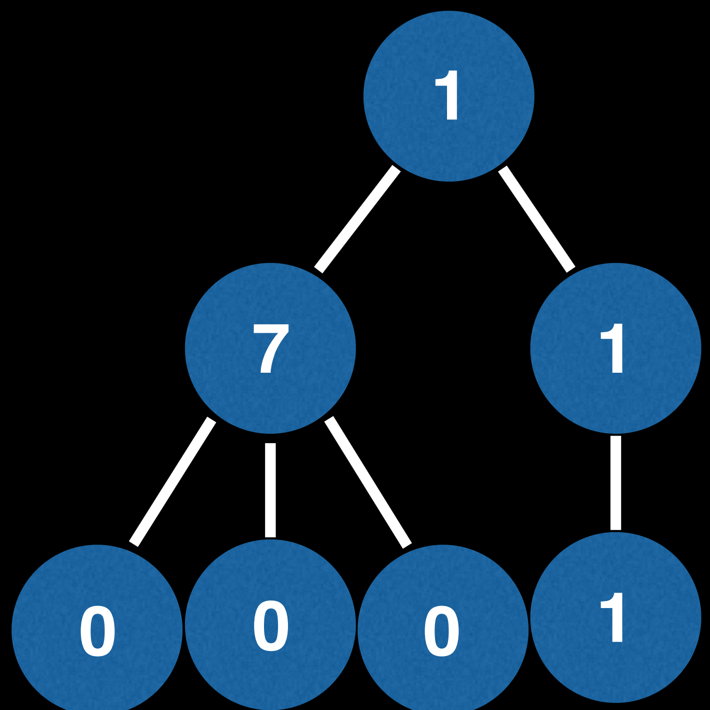


Is this a BT?

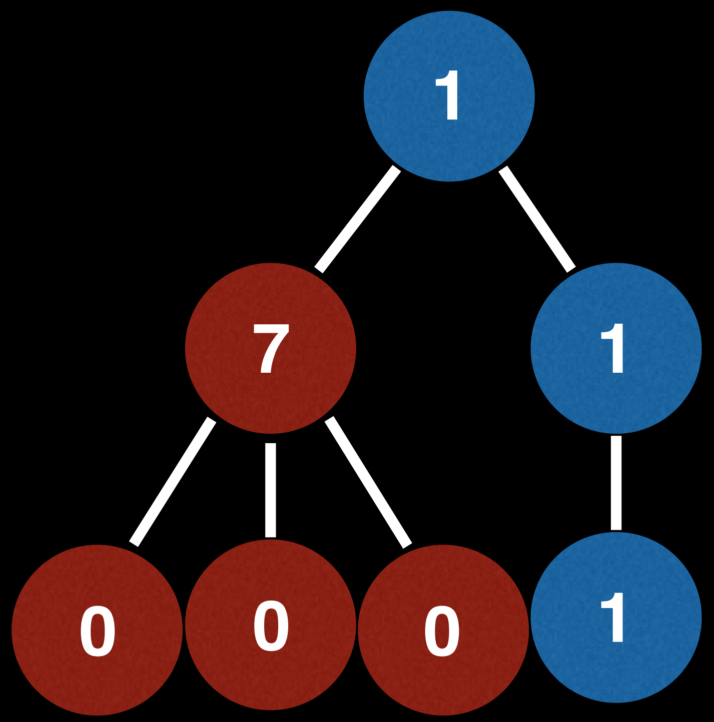


Yes!

Is this a BT?

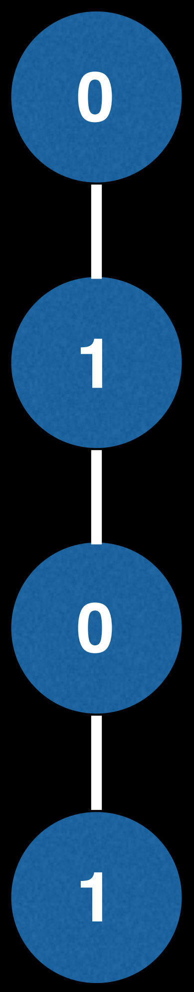


Is this a BT?

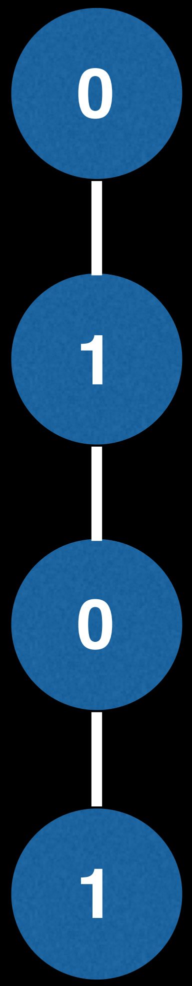


No, there is a node with three children!

Is this a BT?



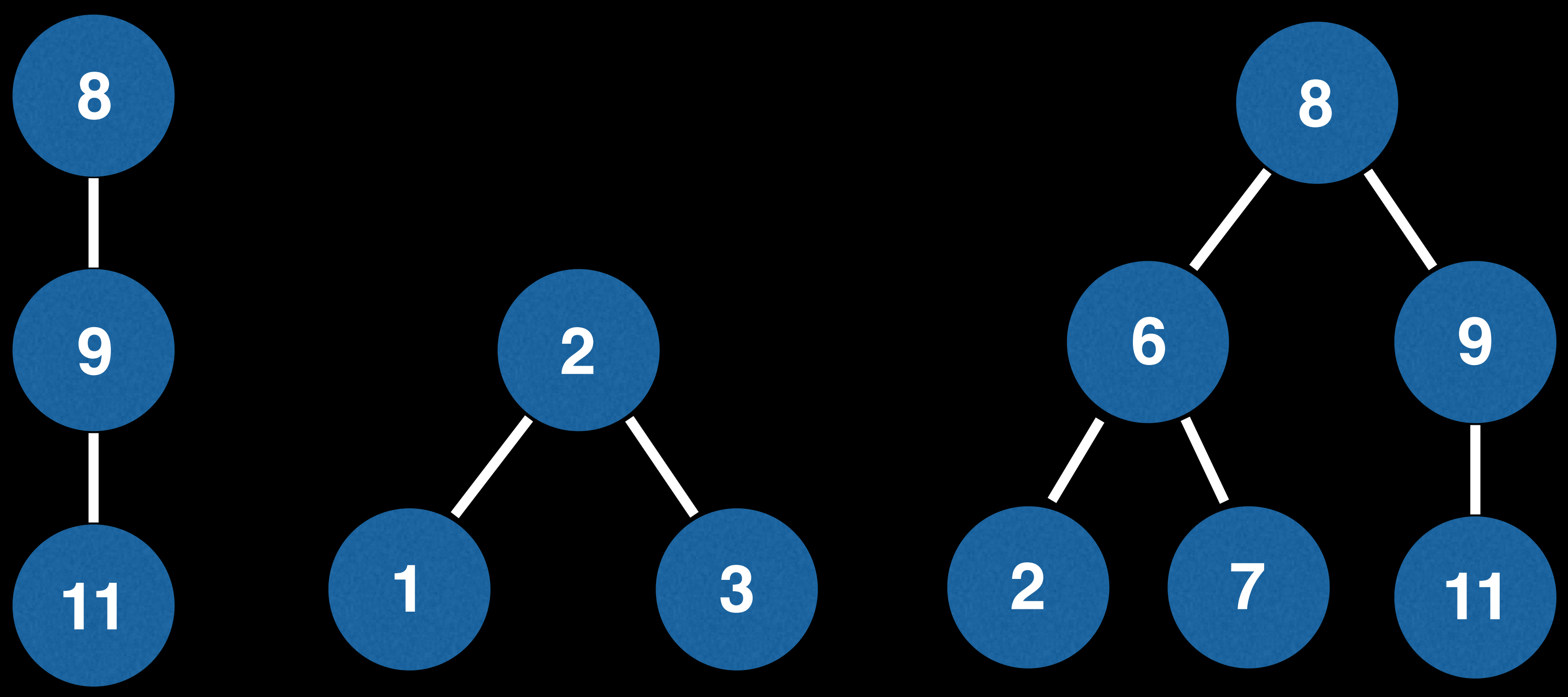
Is this a BT?



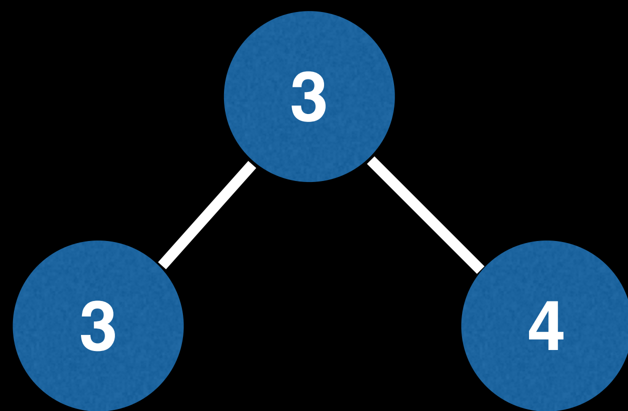
Yes! A degenerate one,
but a BT nonetheless

What is a Binary Search Tree (BST)?

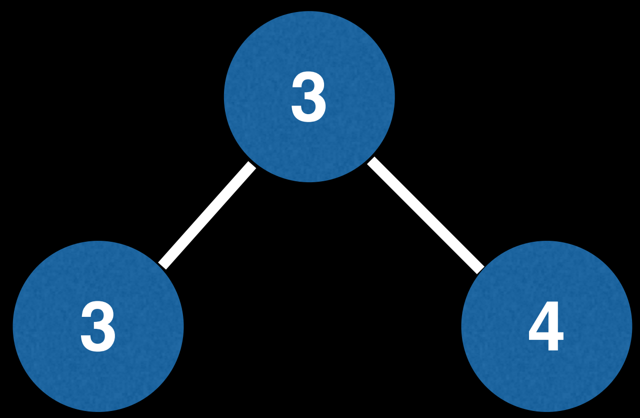
A **binary search tree** is a binary tree that satisfies the **BST invariant**: left subtree has smaller elements and right subtree has larger elements.



Is this a valid BST?

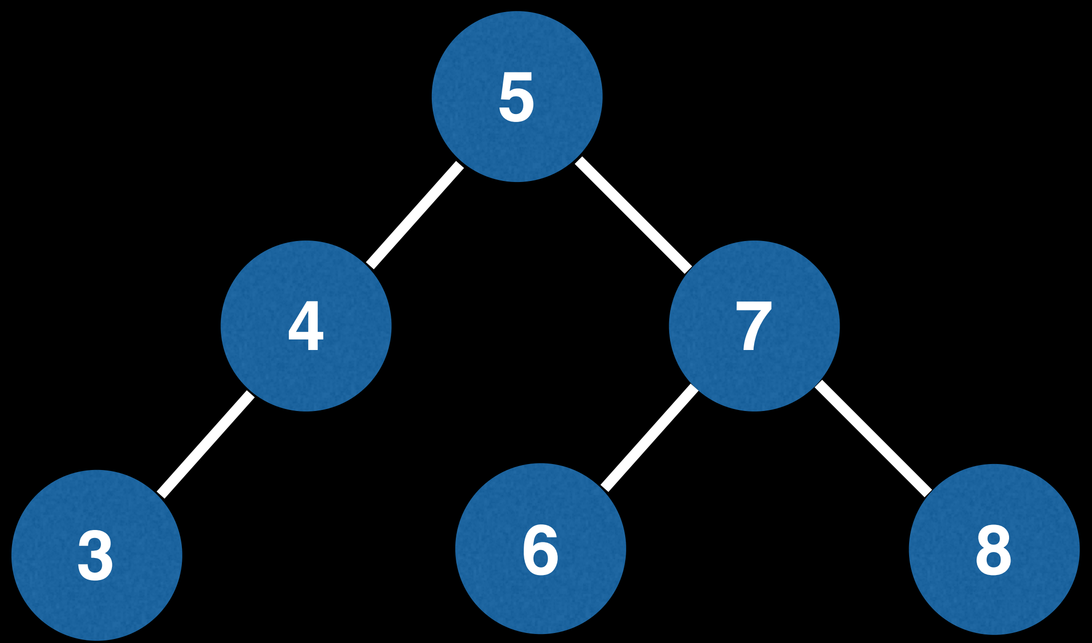


Is this a valid BST?

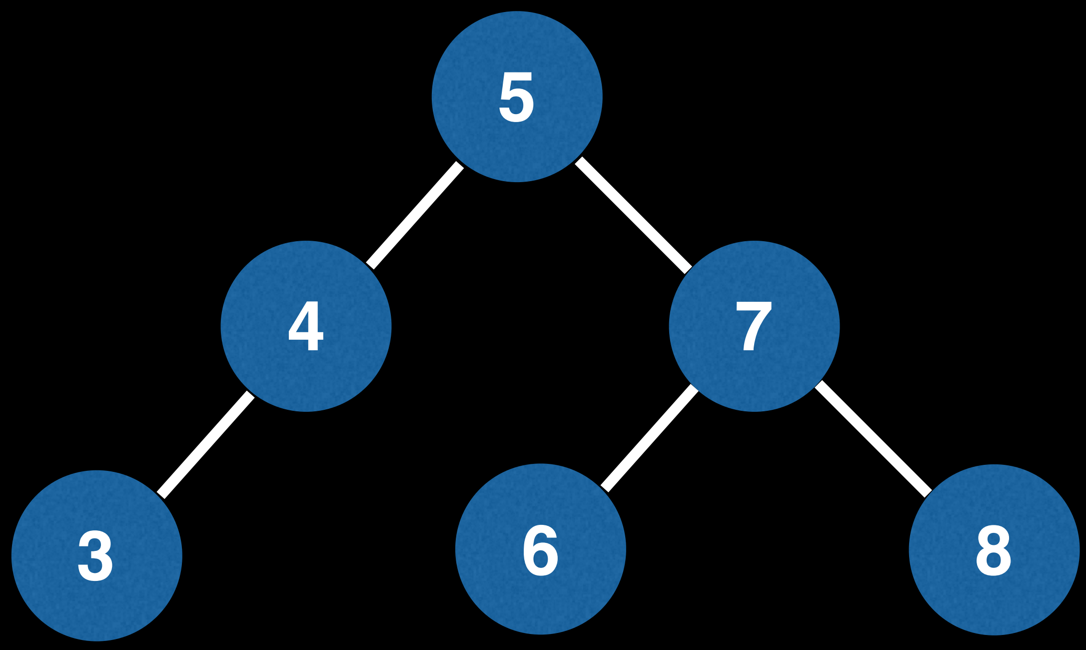


It depends on whether you want to allow duplicate values in your tree. BST operations allow for duplicate values, but most of the time we are only interested in having unique elements inside our tree.

Is this a valid BST?

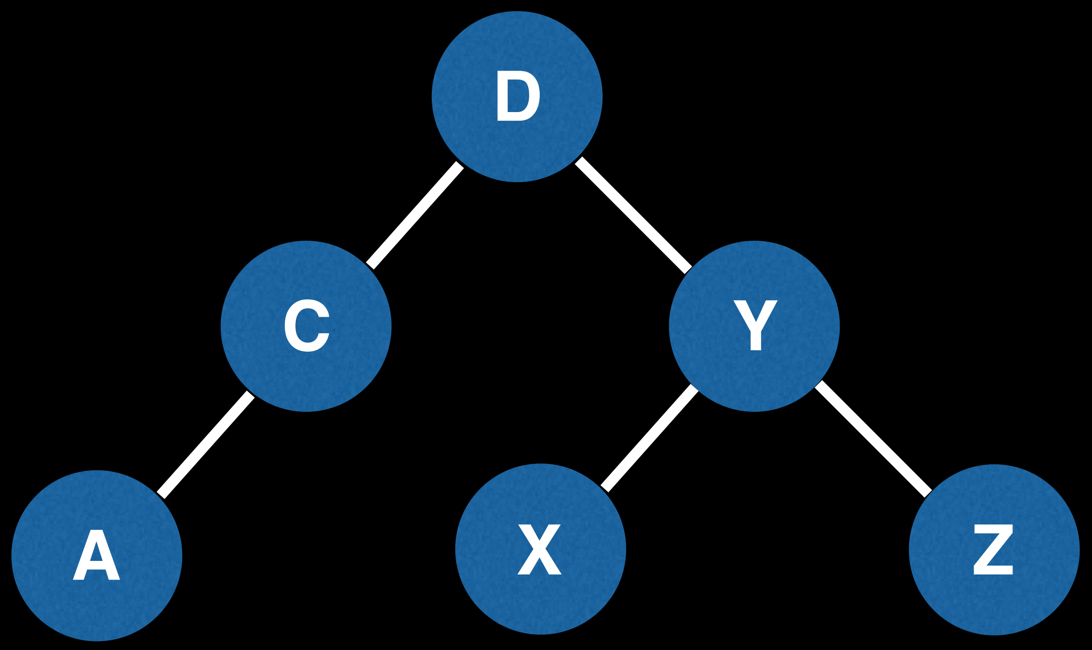


Is this a valid BST?

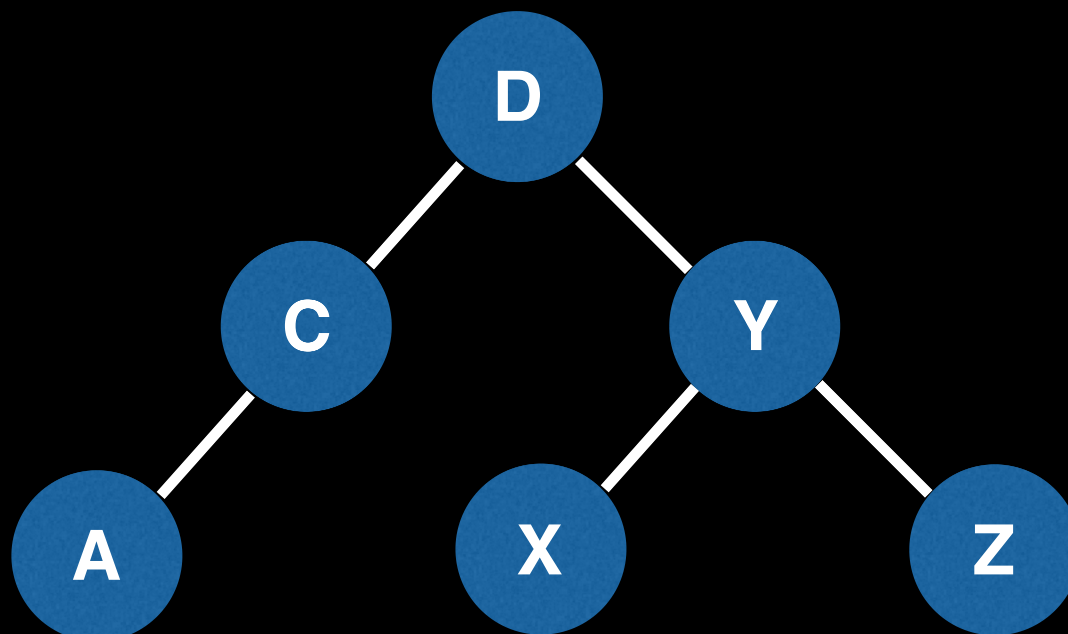


Yes!

Is this a valid BST?

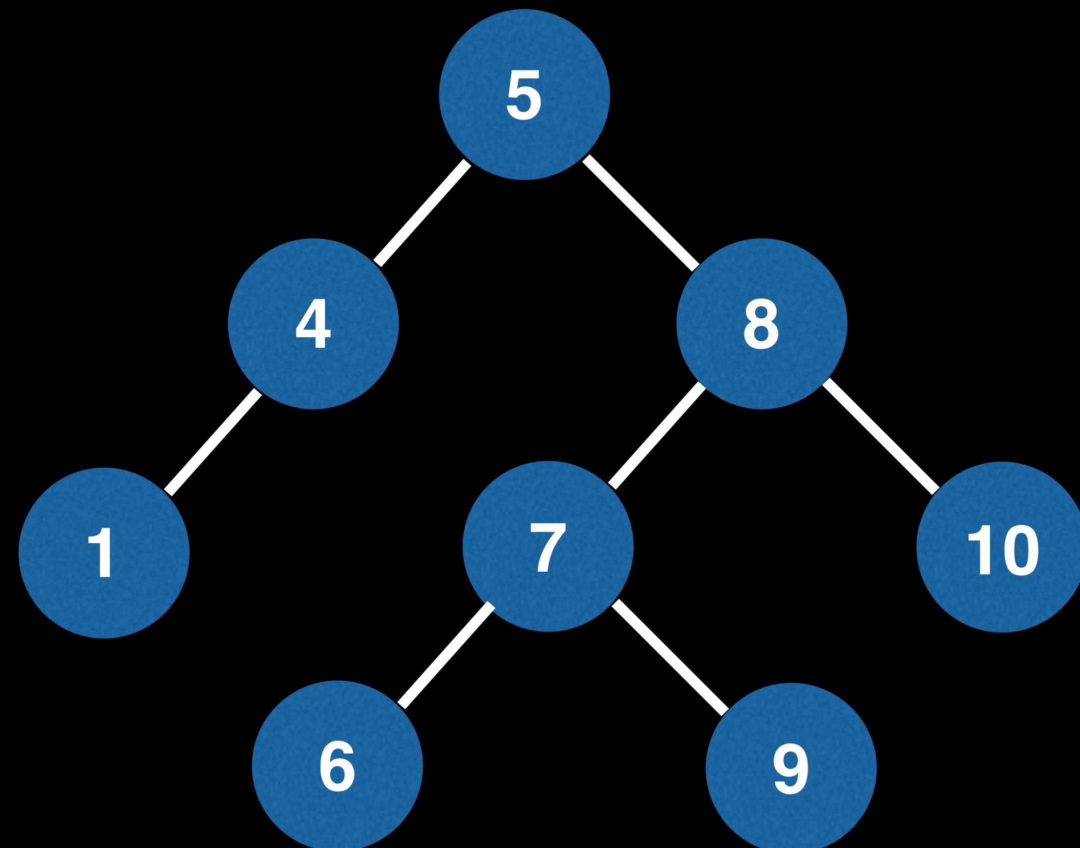


Is this a valid BST?

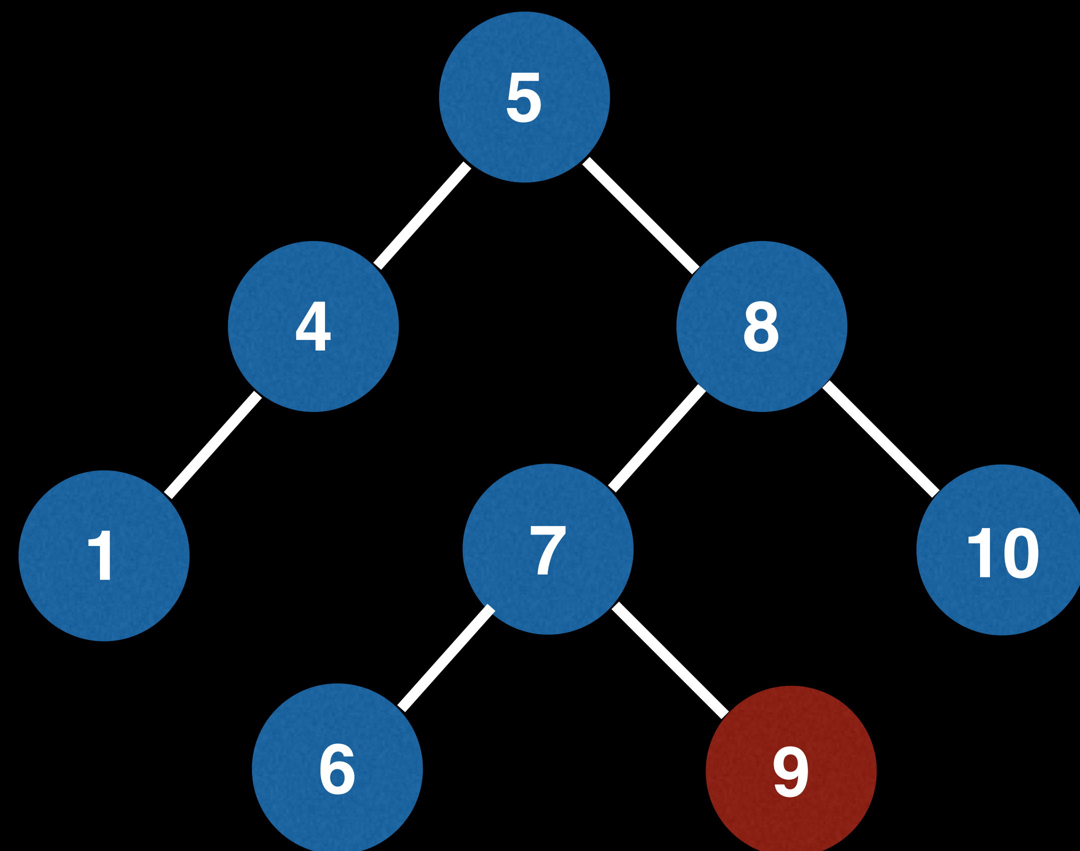


Yes! We are not limited to only using numbers. Any data that can be ordered can be placed inside a BST.

Is this a valid BST?

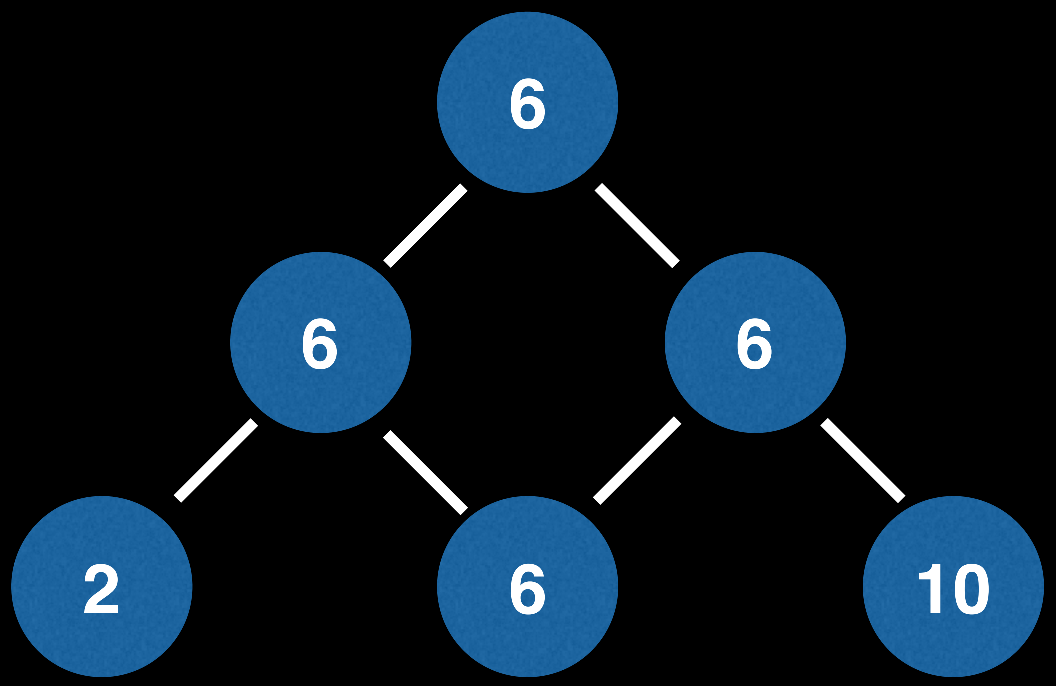


Is this a valid BST?

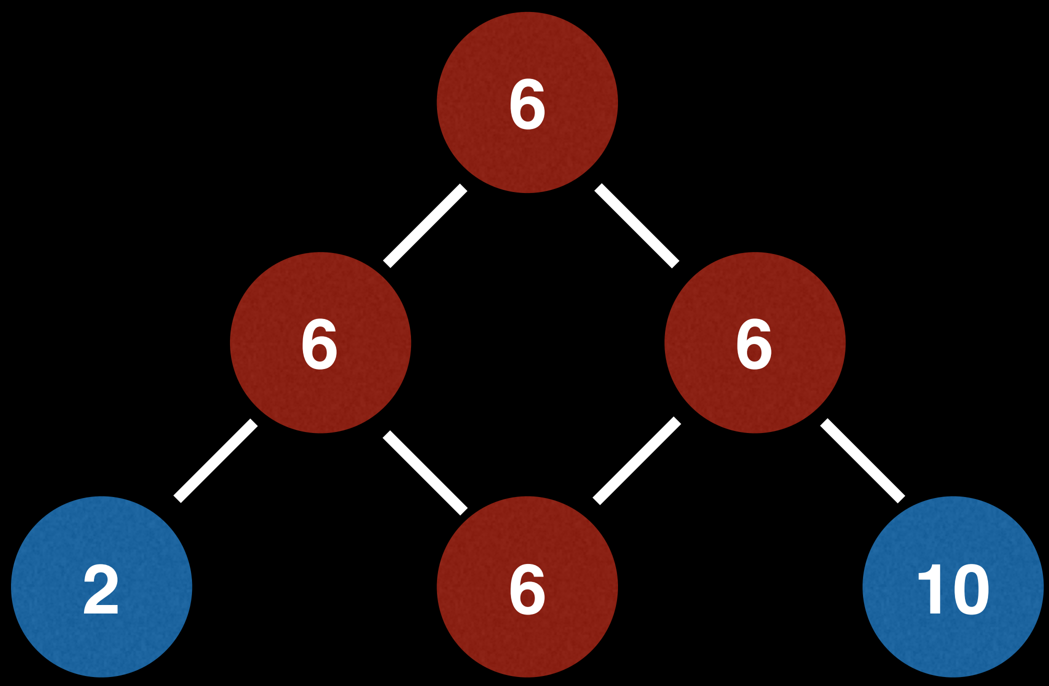


No! Since 9 is larger than 8 then it should be in the right subtree of 8.

Is this a valid BST?

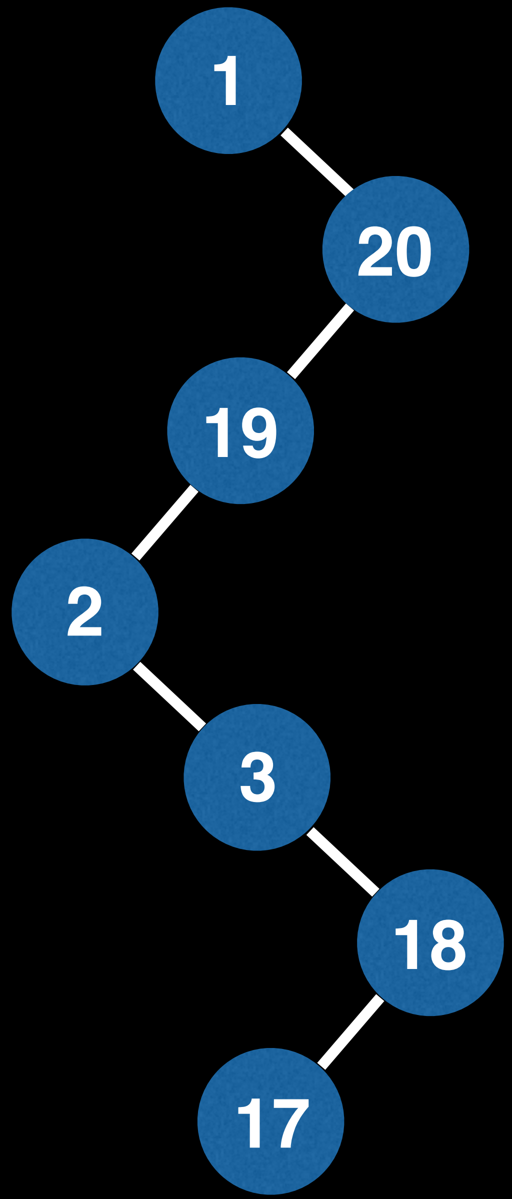


Is this a valid BST?

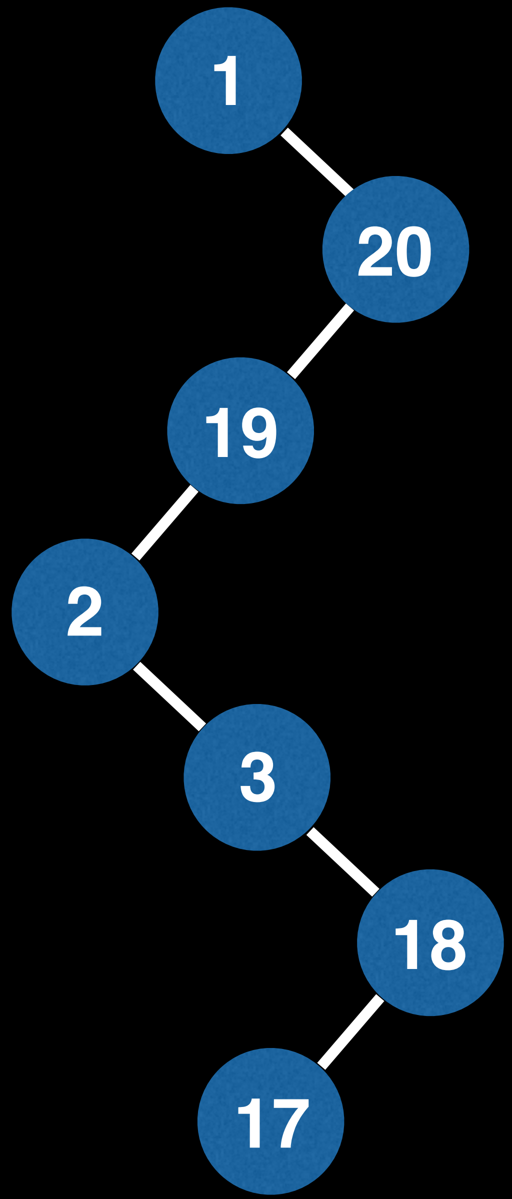


No! This structure is not a tree because it contains a cycle, and all BSTs must be trees.

Is this a valid BST?



Is this a valid BST?



Yes! This structure satisfies the BST invariant.

When and where are Binary Trees used?

- Binary Search Trees (BSTs)
  - Implementation of some map and set ADTs
  - Red Black Trees
  - AVL Trees
  - Splay Trees
  - etc…
- Used in the implementation of binary heaps
- Syntax trees (used by compiler and calculators)
- Treap – a probabilistic DS (uses a randomized BST)

Complexity of BSTs

Operation	Average	Worst
Insert	O(log(n))	O(n)
Delete	O(log(n))	O(n)
Remove	O(log(n))	O(n)
Search	O(log(n))	O(n)

Inserting elements into a Binary Search Tree (BST)

Adding elements to a BST

Binary Search Tree (BST) elements must be **comparable** so that we can order them inside the tree.

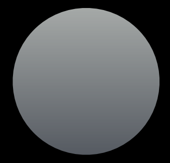
When inserting an element we want to compare its value to the value stored in the current node we're considering to decide on one of the following:

- Recurse down left subtree (< case)
- Recurse down right subtree (> case)
- Handle finding a duplicate value (= case)
- Create a new node (found a null leaf)

Adding elements to a BST

Instructions:

```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```

Adding elements to a BST

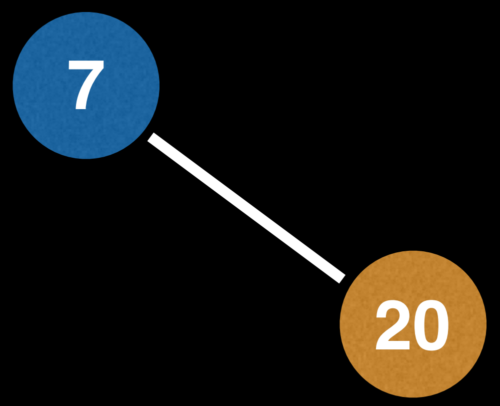
Instructions:

```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```

Adding elements to a BST

Instructions:

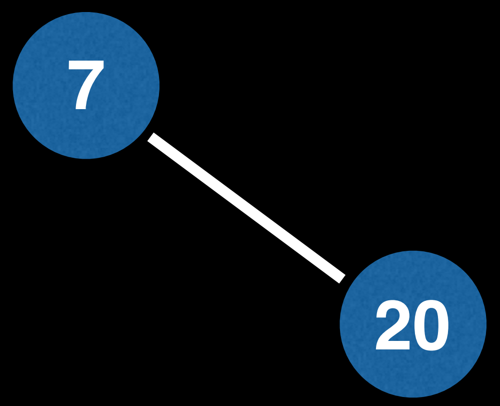
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

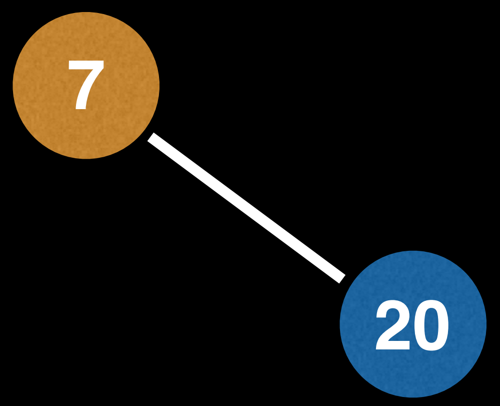
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

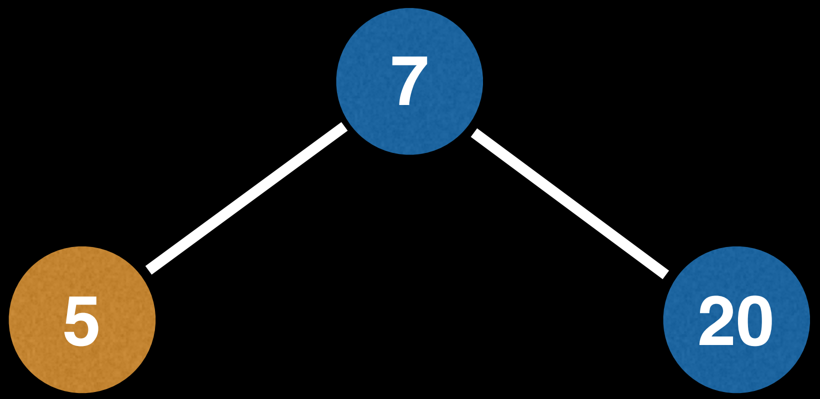
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

<u>Instructions</u>:

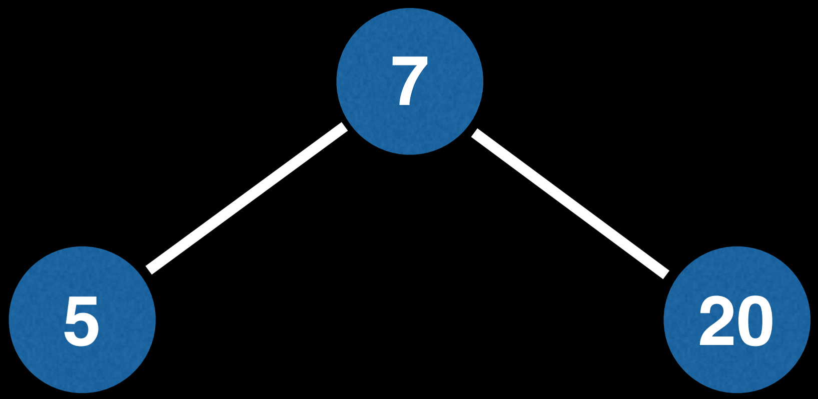
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

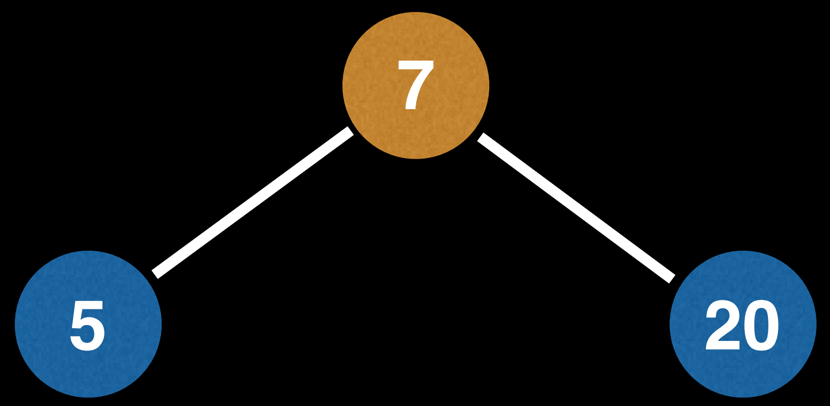
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

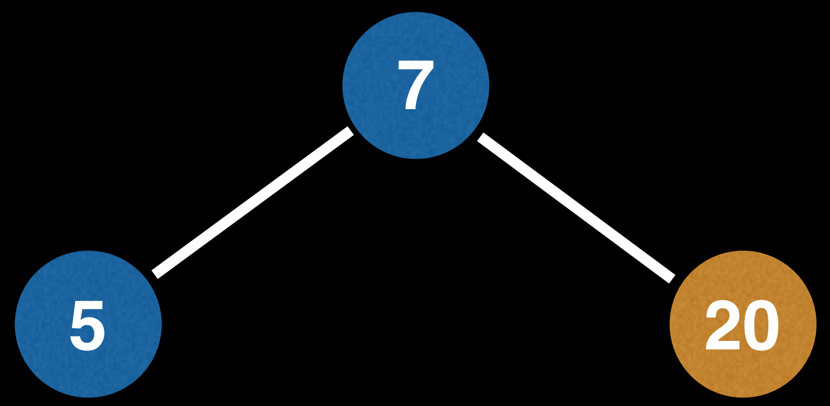
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

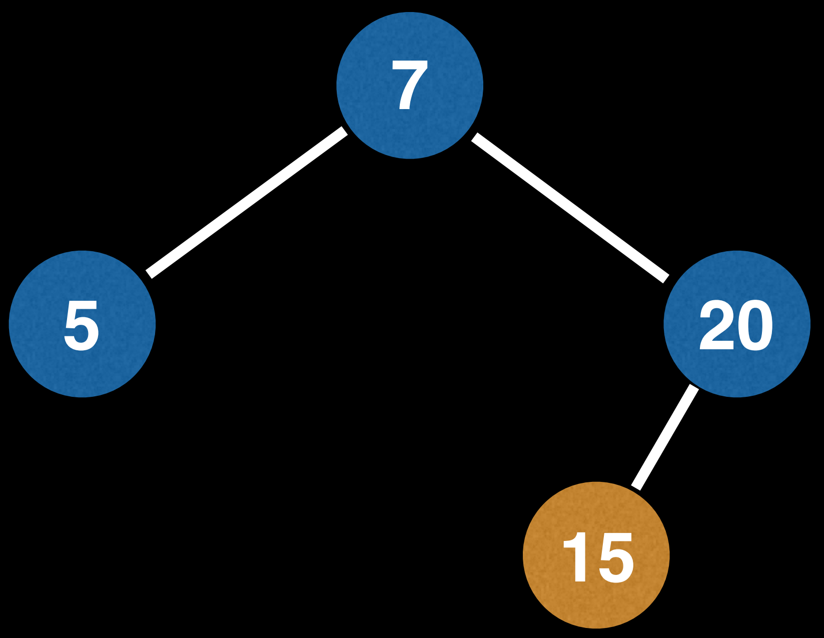
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

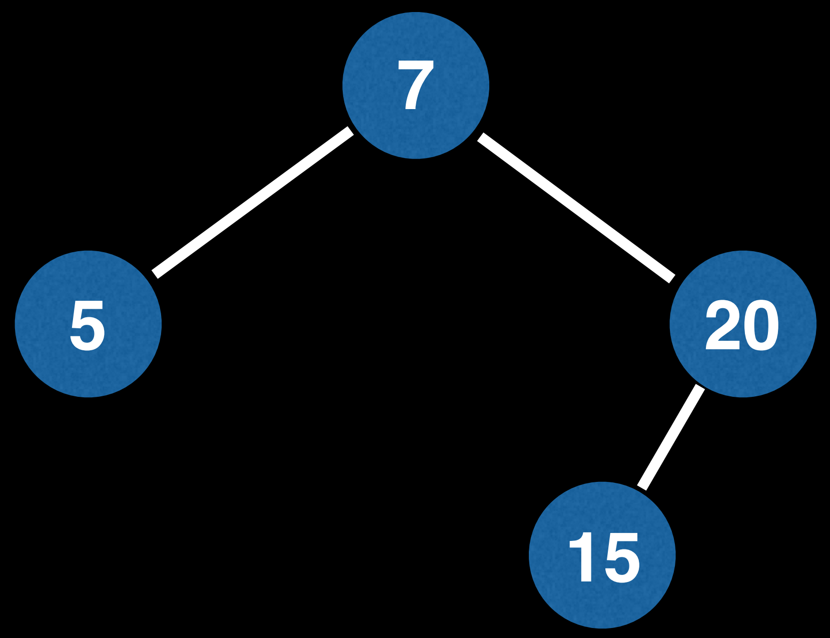
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

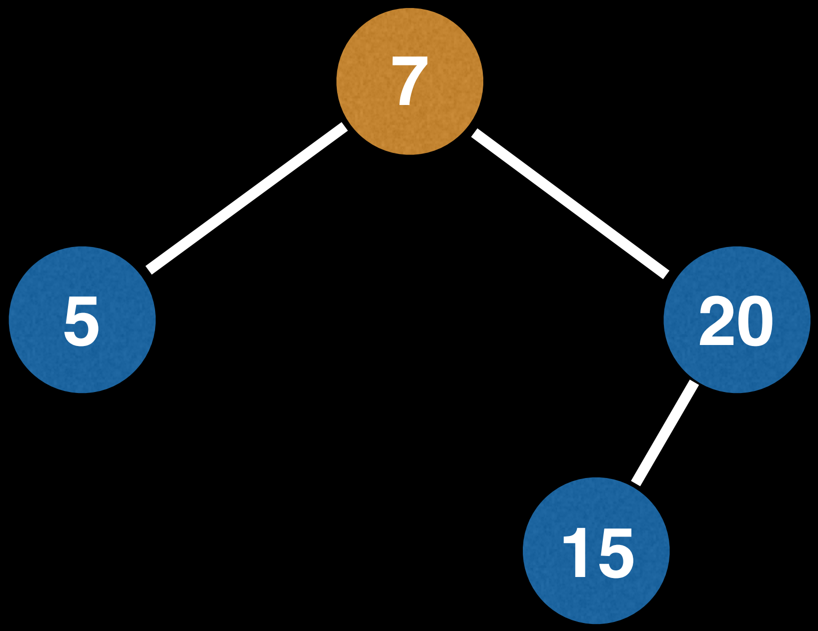
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

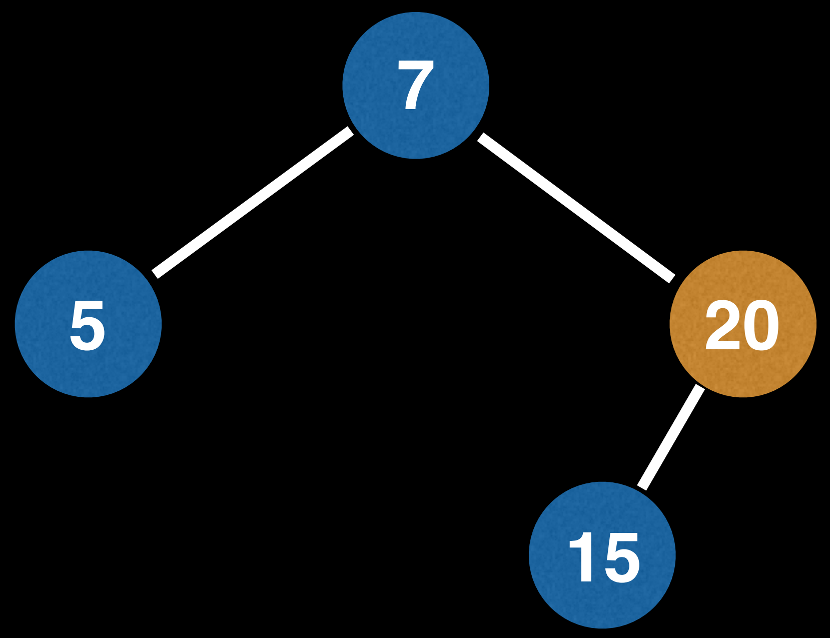
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)



Adding elements to a BST

Instructions:

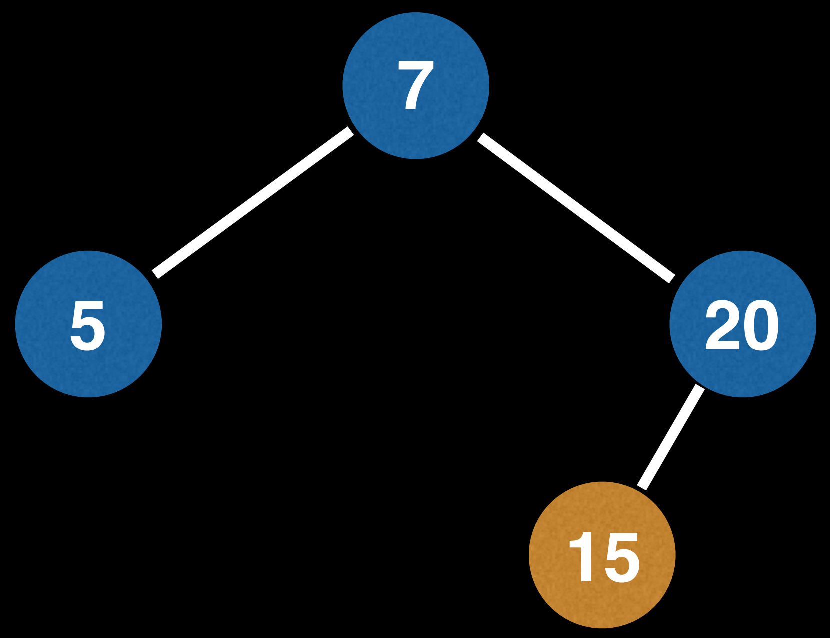
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

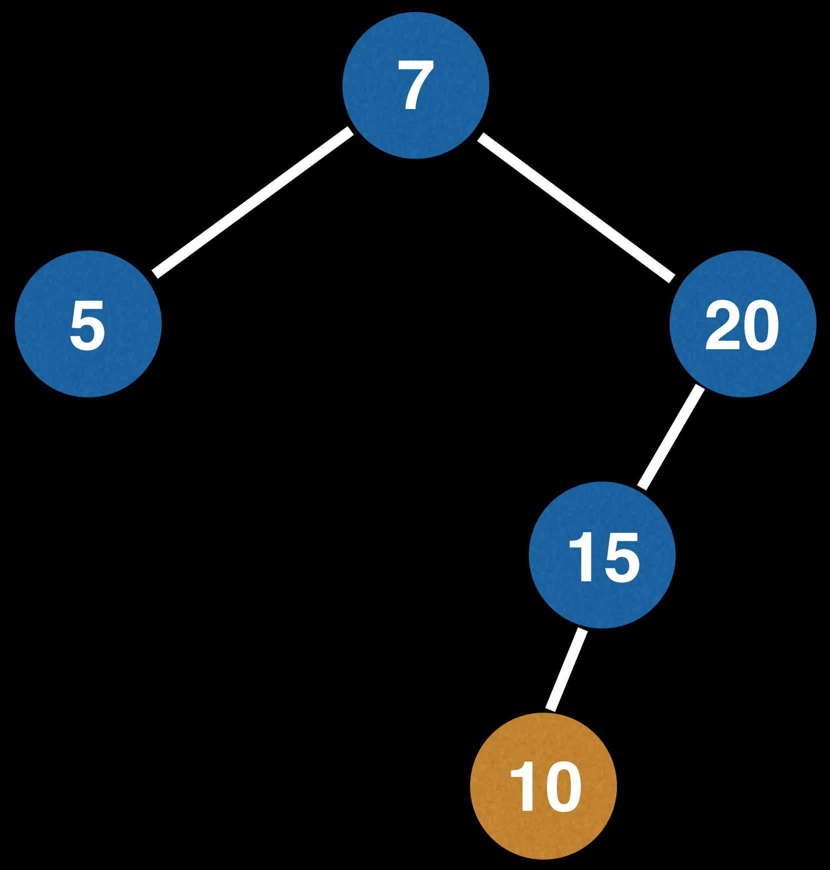
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

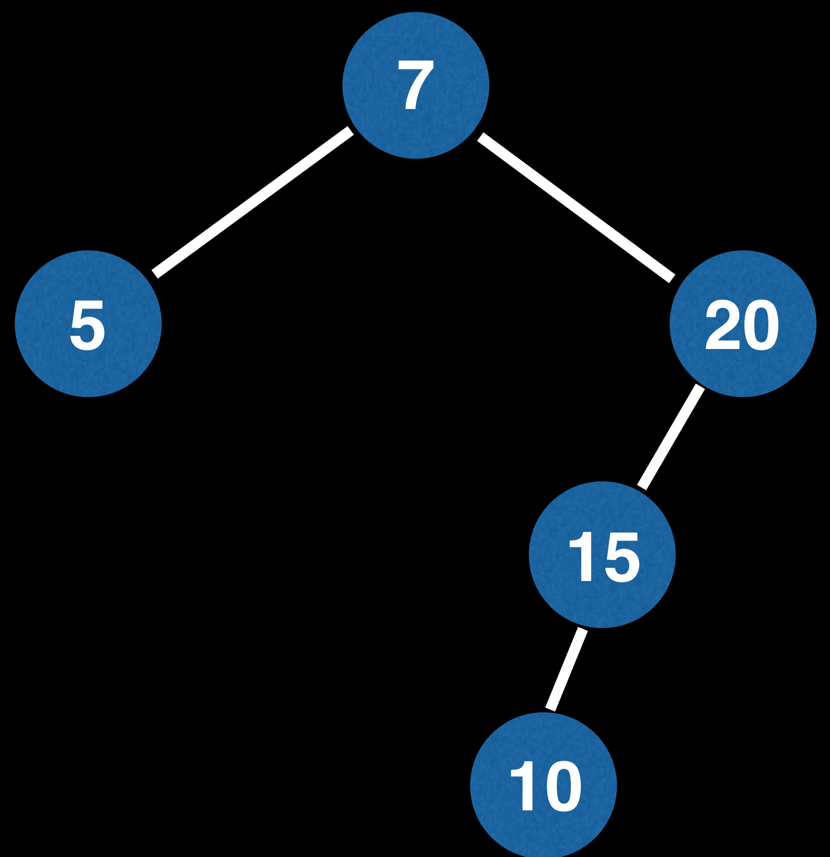
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

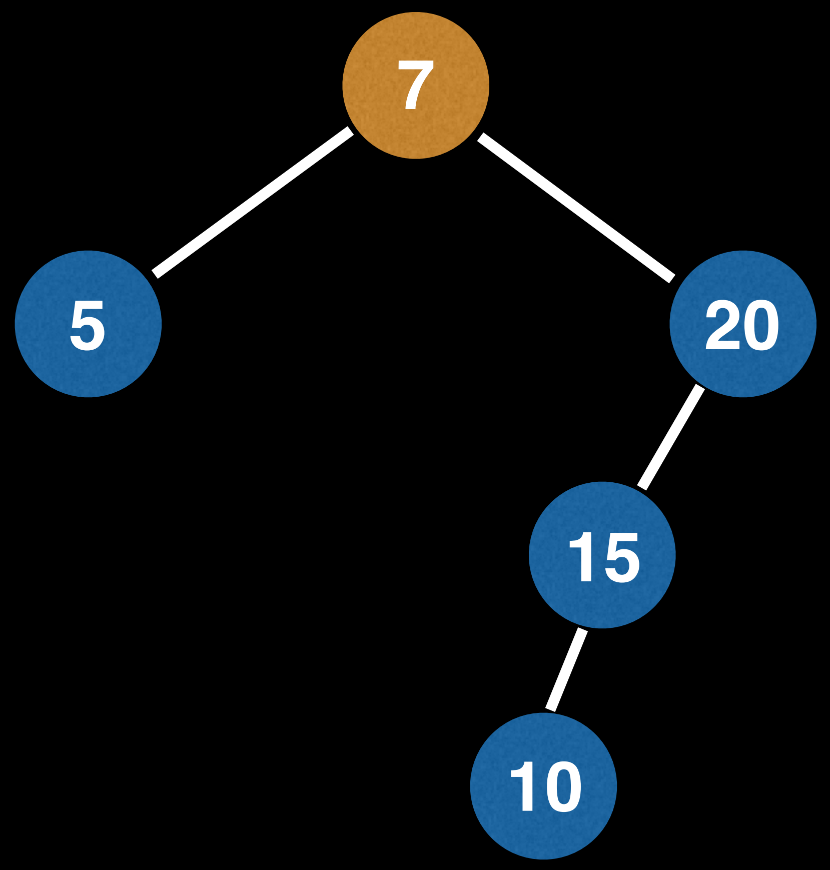
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

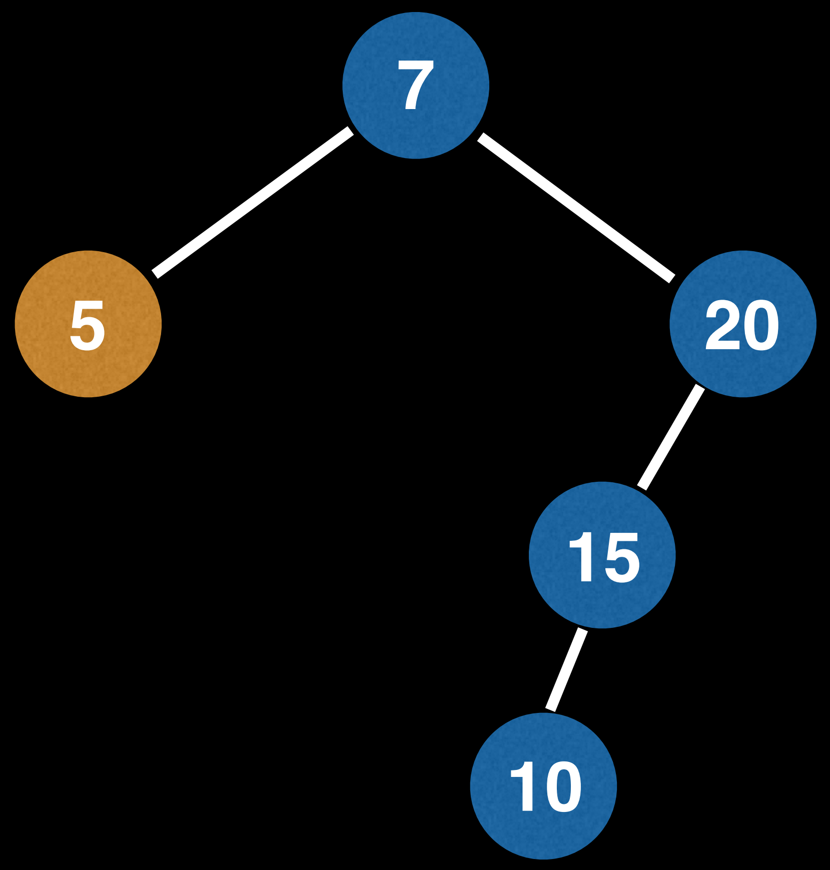
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

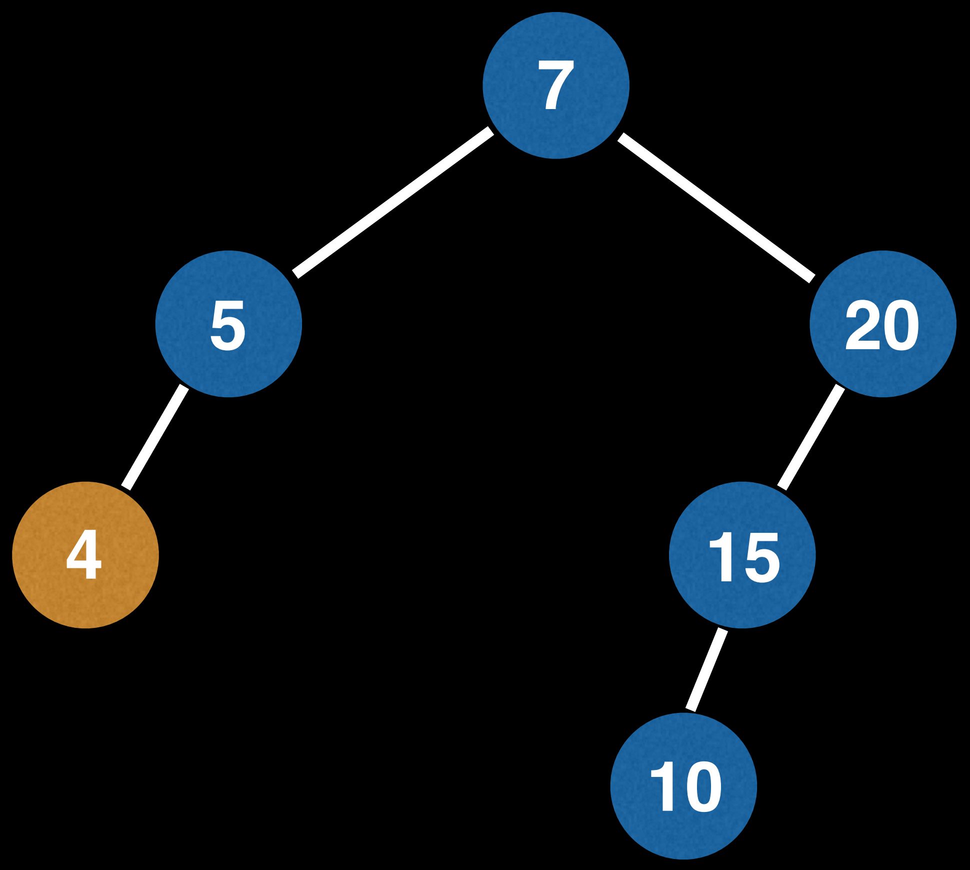
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

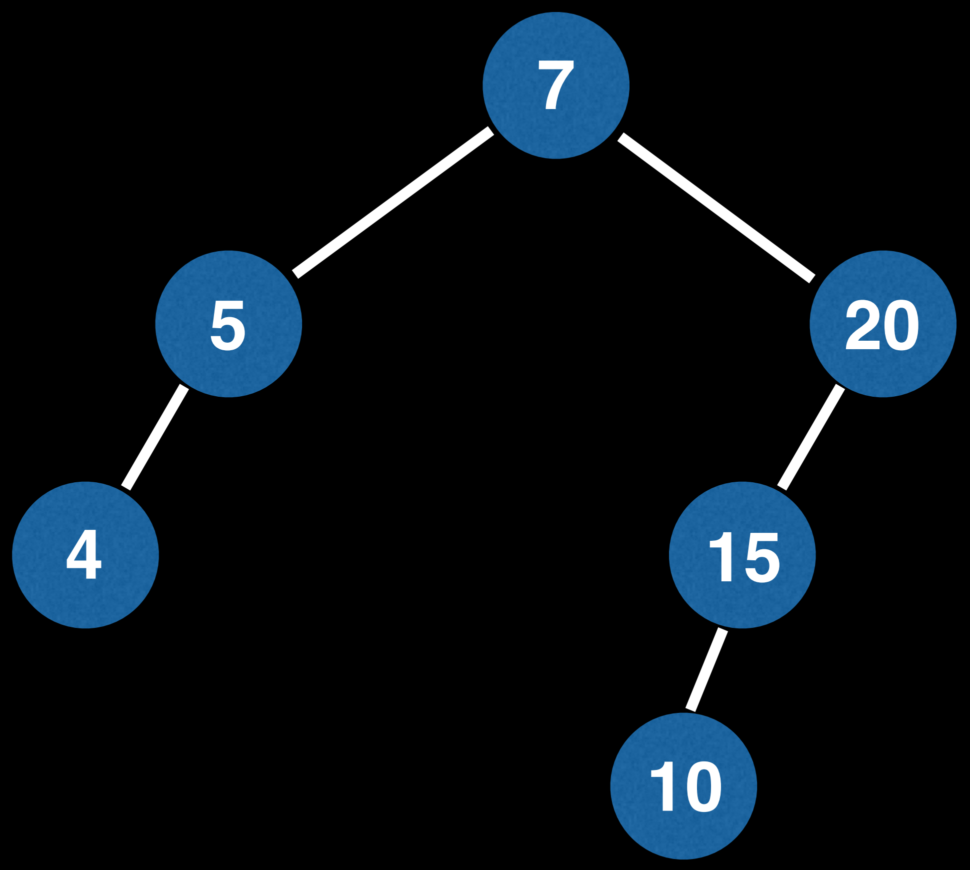
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

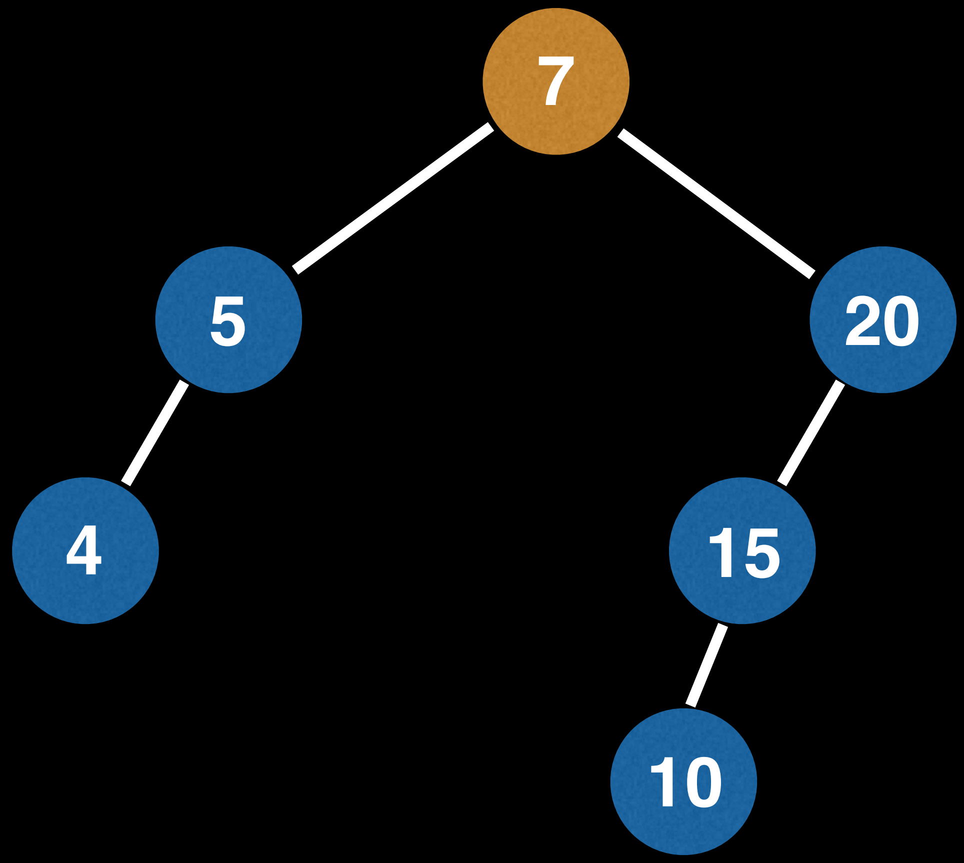
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

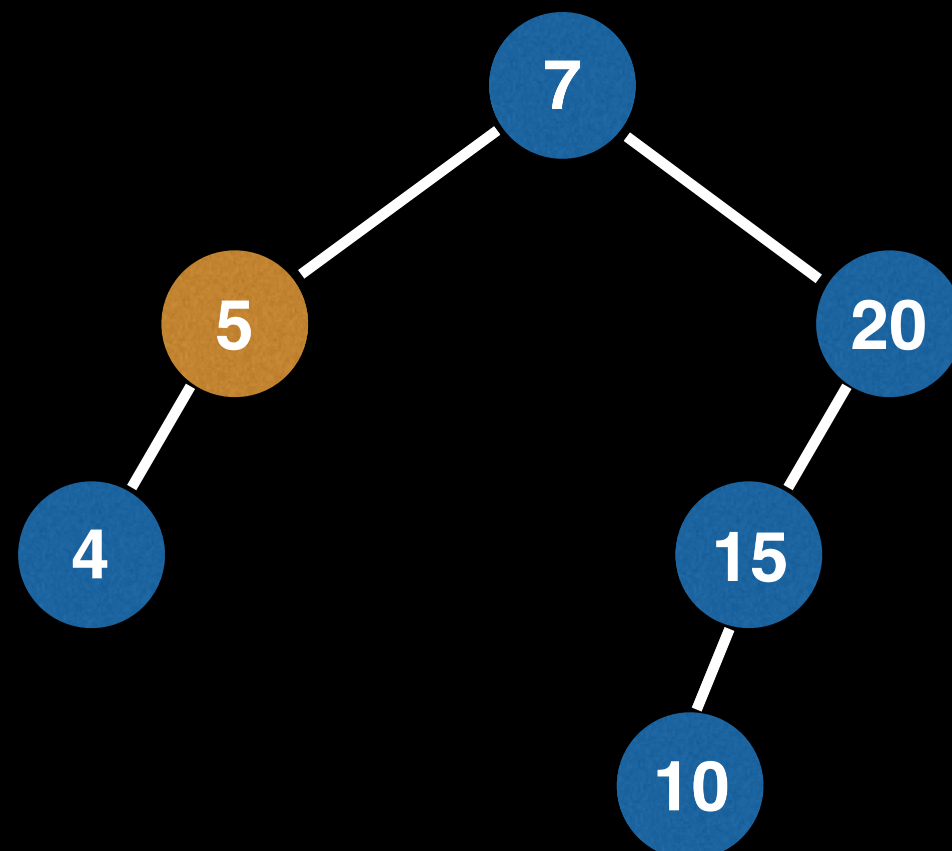
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)



Adding elements to a BST

Instructions:

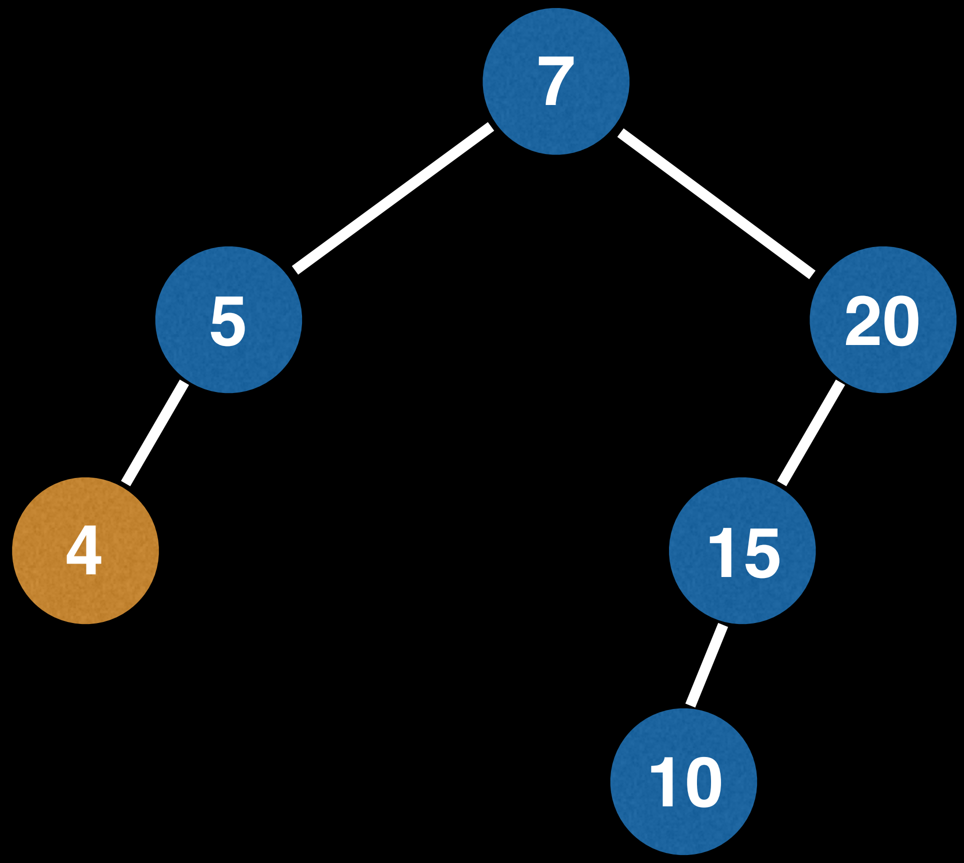
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)



Adding elements to a BST

Instructions:

insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)

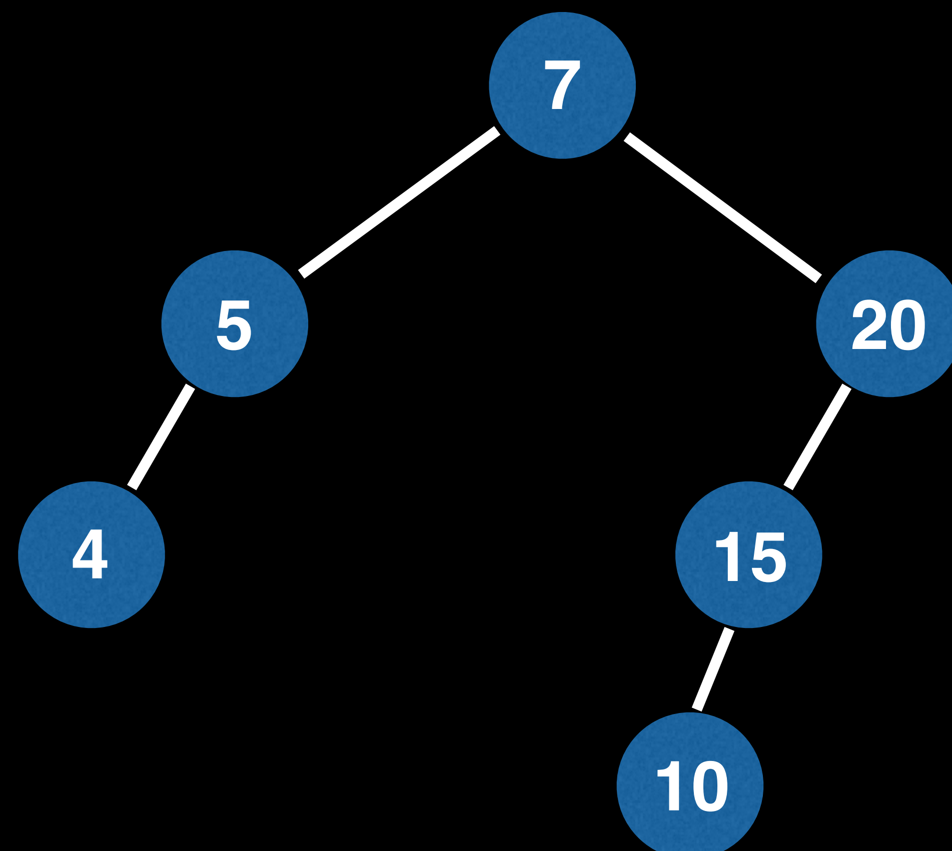


We have encountered a value that is already in the tree. If your tree supports duplicate values then add another node, otherwise do nothing.

Adding elements to a BST

Instructions:

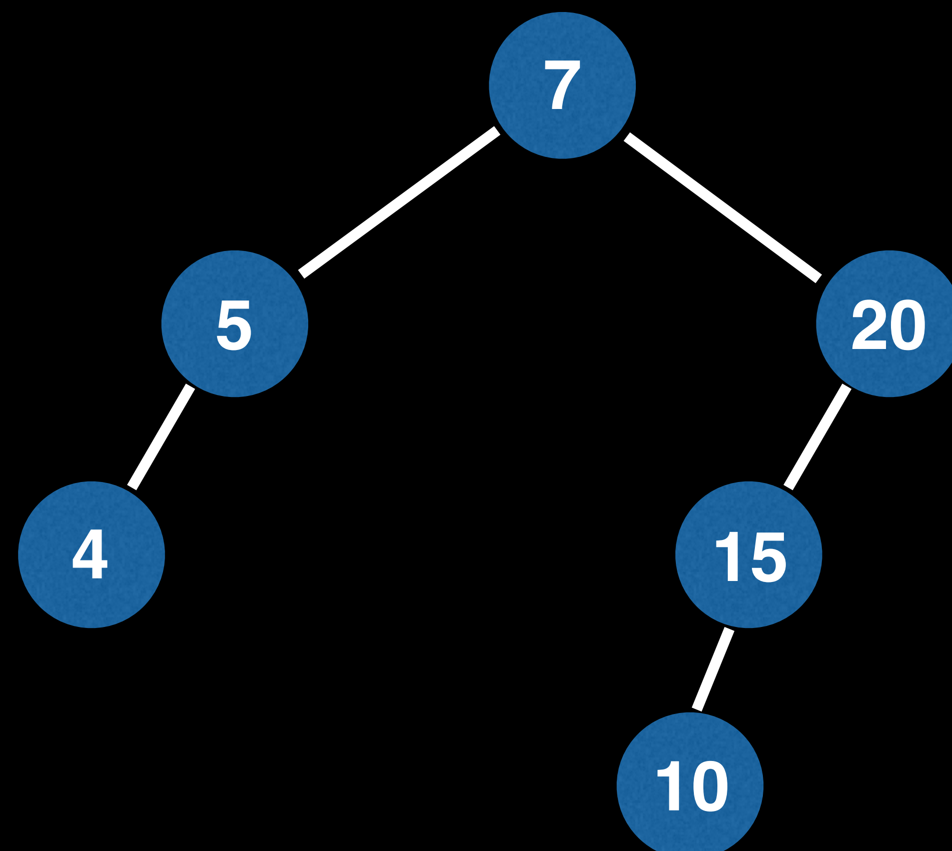
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

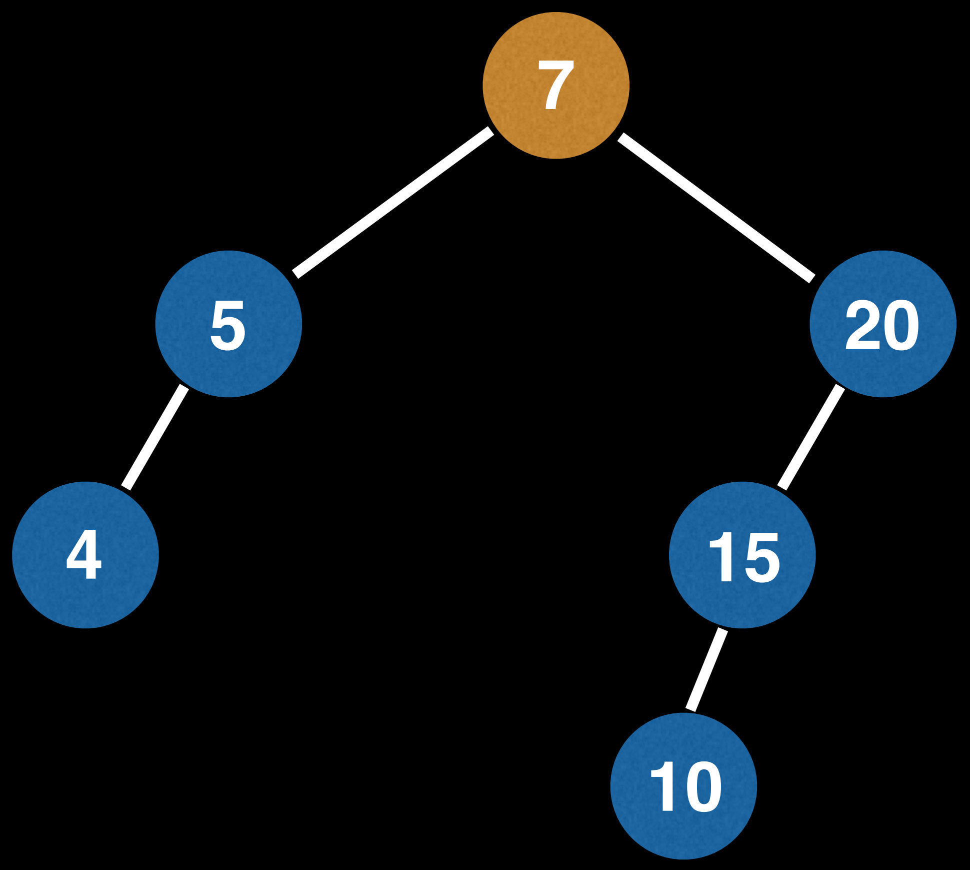
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

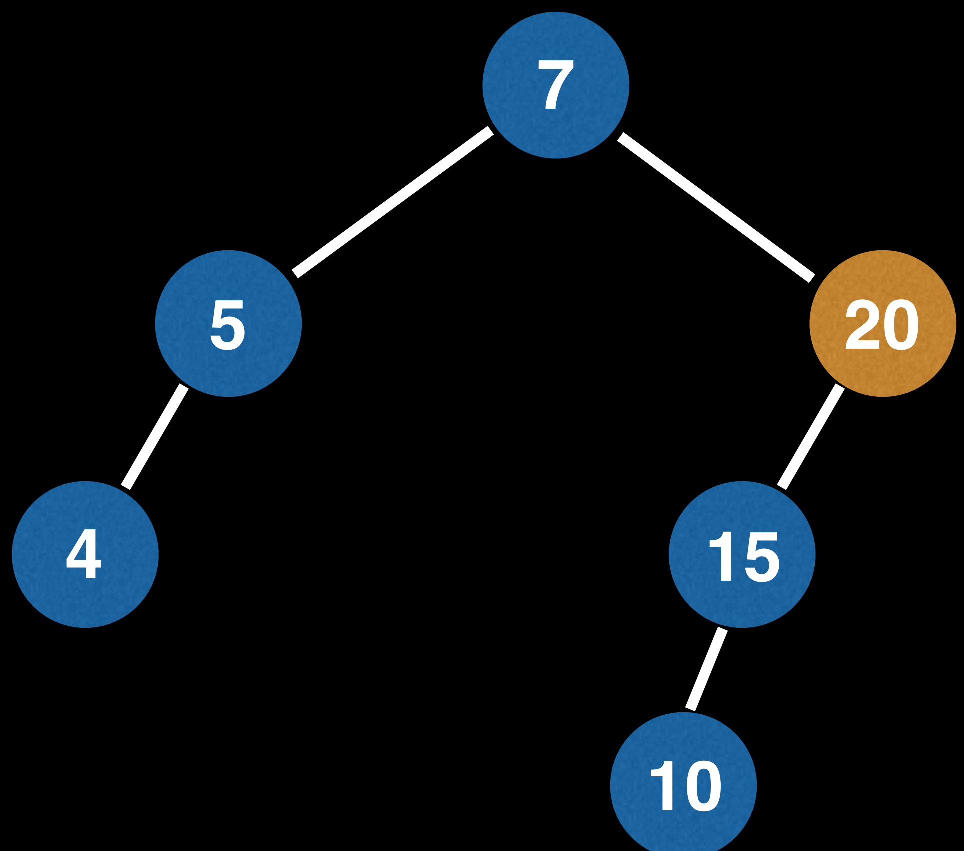
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

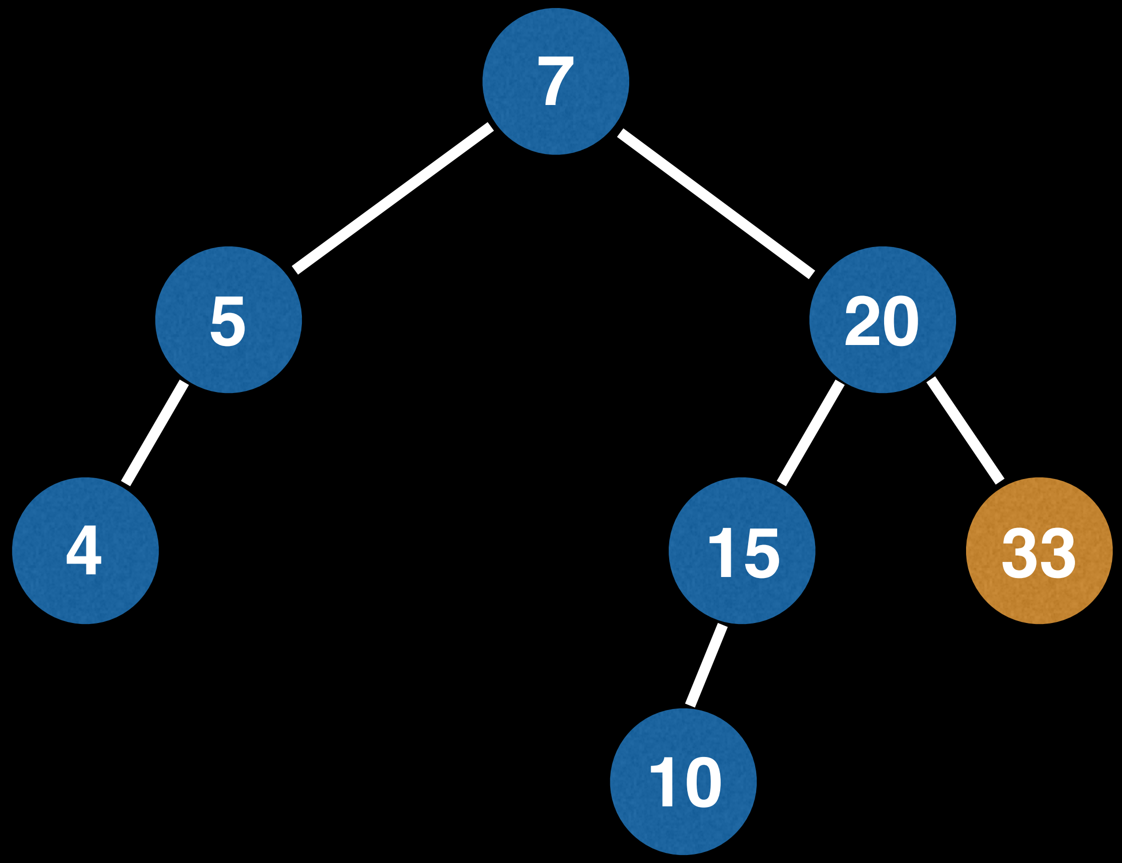
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

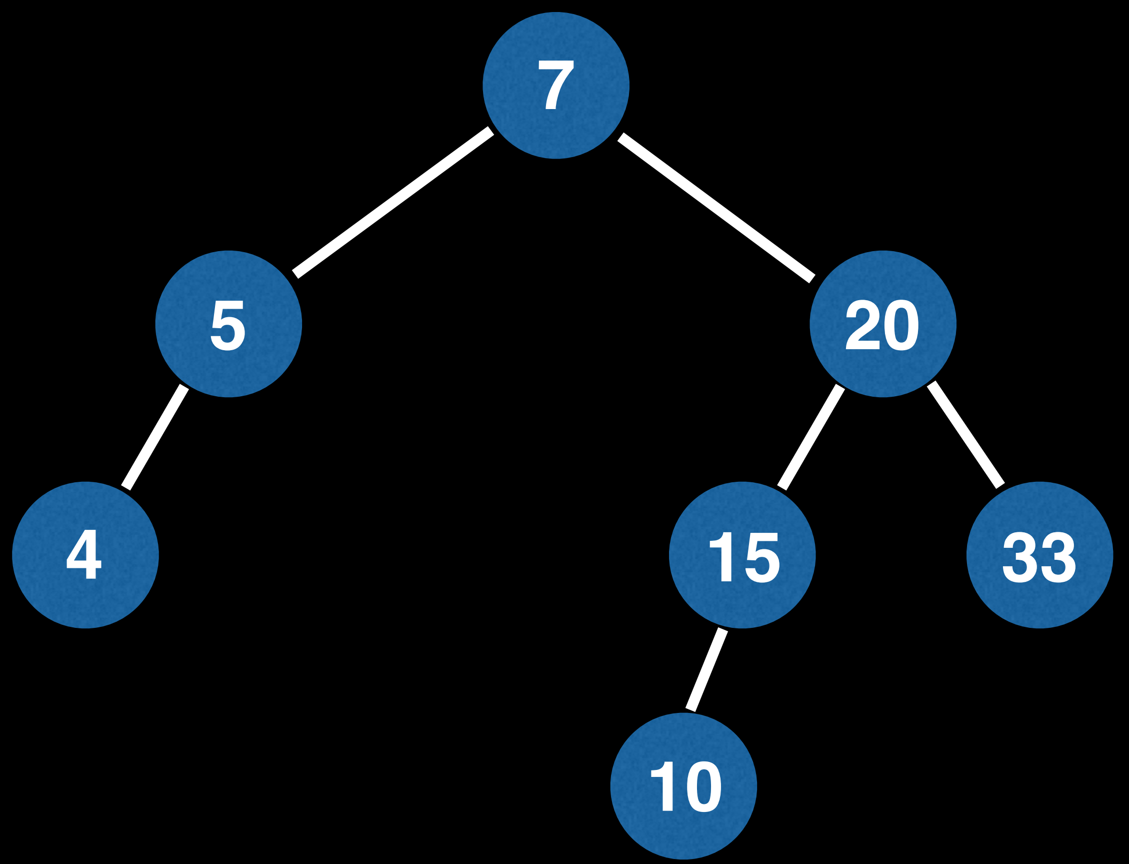
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

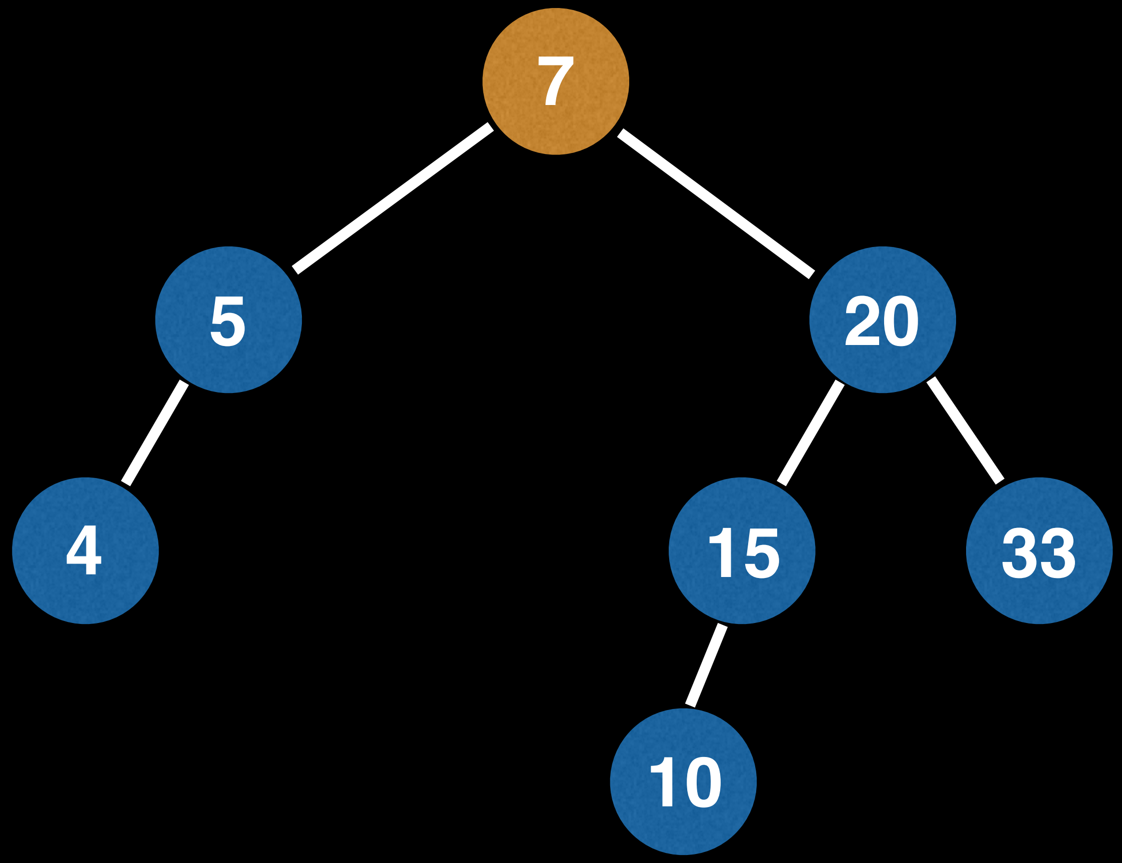
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

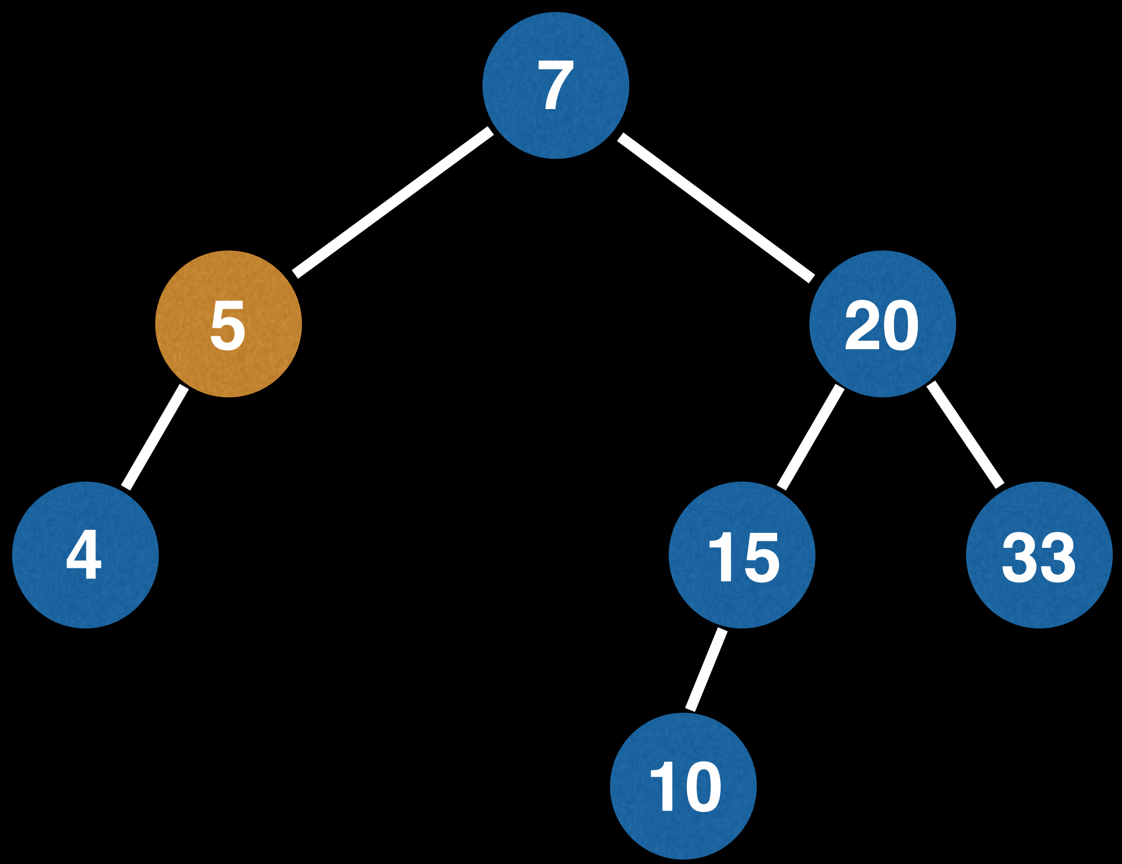
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

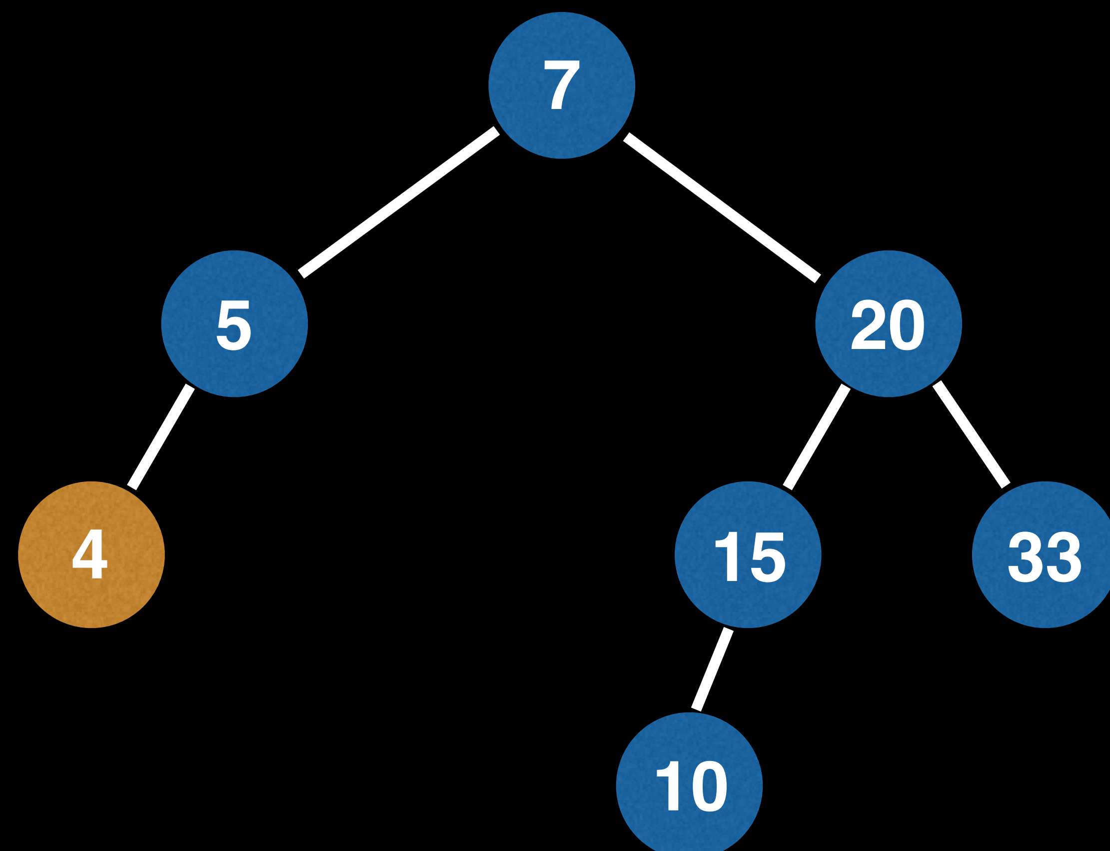
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

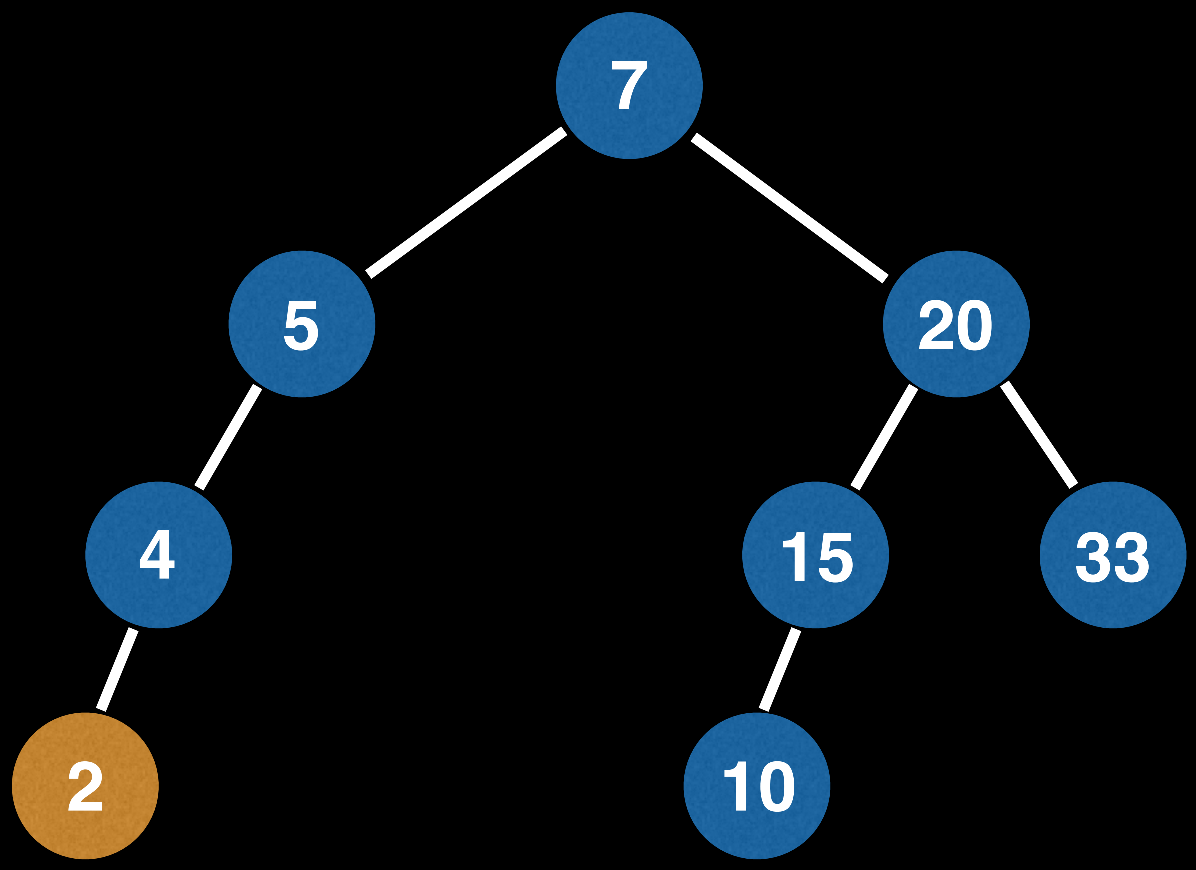
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

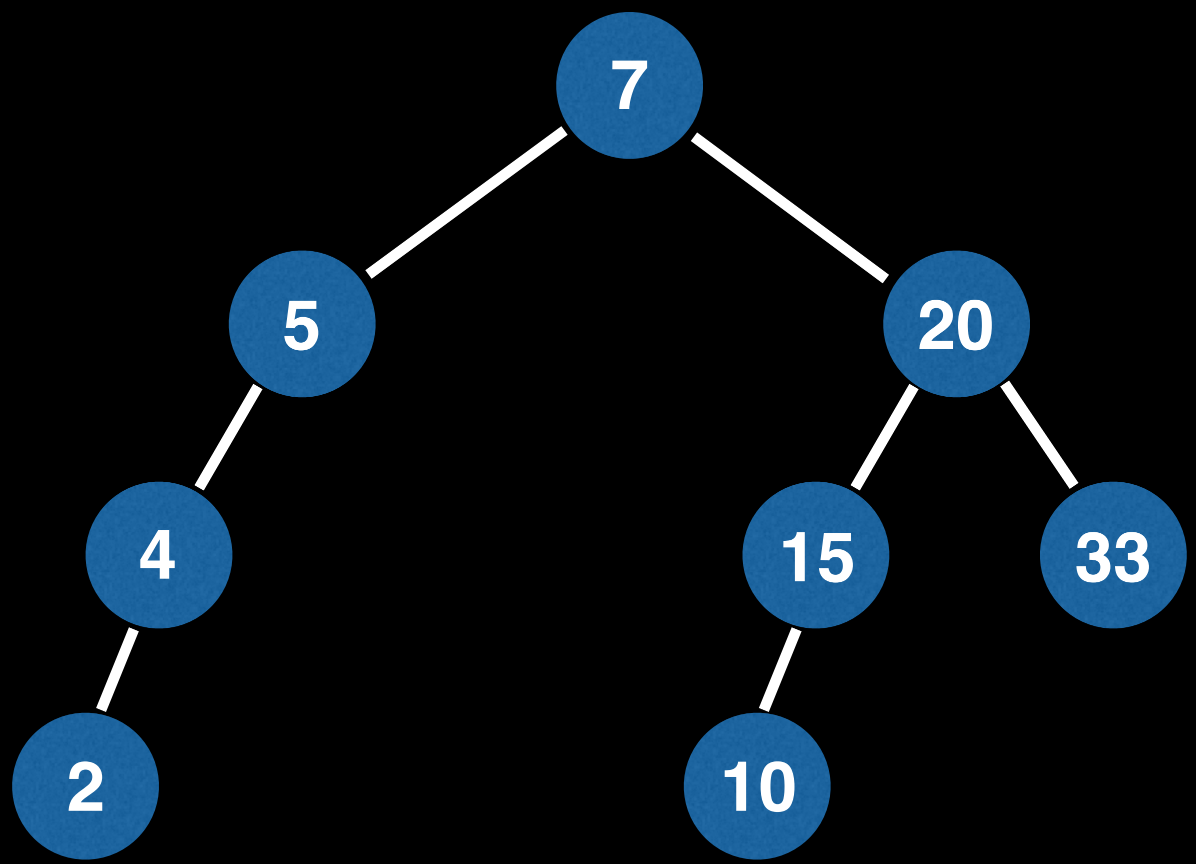
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

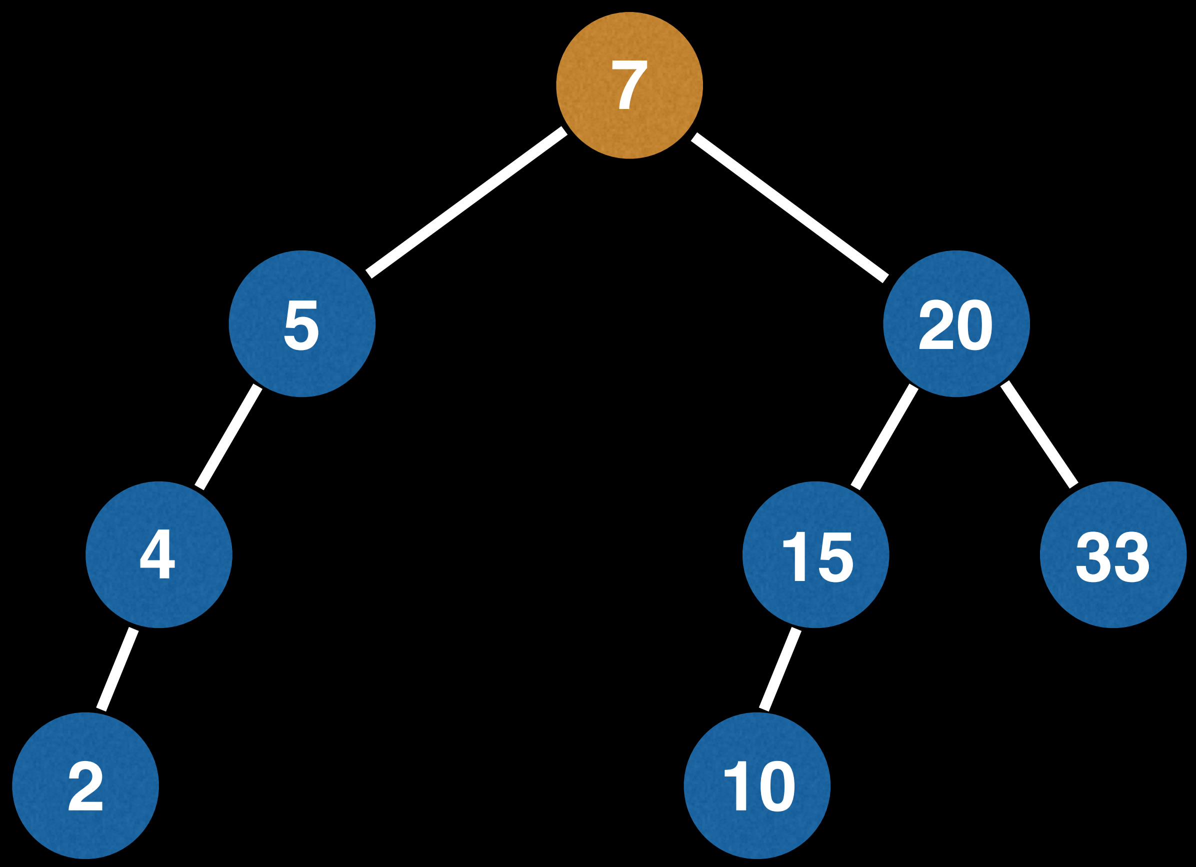
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

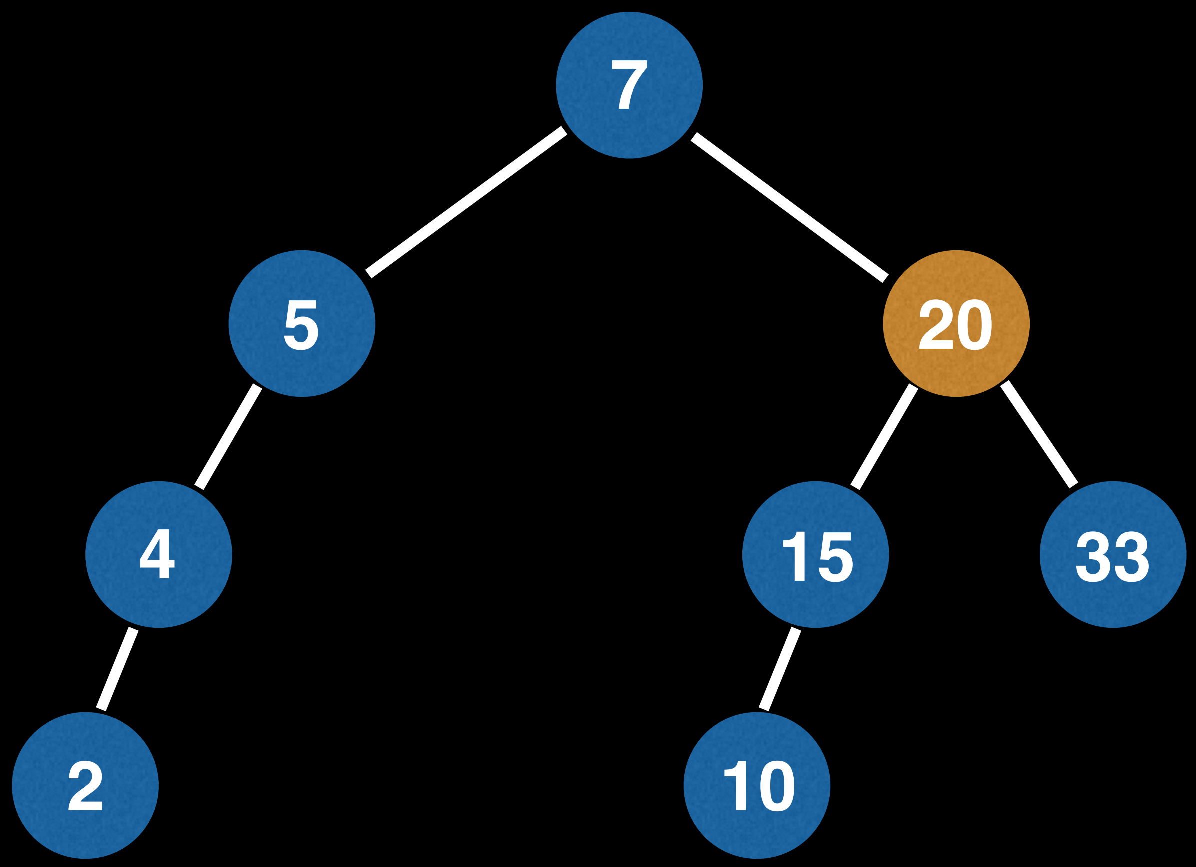
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)



Adding elements to a BST

Instructions:

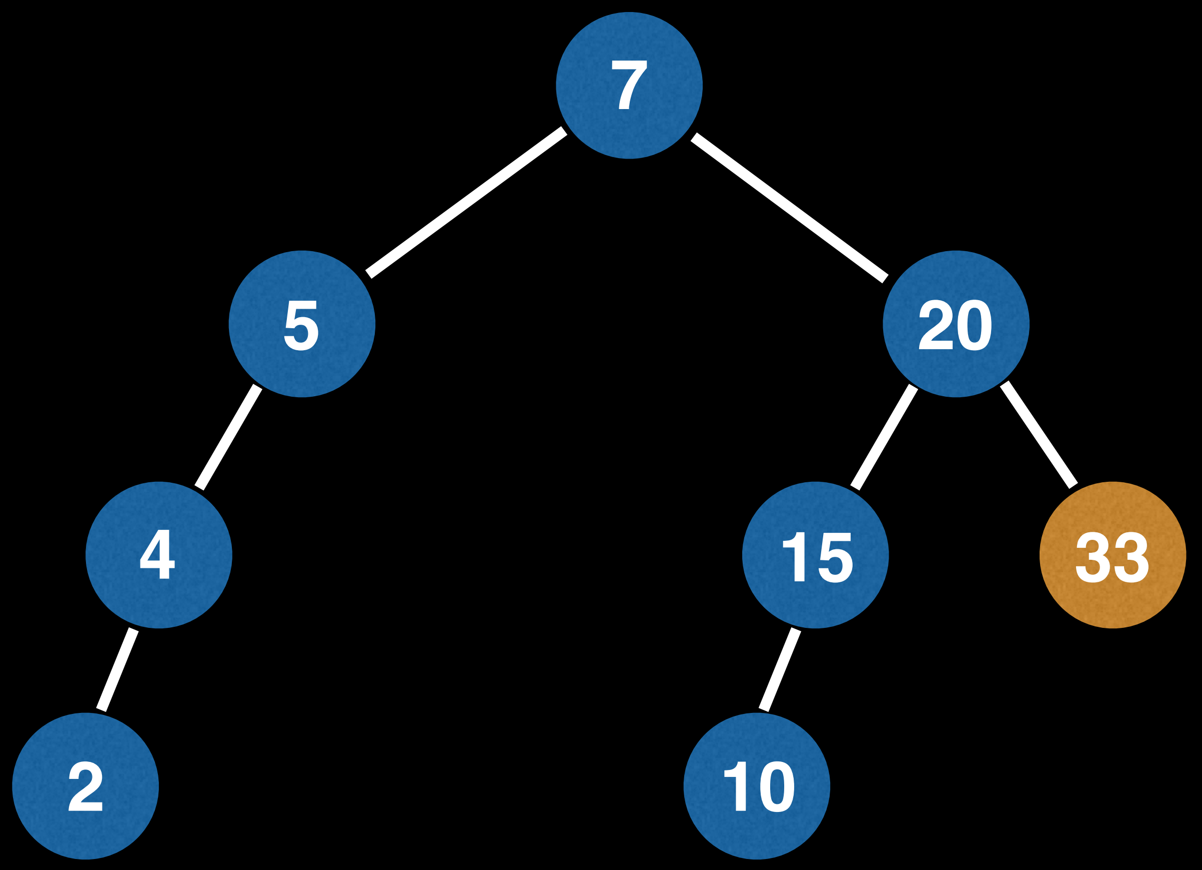
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)



Adding elements to a BST

Instructions:

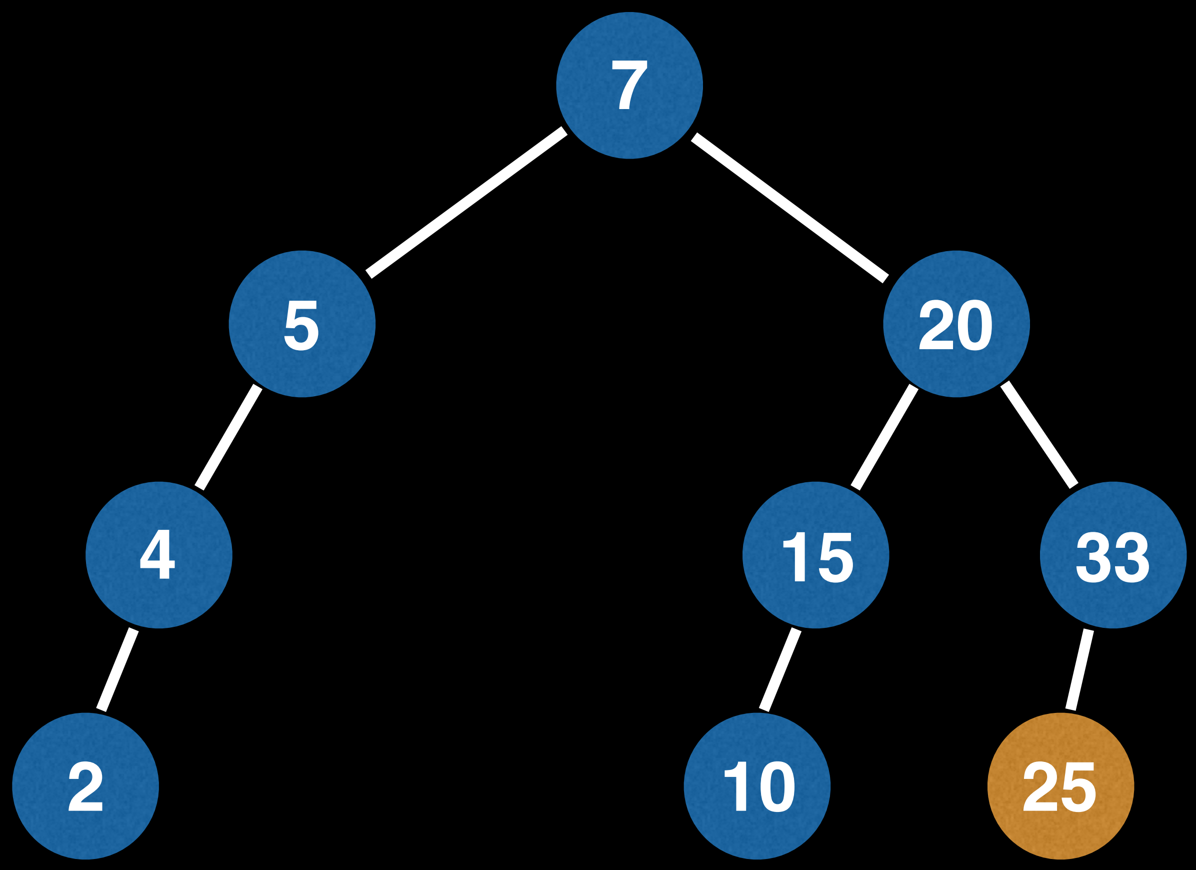
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

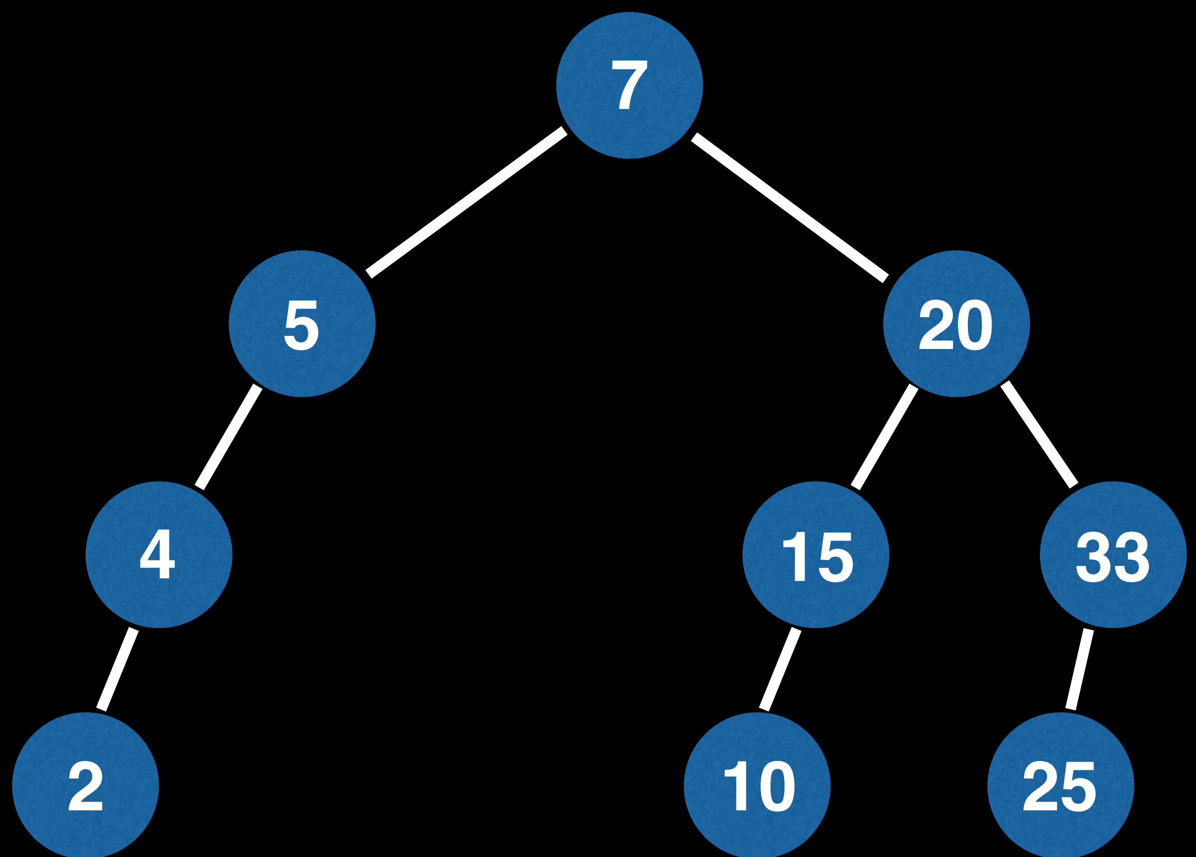
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

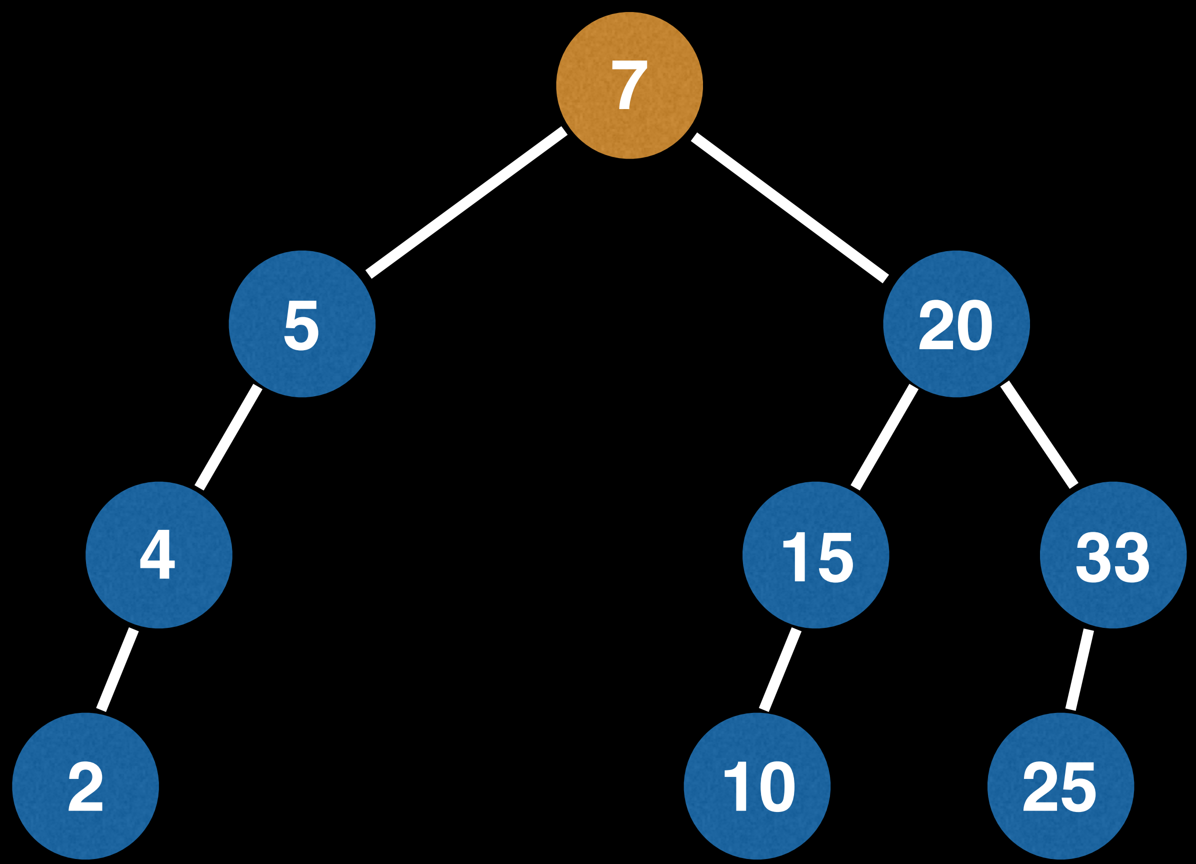
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

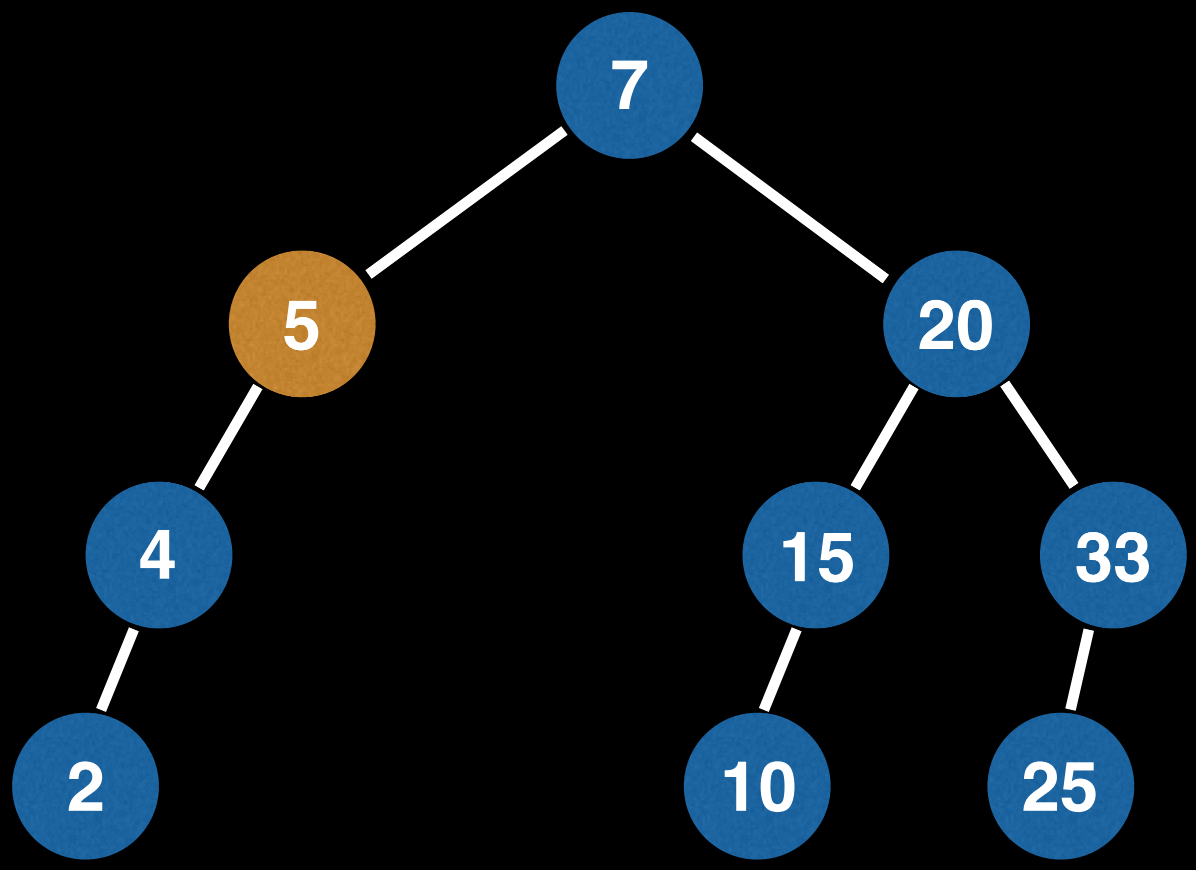
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

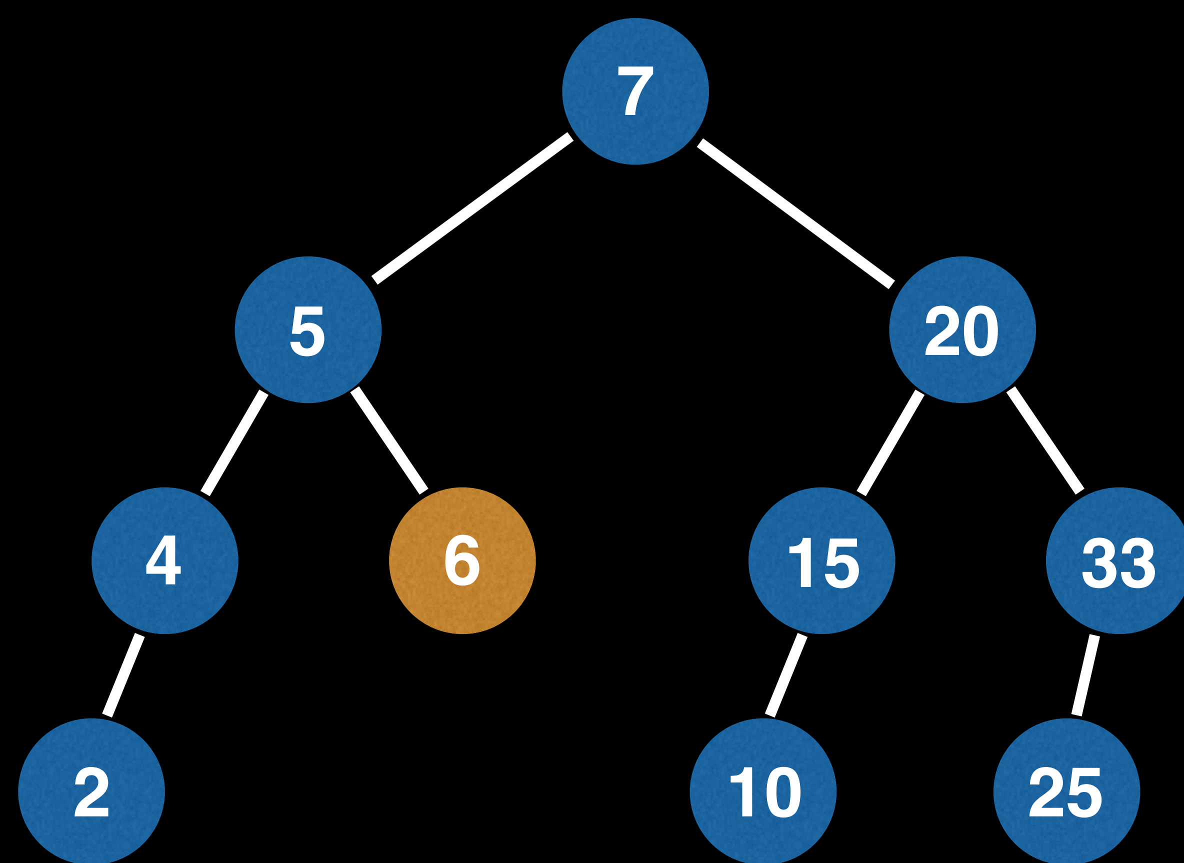
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

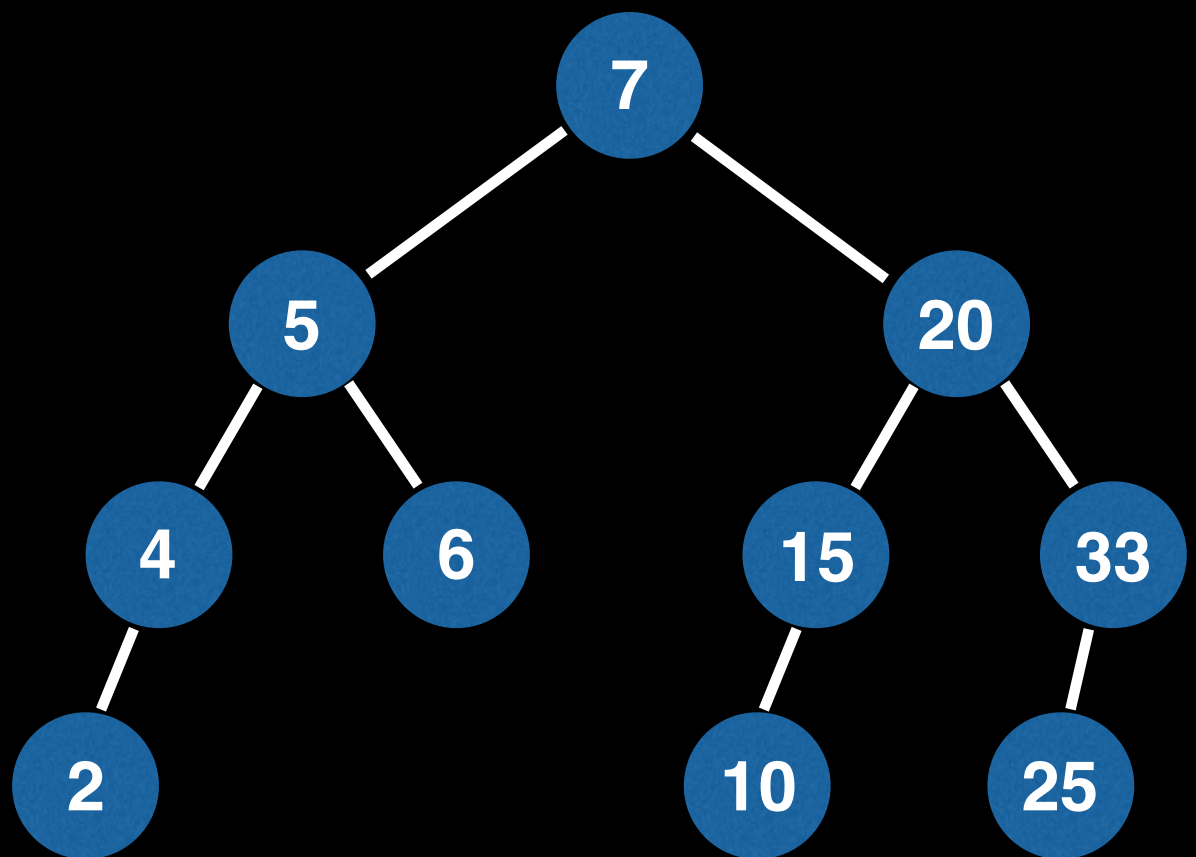
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

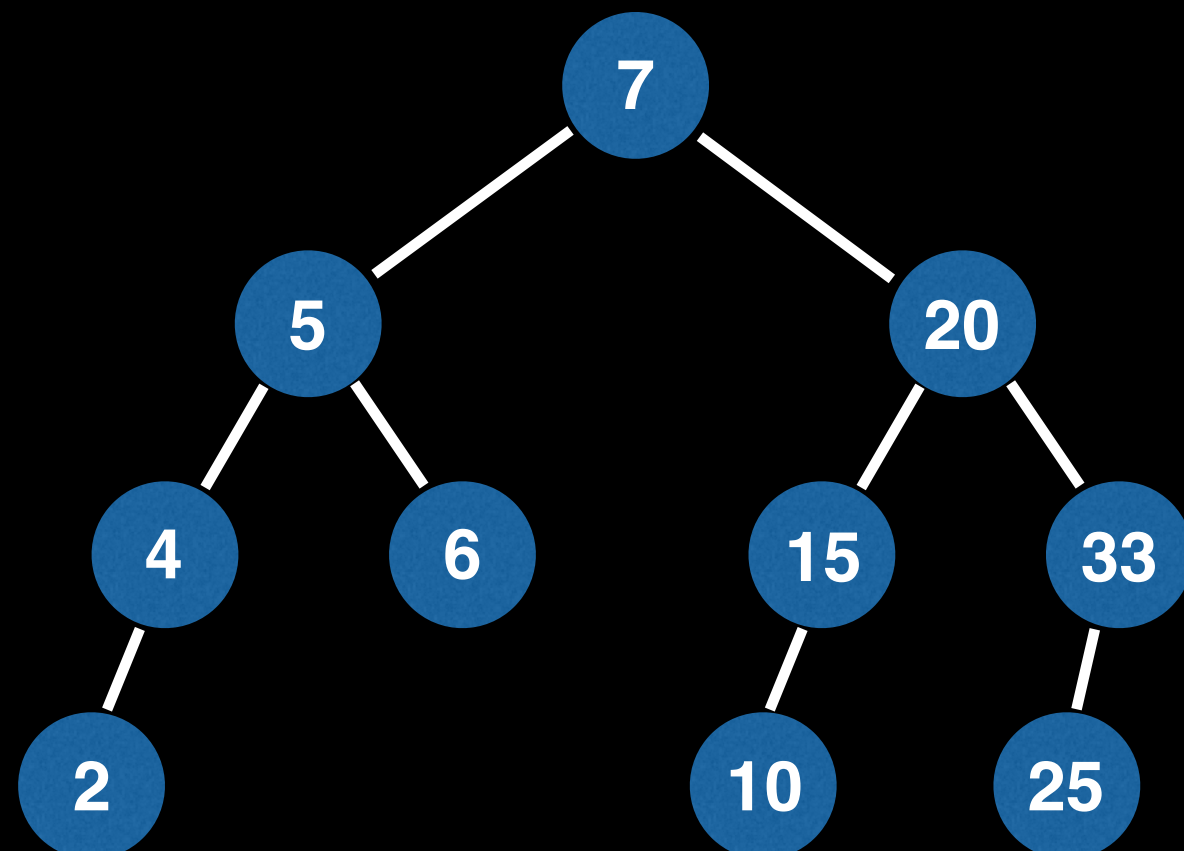
```
insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)
```



Adding elements to a BST

Instructions:

insert(7)
insert(20)
insert(5)
insert(15)
insert(10)
insert(4)
insert(4)
insert(33)
insert(2)
insert(25)
insert(6)

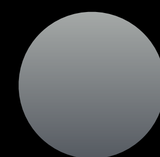


On average the insertion time will be **logarithmic**, but in the worst case this could degrade to **linear** time.

Adding elements to a BST

Instructions:

```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

<u>Instructions</u>:

```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```

Adding elements to a BST

Instructions:

insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)

Adding elements to a BST

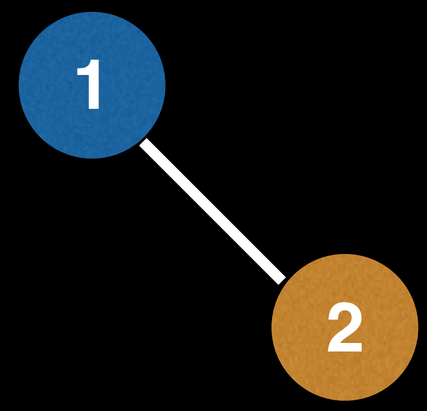
Instructions:

insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)

Adding elements to a BST

Instructions:

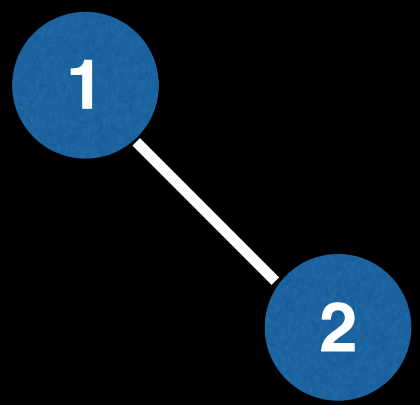
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

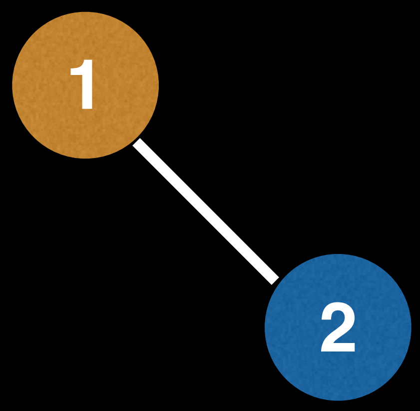
Instructions:

```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

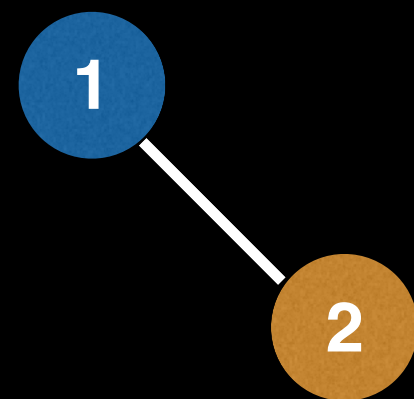



```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```

Adding elements to a BST

Instructions:

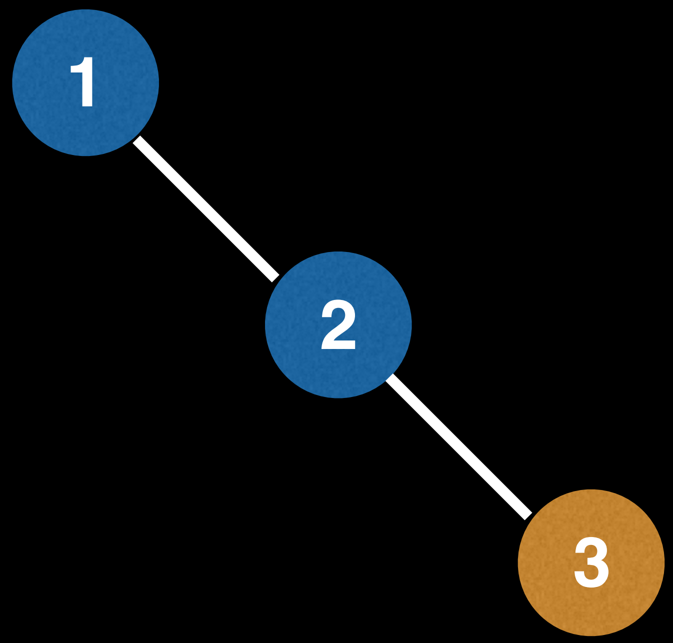
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

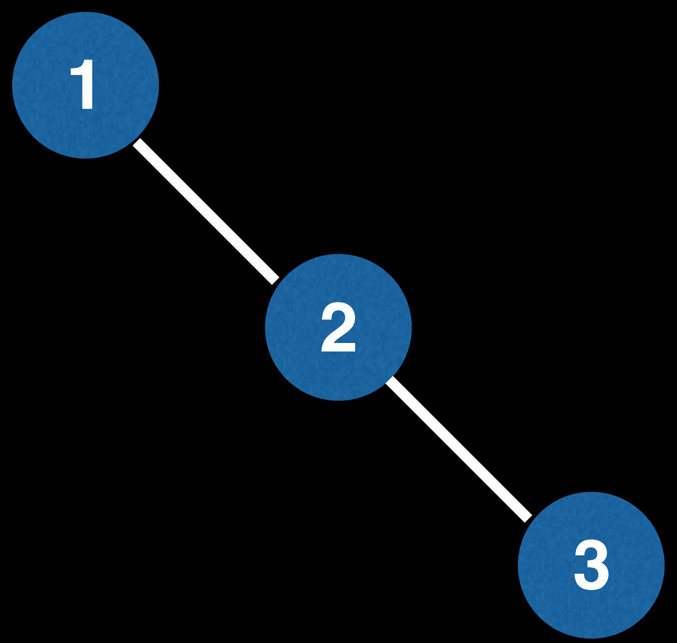
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

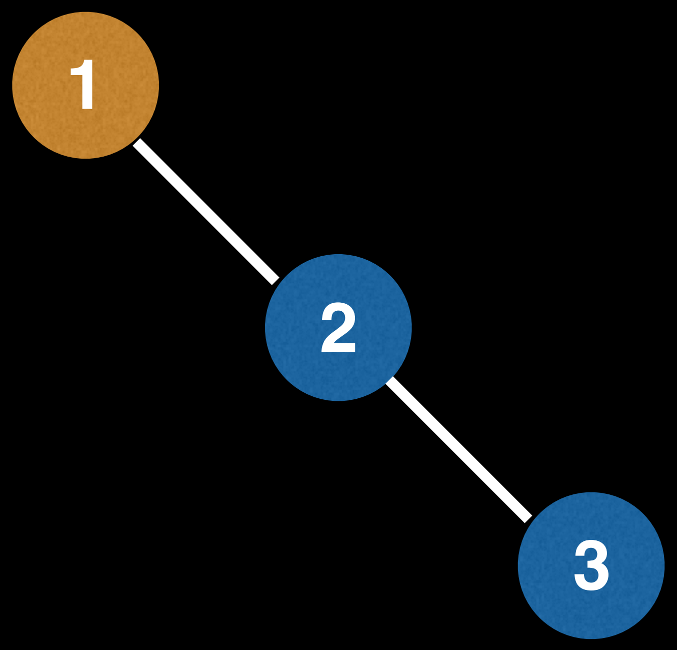
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

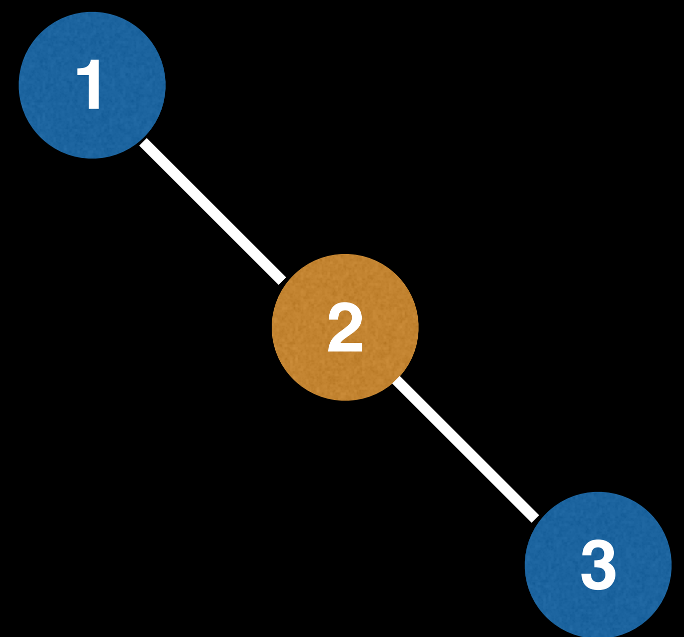
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)



Adding elements to a BST

Instructions:

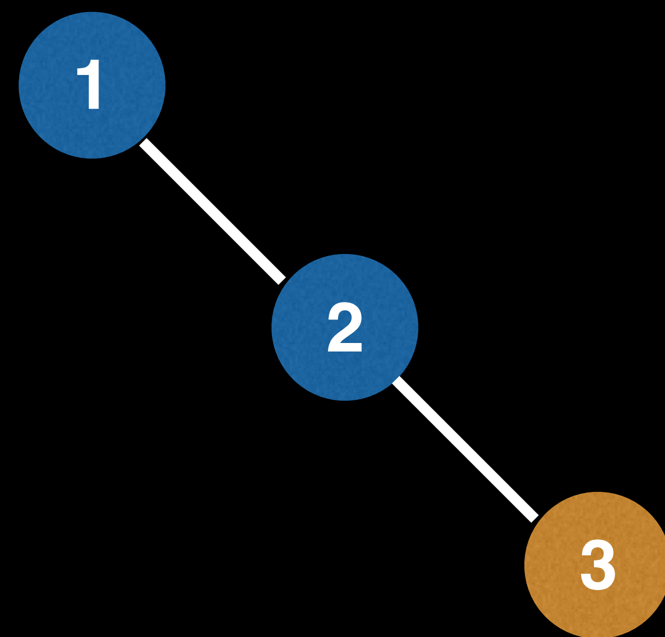
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)



Adding elements to a BST

Instructions:

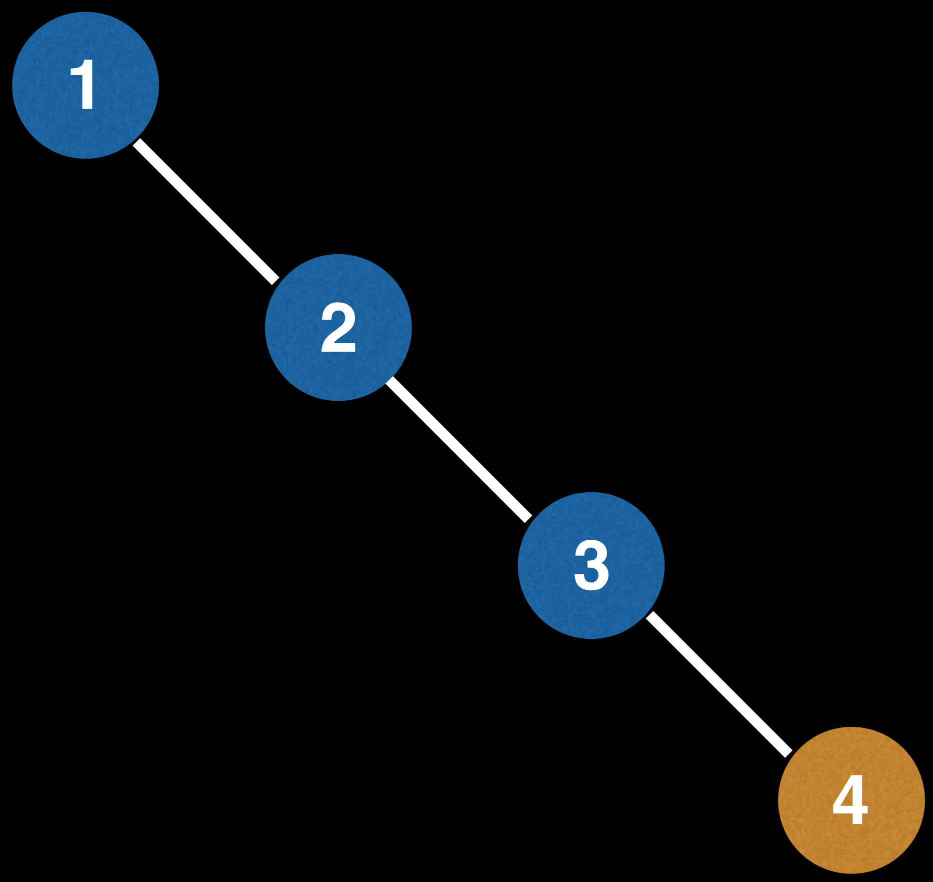
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

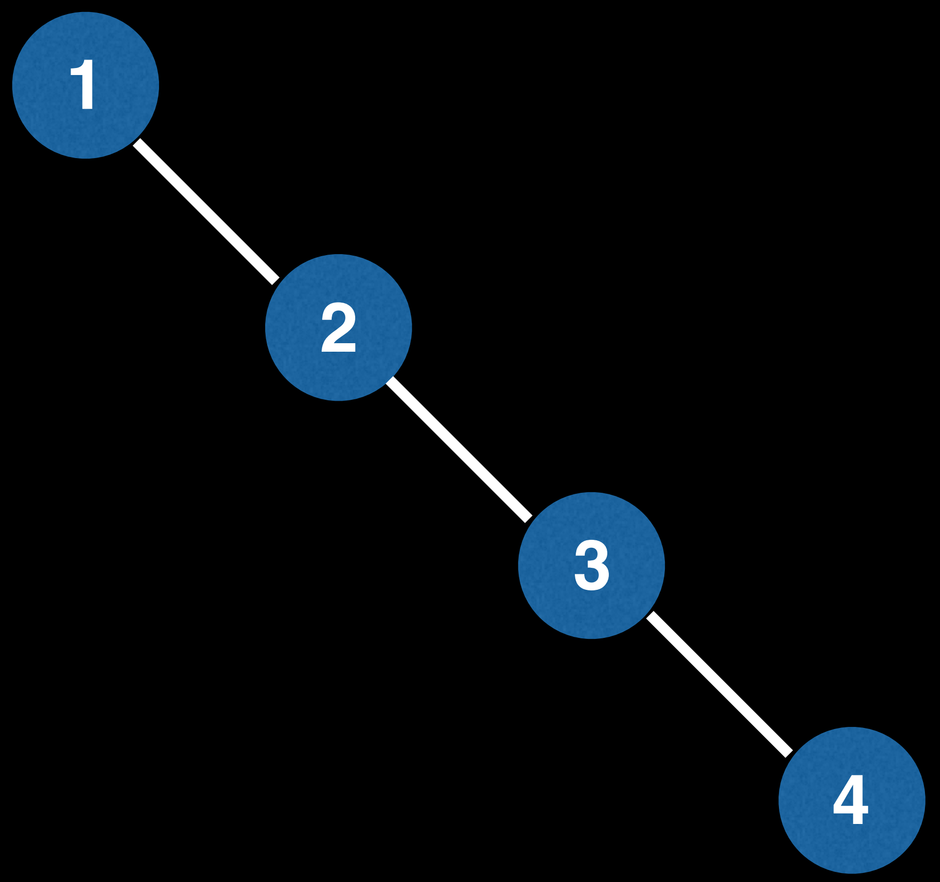
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

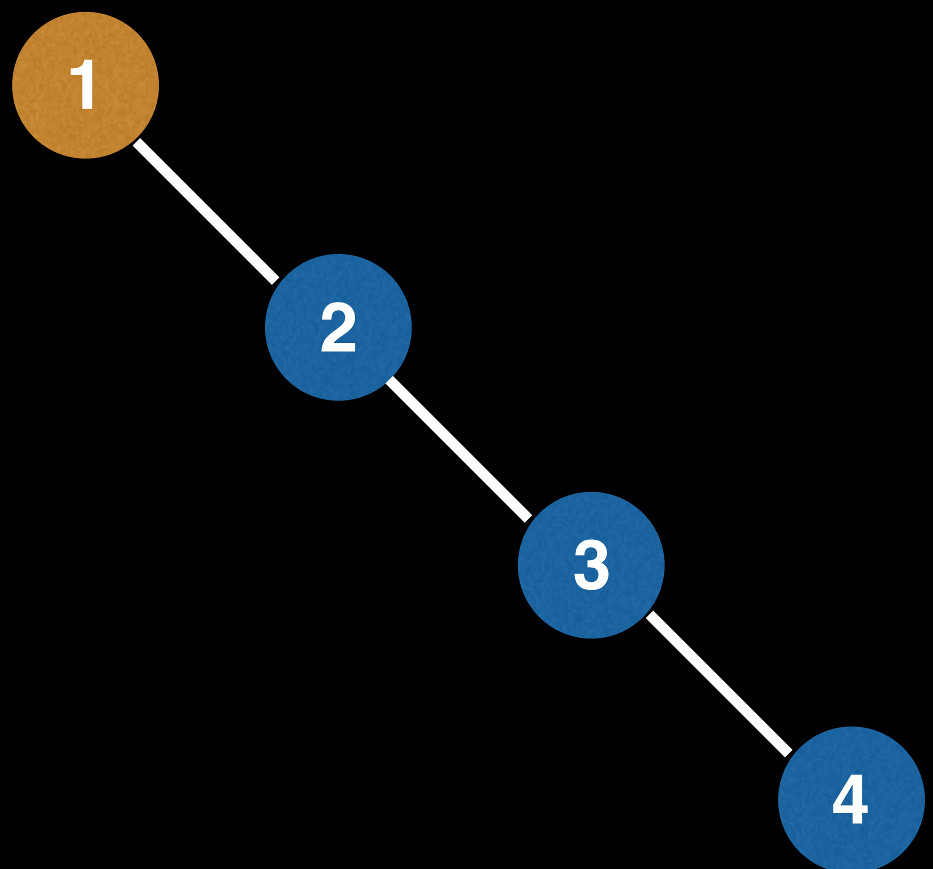
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

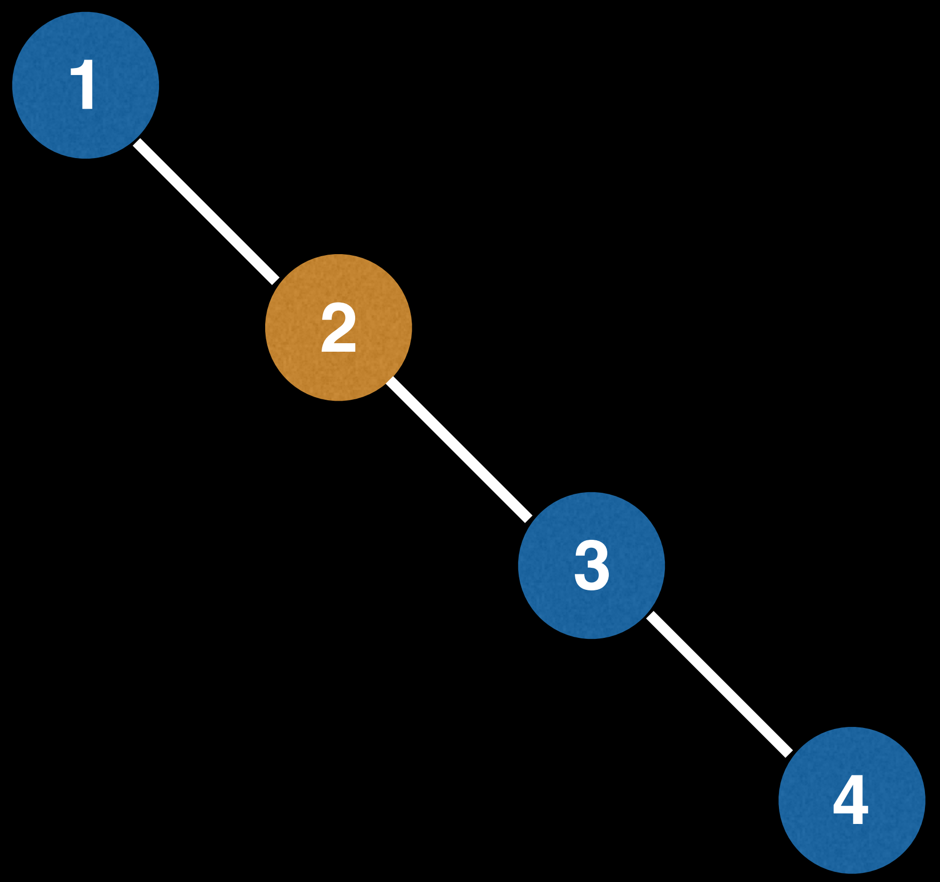
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)



Adding elements to a BST

Instructions:

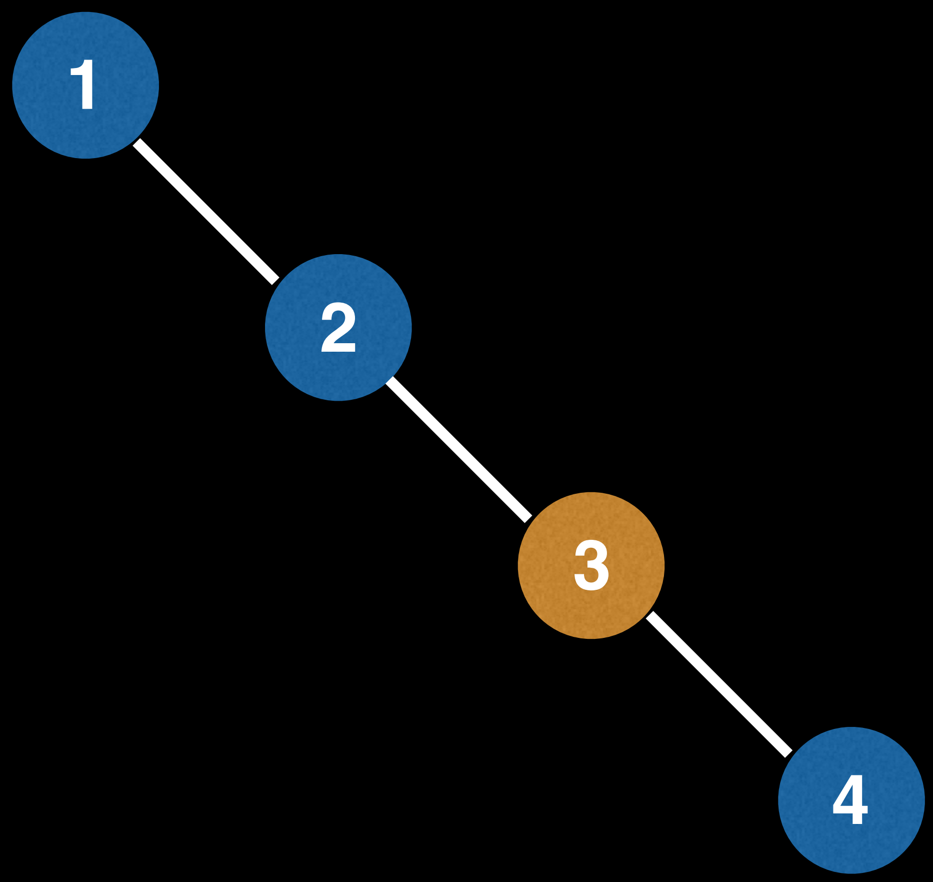
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

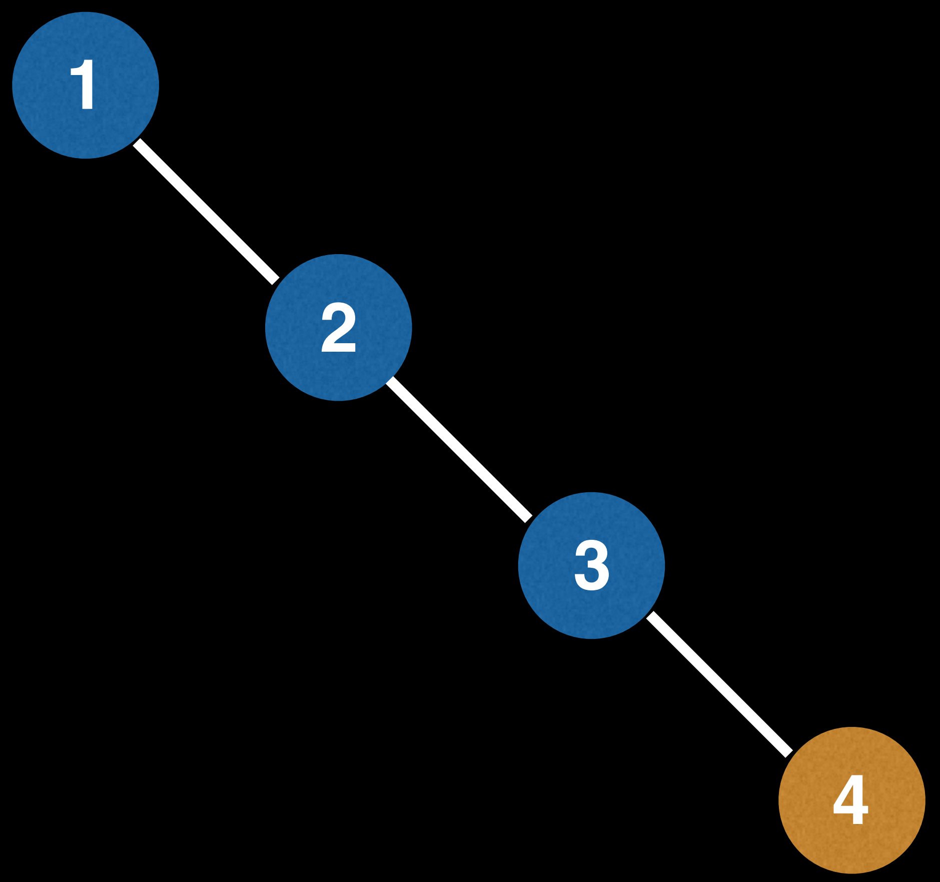
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

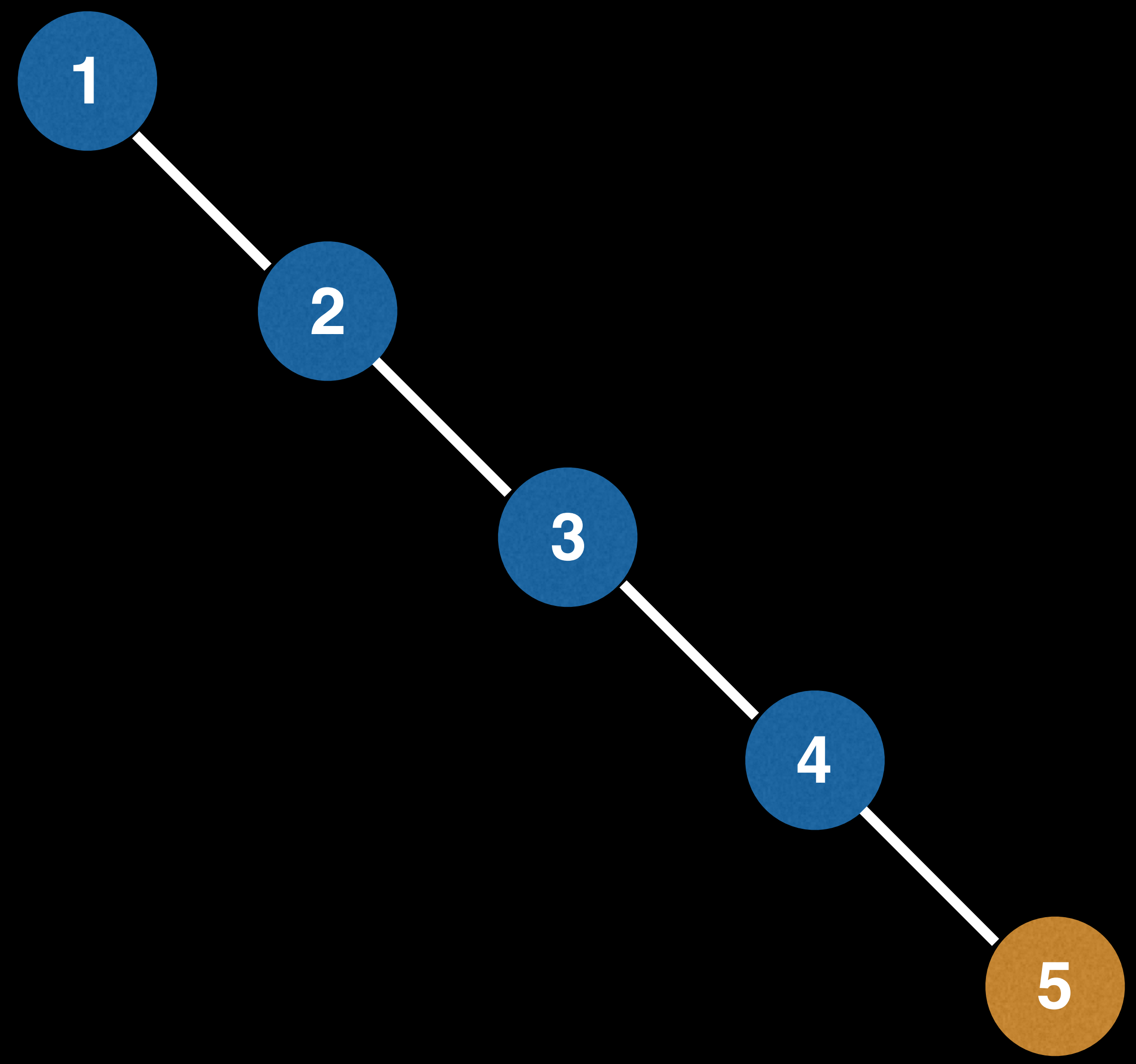
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

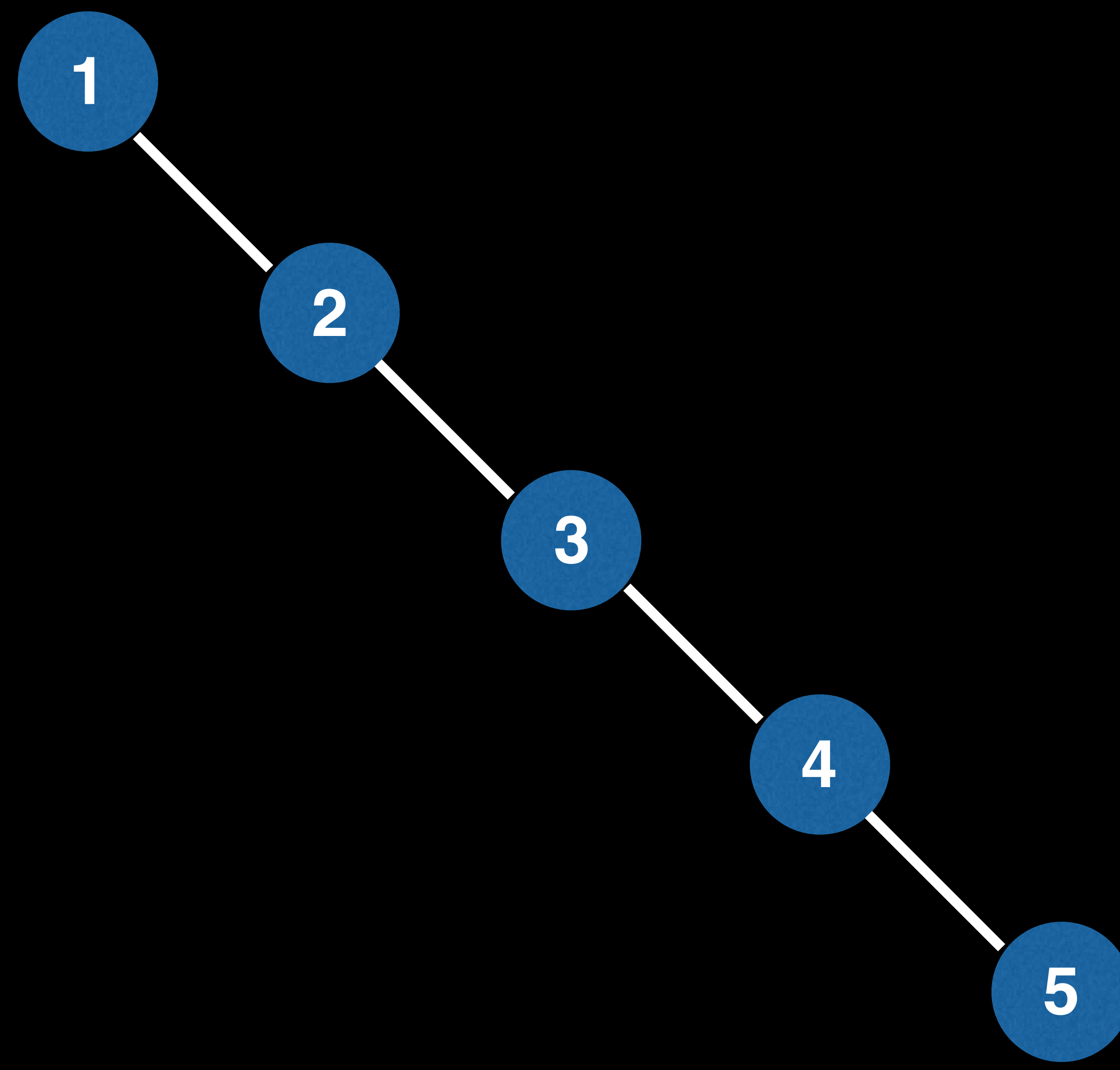
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

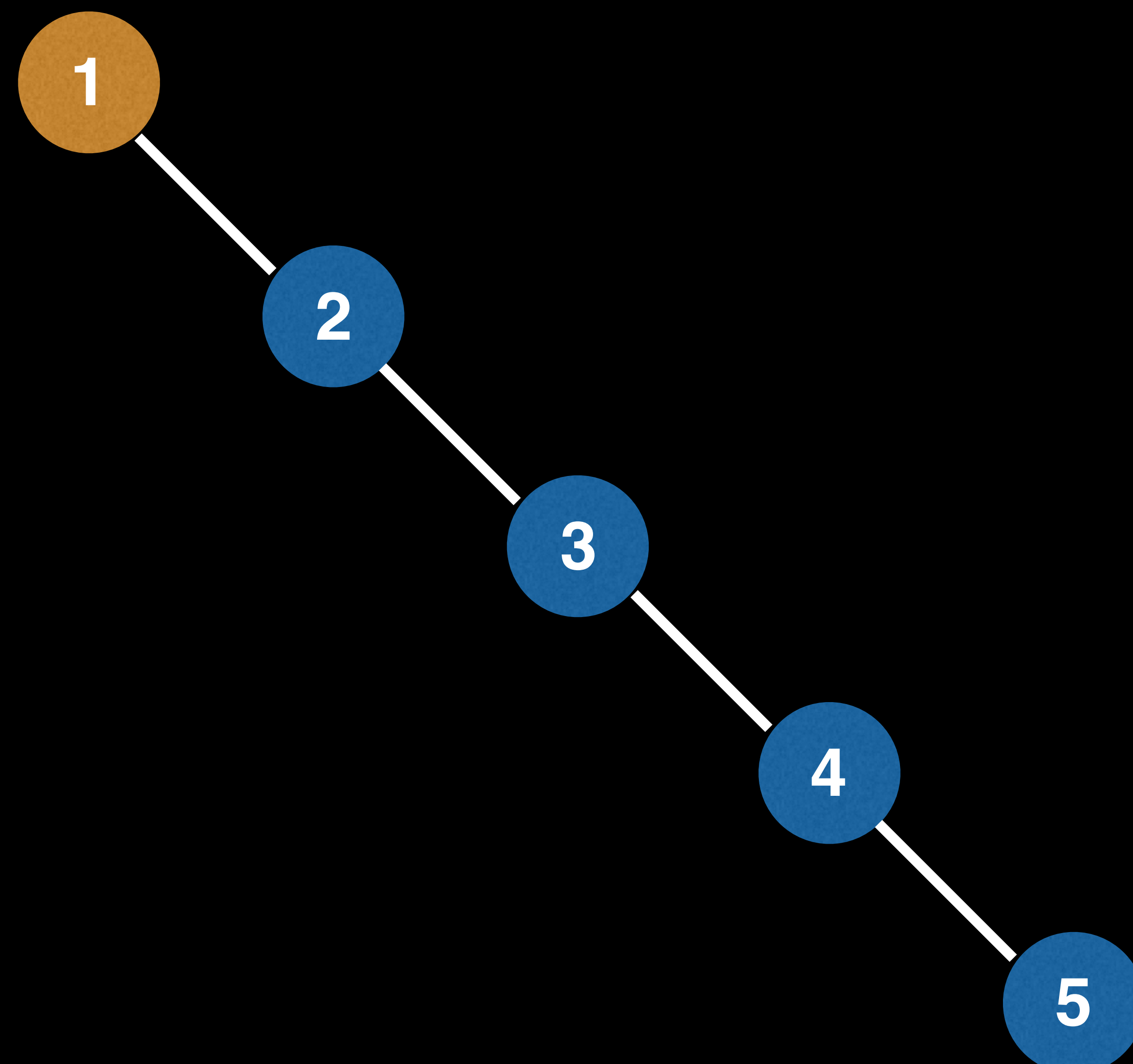
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

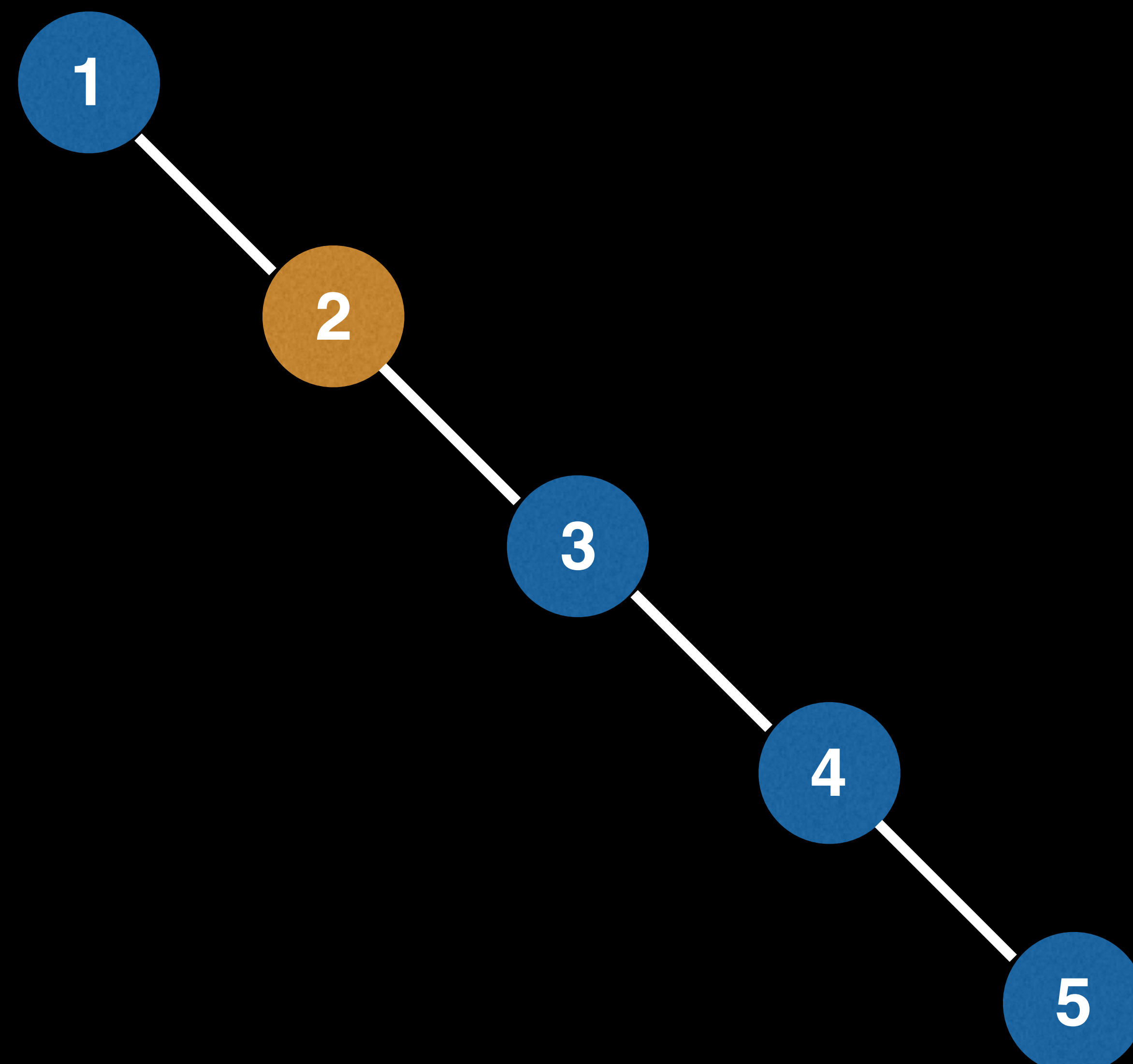
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

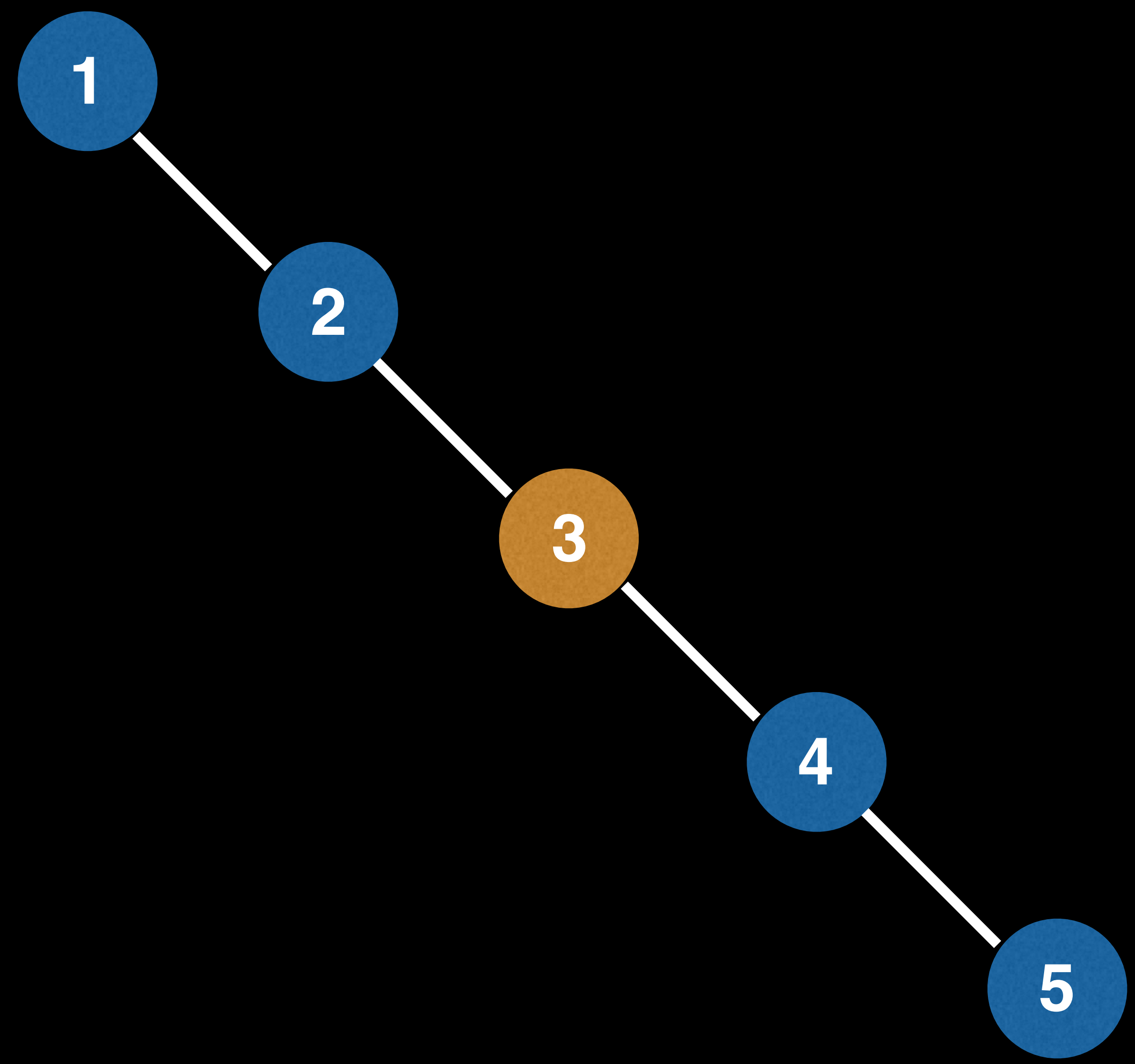
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

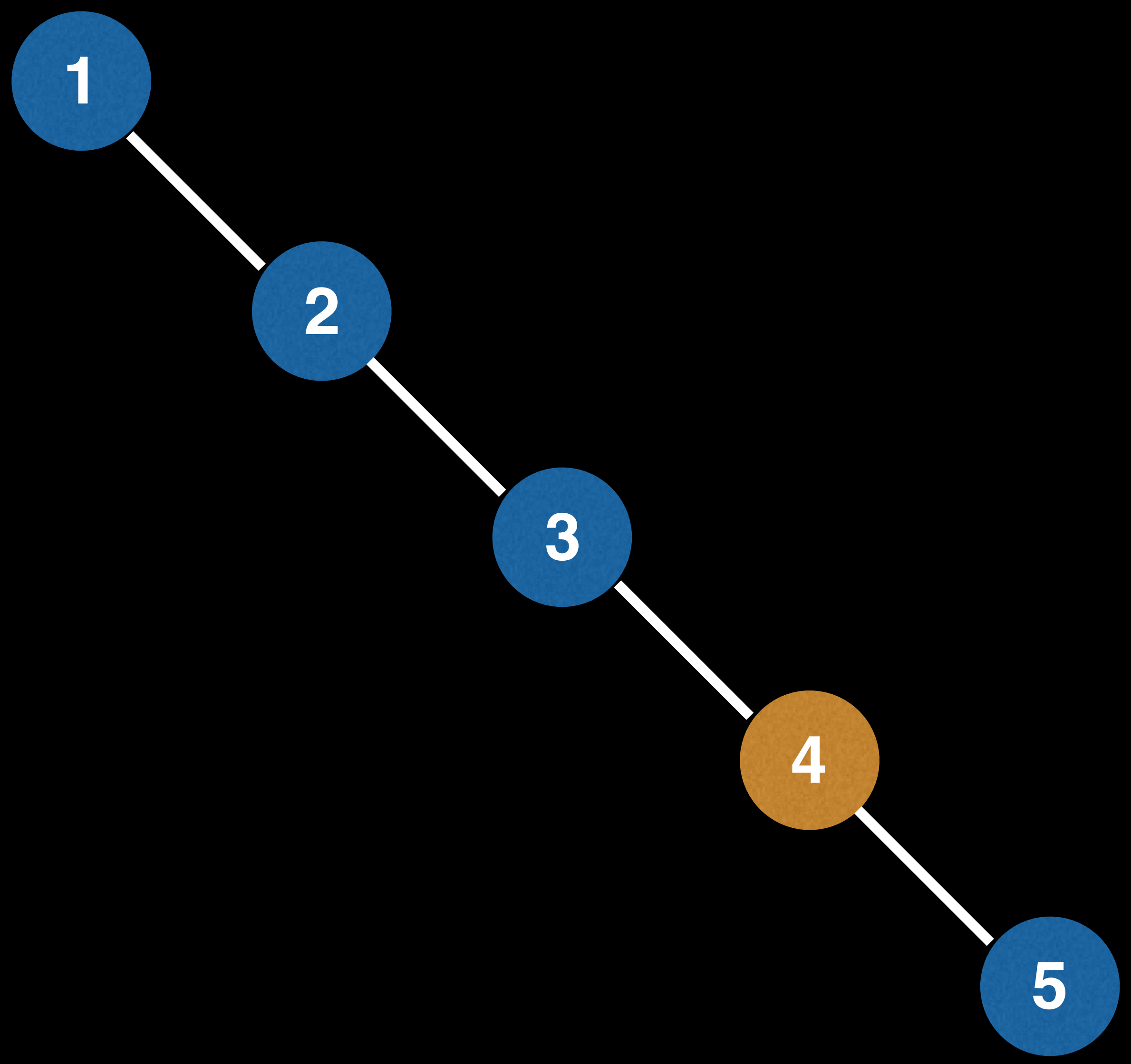
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

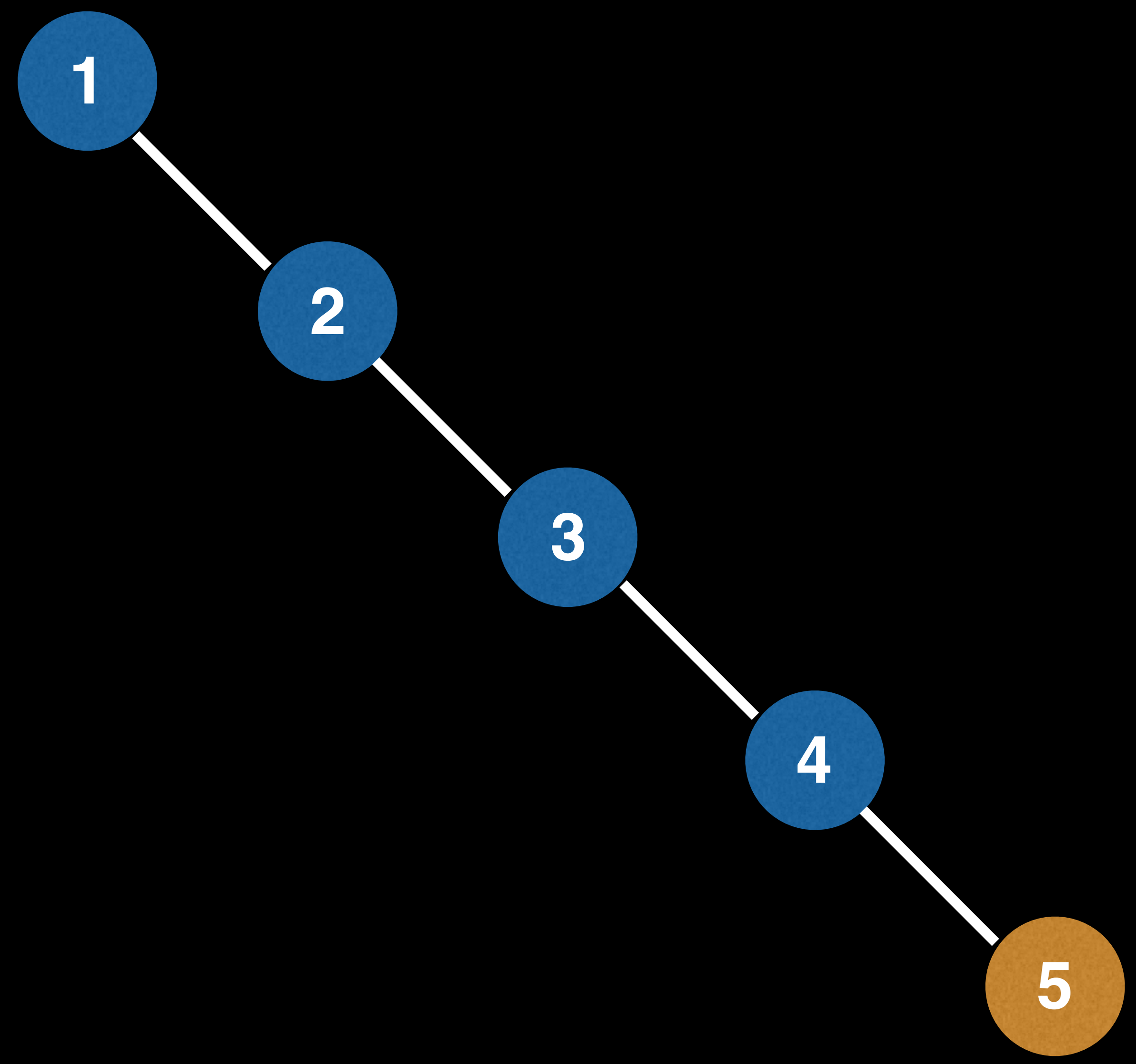
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)



Adding elements to a BST

Instructions:

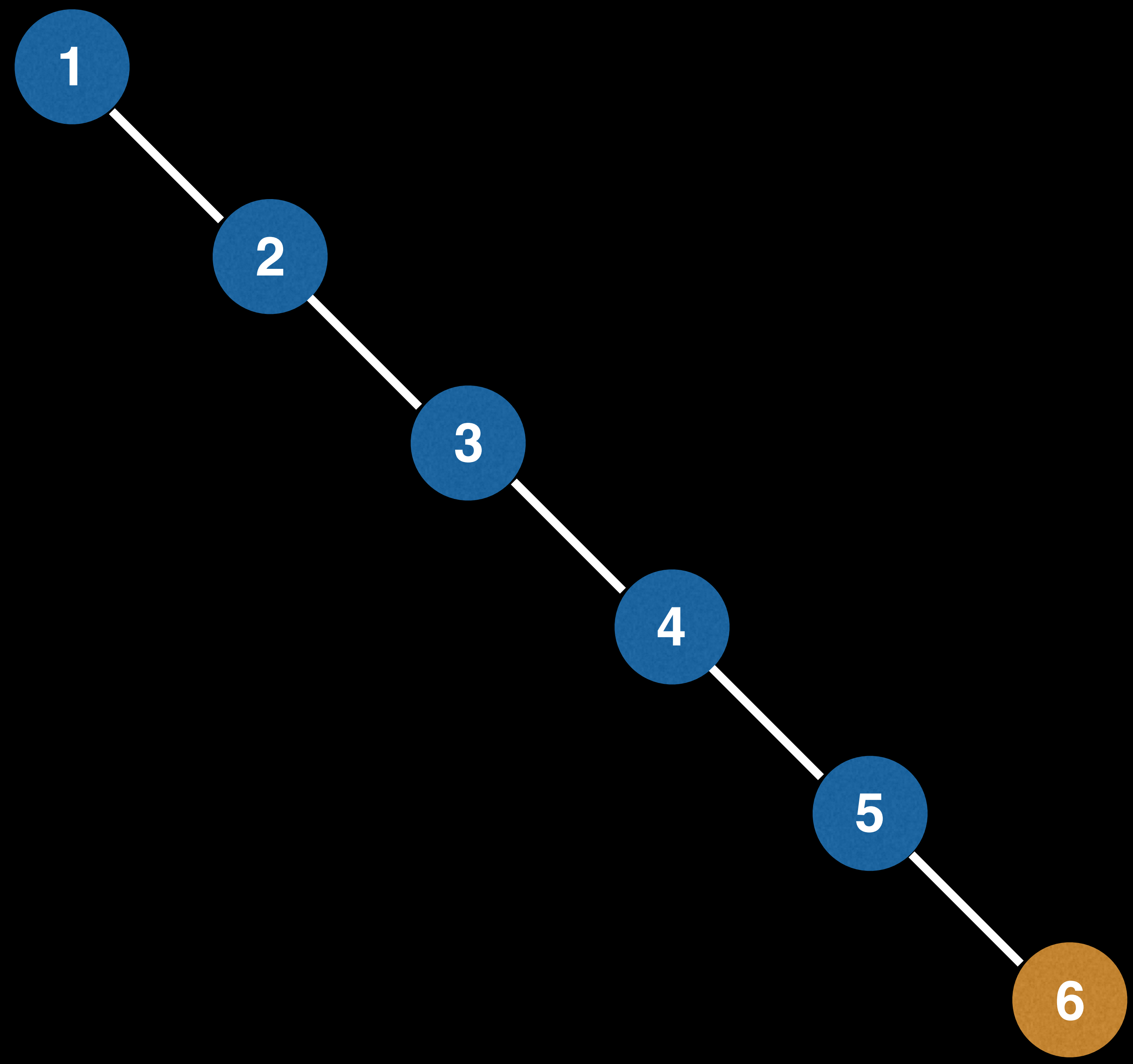
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)



Adding elements to a BST

Instructions:

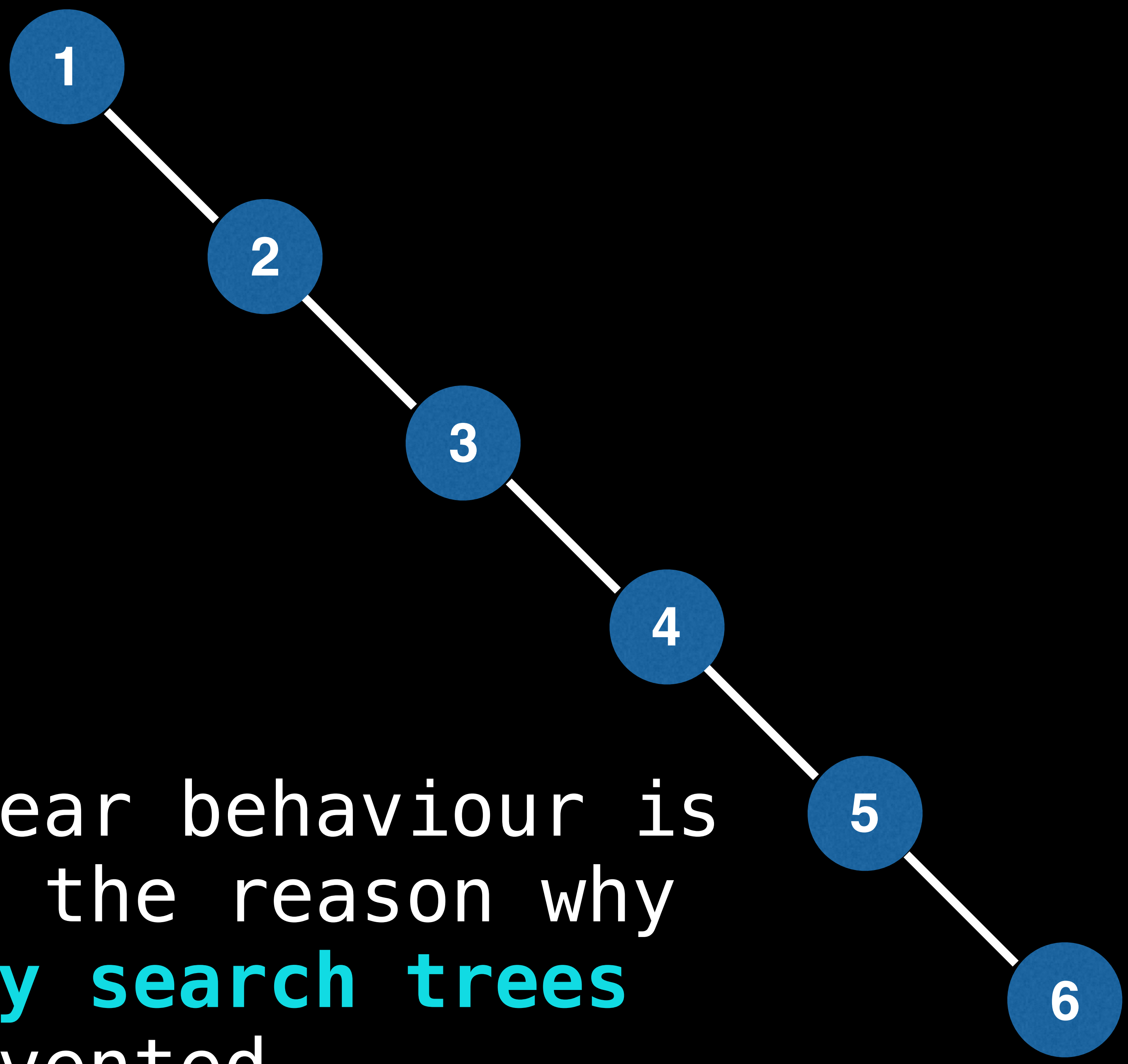
```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



Adding elements to a BST

Instructions:

```
insert(1)
insert(2)
insert(3)
insert(4)
insert(5)
insert(6)
```



This type of linear behaviour is **very bad** and is the reason why **balanced binary search trees** were invented.

Removing elements from a Binary Search Tree (BST)

Removing elements from a BST

Removing elements from a Binary Search Tree (BST) can be seen as a two step process.

1) **Find** the element we wish to remove (if it exists)

2) **Replace** the node we want to remove with its successor (if any) to maintain the BST invariant.

Recall the **BST invariant**: left subtree has smaller elements and right subtree has larger elements.

Find phase

When searching our BST for a node with a particular value one of four things will happen:

1) We hit a **null node** at which point we know the value does not exist within our BST

2) Comparator value **equal to 0** (found it!)

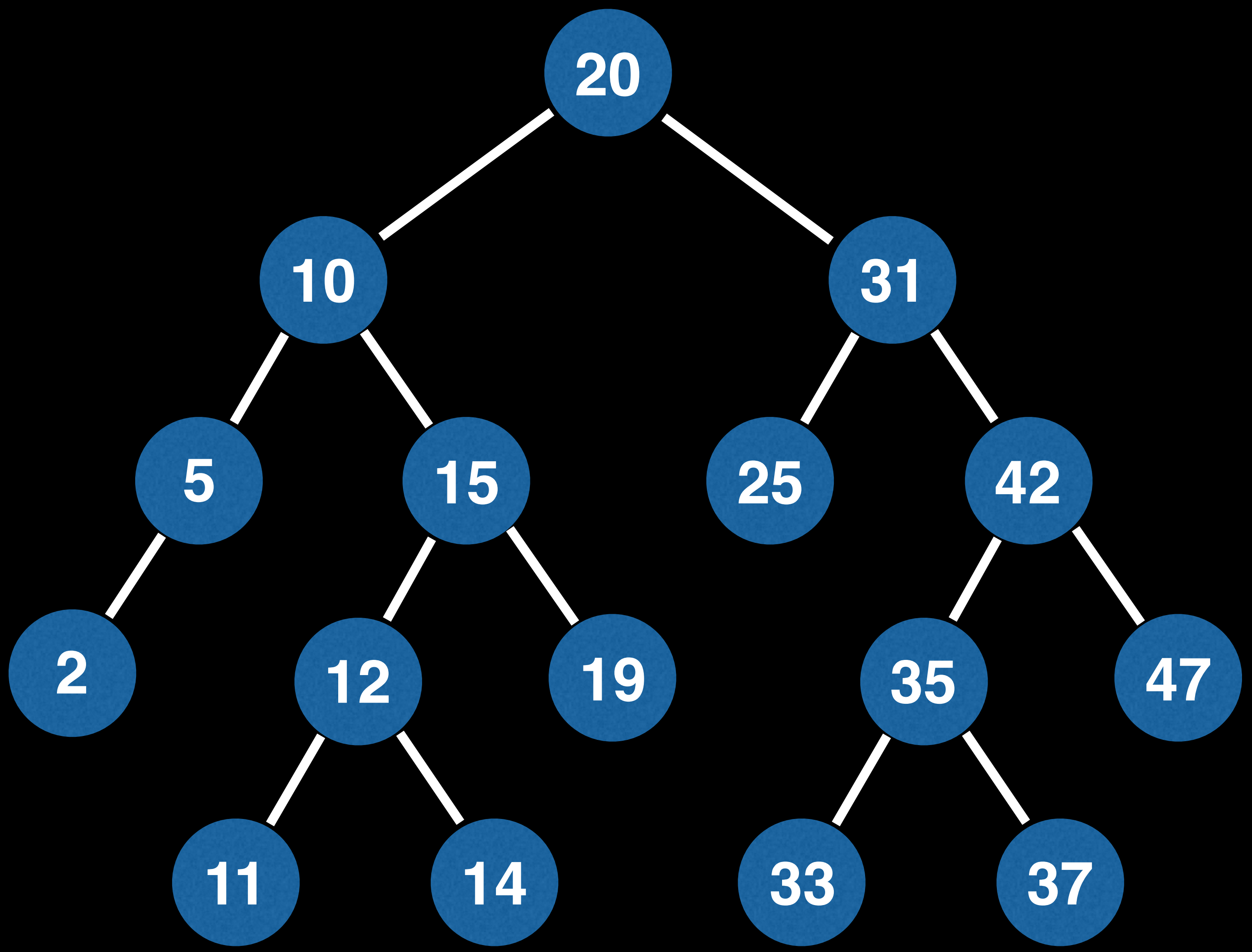
3) Comparator value **less than 0** (the value, if it exists, is in the left subtree)

4) Comparator value **greater than 0** (the value, if it exists, is in the right subtree)

Find phase

Find queries:

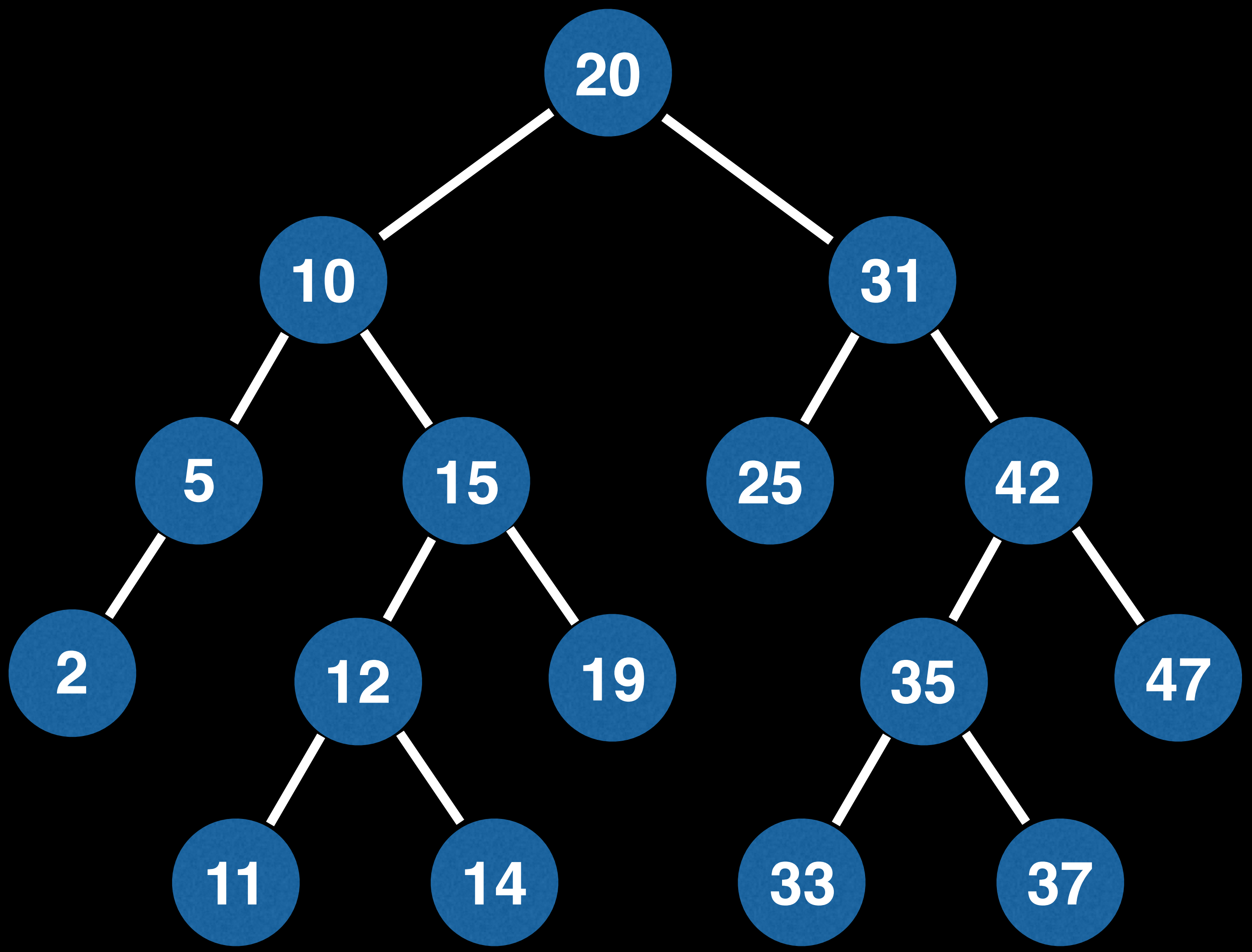
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

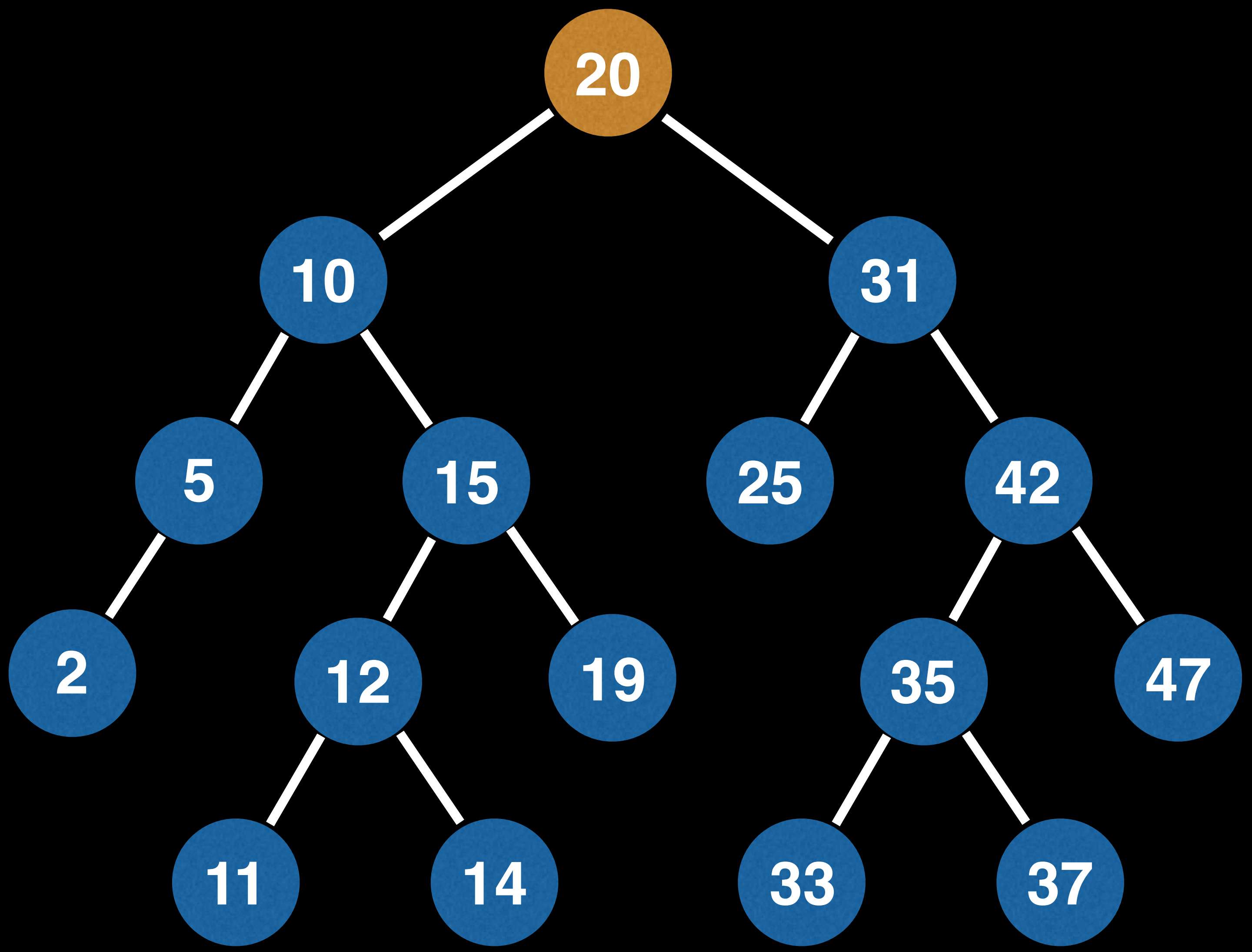
```
find(14) ←
find(25)
find(37)
find(17)
```



Find phase

Find queries:

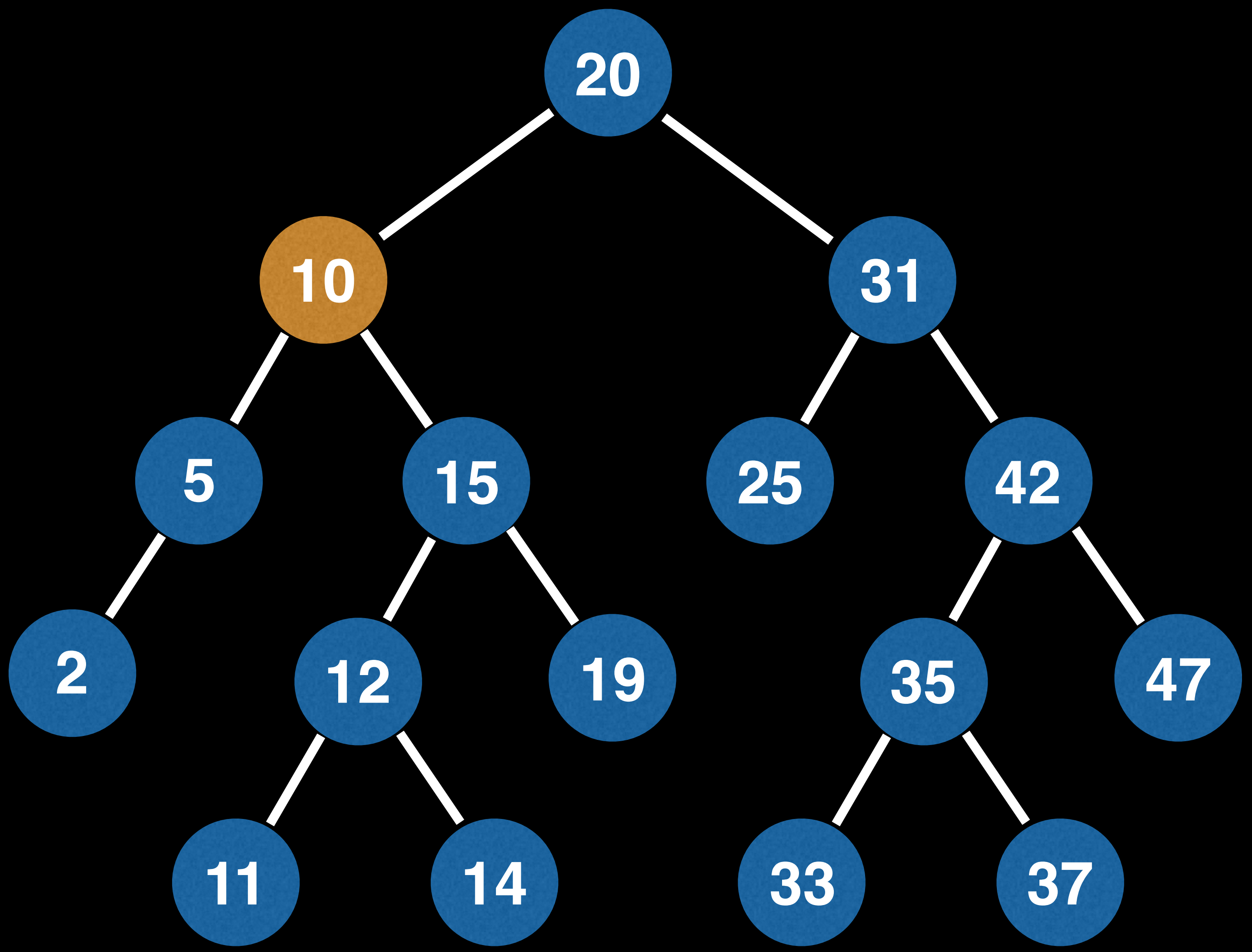
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

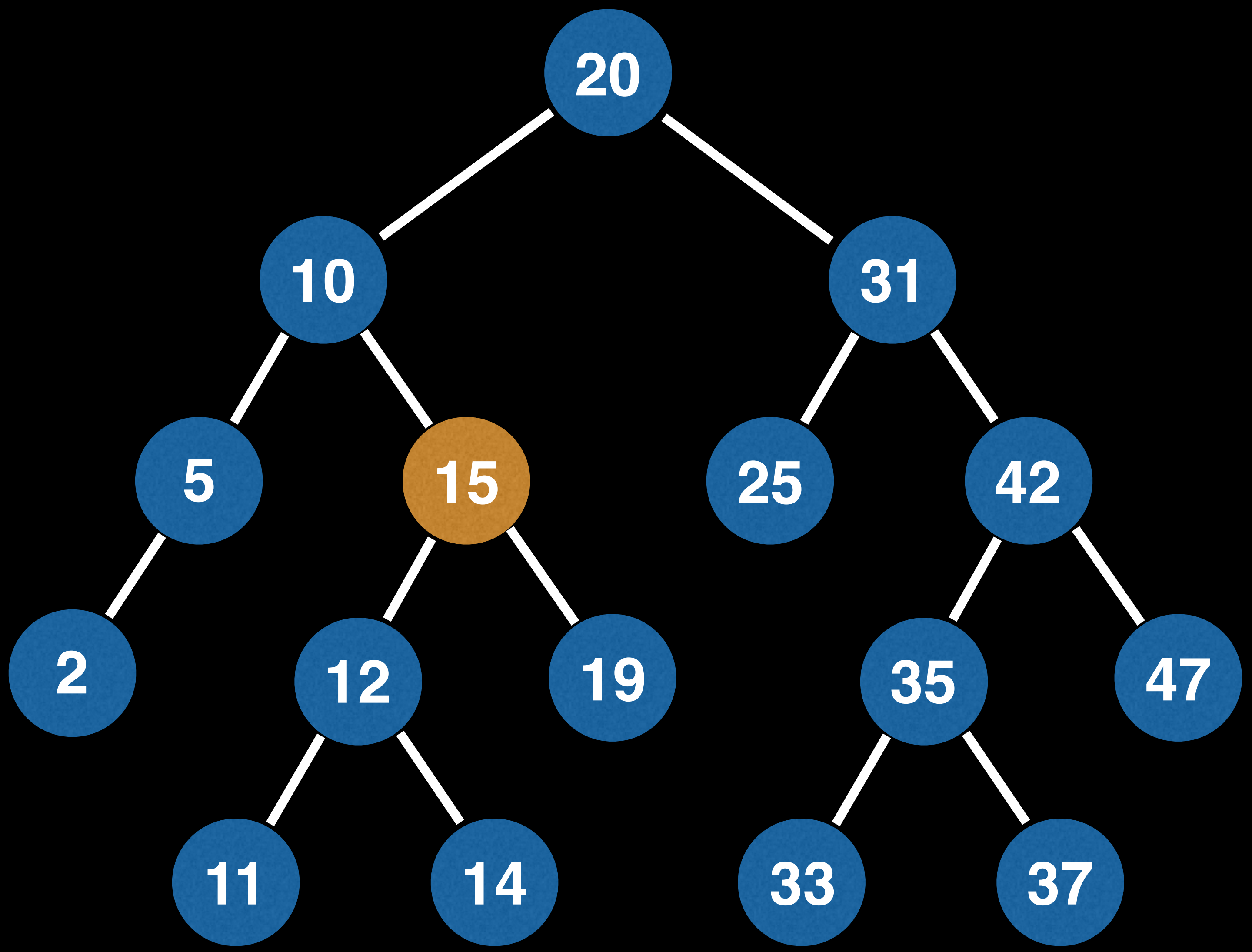
find(14)
find(25)
find(37)
find(17)



Find phase

Find queries:

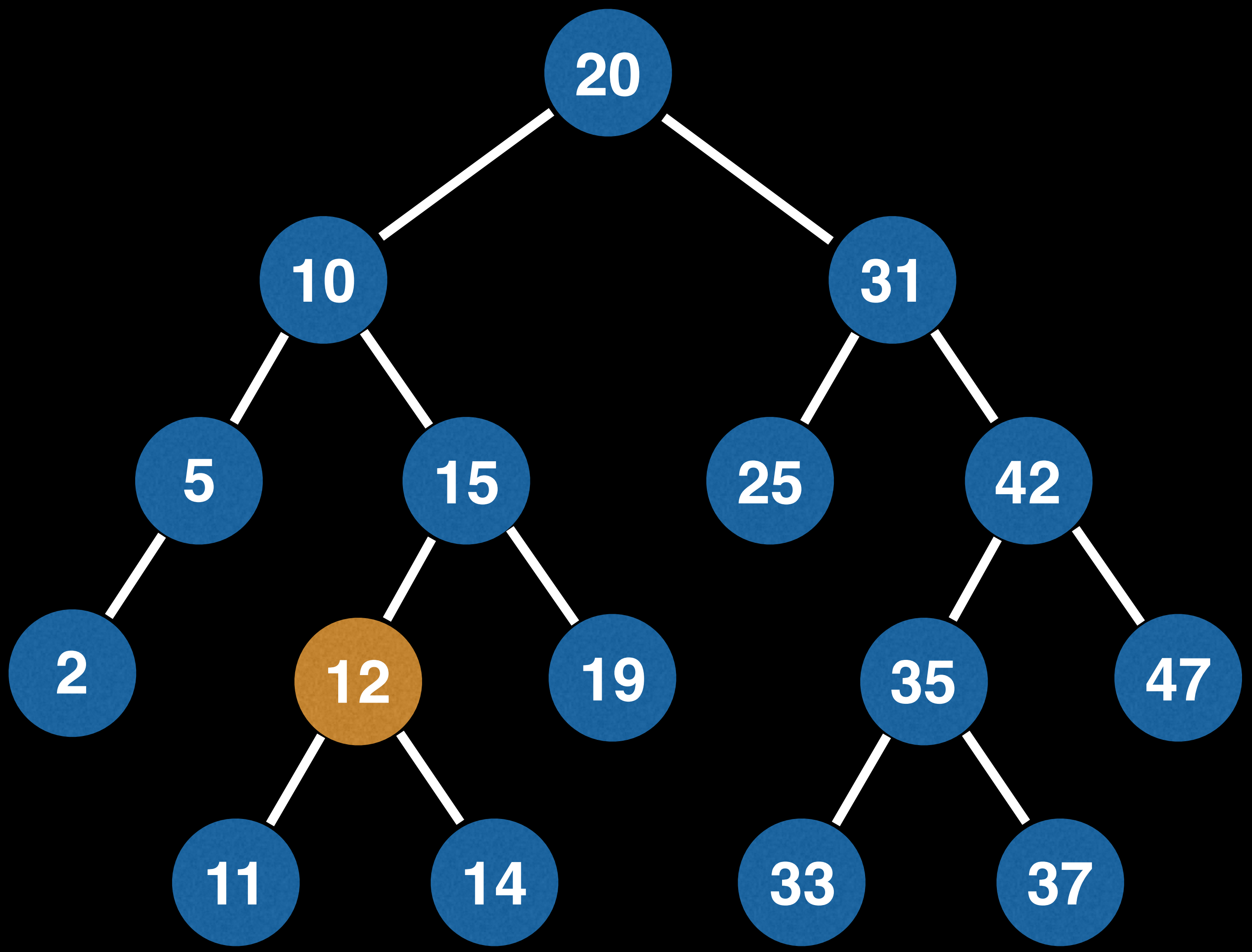
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

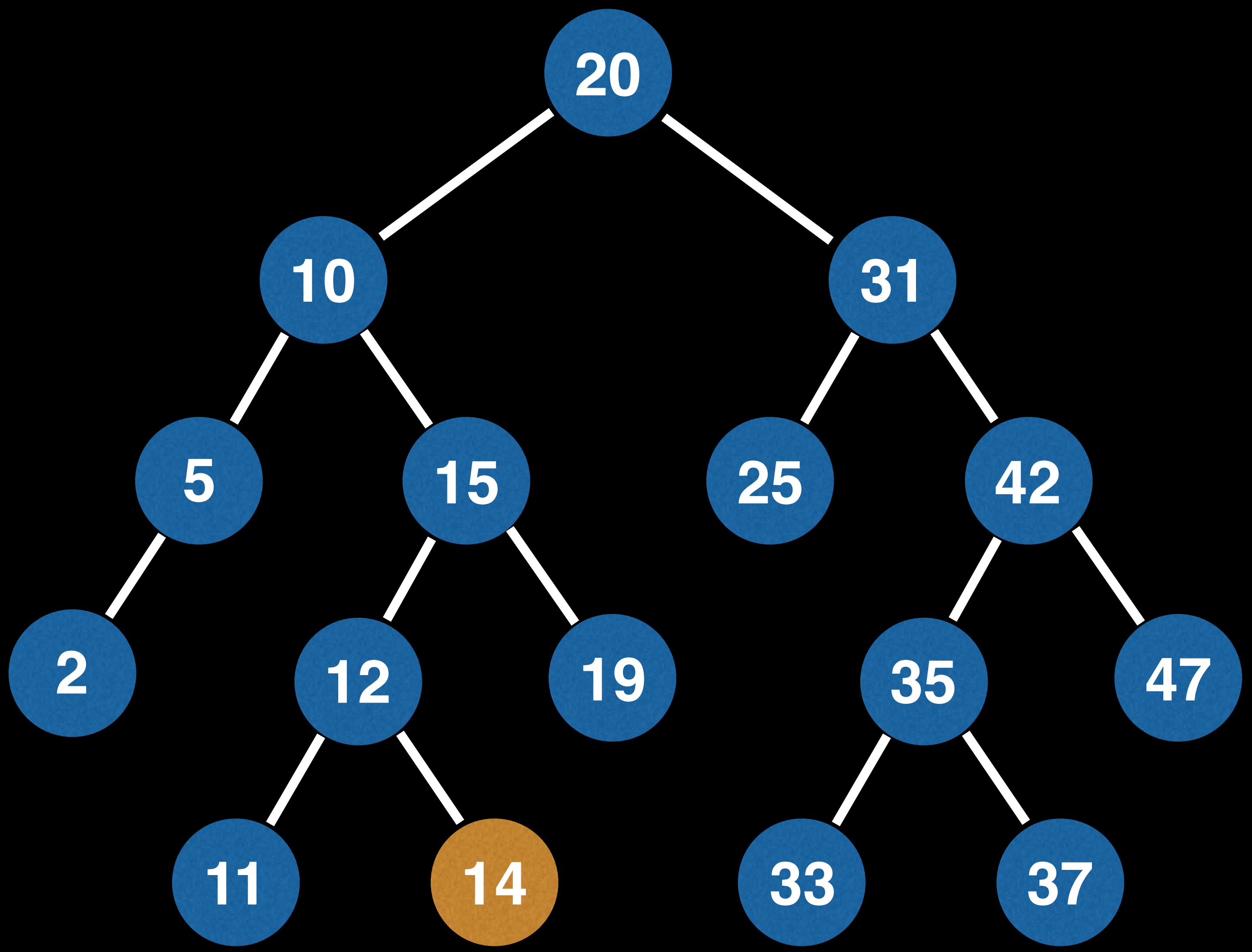
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

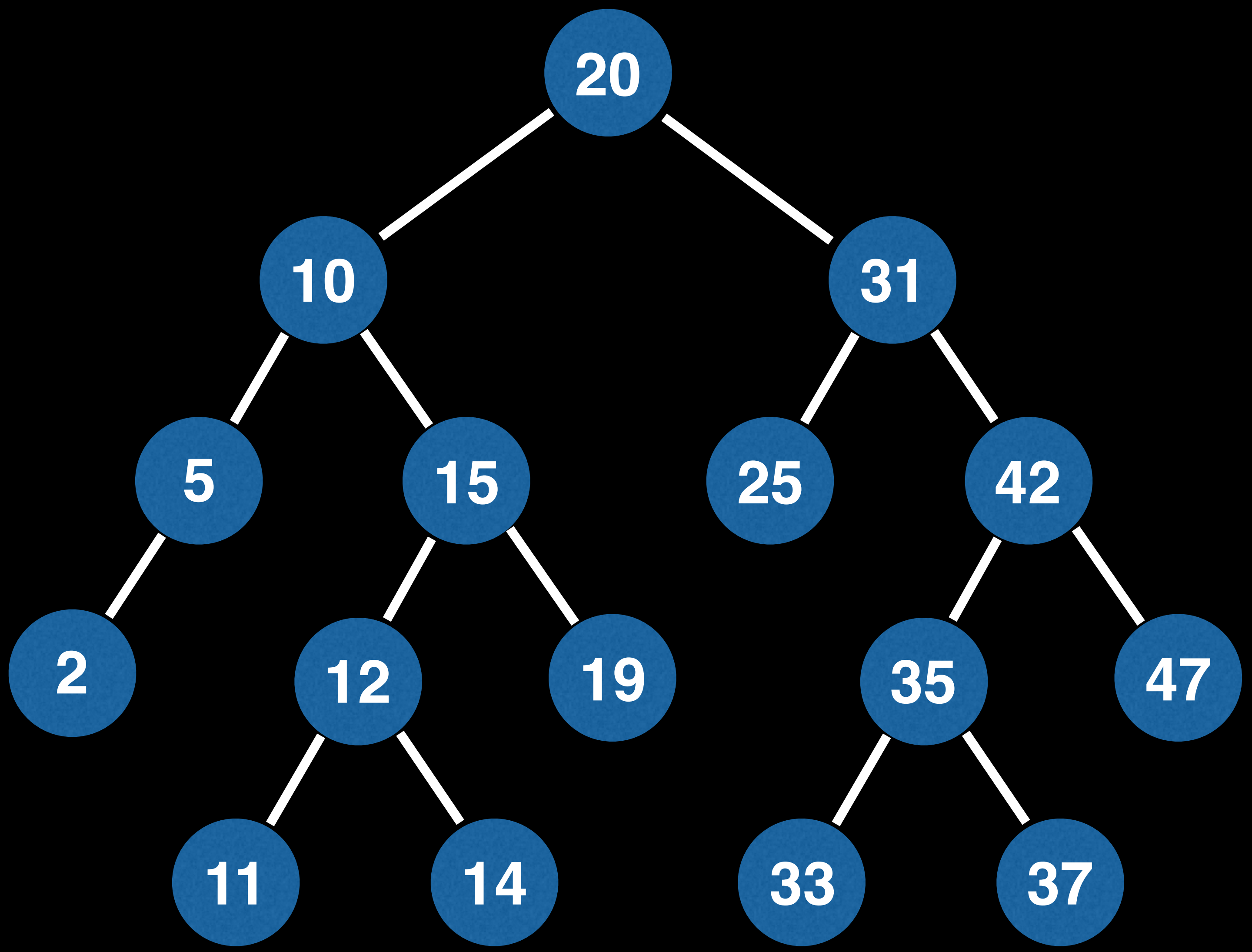
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

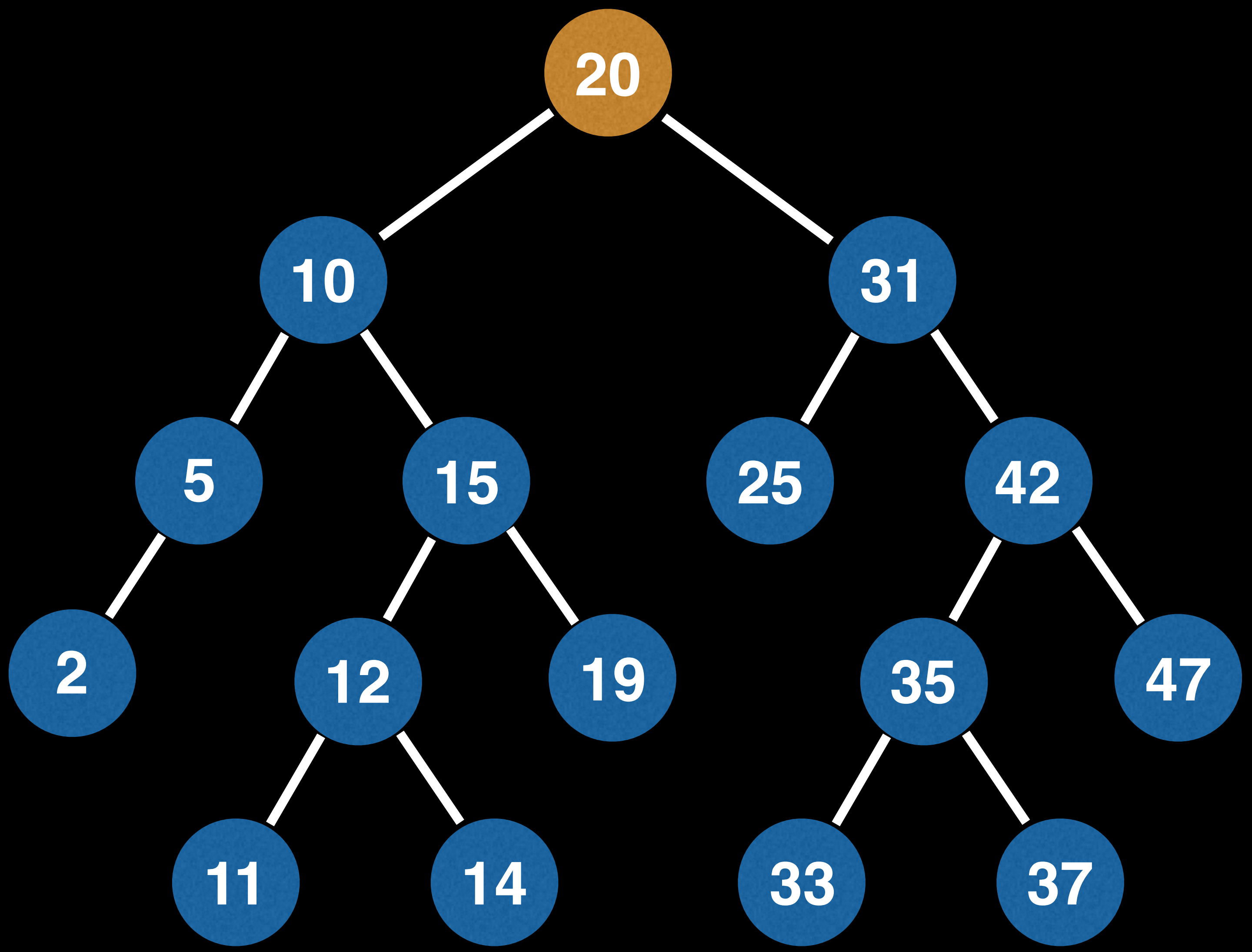
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

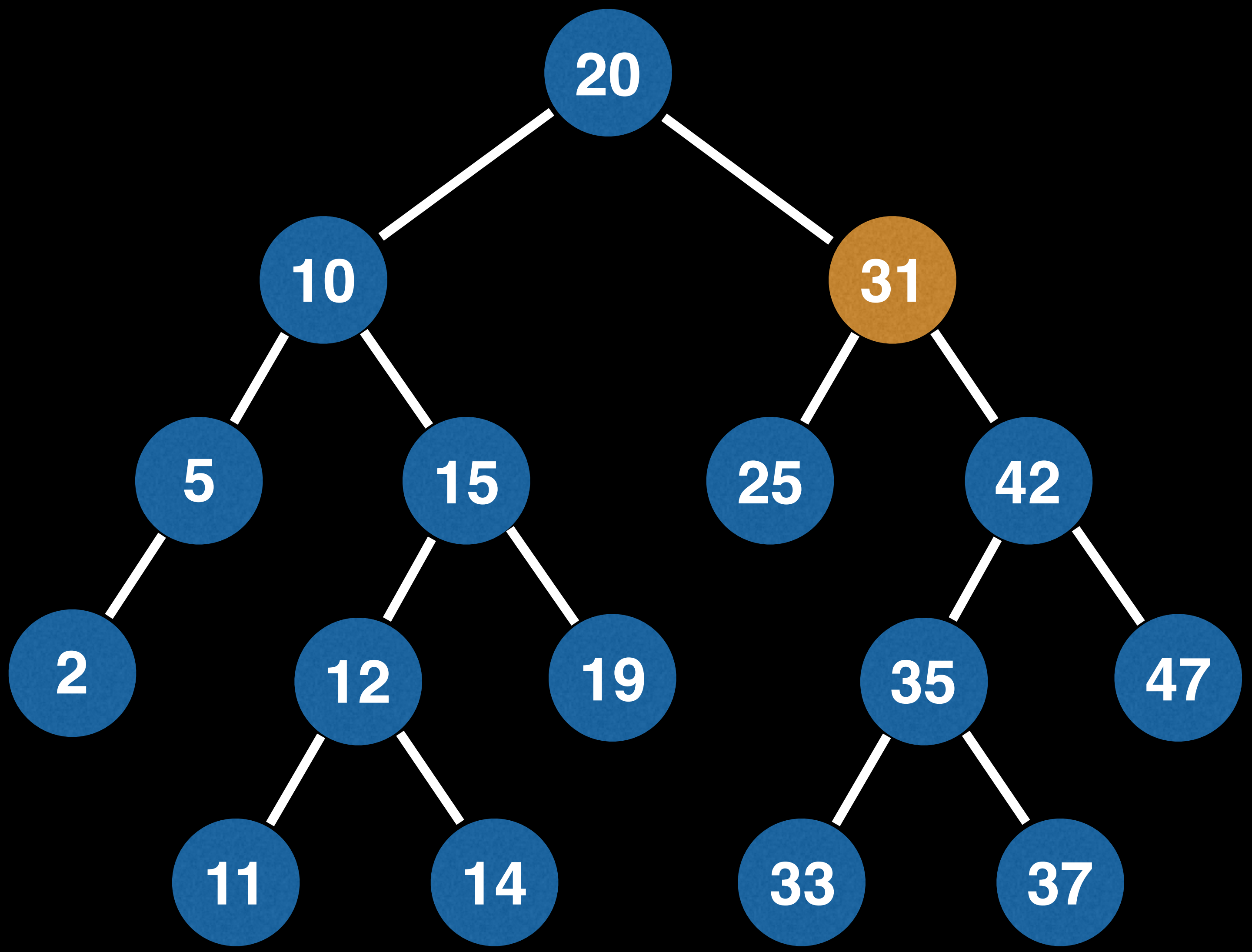
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

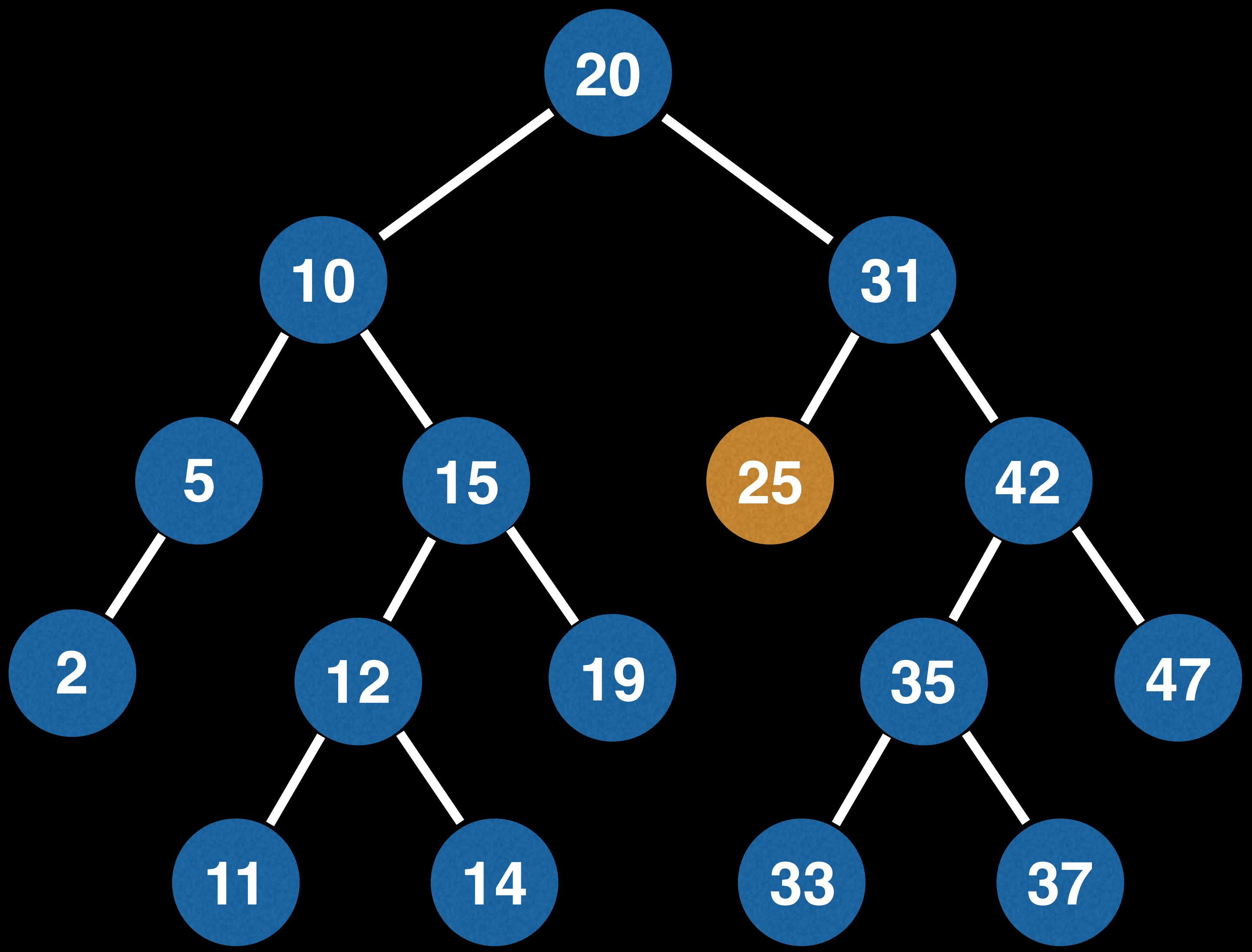
```
find(14)
find(25) ←
find(37)
find(17)
```



Find phase

Find queries:

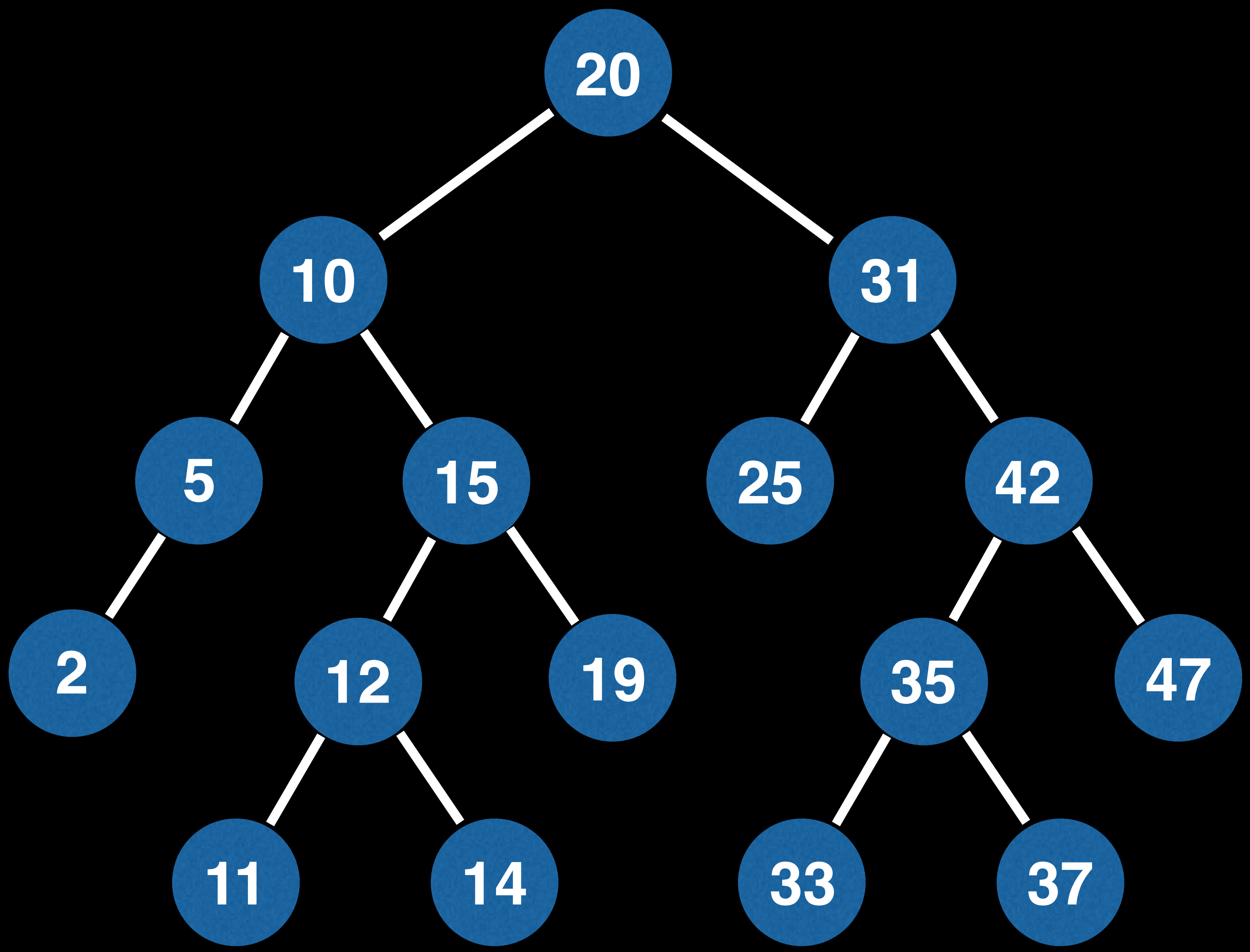
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

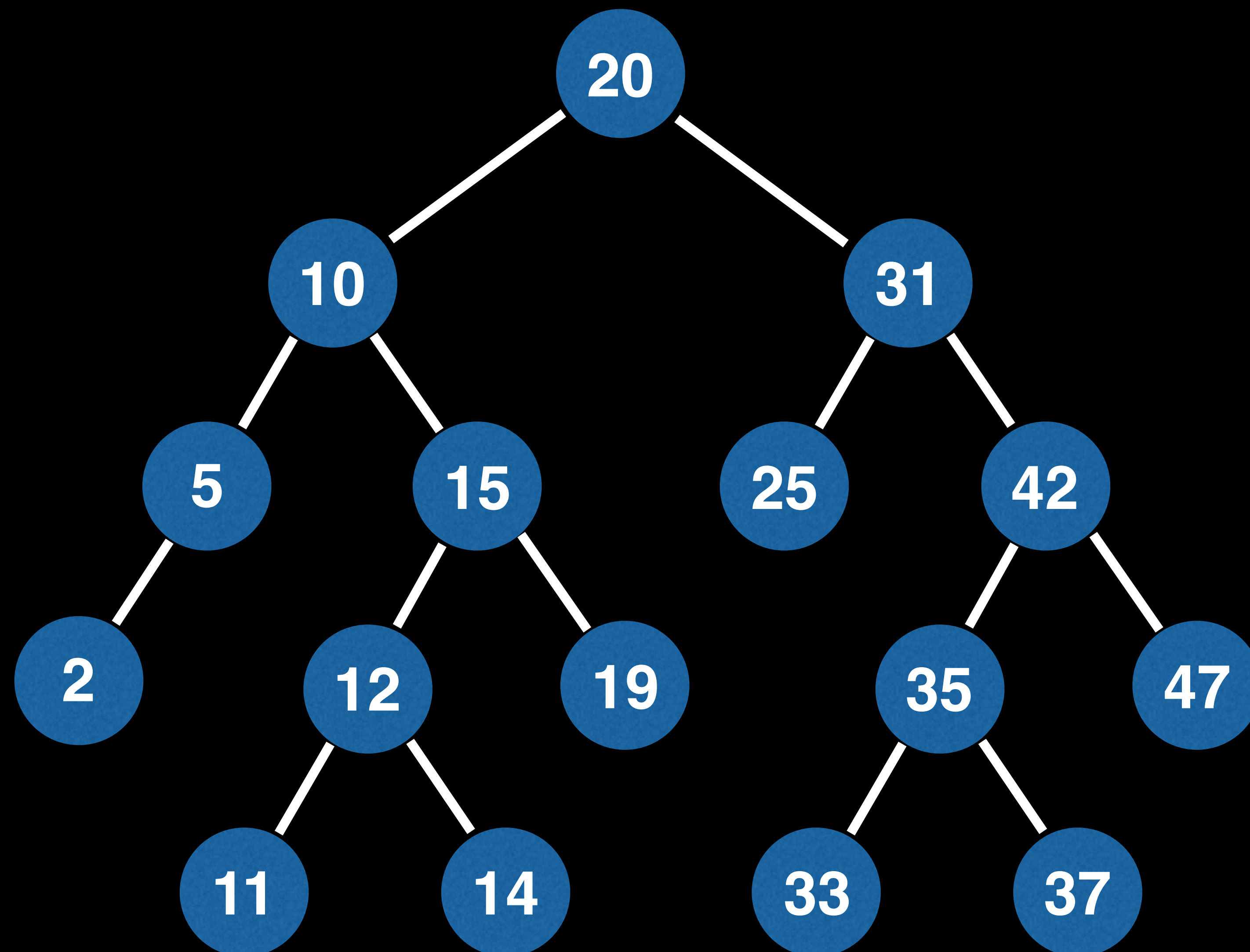
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

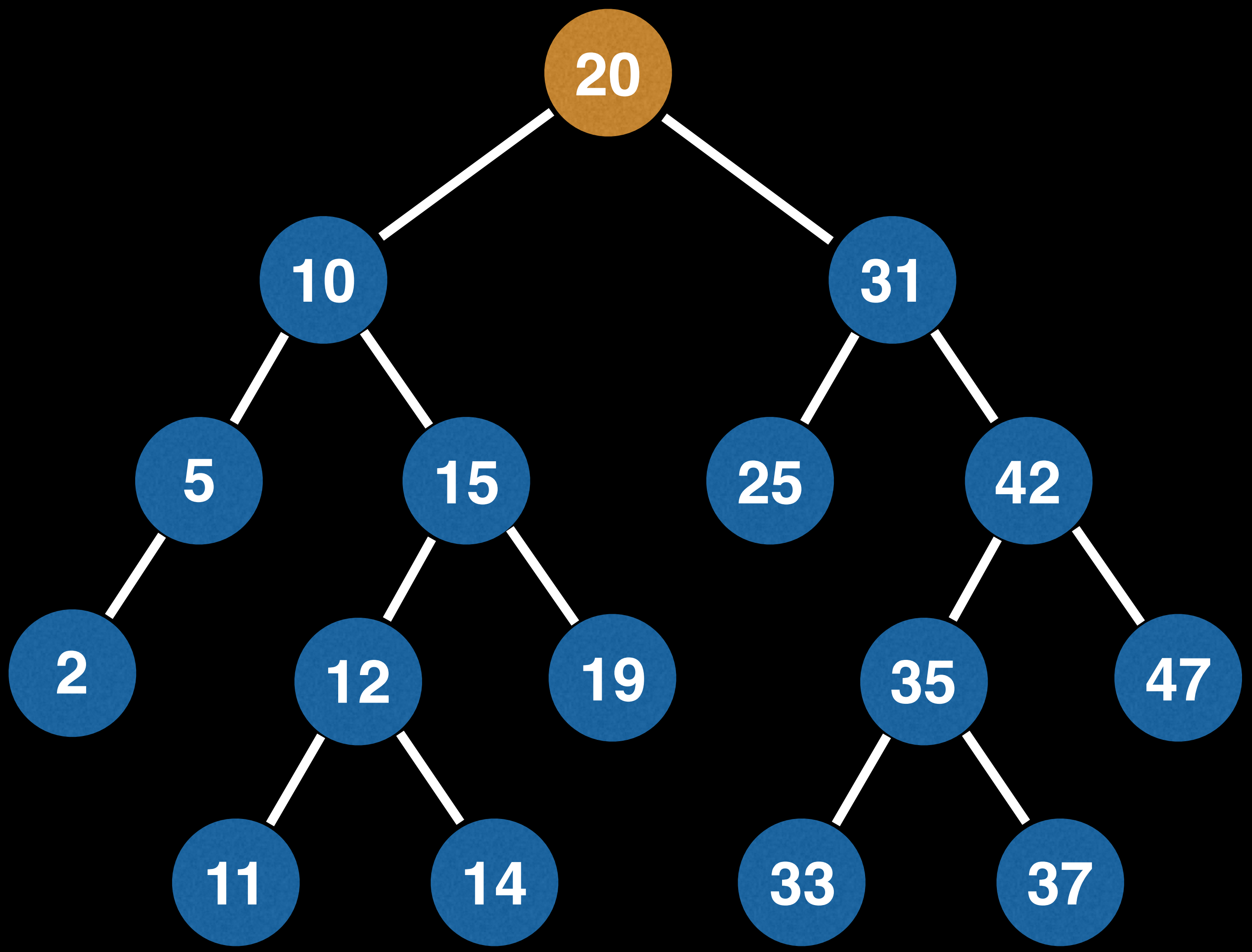
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

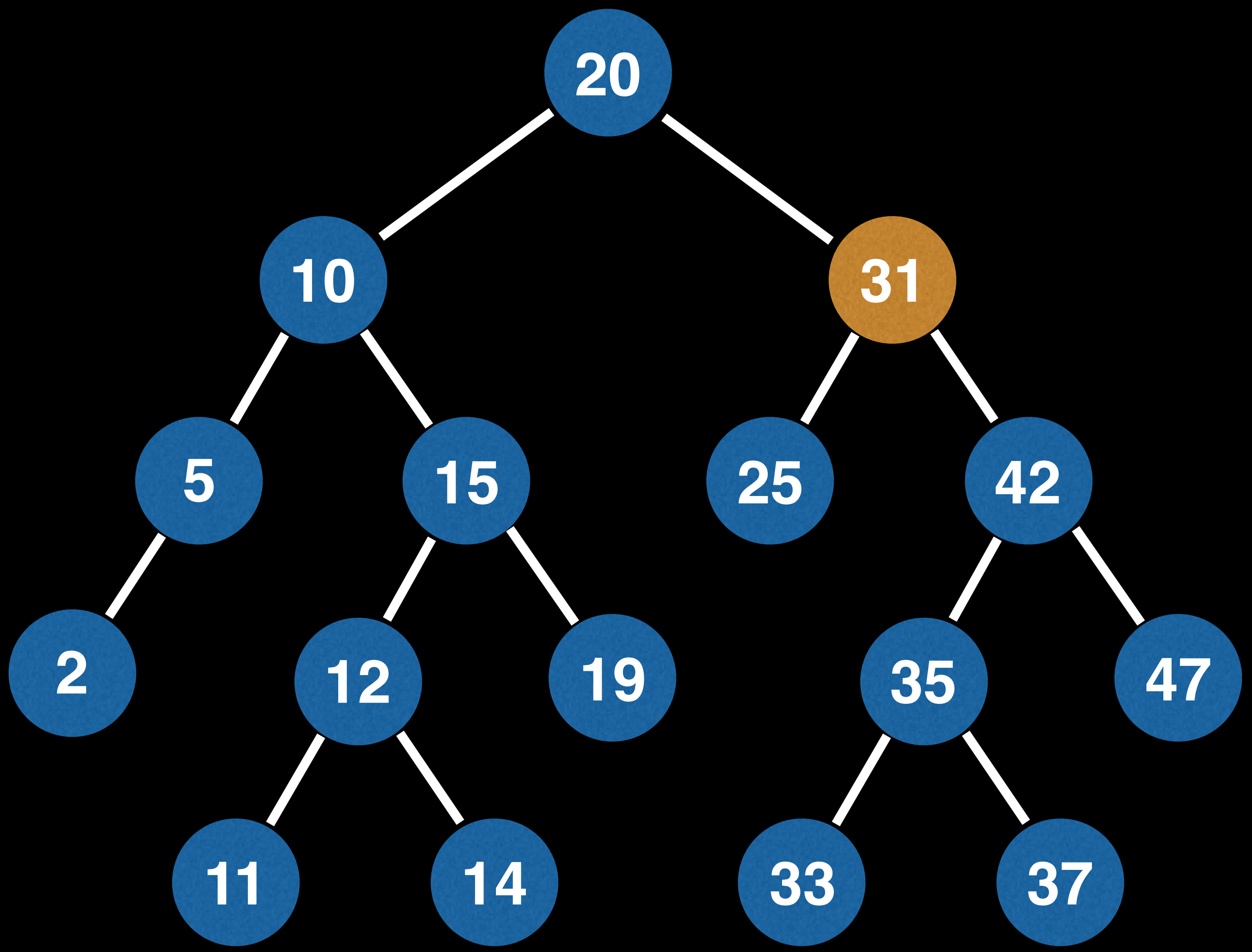
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

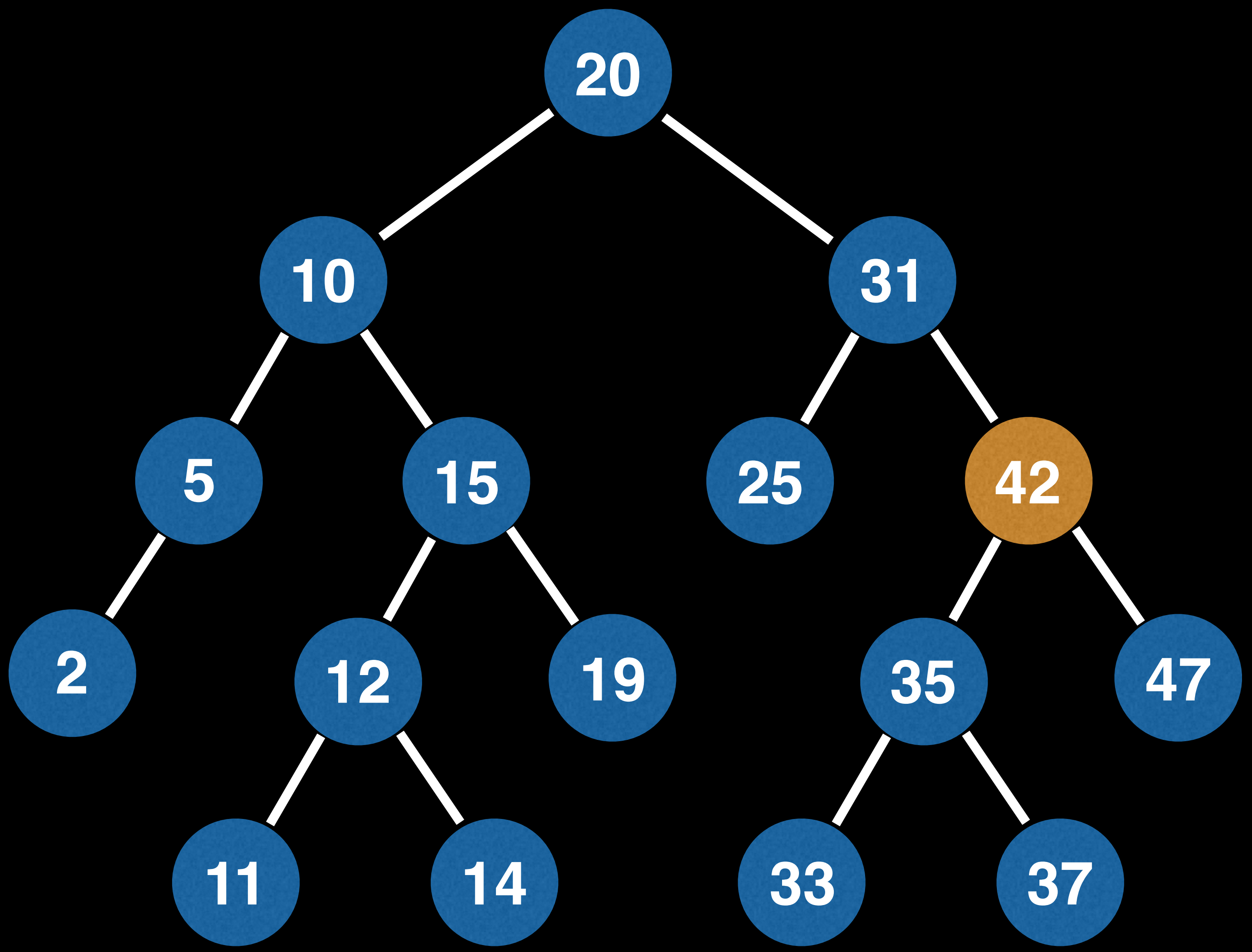
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

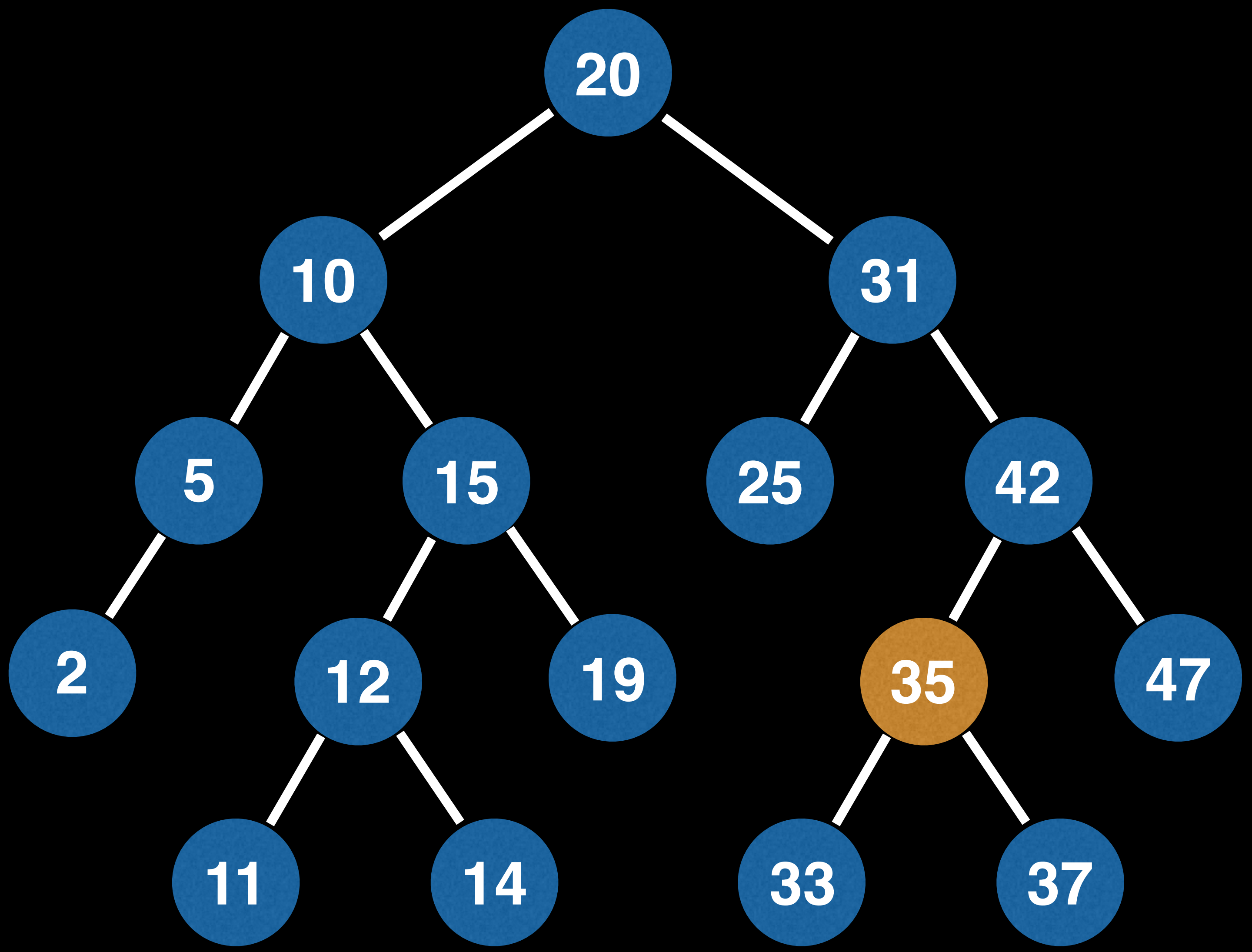
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

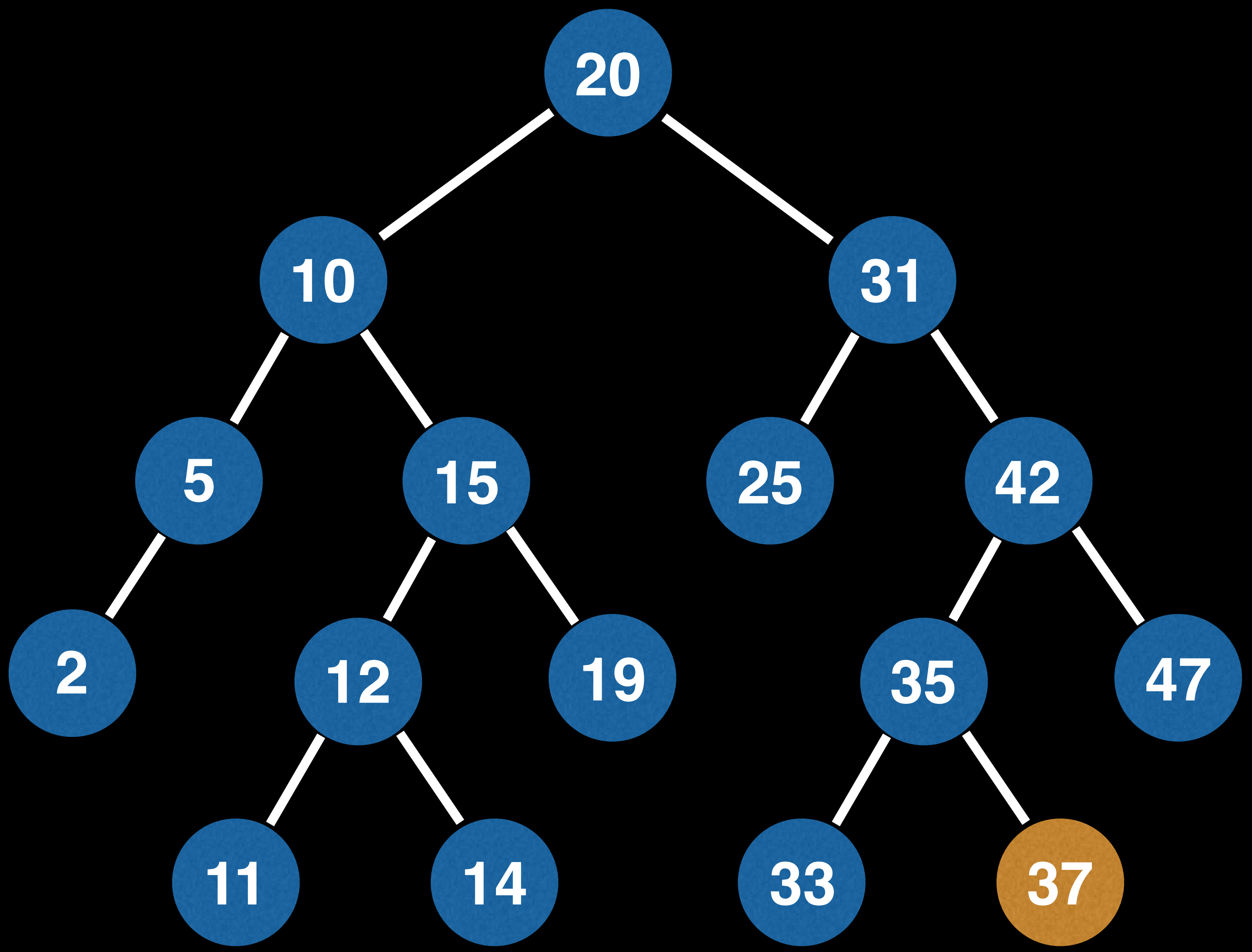
```
find(14)
find(25)
find(37) ←
find(17)
```



Find phase

Find queries:

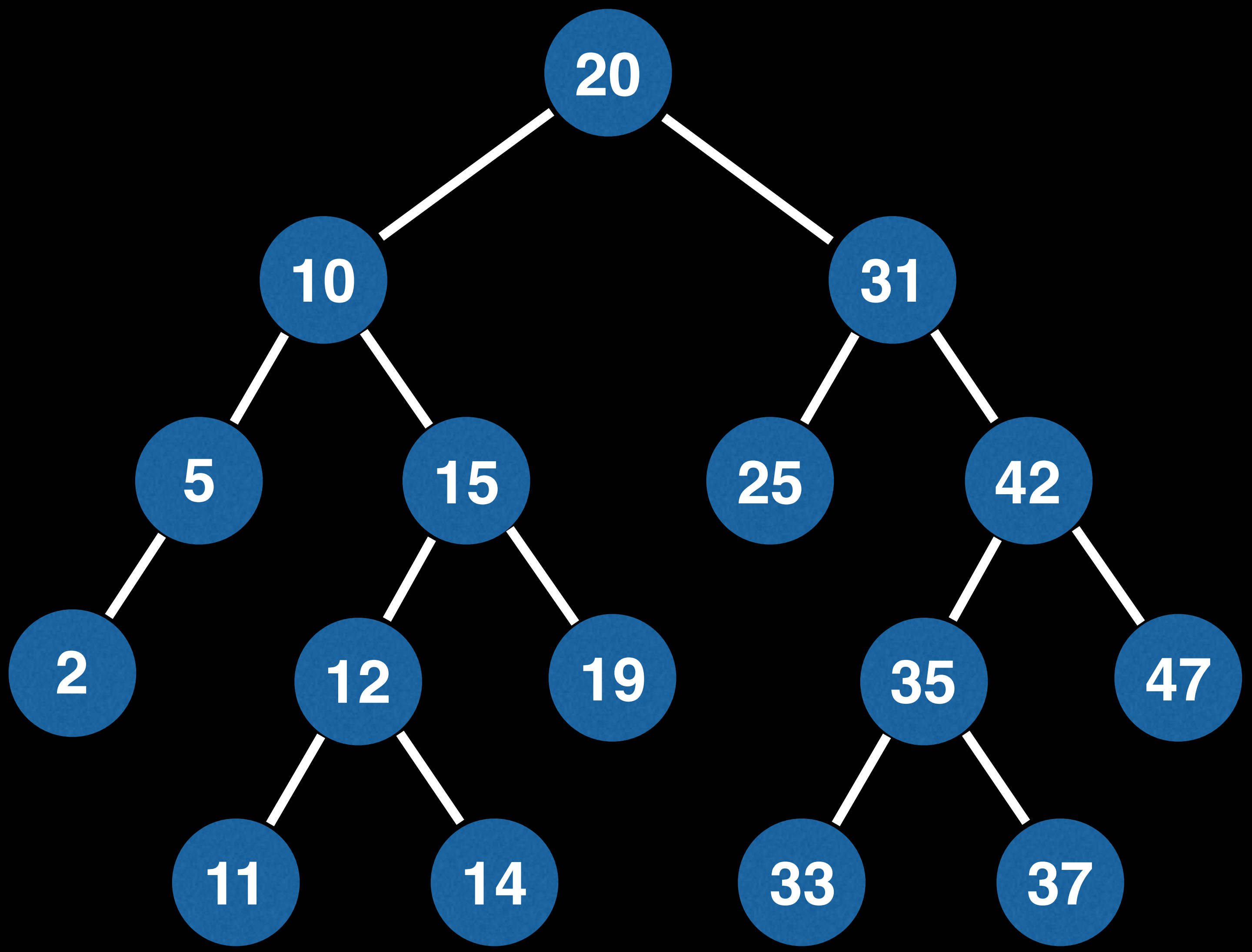
```
find(14)
find(25)
find(37) ←
find(17)
```



Find phase

Find queries:

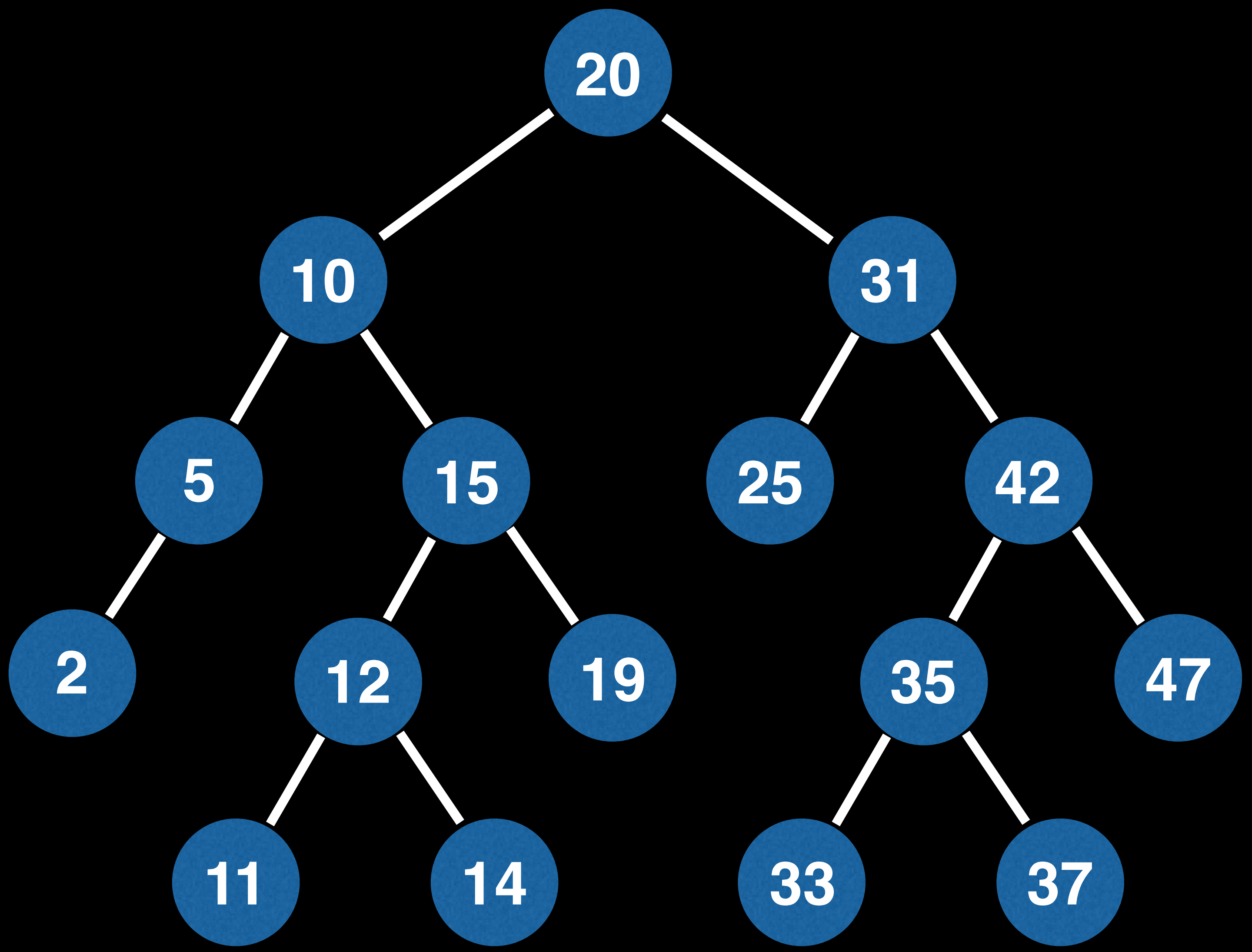
```
find(14)
find(25)
find(37) ←
find(17)
```



Find phase

Find queries:

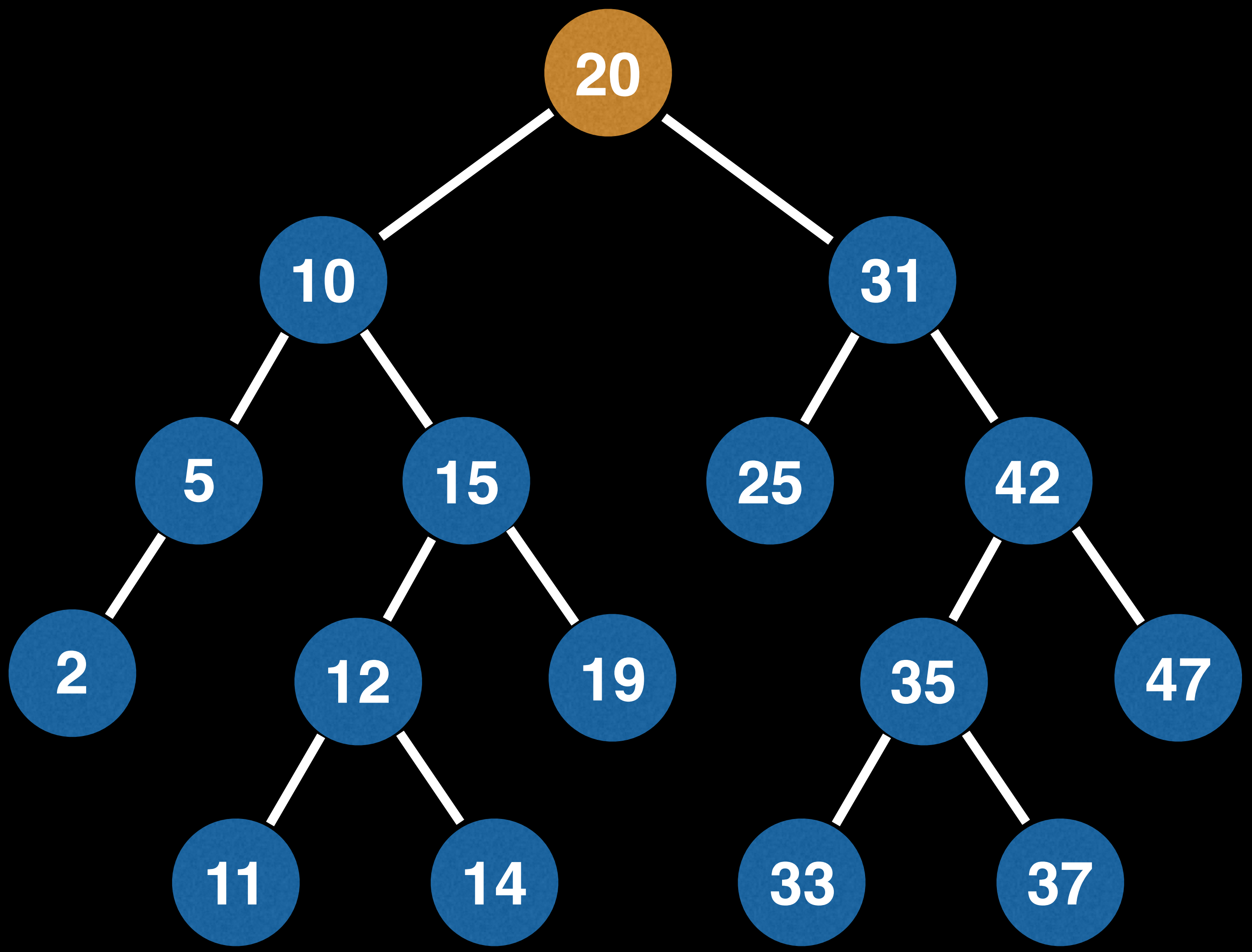
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

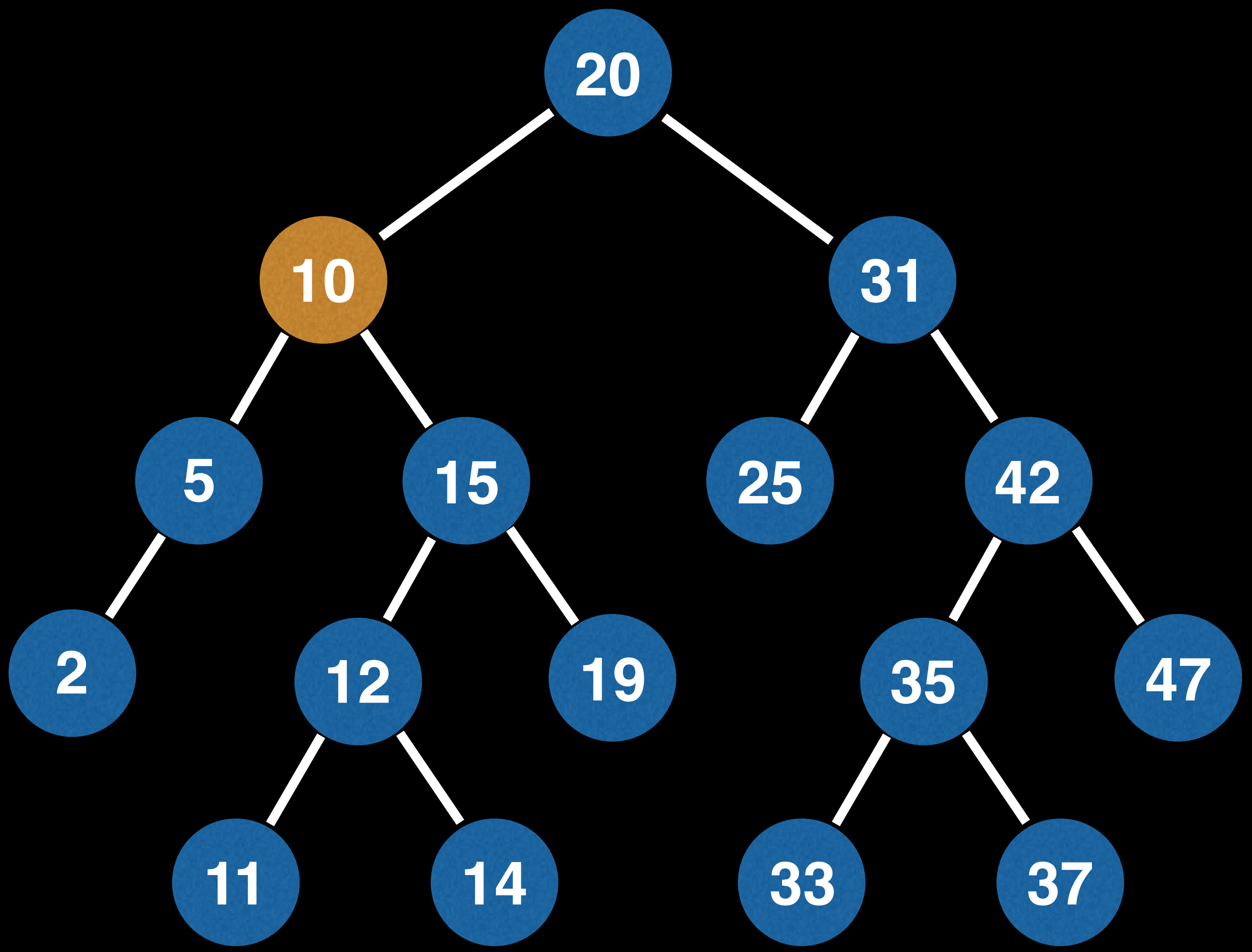
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

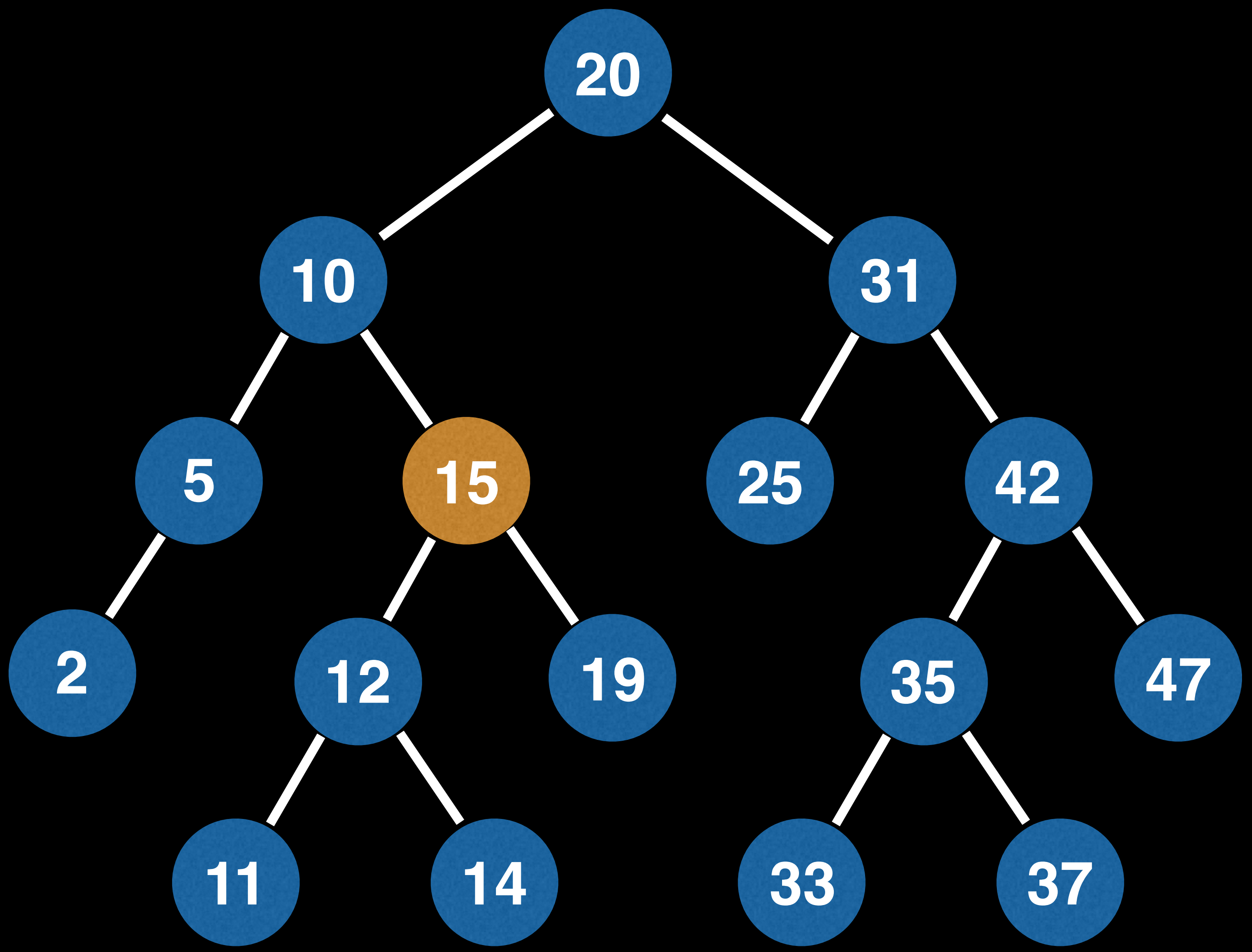
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

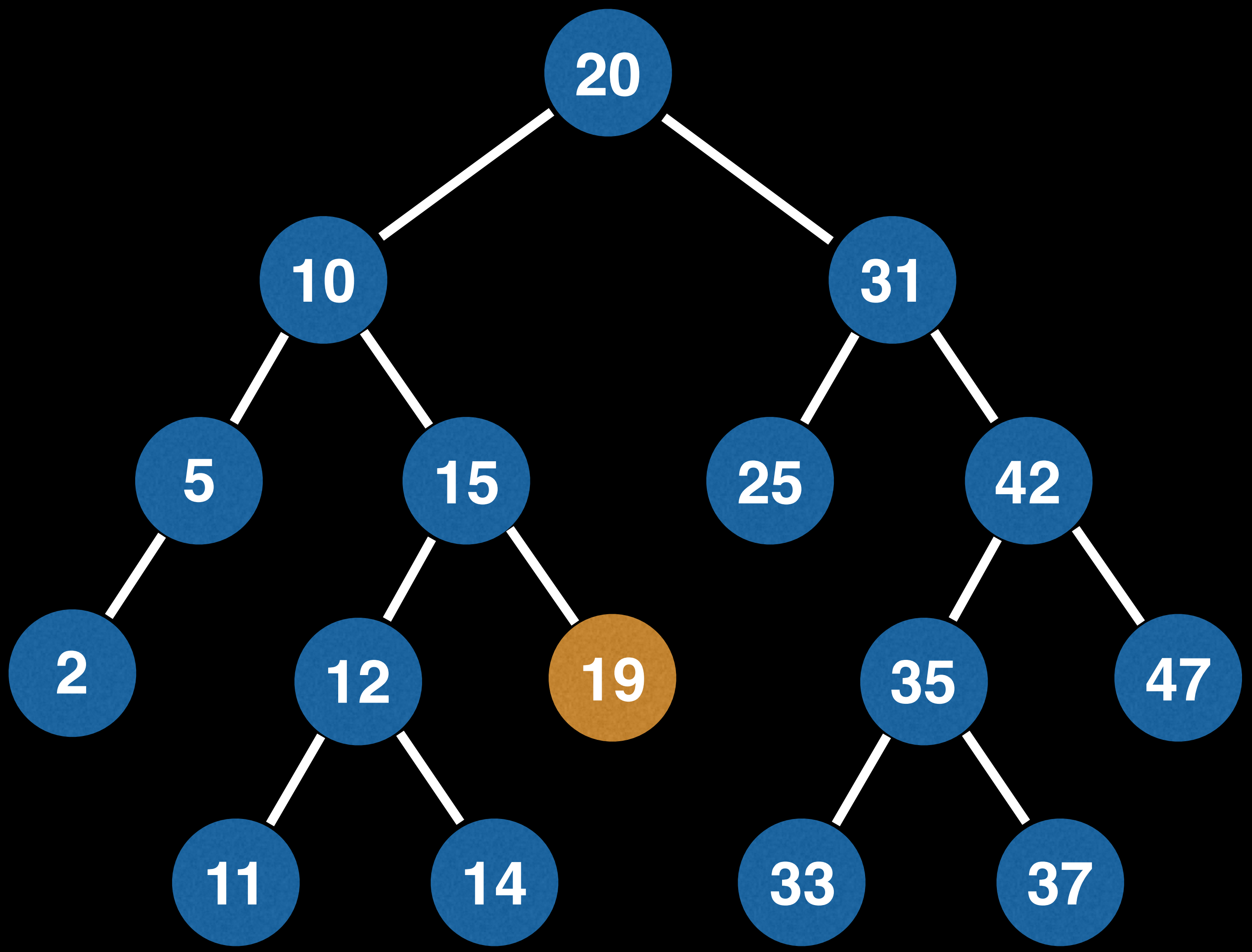
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

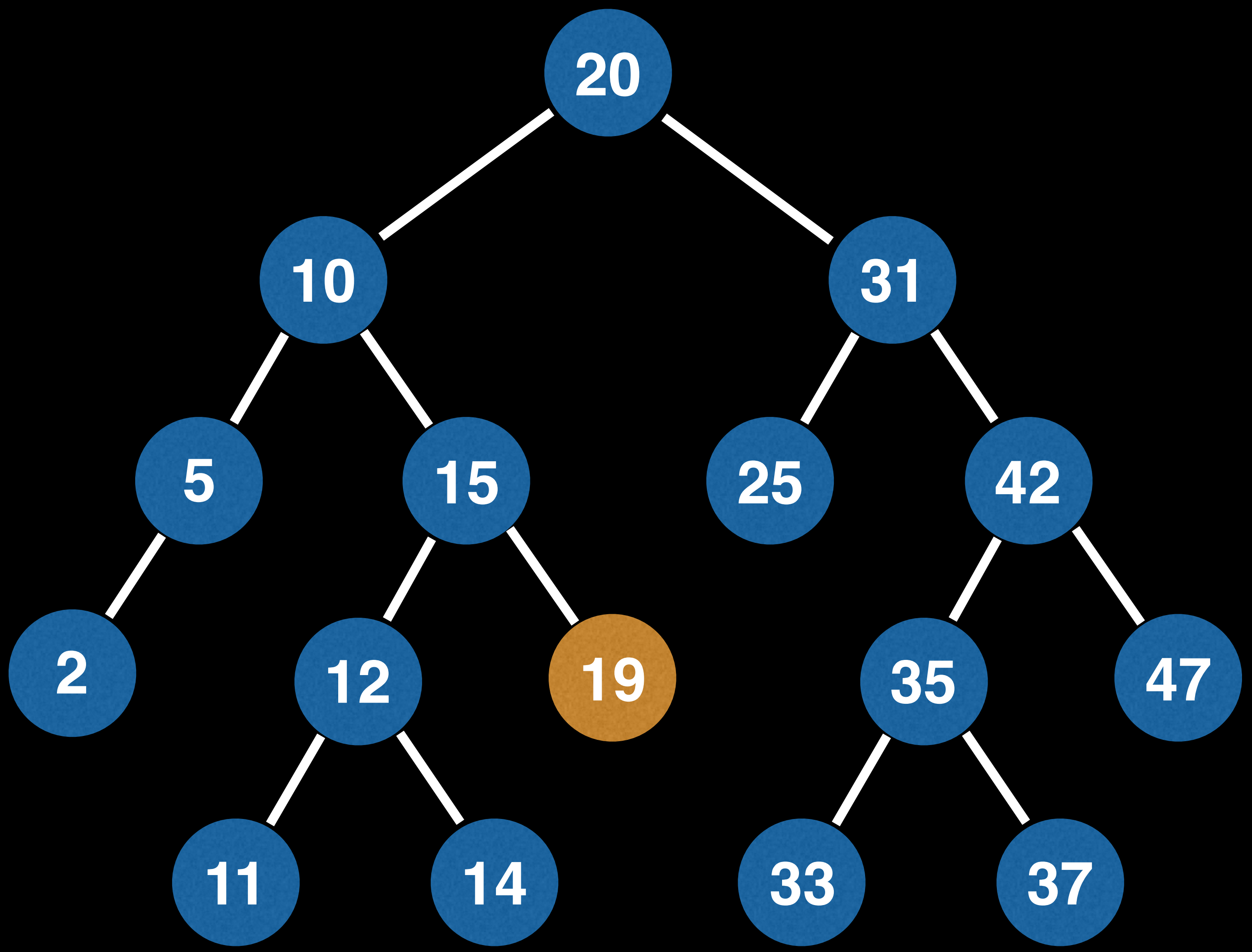
```
find(14)
find(25)
find(37)
find(17)
```



Find phase

Find queries:

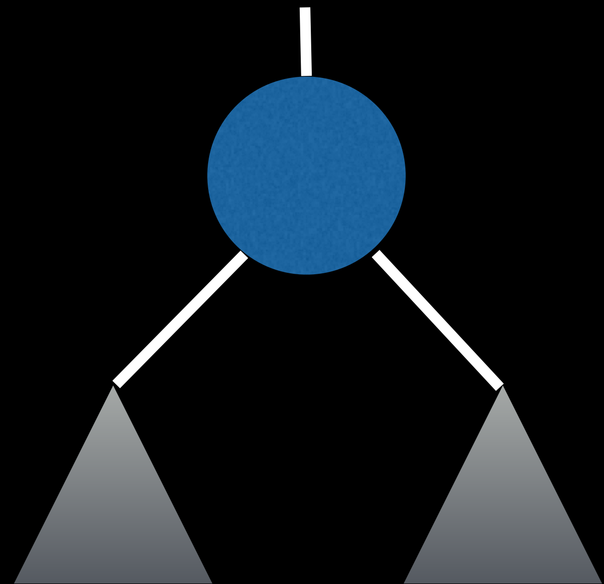
```
find(14)
find(25)
find(37)
find(17)
```



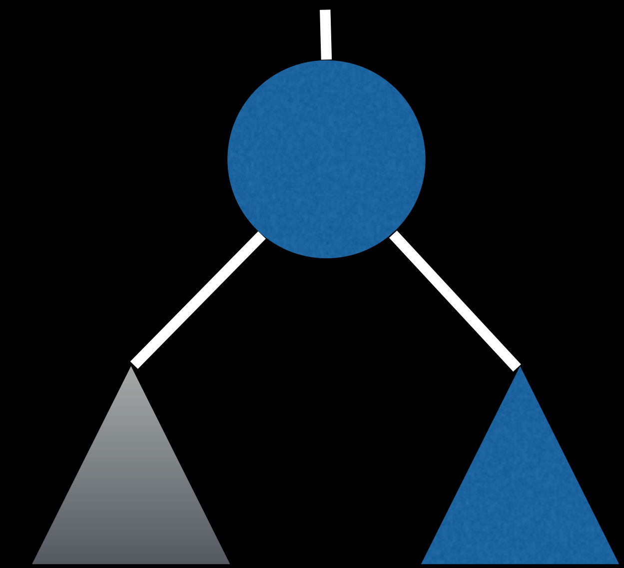
At this point we discover that 17 does not exist!

Remove phase

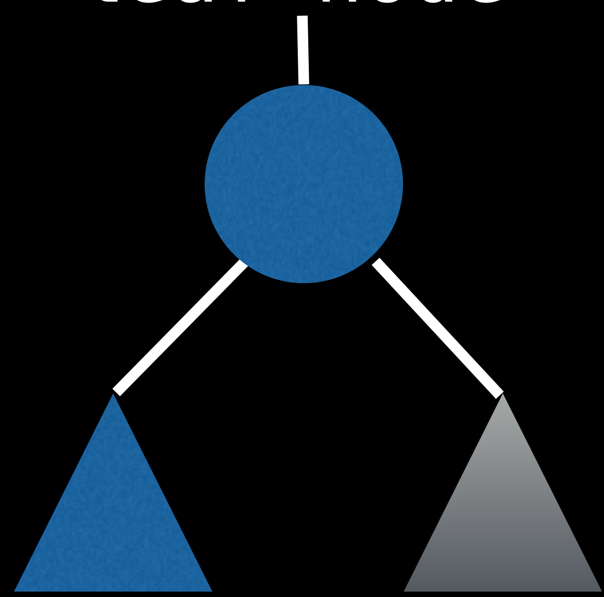
Four Cases



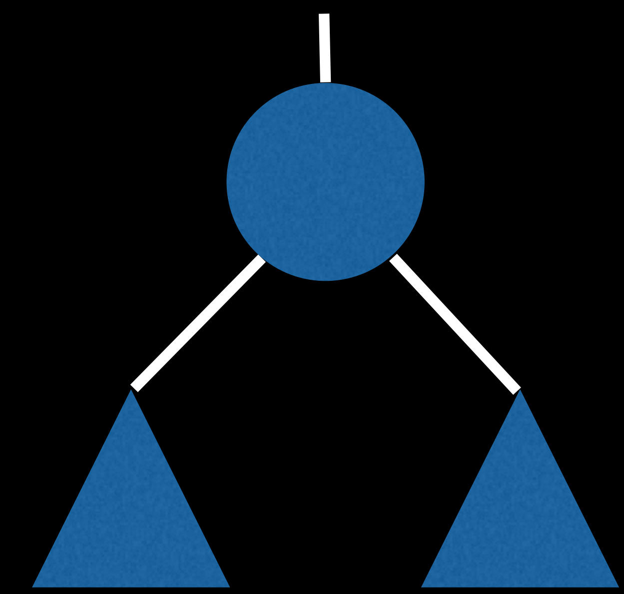
Node to remove is a leaf node



Node to remove has a right subtree but no left subtree

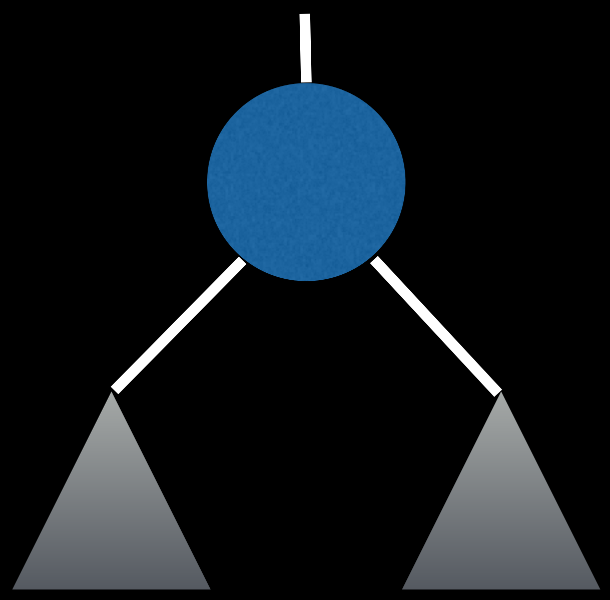


Node to remove has a left subtree but no right subtree



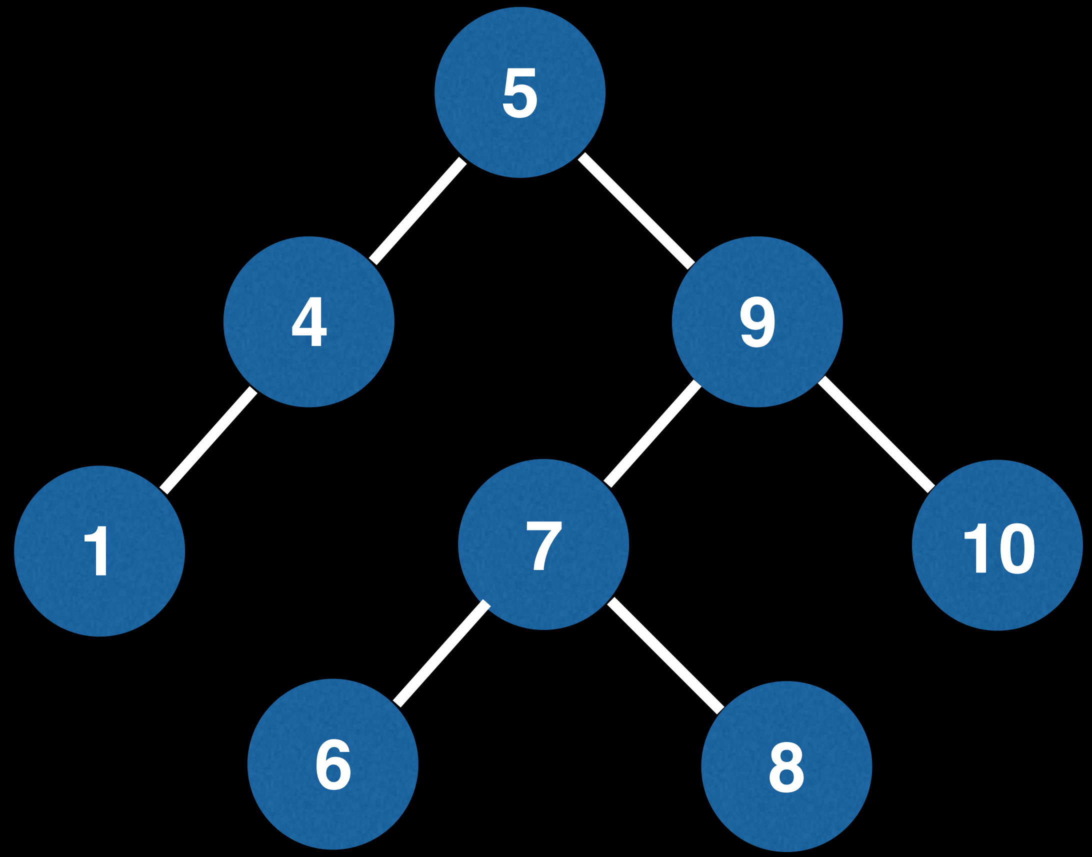
Node to remove has a both a left subtree and a right subtree

Remove phase

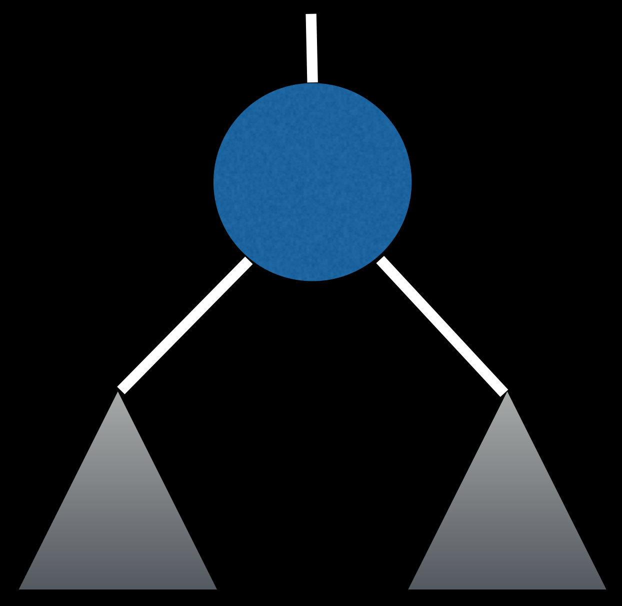


Case I: Leaf node

If the node we wish to remove is a leaf node then we may do so without side effect :)



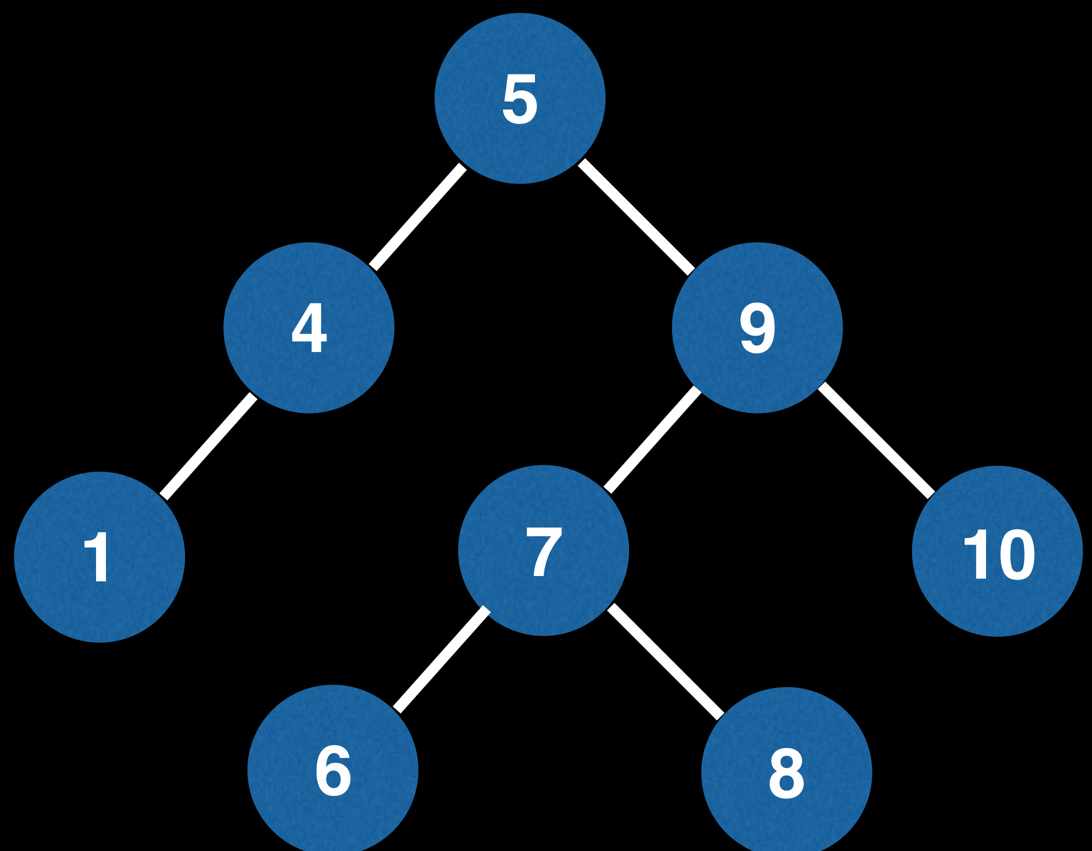
Remove phase



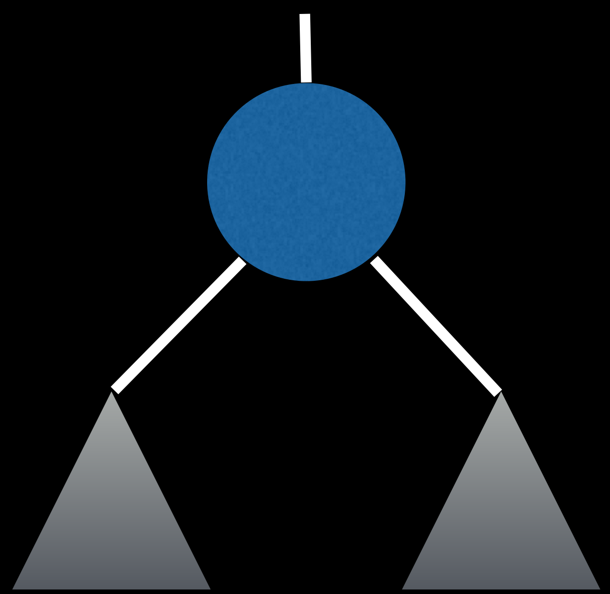
Case I: Leaf node

If the node we wish to remove is a leaf node then we may do so without side effect :)

Suppose we want to remove 8 from the BST on the right. First we would find 8 then remove it immediately since it's a leaf node

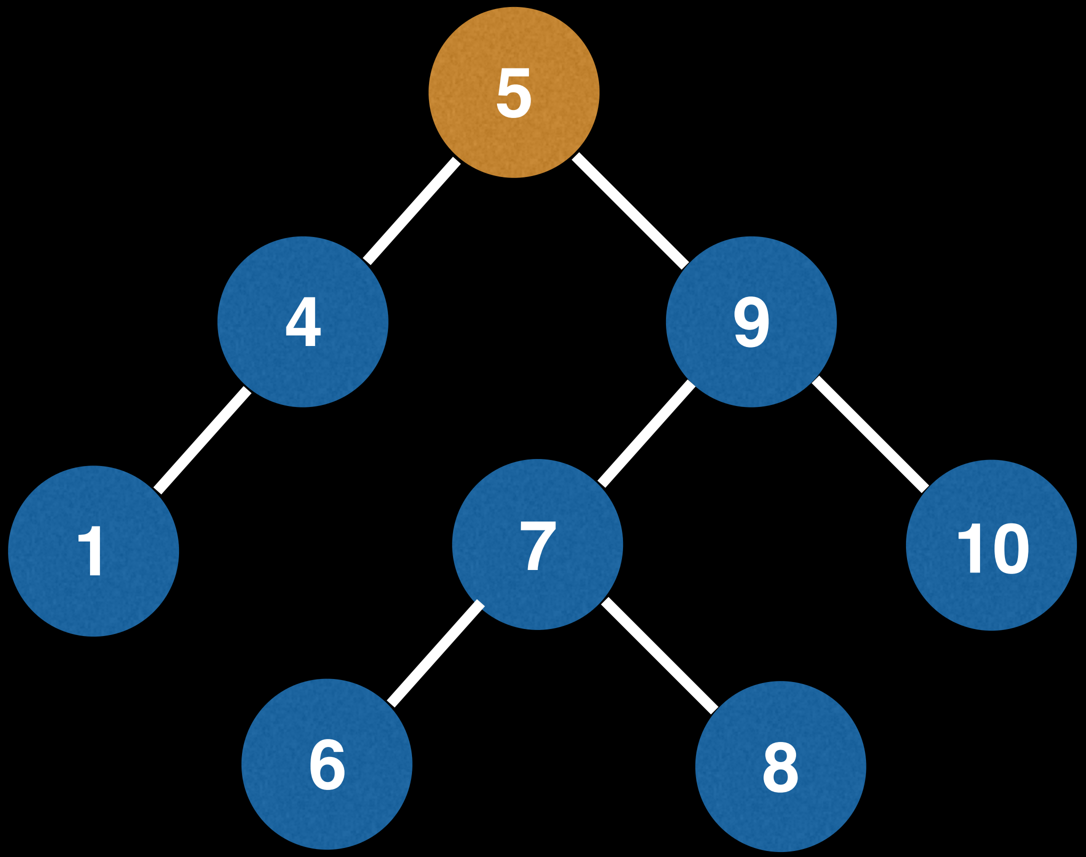


Remove phase



Case I: Leaf node

If the node we wish to remove is a leaf node then we may do so without side effect :)

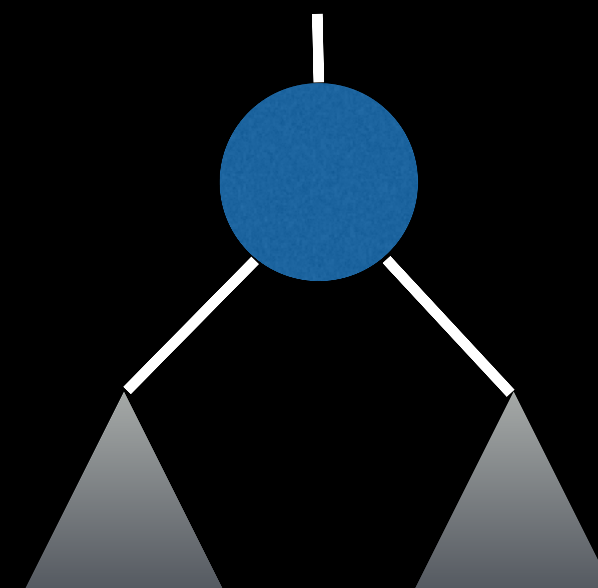


Suppose we want to remove 8 from the BST on the right. First we would find 8 then remove it immediately since it's a leaf node

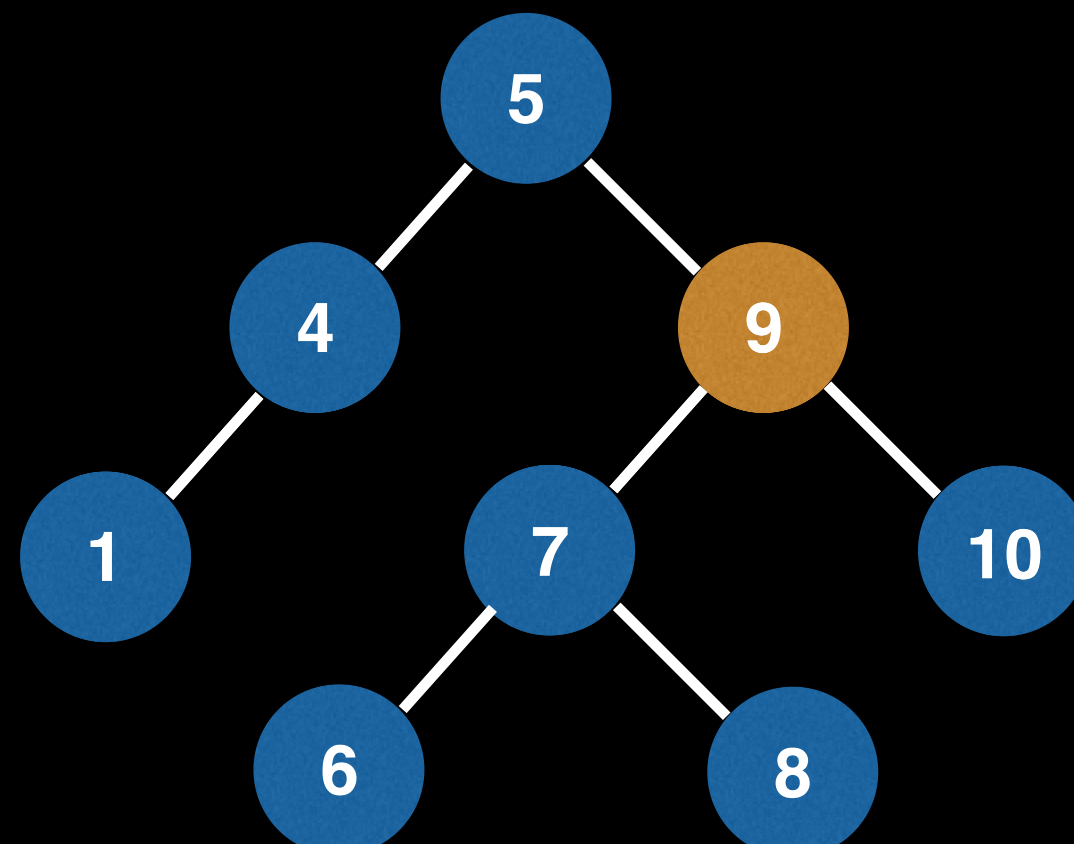
Remove phase

Case I: Leaf node

If the node we wish to remove is a leaf node then we may do so without side effect :)

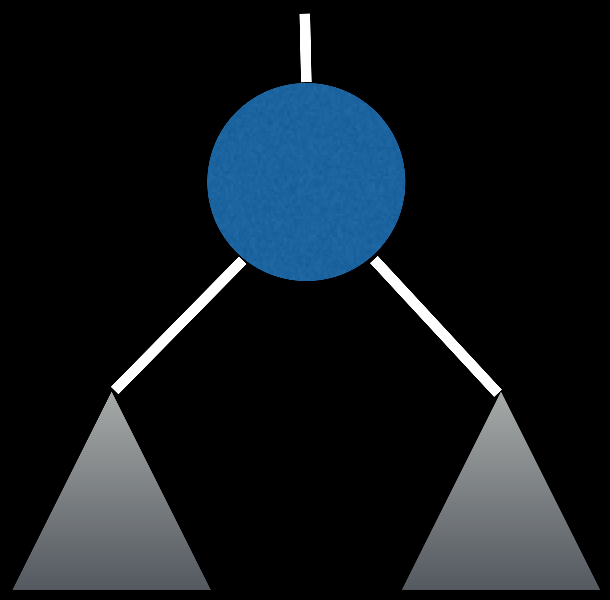


Suppose we want to remove 8 from the BST on the right. First we would find 8 then remove it immediately since it’s a leaf node



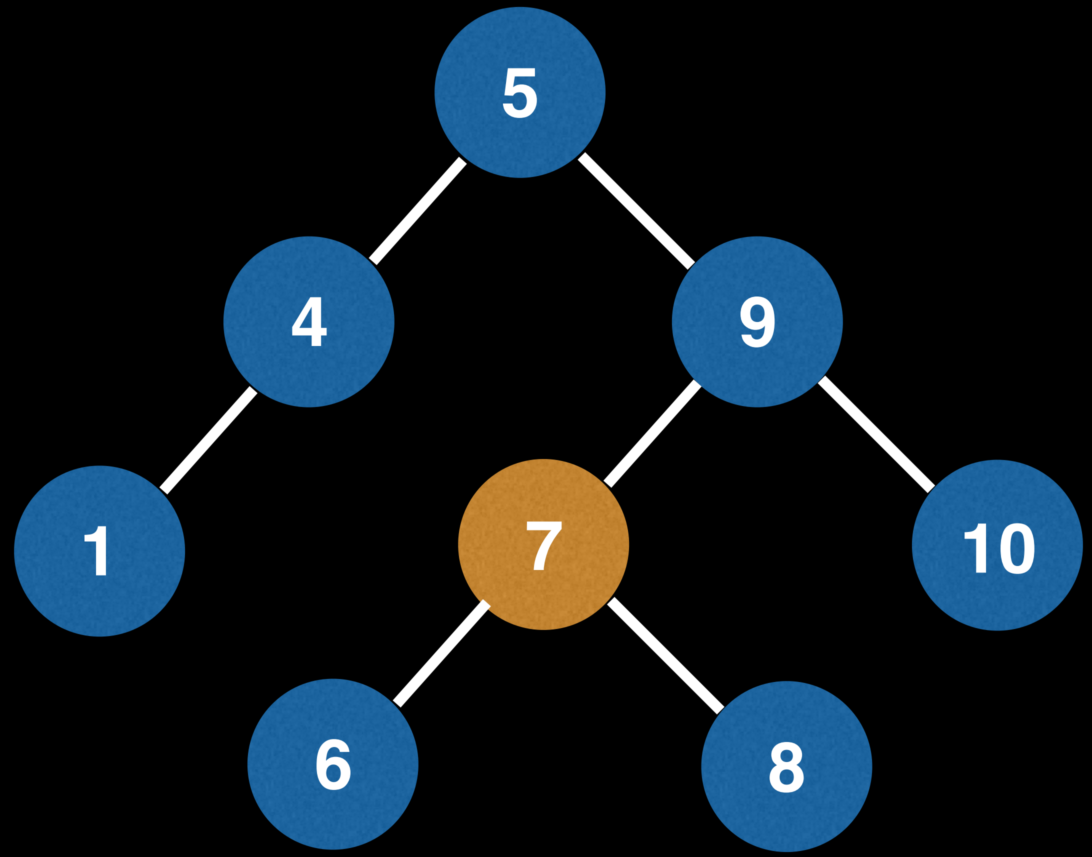
Remove phase

Case I: Leaf node



If the node we wish to remove is a leaf node then we may do so without side effect :)

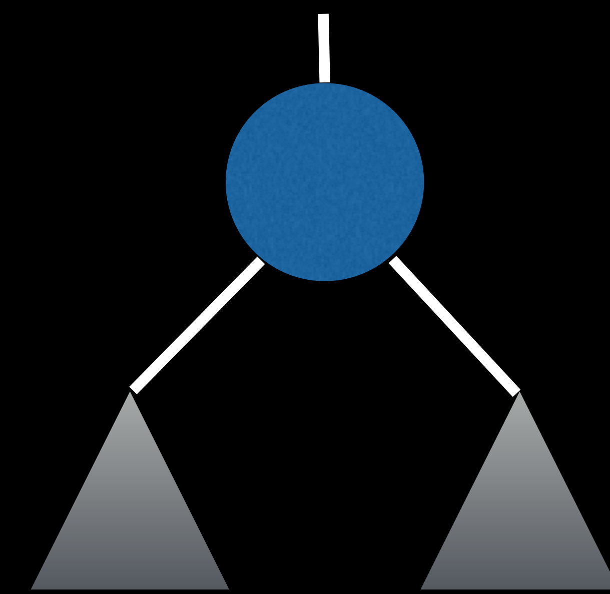
Suppose we want to remove 8 from the BST on the right. First we would find 8 then remove it immediately since it's a leaf node



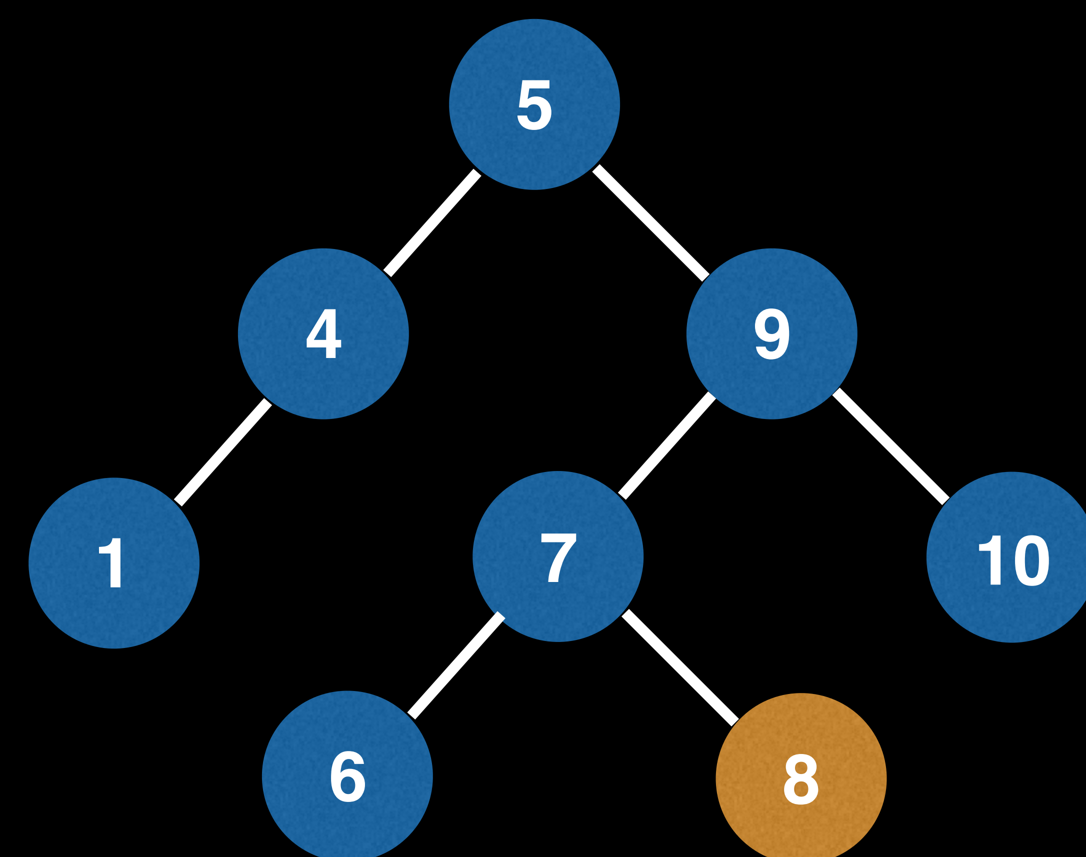
Remove phase

Case I: Leaf node

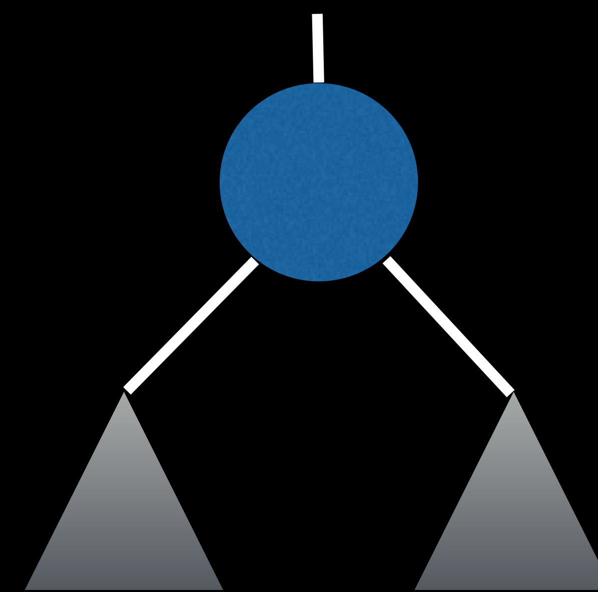
If the node we wish to remove is a leaf node then we may do so without side effect :)



Suppose we want to remove 8 from the BST on the right. First we would find 8 then remove it immediately since it’s a leaf node



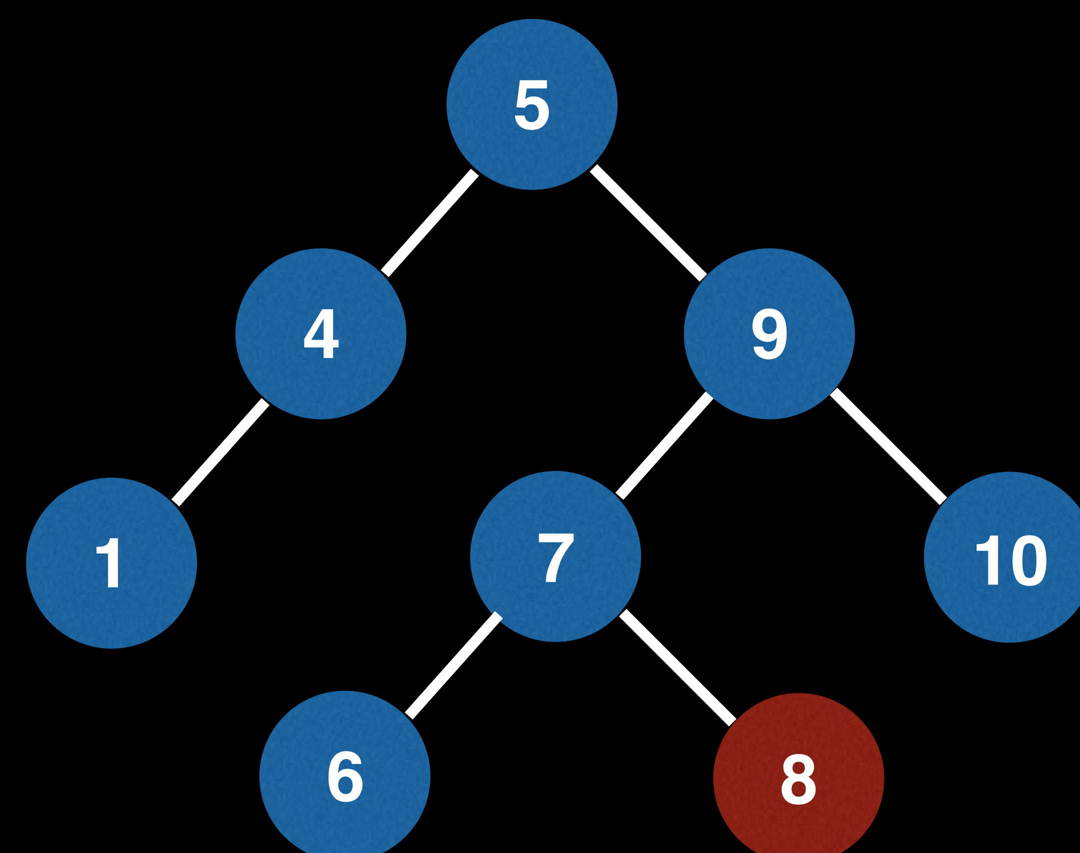
Remove phase



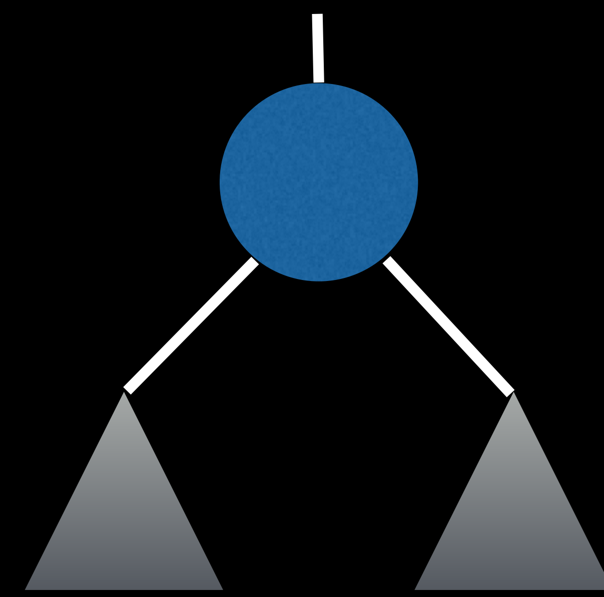
Case I: Leaf node

If the node we wish to remove is a leaf node then we may do so without side effect :)

Suppose we want to remove 8 from the BST on the right. First we would find 8 then remove it immediately since it's a leaf node



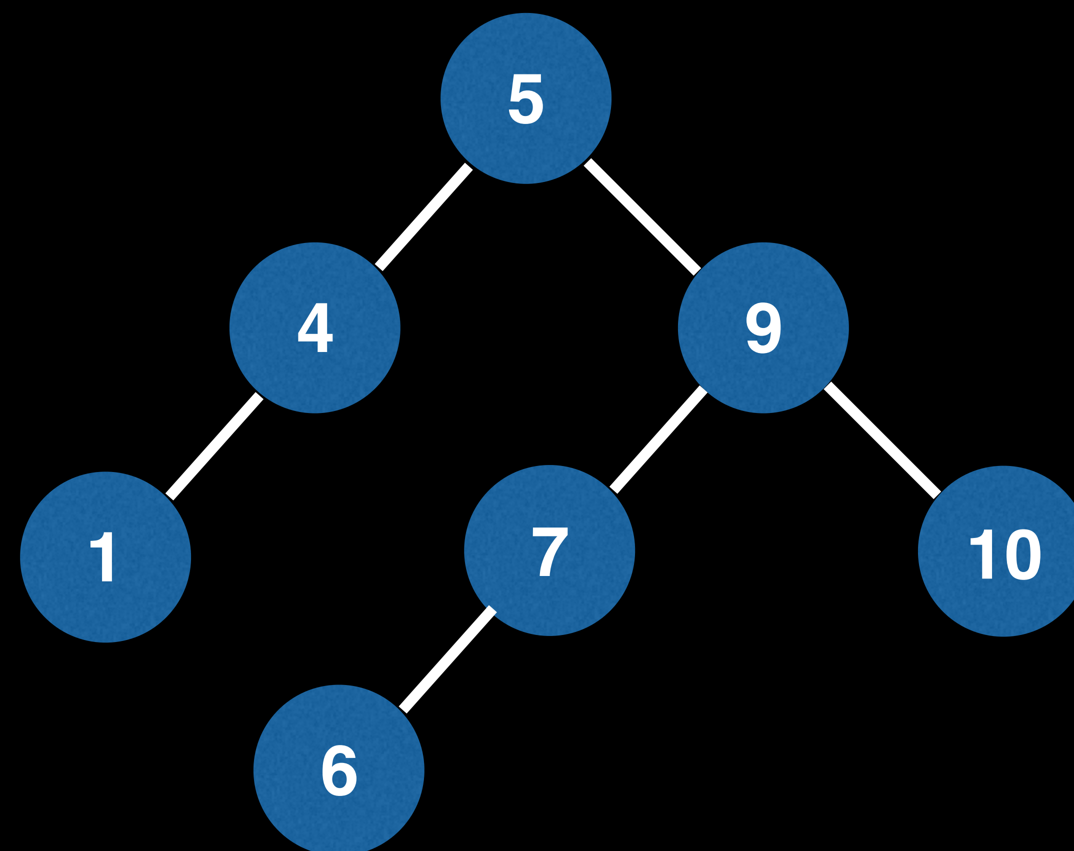
Remove phase



Case I: Leaf node

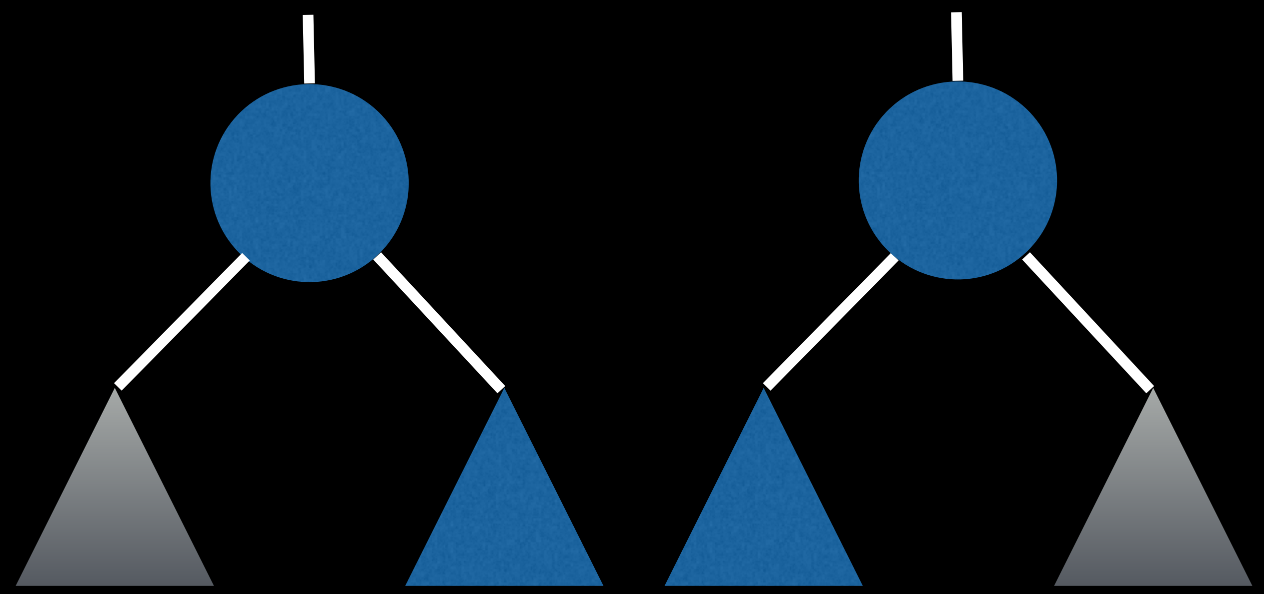
If the node we wish to remove is a leaf node then we may do so without side effect :)

Suppose we want to remove 8 from the BST on the right. First we would find 8 then remove it immediately since it’s a leaf node



Remove phase

Cases II & III: either the left/right child node is a subtree

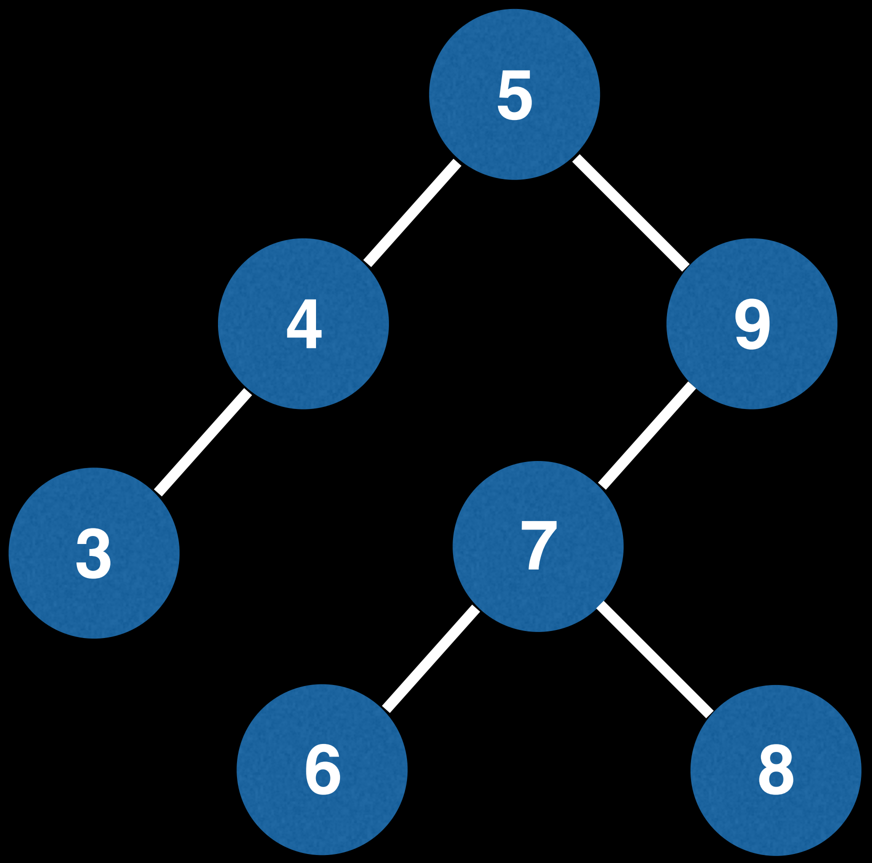


The successor of the node we are trying to remove in these cases will be the **root node of the left/right subtree**.

It may be the case that you are removing the root node of the BST in which case its immediate child becomes the new root as you would expect.

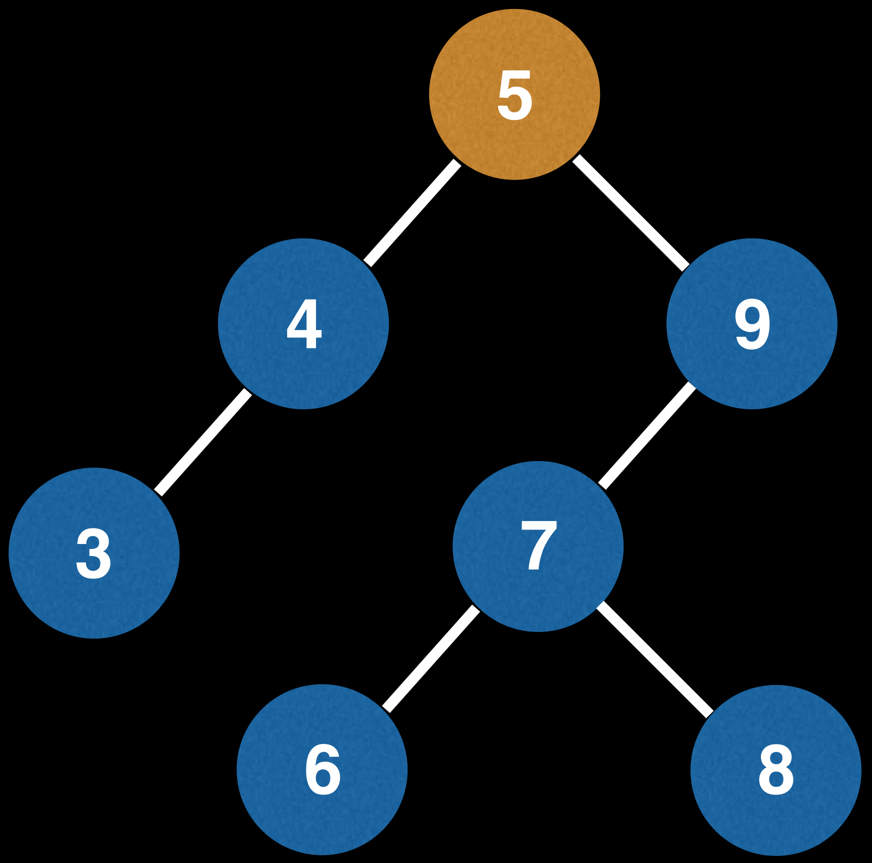
Remove phase

Suppose we wish to remove 9,
then we encounter case II
with a left subtree



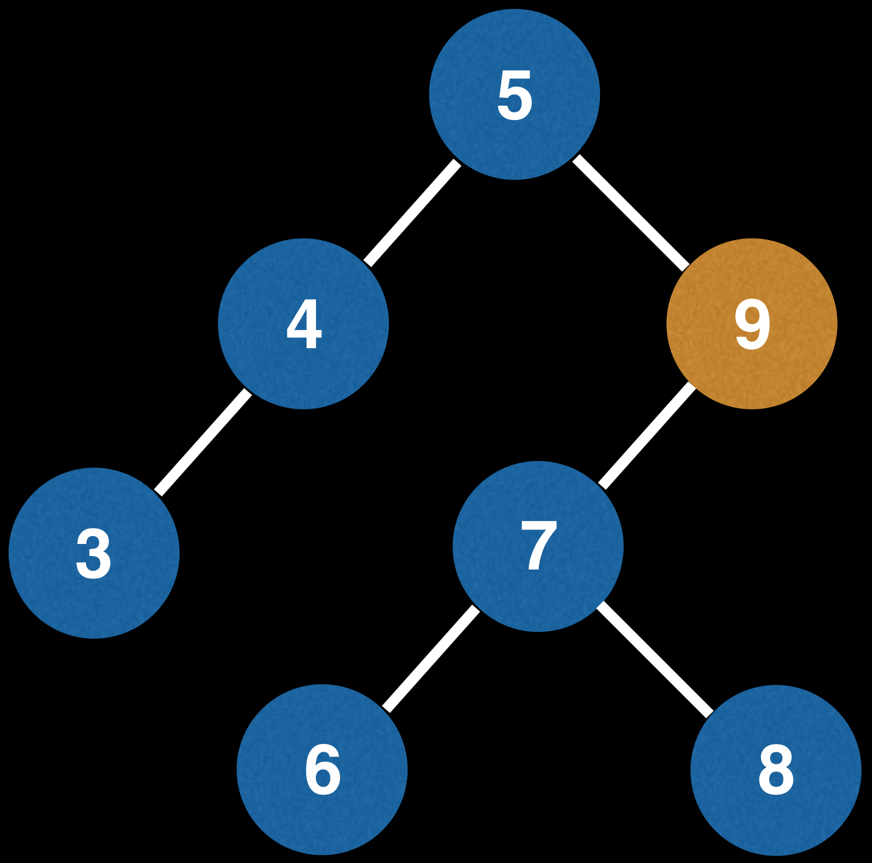
Remove phase

Suppose we wish to remove 9,
then we encounter case II
with a left subtree



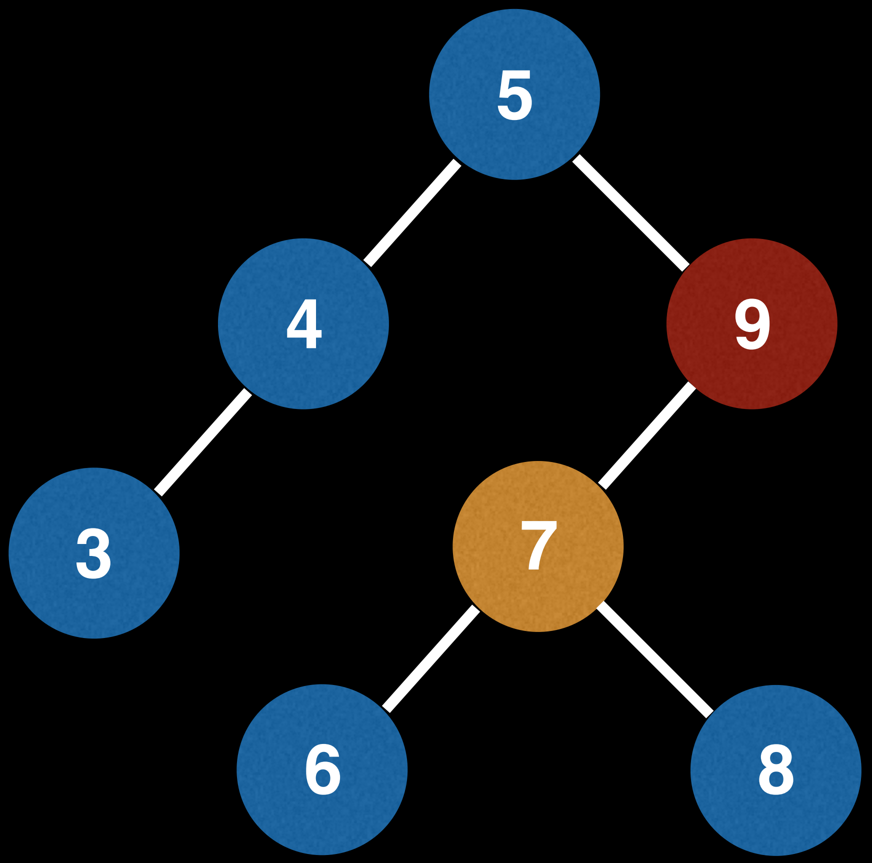
Remove phase

Suppose we wish to remove 9,
then we encounter case II
with a left subtree



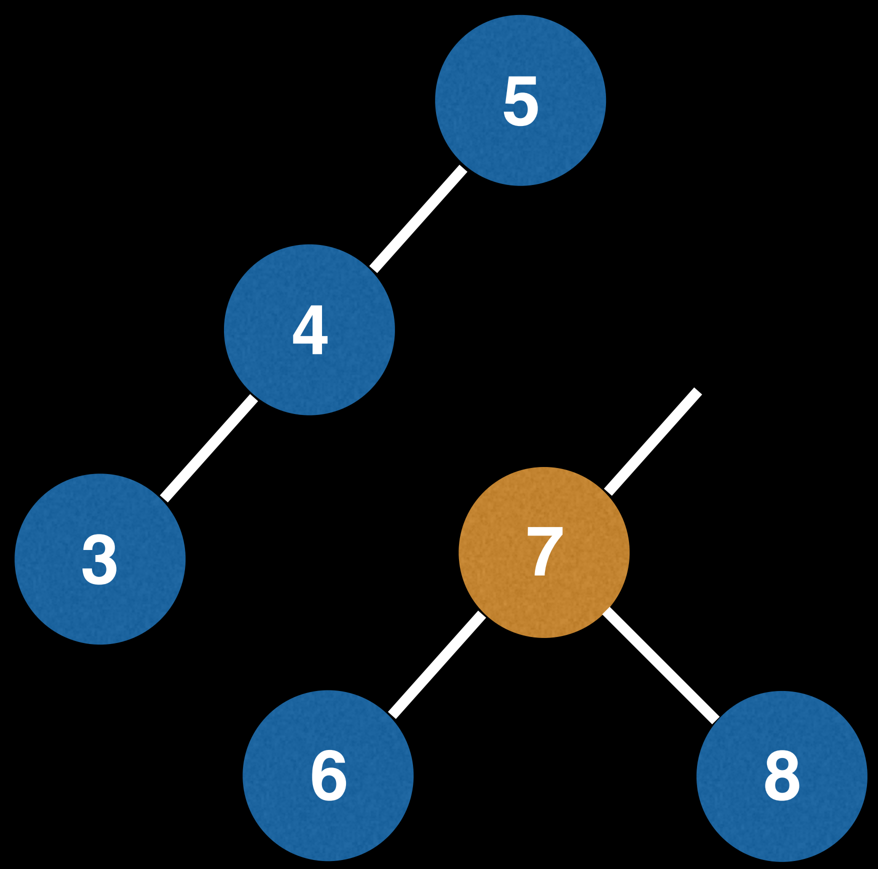
Remove phase

Suppose we wish to remove 9,
then we encounter case II
with a left subtree



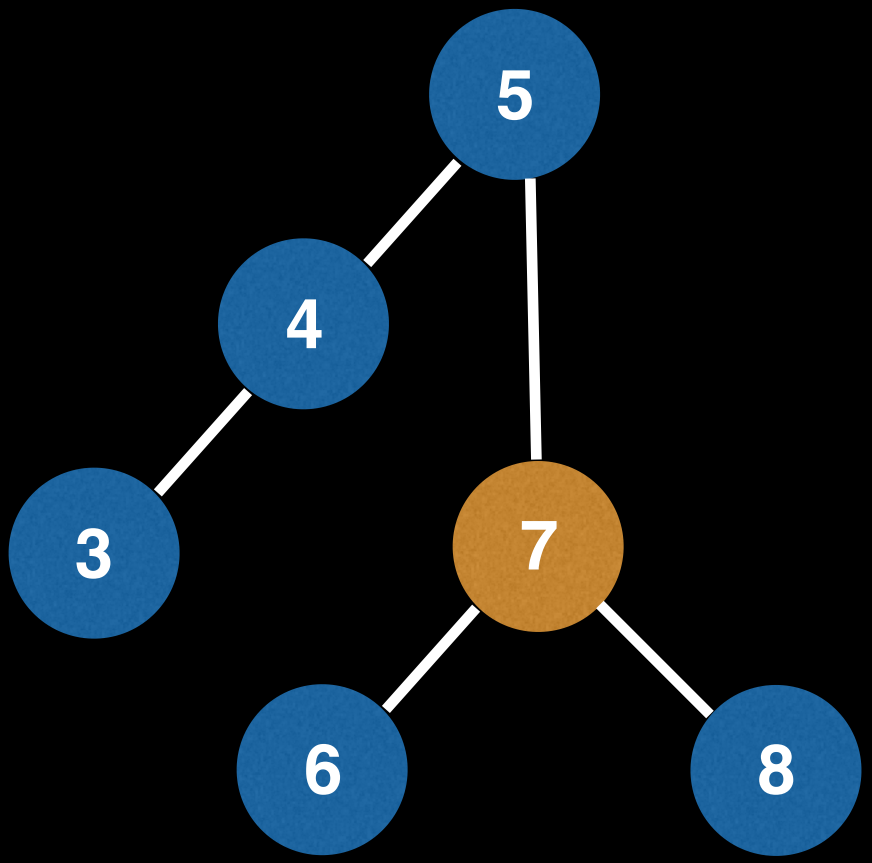
Remove phase

Suppose we wish to remove 9,
then we encounter case II
with a left subtree



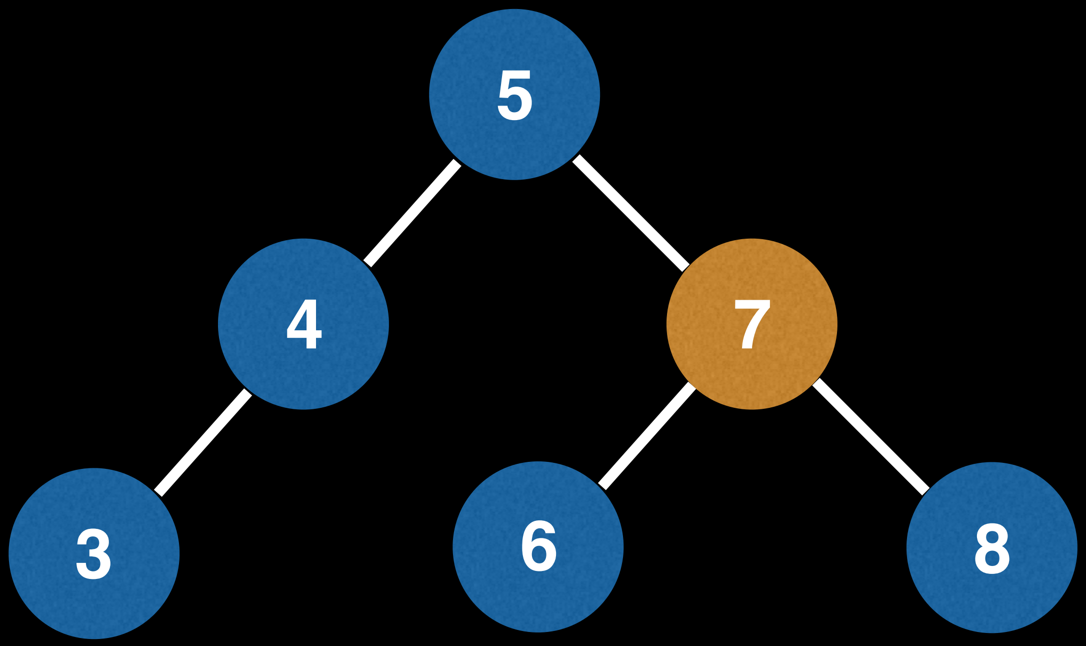
Remove phase

Suppose we wish to remove 9,
then we encounter case II
with a left subtree



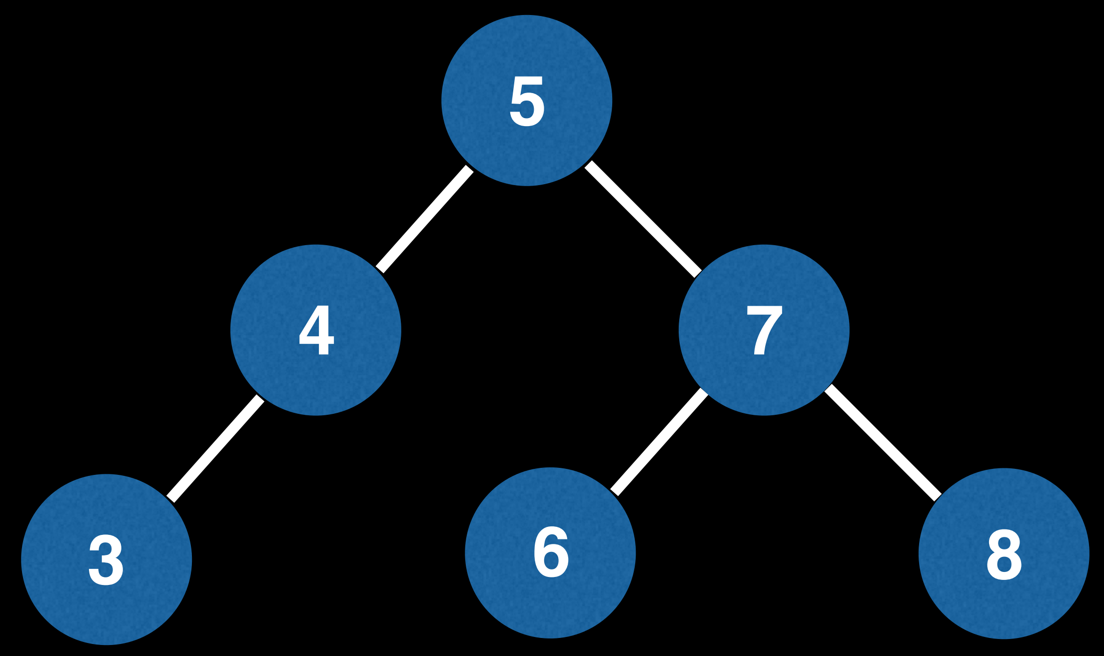
Remove phase

Suppose we wish to remove 9,
then we encounter case II
with a left subtree



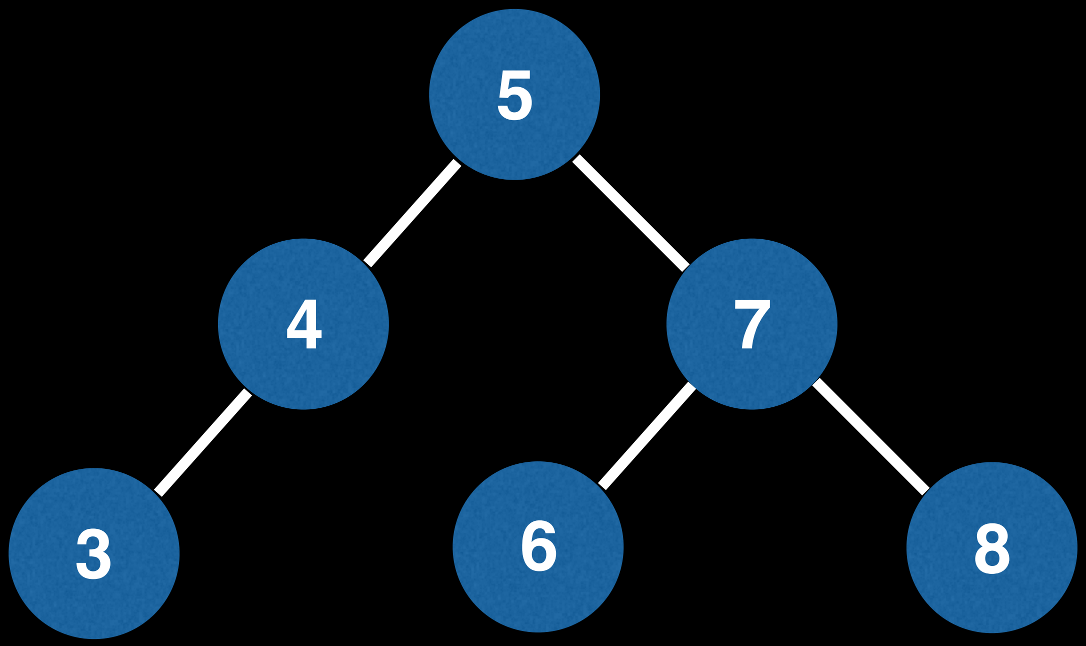
Remove phase

Suppose we wish to remove 9,
then we encounter case II
with a left subtree



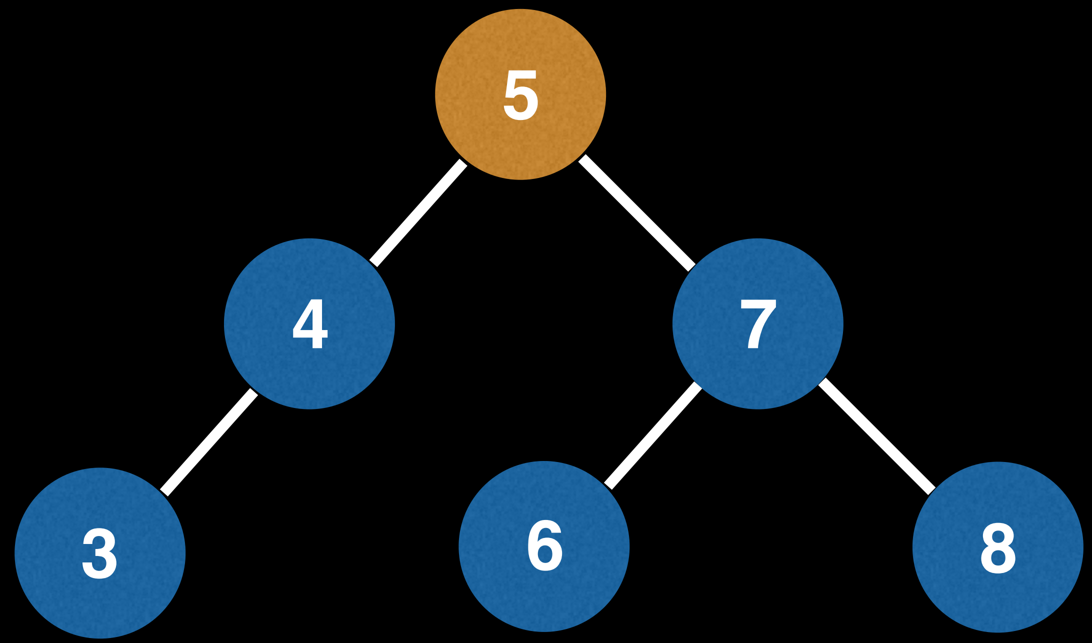
Remove phase

Now let's remove 4!



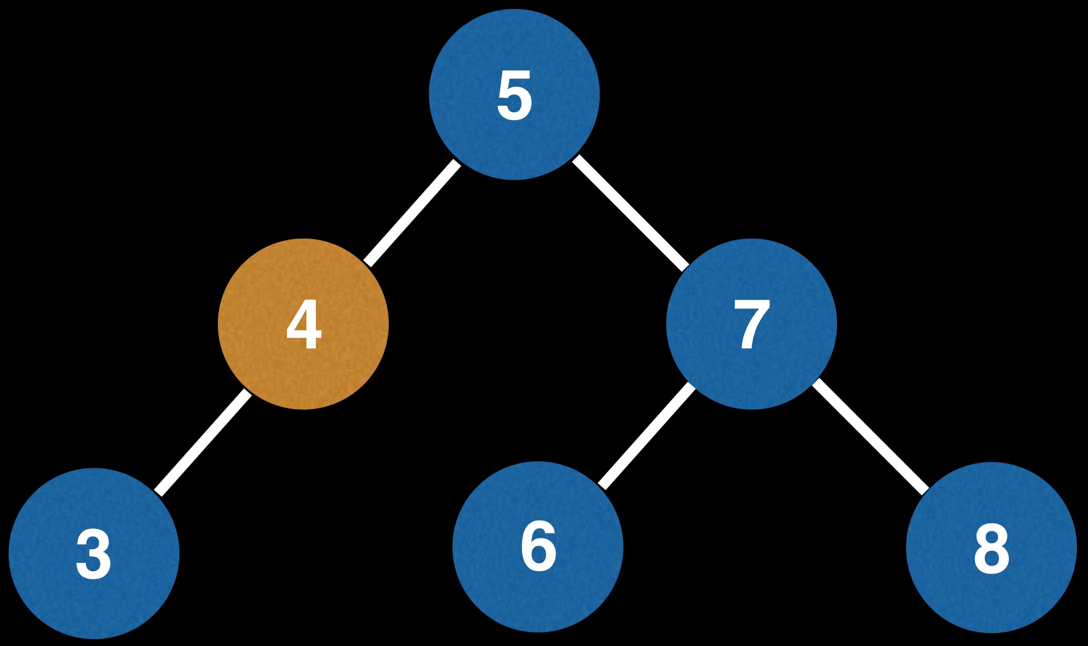
Remove phase

Now let's remove 4!



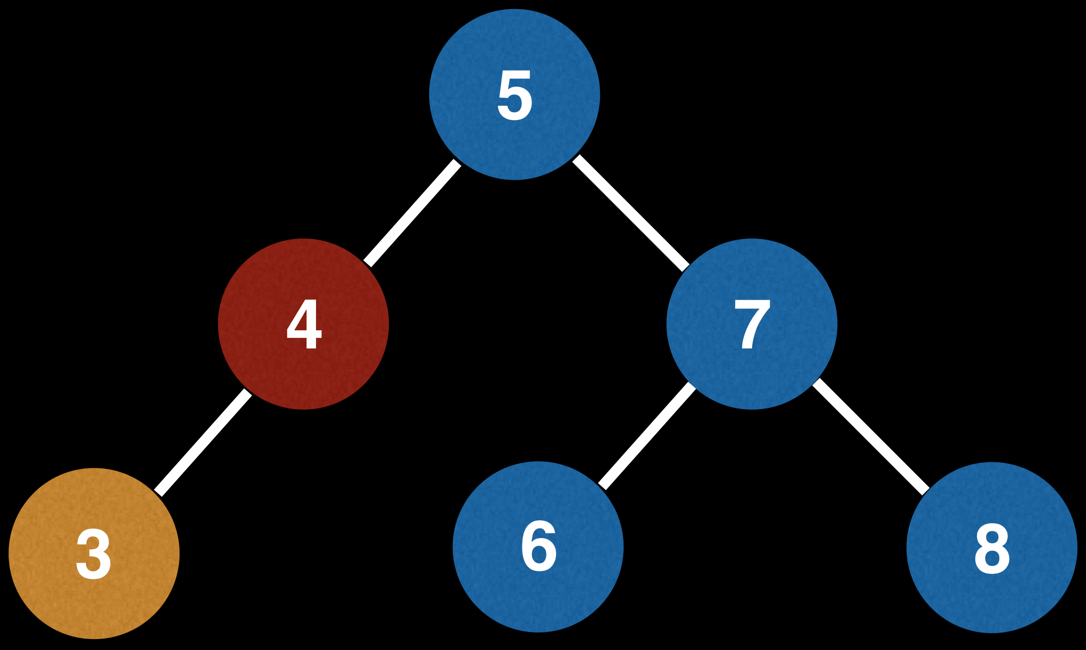
Remove phase

Now let's remove 4!



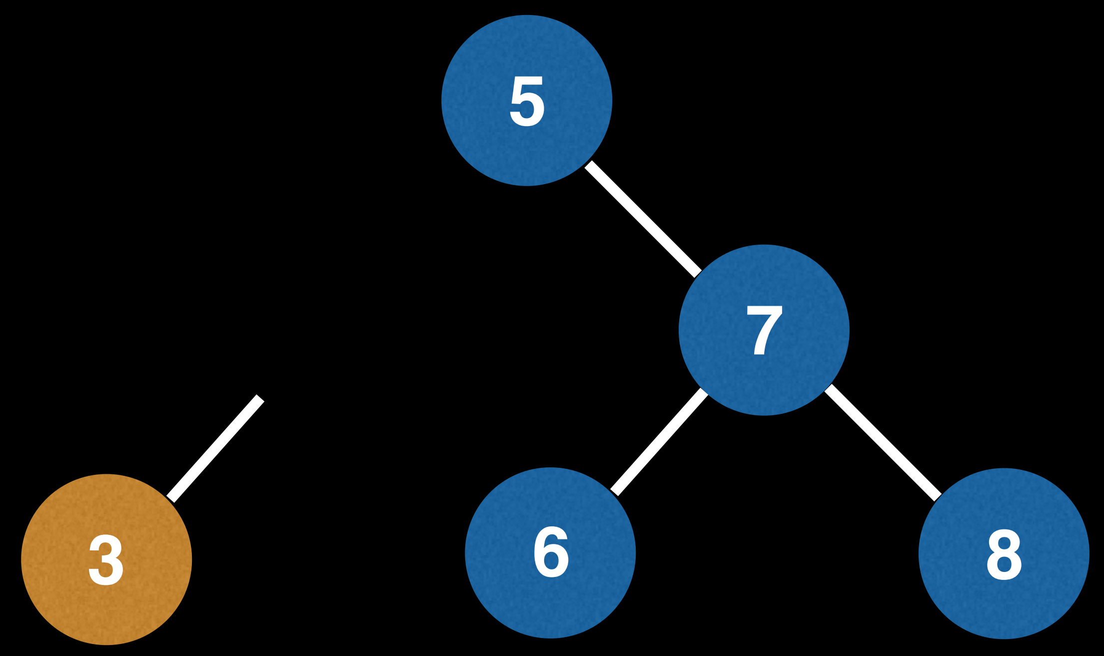
Remove phase

Now let’s remove 4!



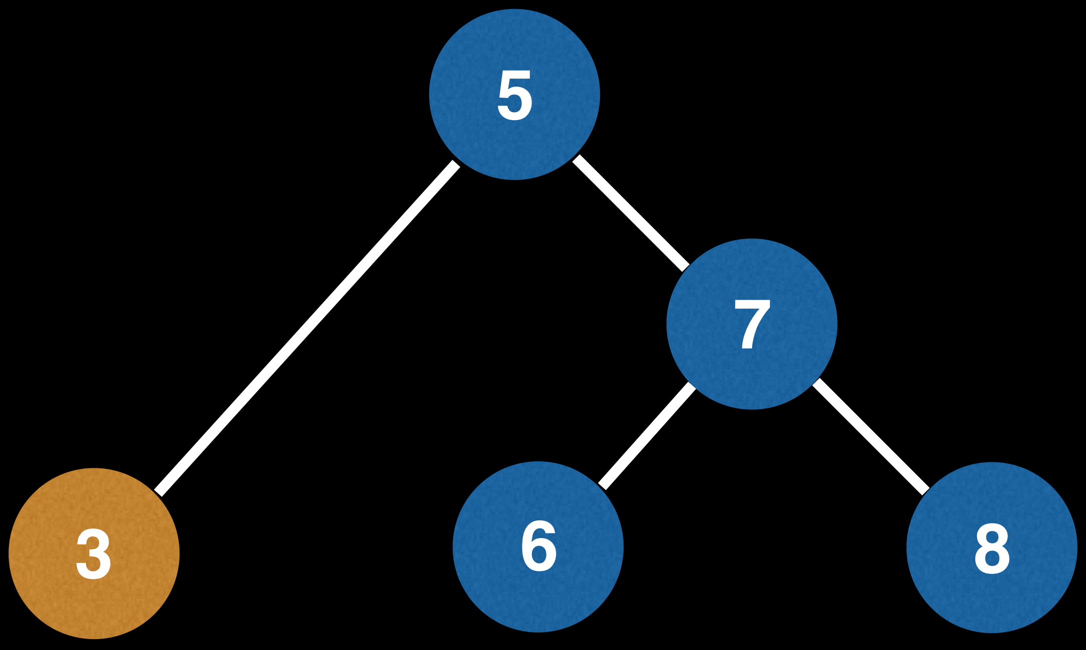
Remove phase

Now let’s remove 4!



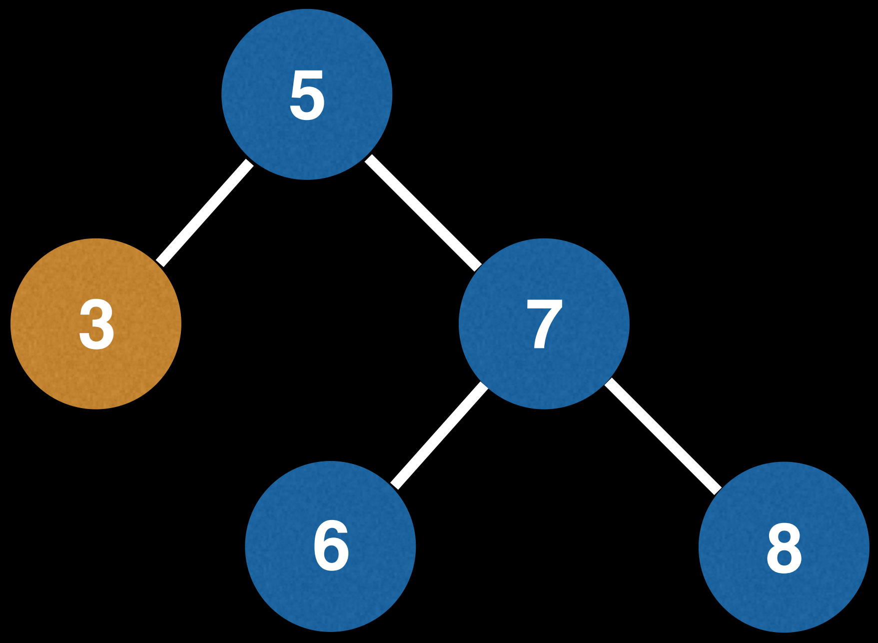
Remove phase

Now let's remove 4!



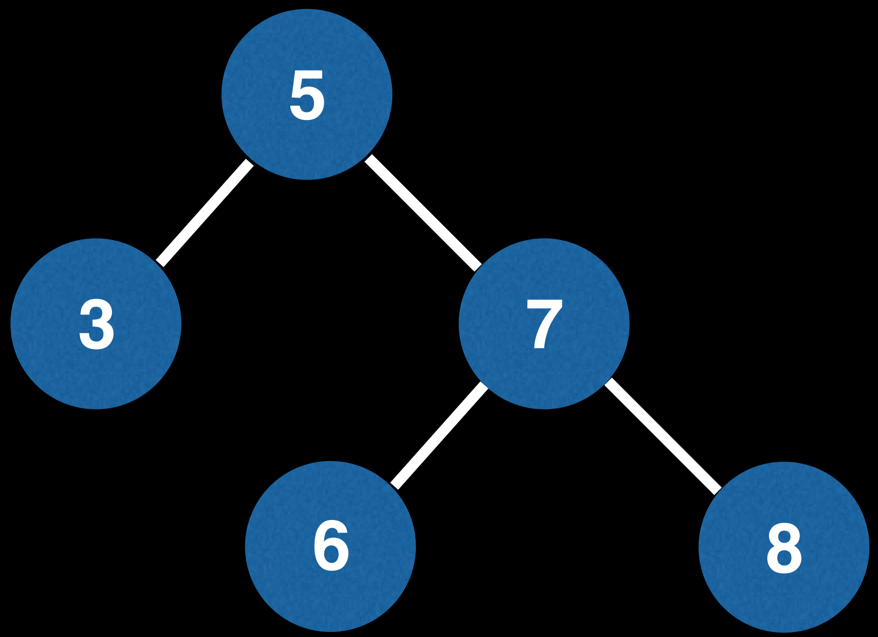
Remove phase

Now let's remove 4!



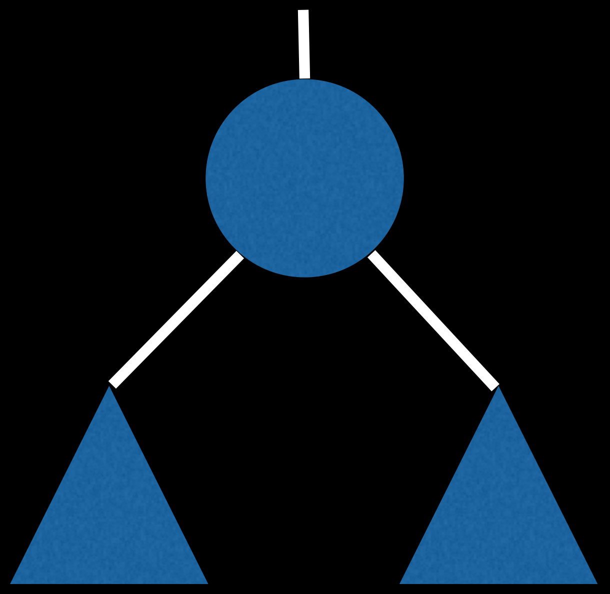
Remove phase

Now let's remove 4!



Remove phase

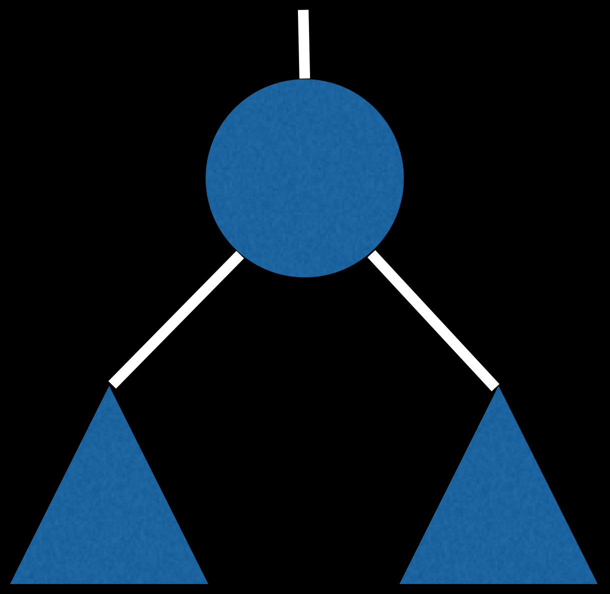
Case IV: Node to remove has both a left subtree and a right subtree



Q: In which subtree will the successor of the node we are trying to remove be?

Remove phase

Case IV: Node to remove has both a left subtree and a right subtree



Q: In which subtree will the successor of the node we are trying to remove be?

A: The answer is both! The successor can either be the **largest value** in the **left subtree** OR the **smallest value** in the **right subtree**.

Remove phase

A justification for why there could be more than one successor is:

The **largest value** in the **left subtree** satisfies the BST invariant since it:

1) Is larger than everything in left subtree. This follows immediately from the definition of being the largest.

2) Is smaller than everything in right subtree because it was found in the left subtree

but also…

Remove phase

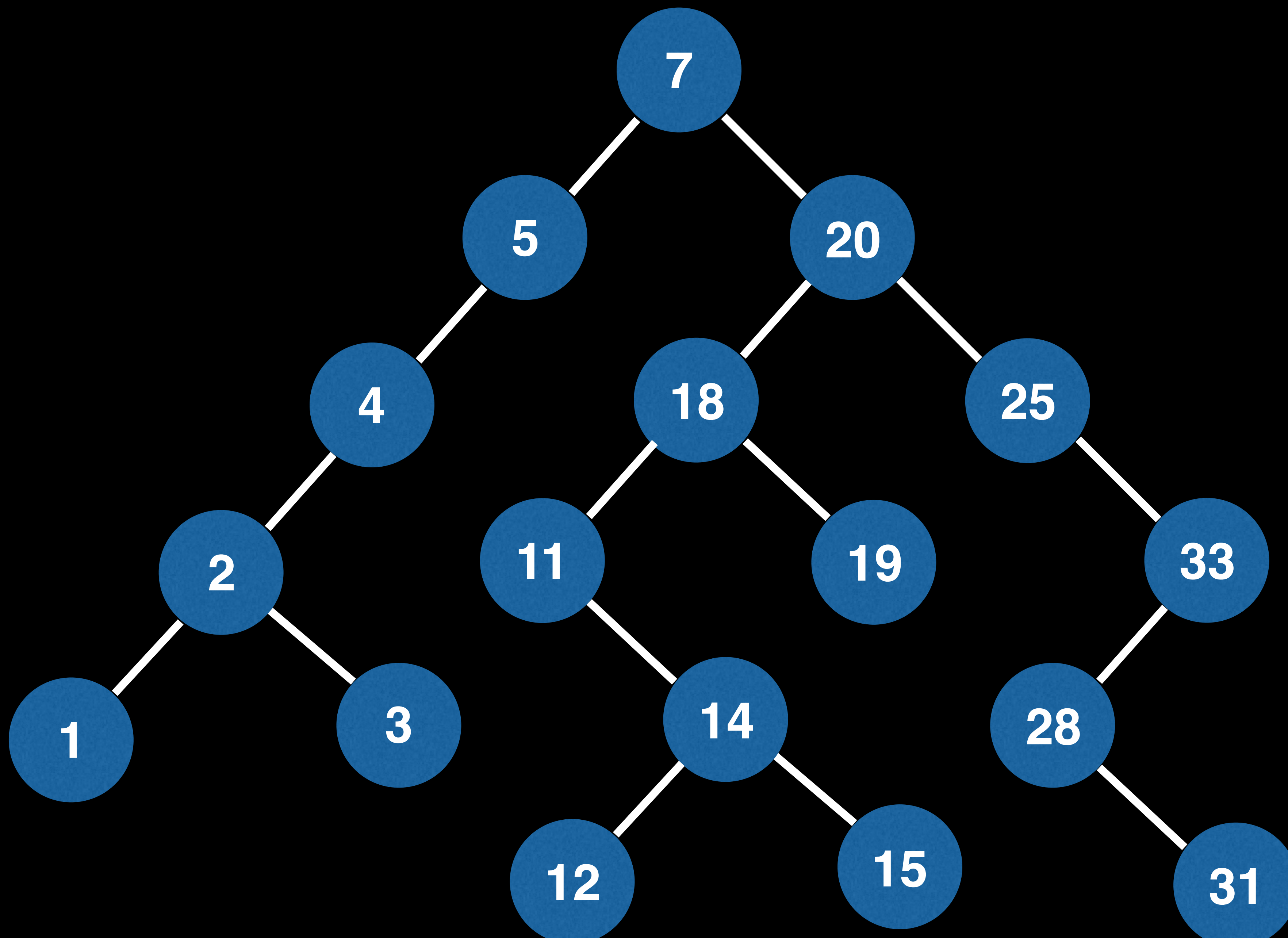
The **smallest value** in the **right subtree** satisfies the BST invariant since it:

1) Is smaller than everything in right subtree. This follows immediately from the definition of being the smallest.

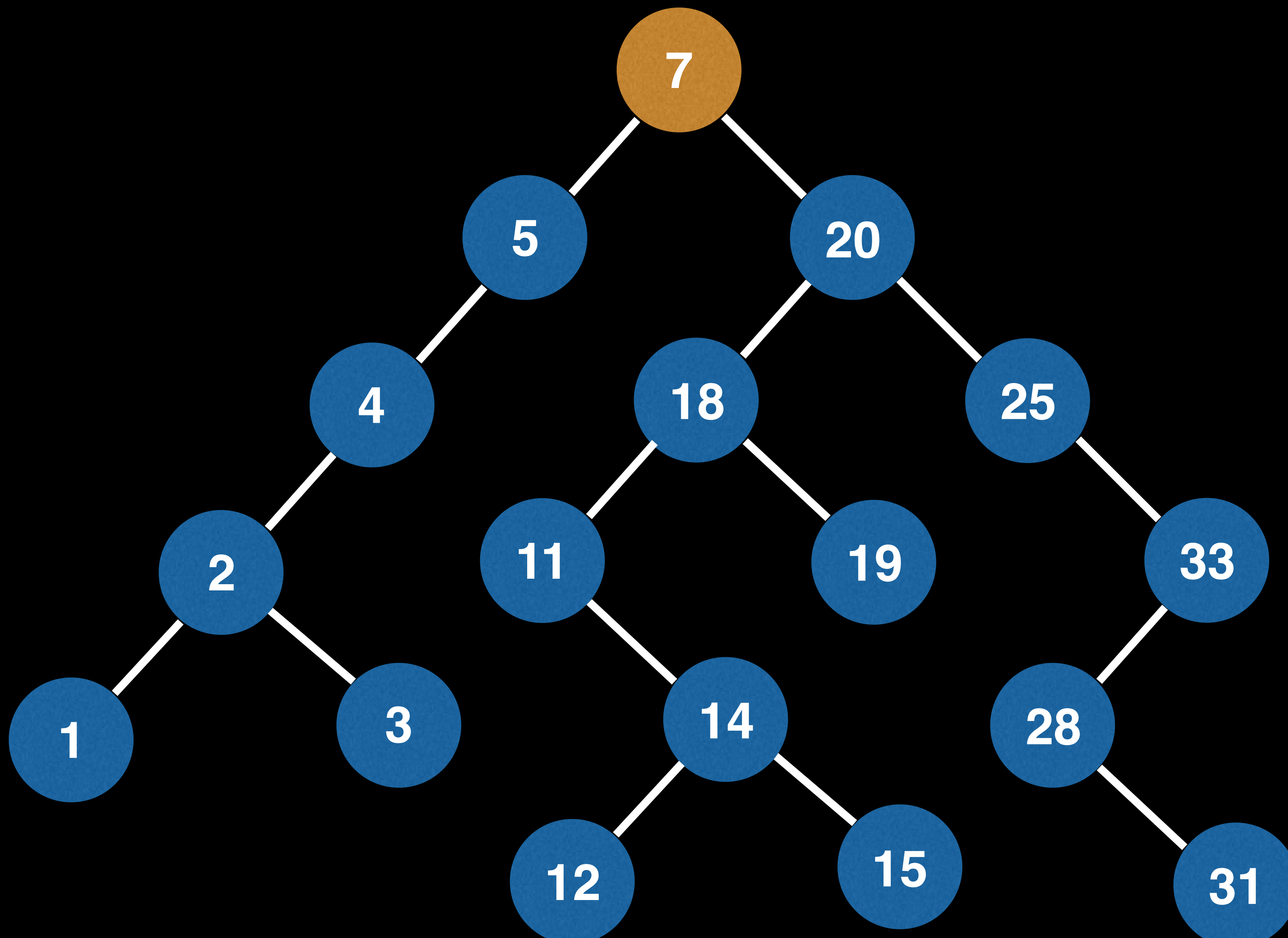
2) Is larger than everything in left subtree because it was found in the right subtree

So there are two possible successors, yea!

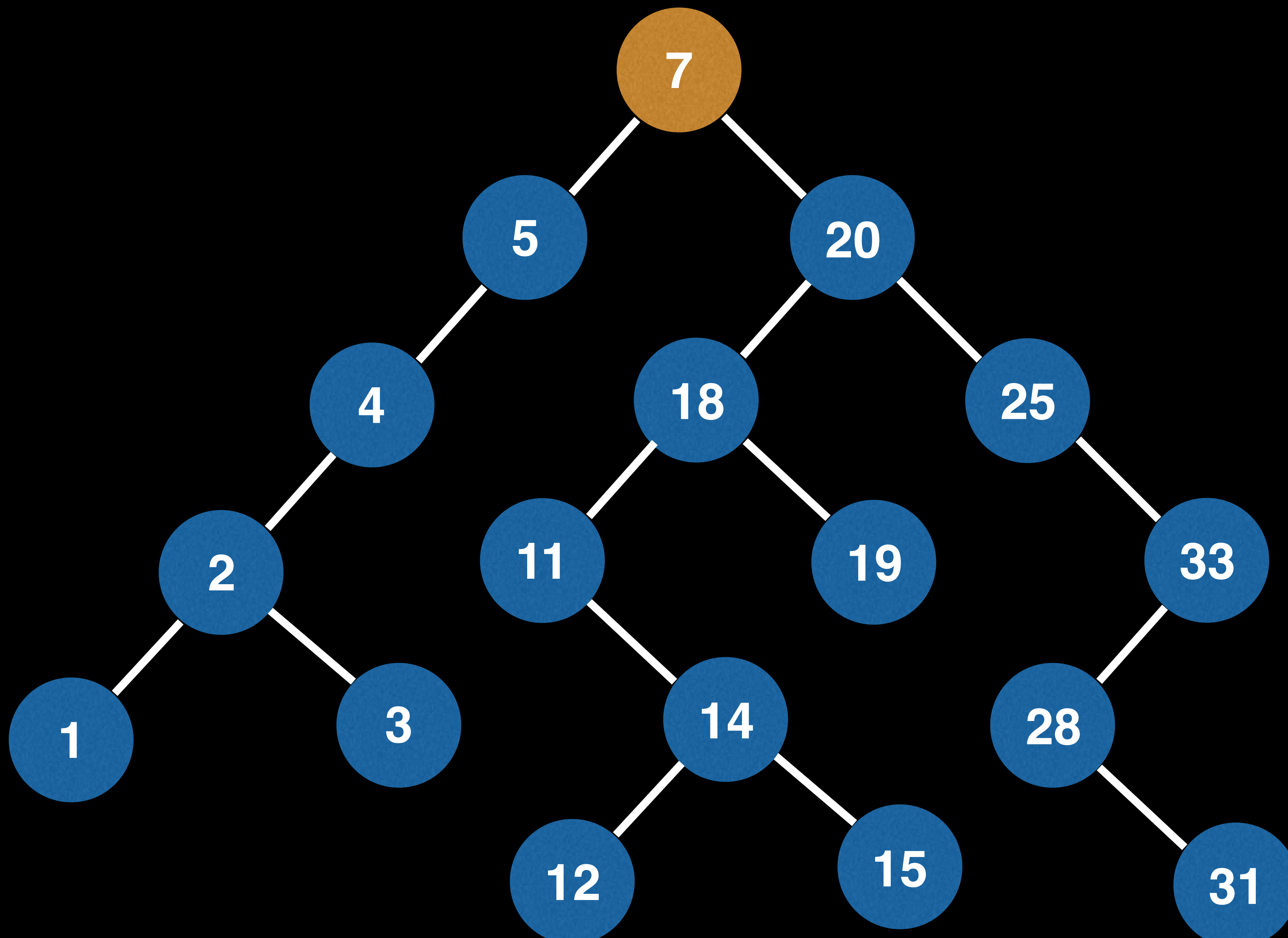
Let’s remove 7



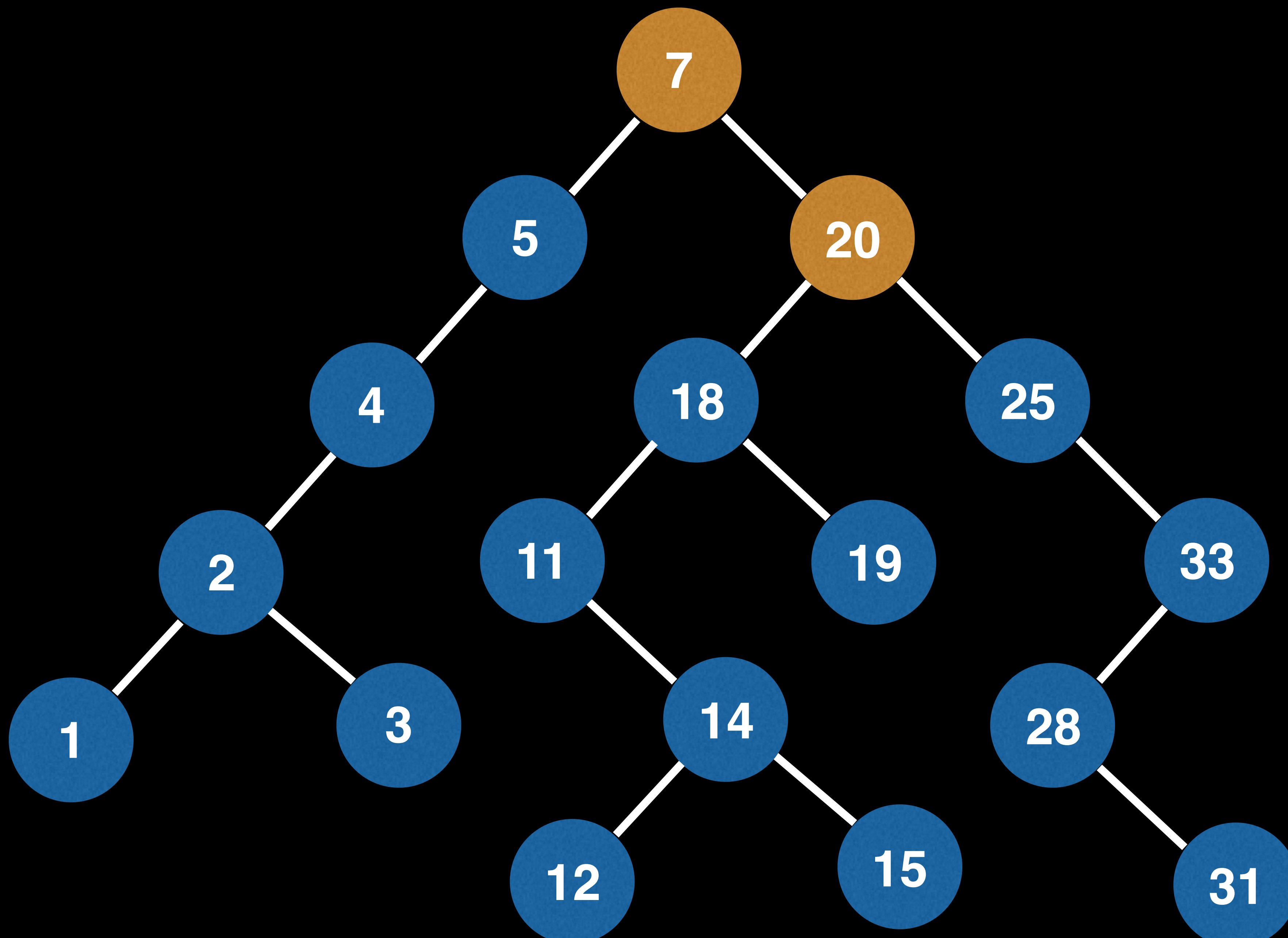
Let’s remove 7



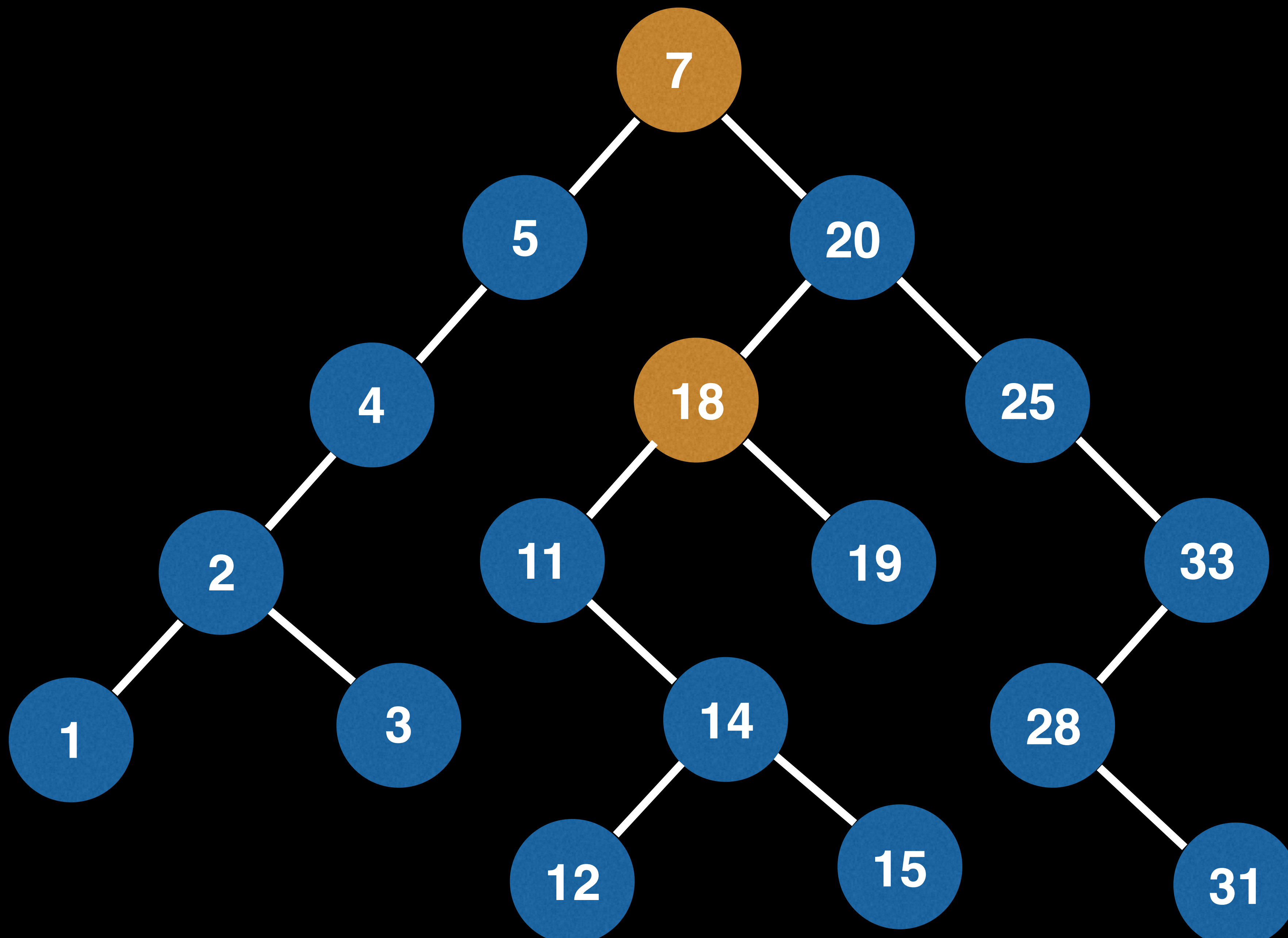
Now choose successor to be either the smallest value in right subtree or largest in left subtree. Let’s do the former. To do this dig as far left as possible in the right subtree.



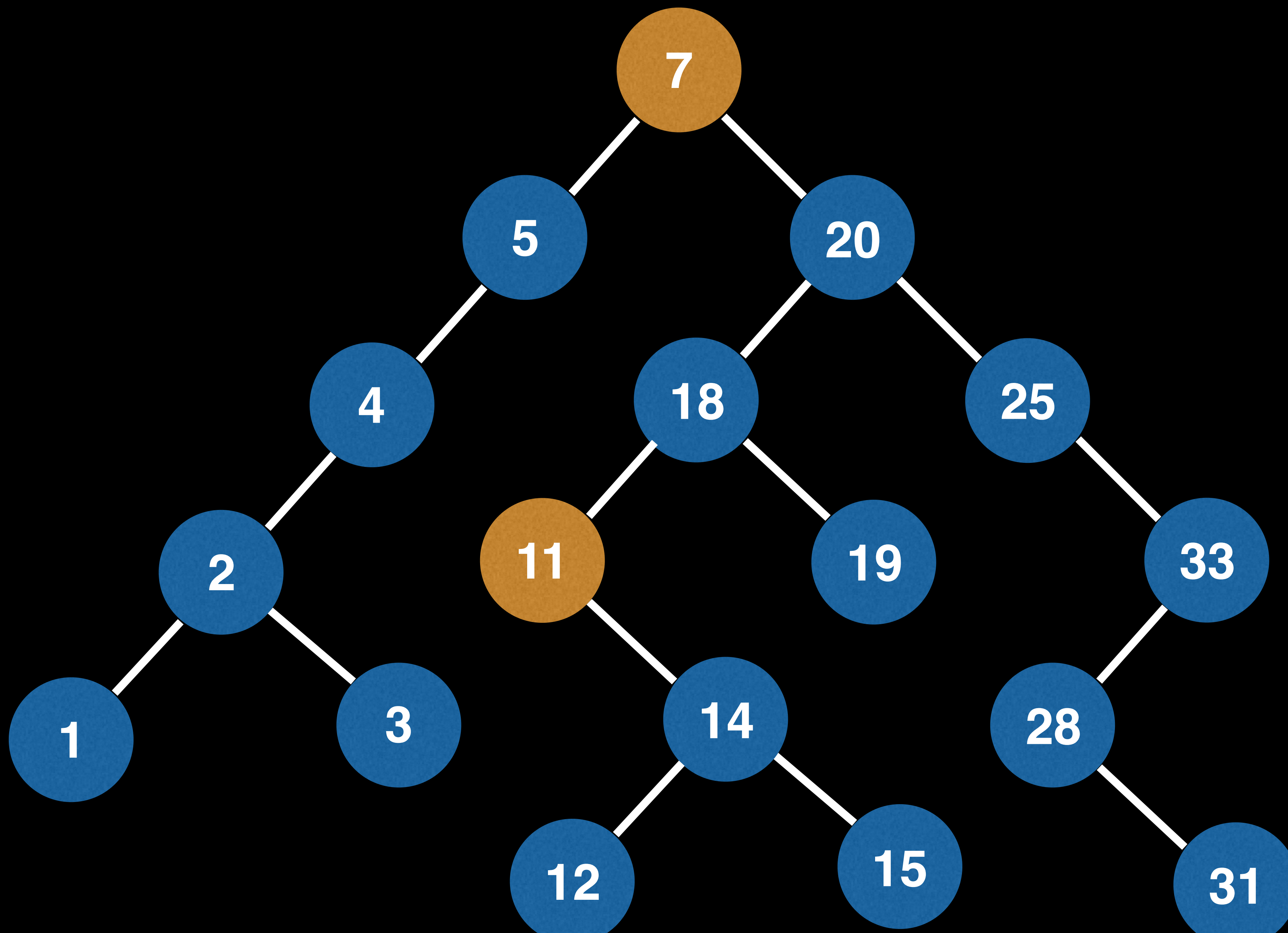
Now choose successor to be either the smallest value in right subtree or largest in left subtree. Let’s do the former. To do this dig as far left as possible in the right subtree.



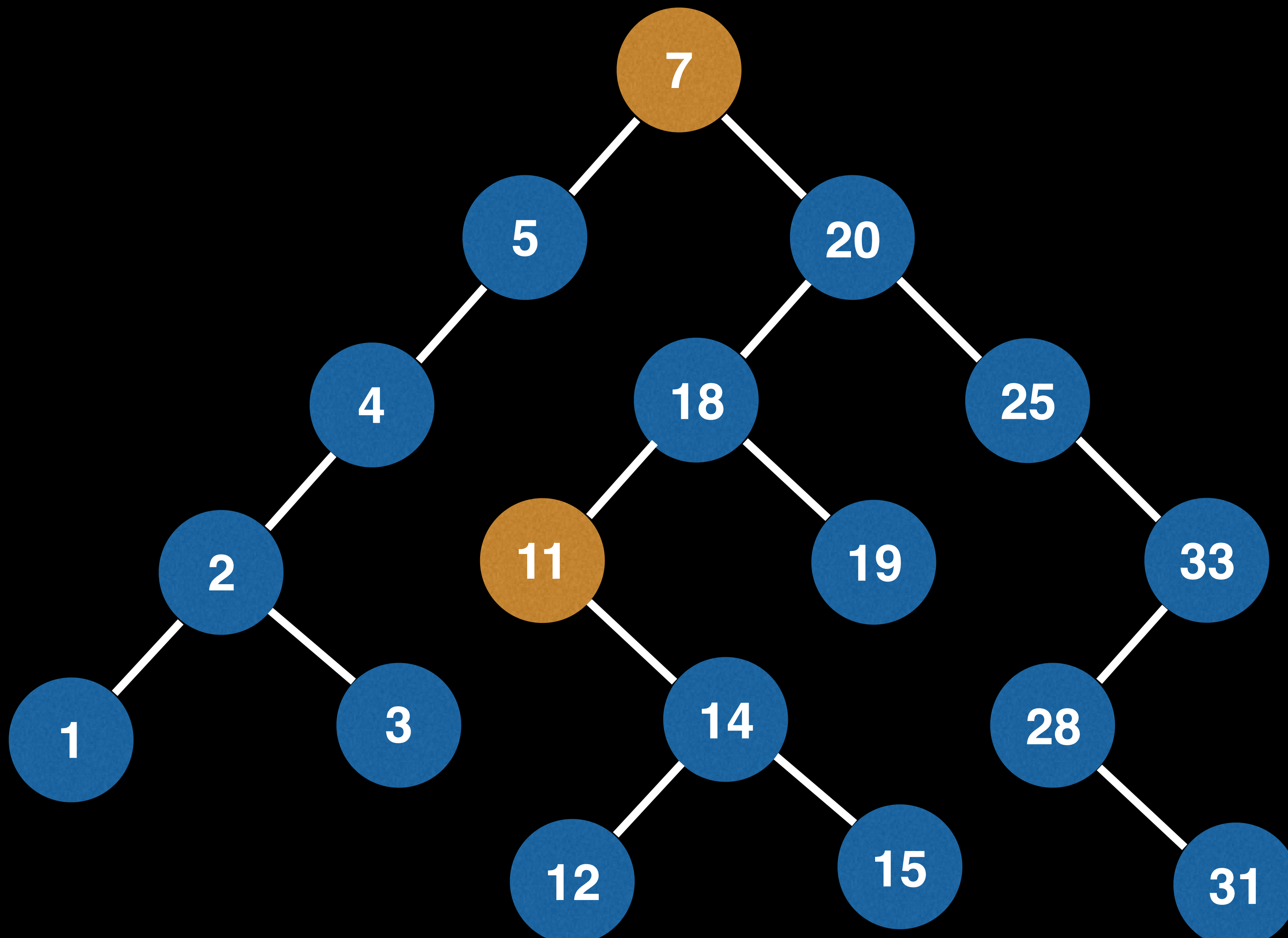
Now choose successor to be either the smallest value in right subtree or largest in left subtree. Let’s do the former. To do this dig as far left as possible in the right subtree.



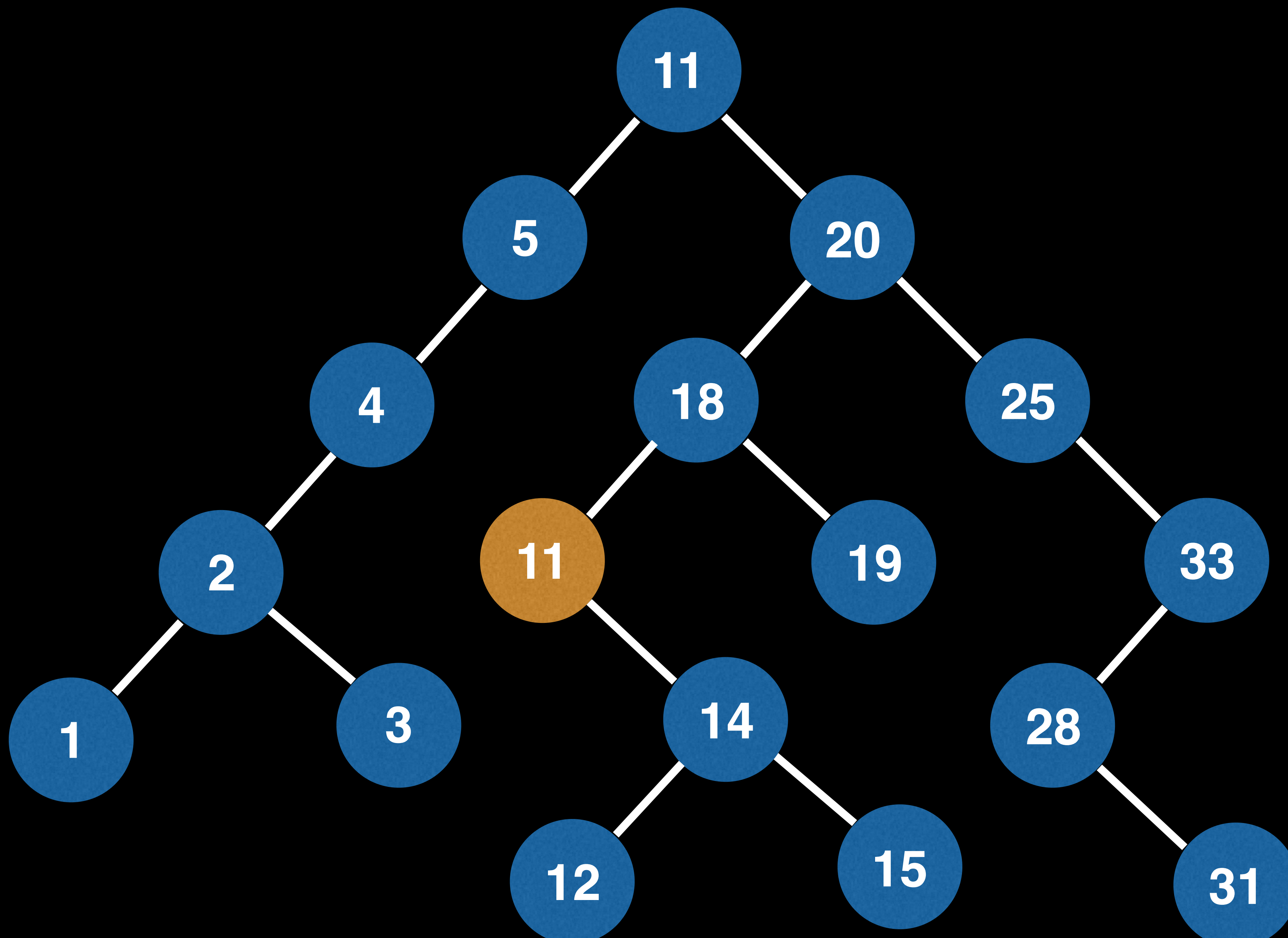
Now choose successor to be either the smallest value in right subtree or largest in left subtree. Let’s do the former. To do this dig as far left as possible in the right subtree.



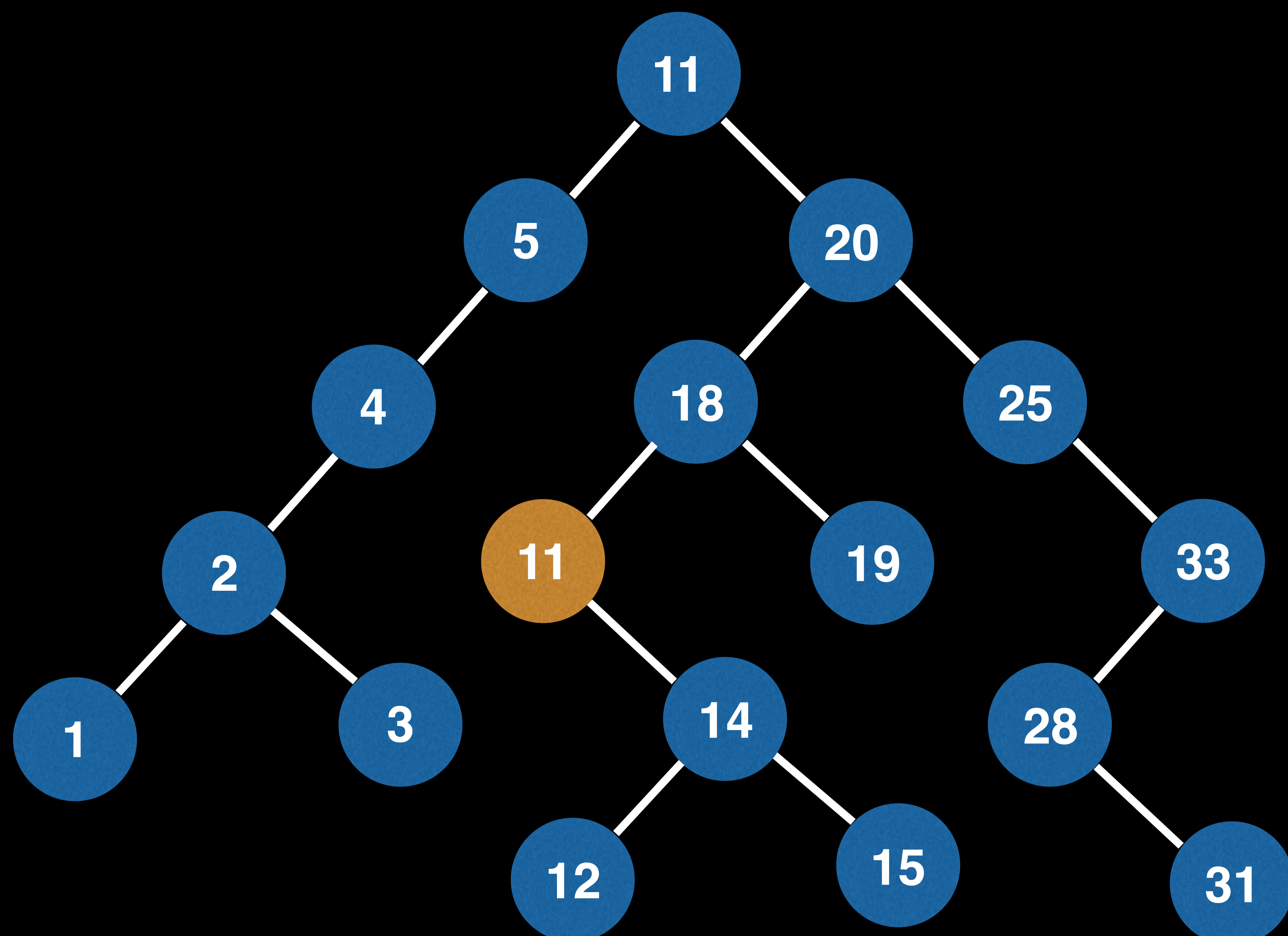
Copy the value from the node found in right subtree (11) to the node we want to remove.



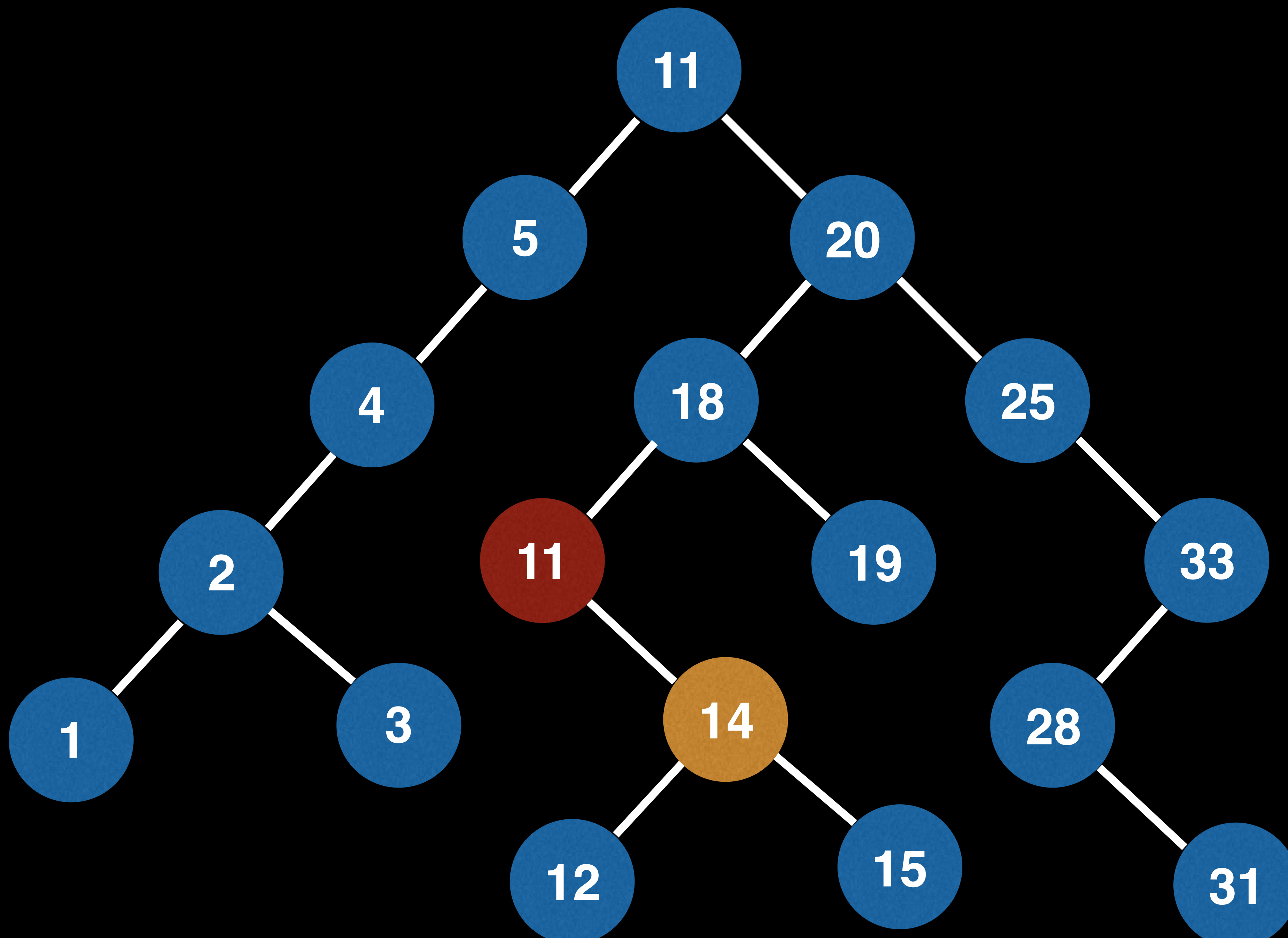
Copy the value from the node found in right subtree (11) to the node we want to remove.



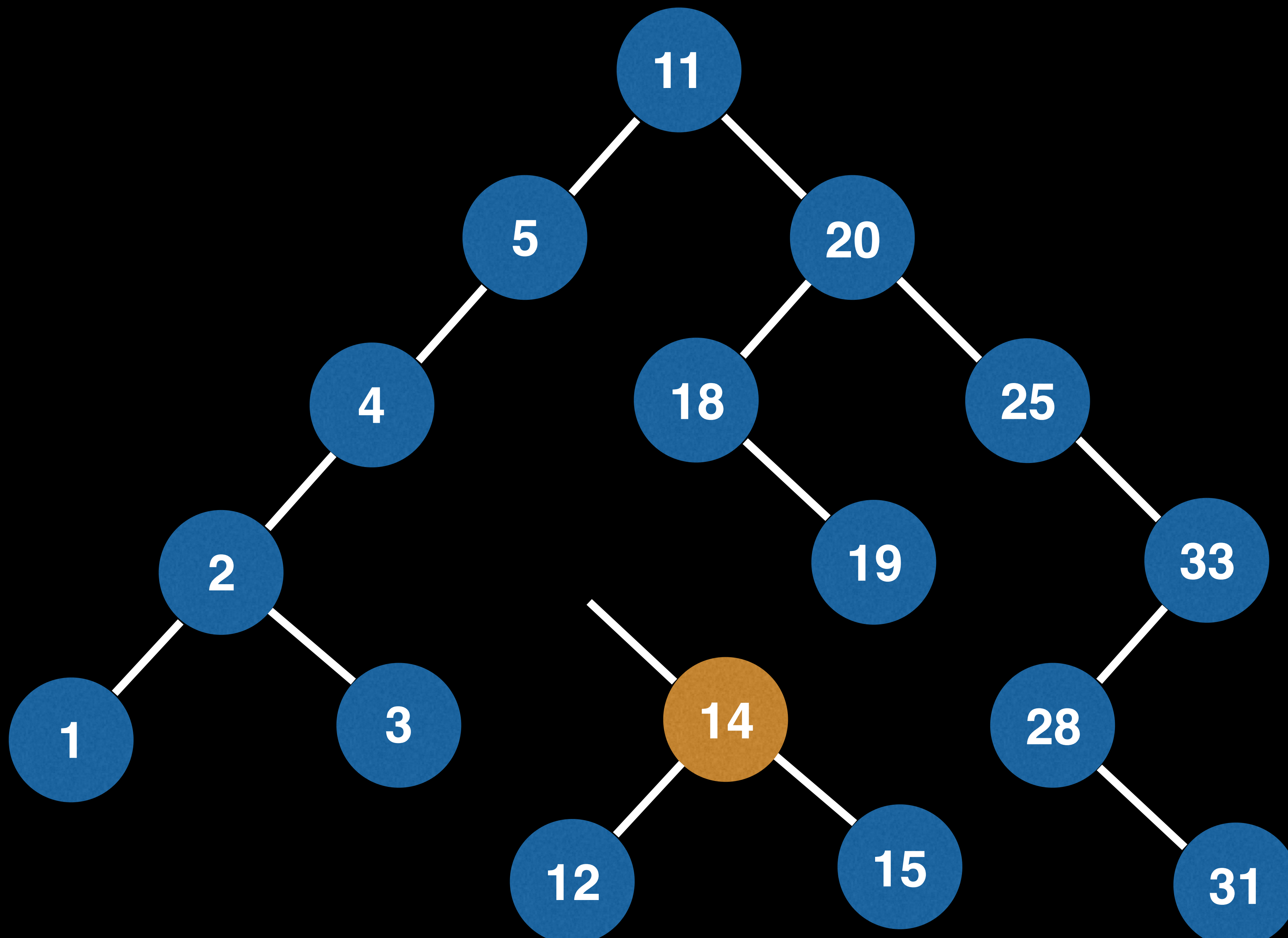
Now we have to remove the 11 we found in the right subtree. Luckily, the node we find will always be either a Case I, II, III removal



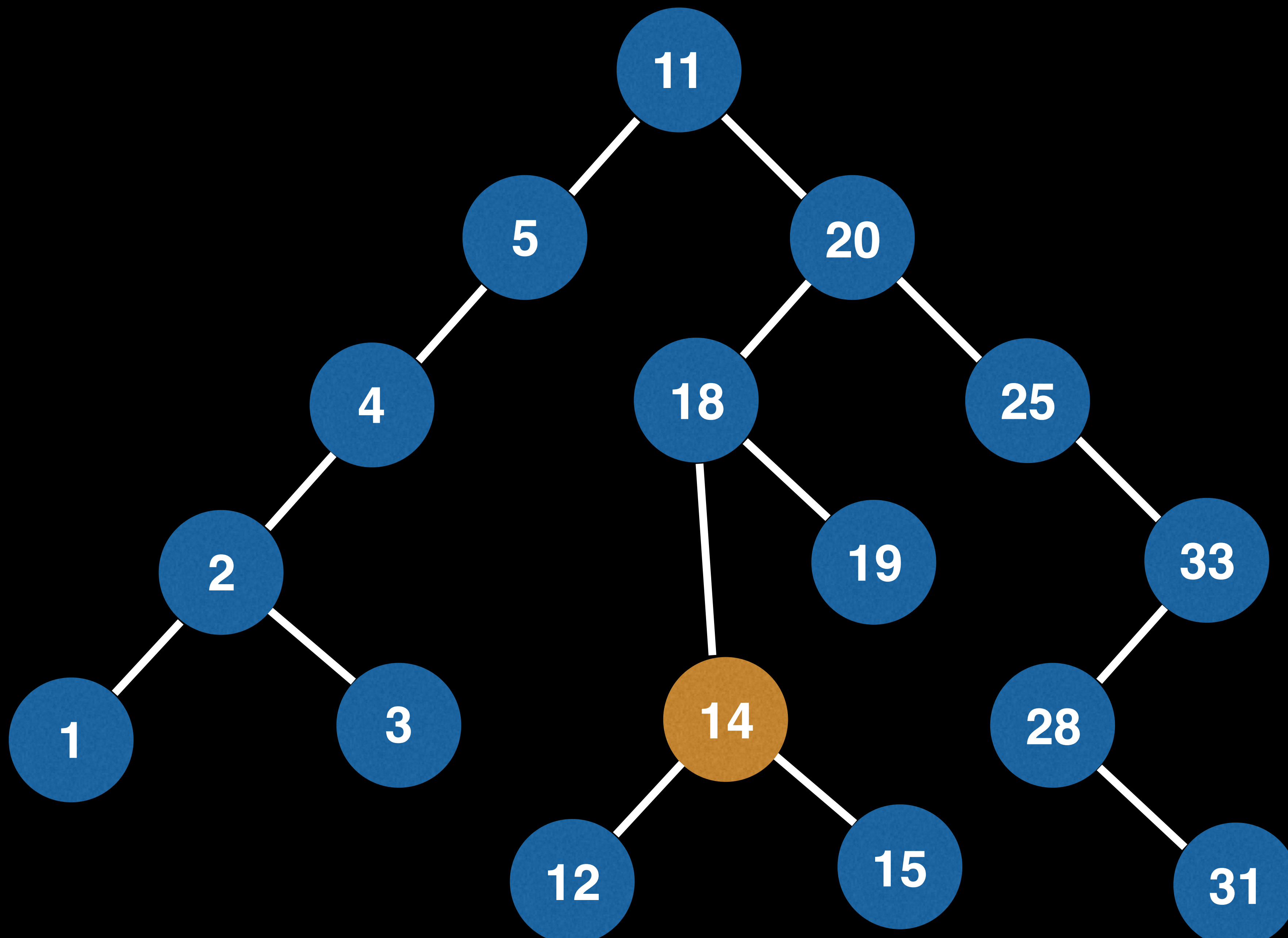
Now we have to remove the 11 we found in the right subtree. Luckily, the node we find will always be either a Case I, II, III removal



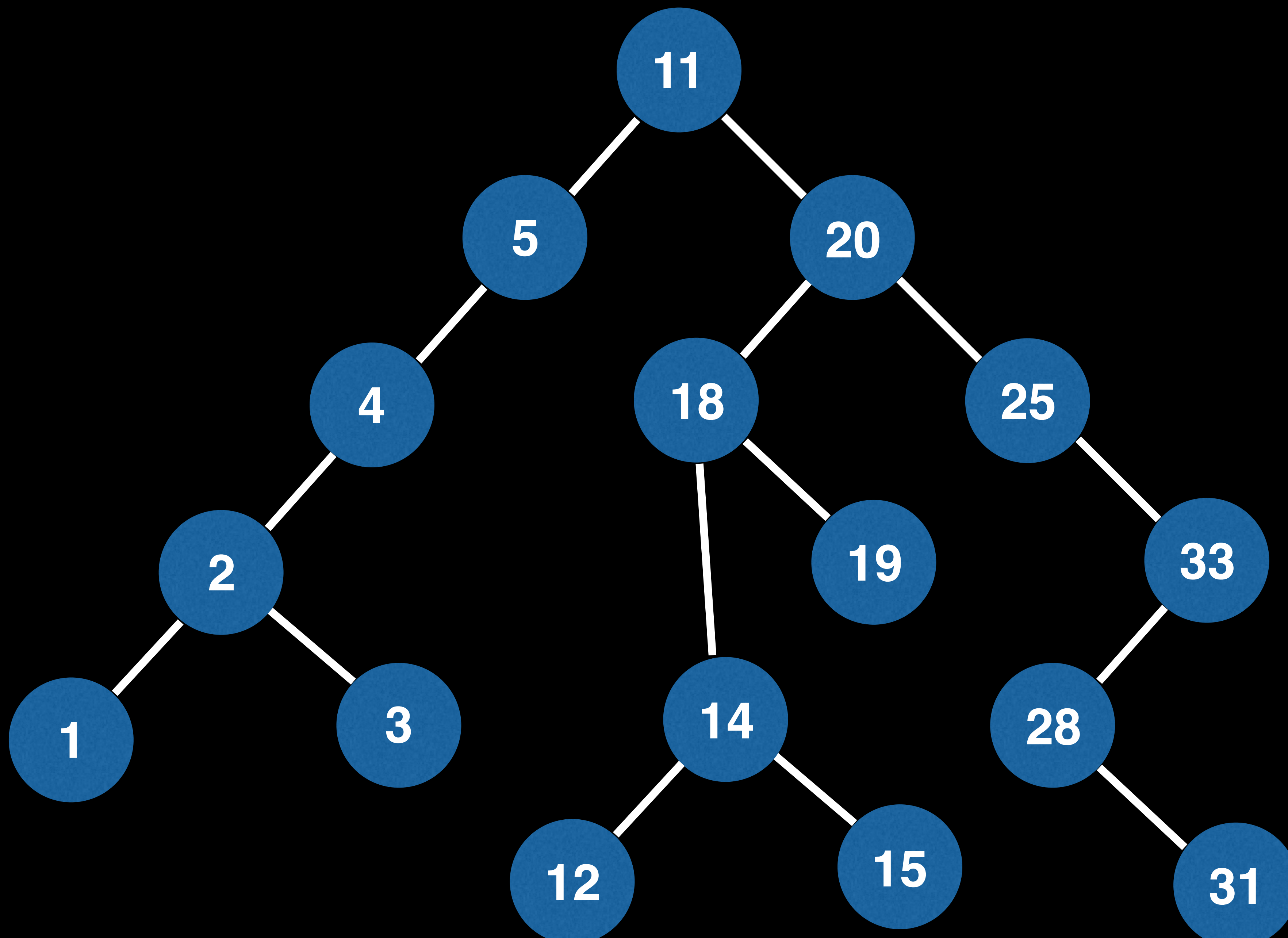
Now we have to remove the 11 we found in the right subtree. Luckily, the node we find will always be either a Case I, II, III removal



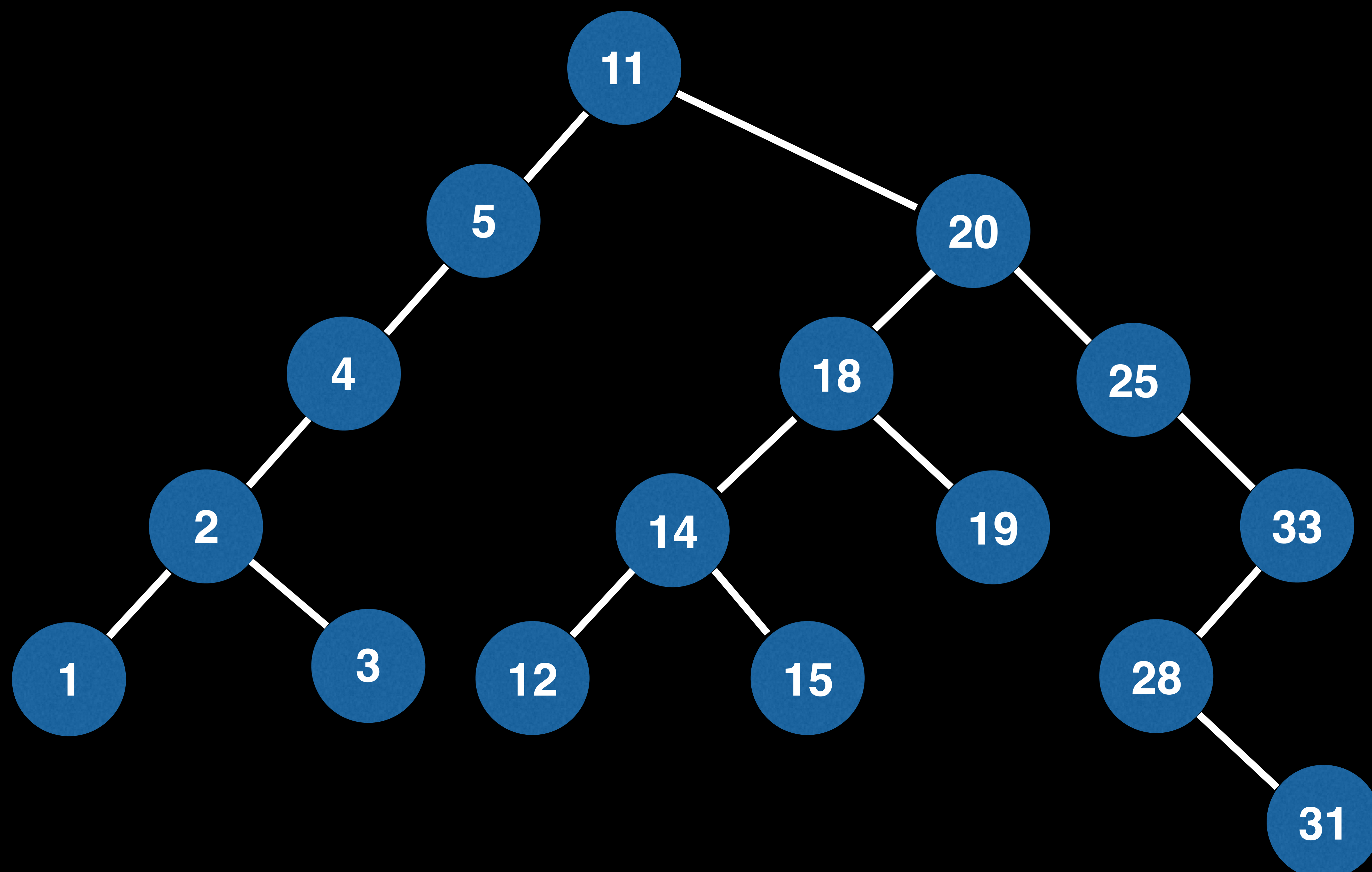
Now we have to remove the 11 we found in the right subtree. Luckily, the node we find will always be either a Case I, II, III removal



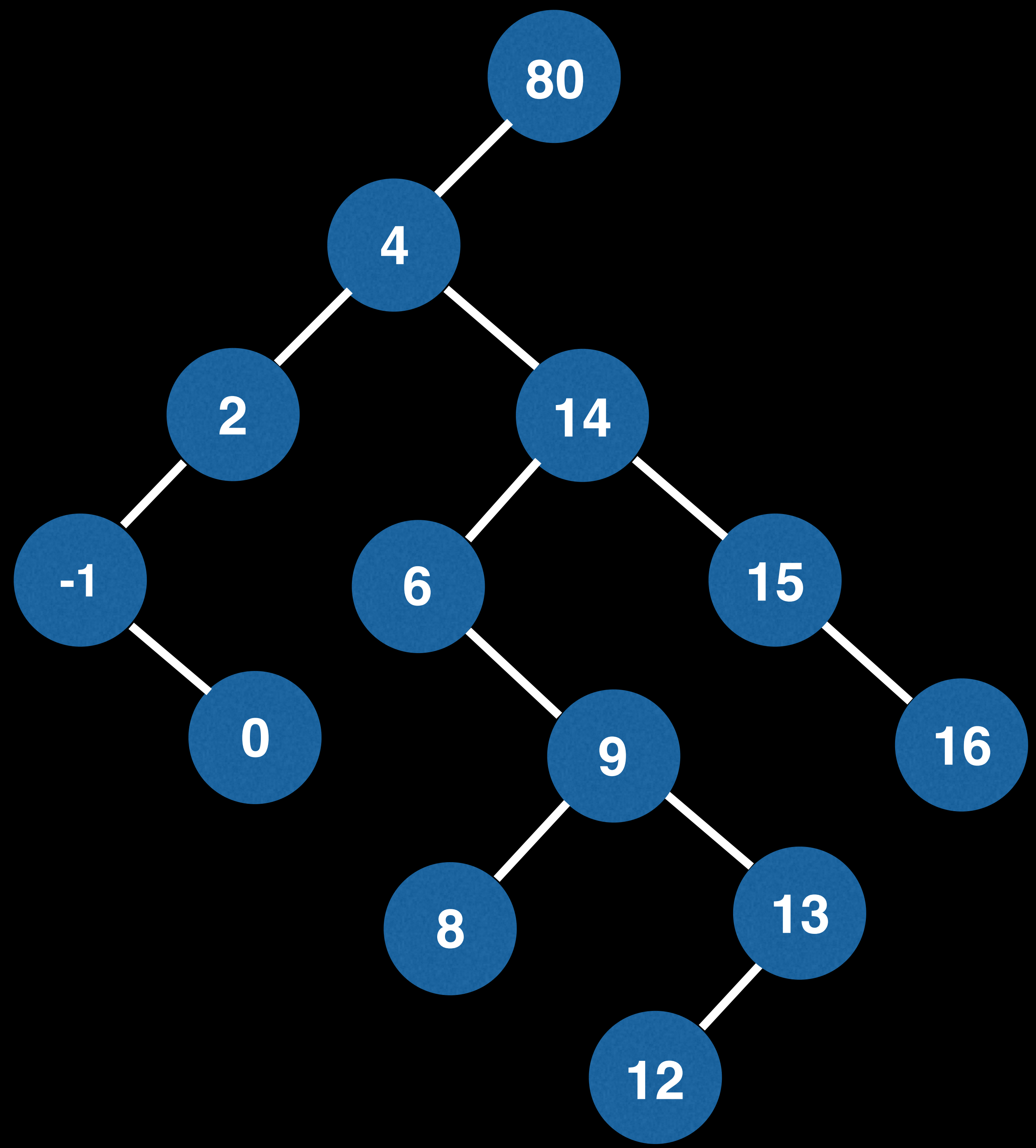
Now we have to remove the 11 we found in the right subtree. Luckily, the node we find will always be either a Case I, II, III removal



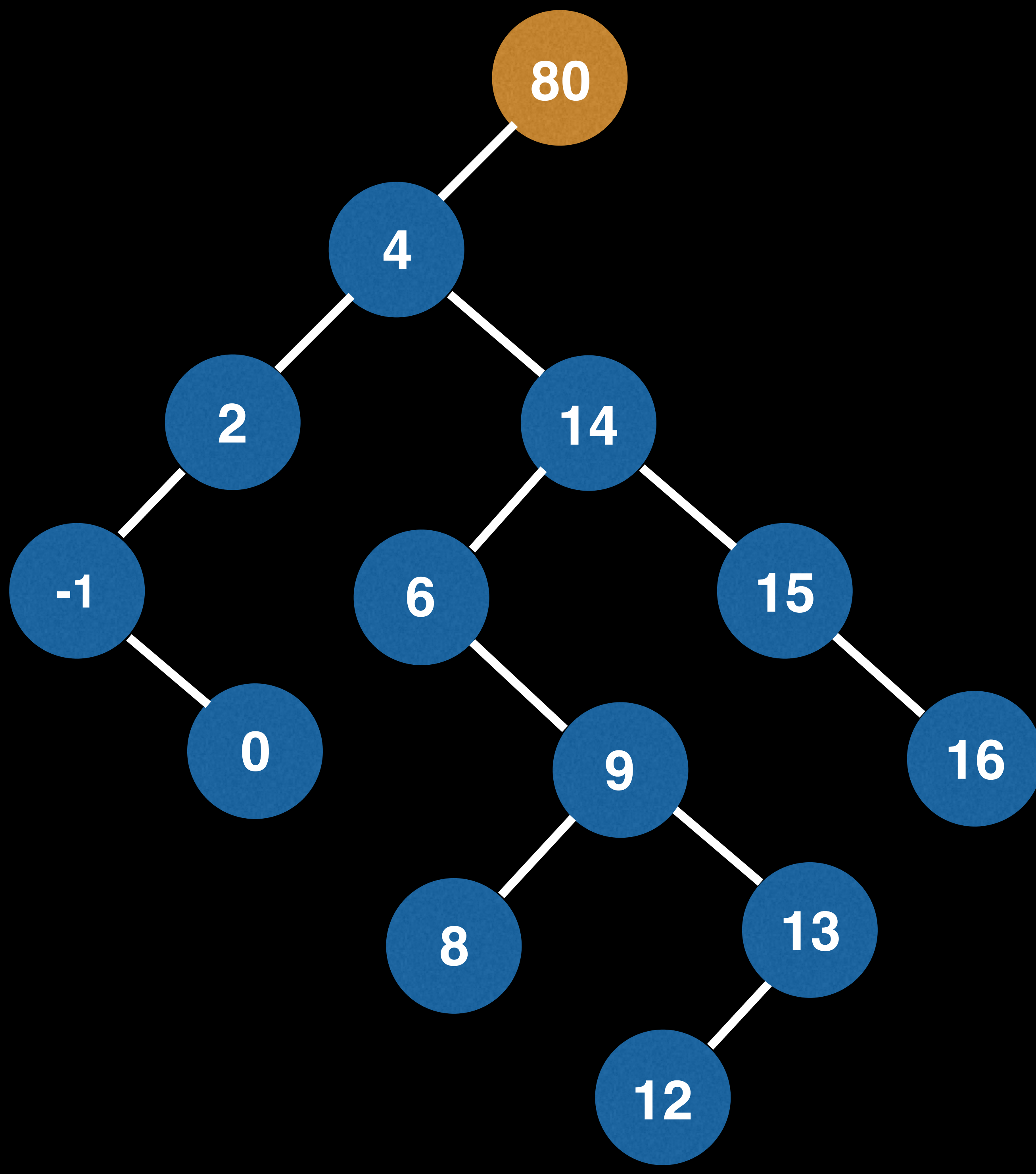
Now we have to remove the 11 we found in the right subtree. Luckily, the node we find will always be either a Case I, II, III removal



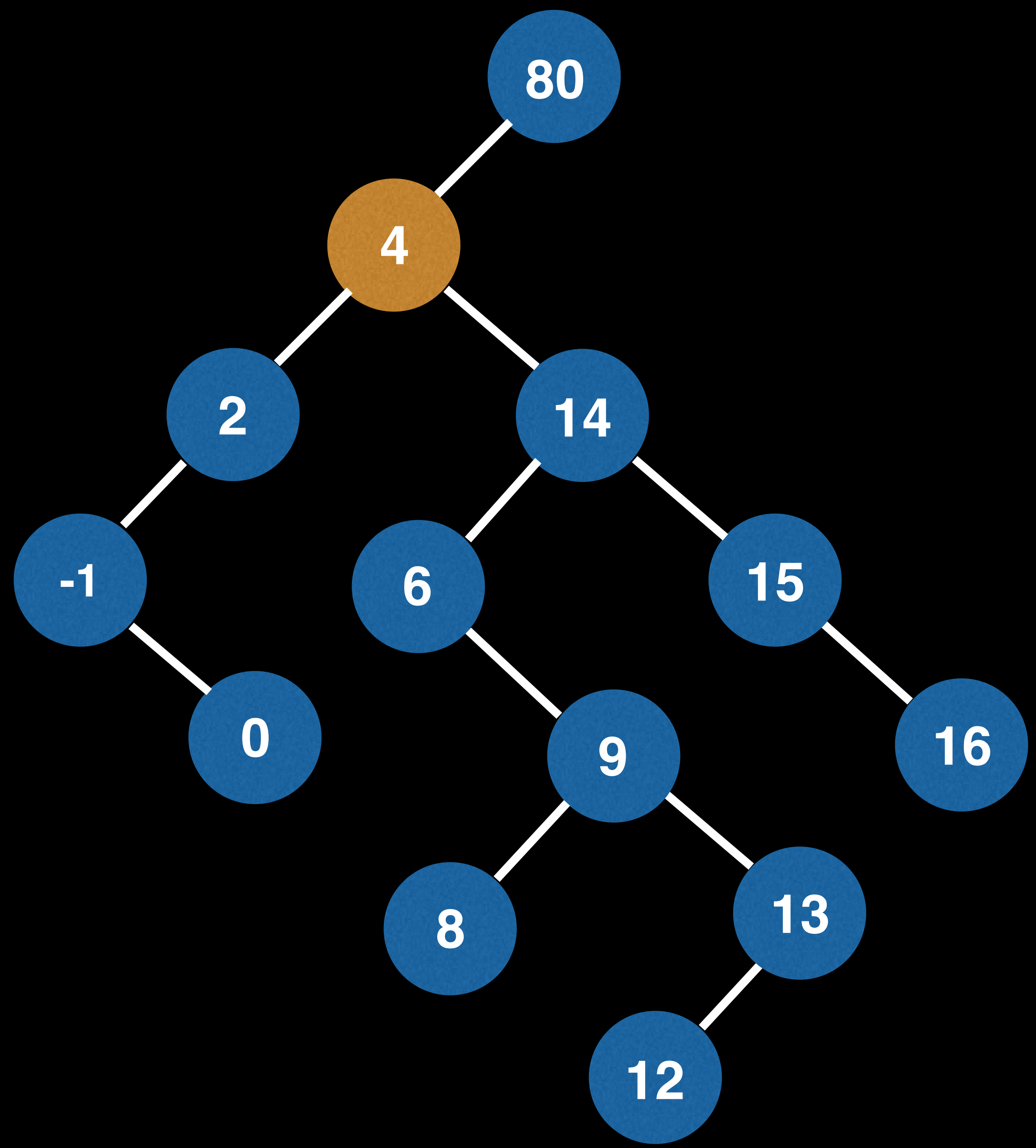
In this example let's remove 14. First begin by finding where 14 is located.



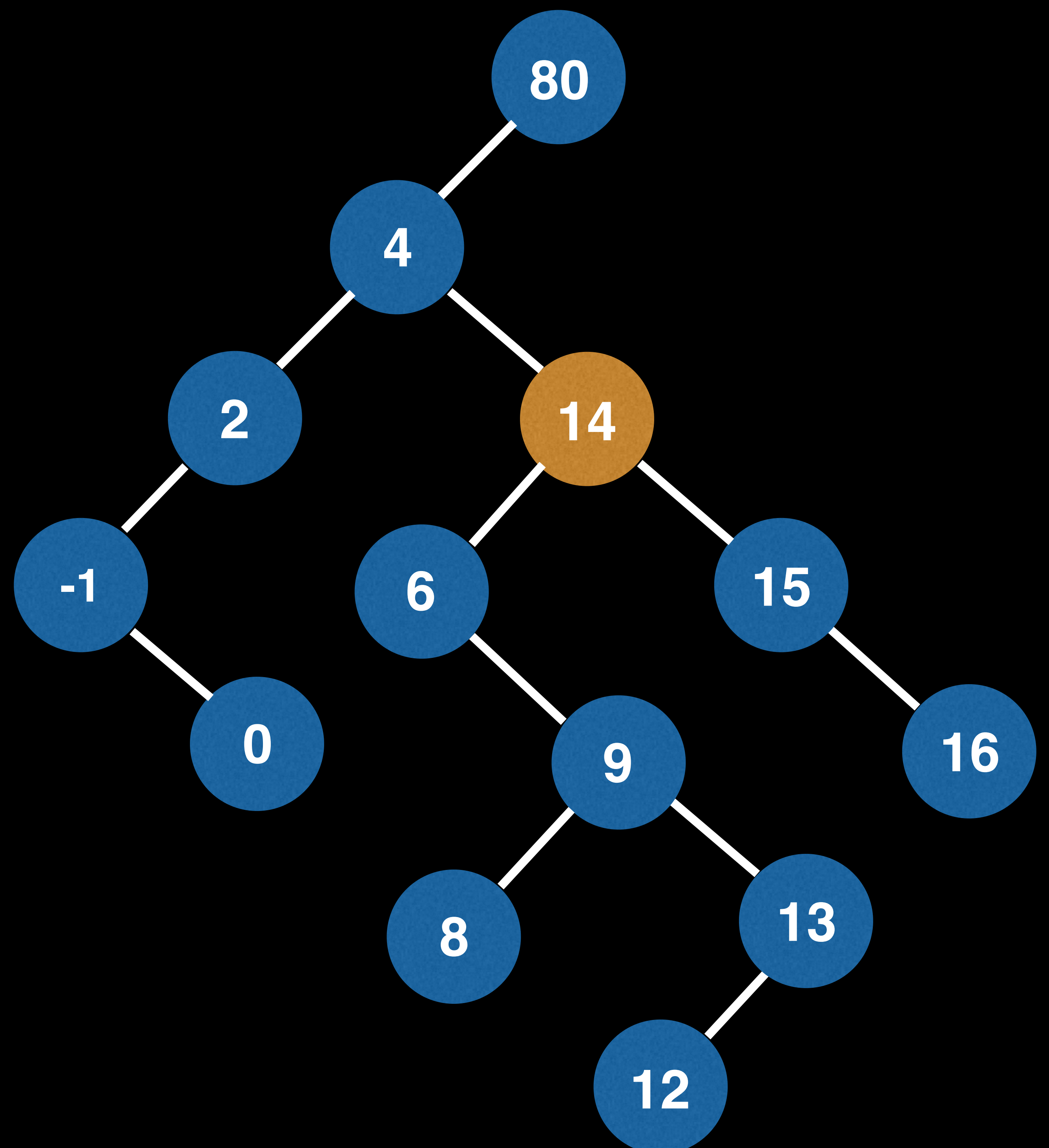
In this example let's remove 14. First begin by finding where 14 is located.



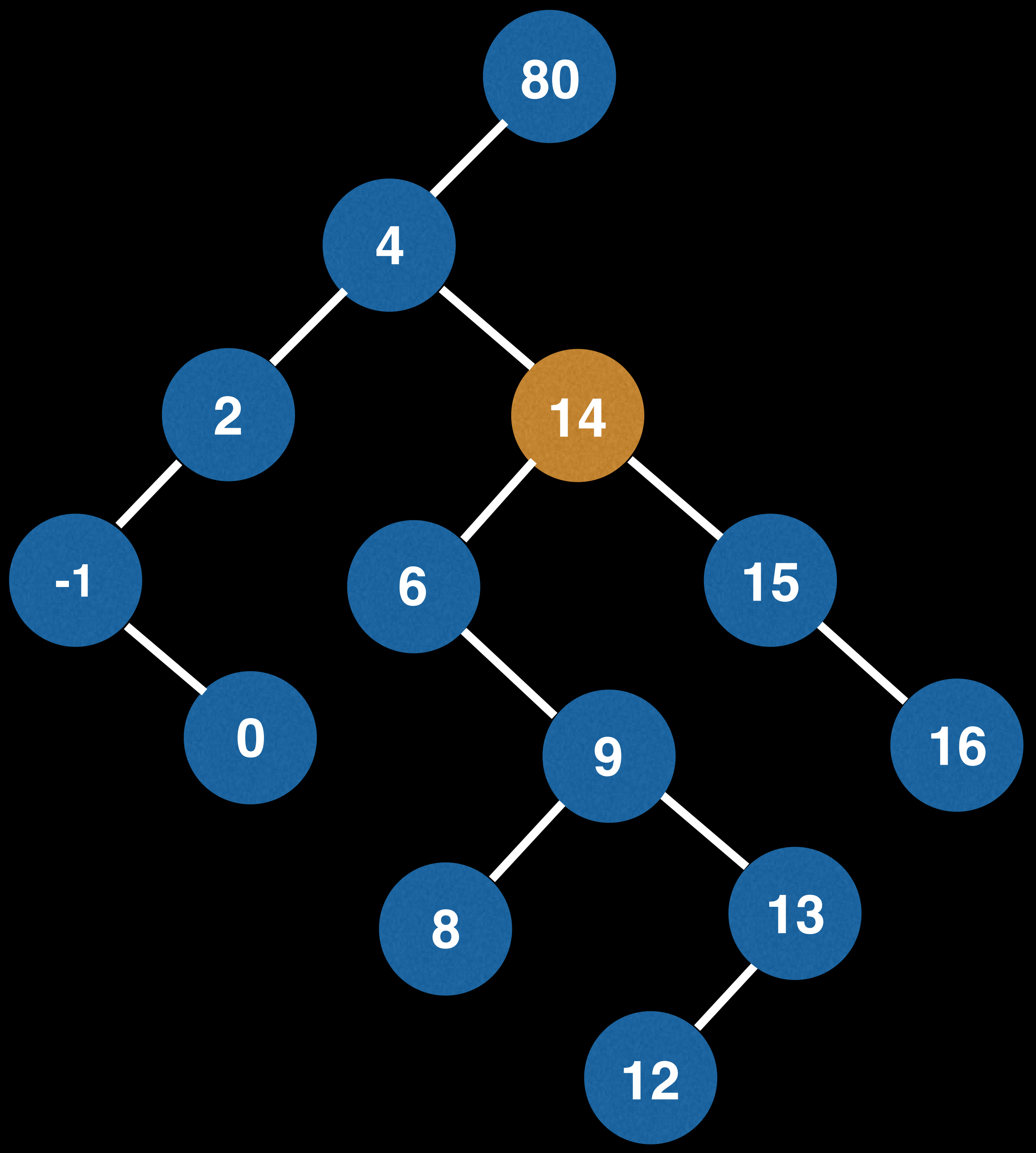
In this example let's remove 14. First begin by finding where 14 is located.



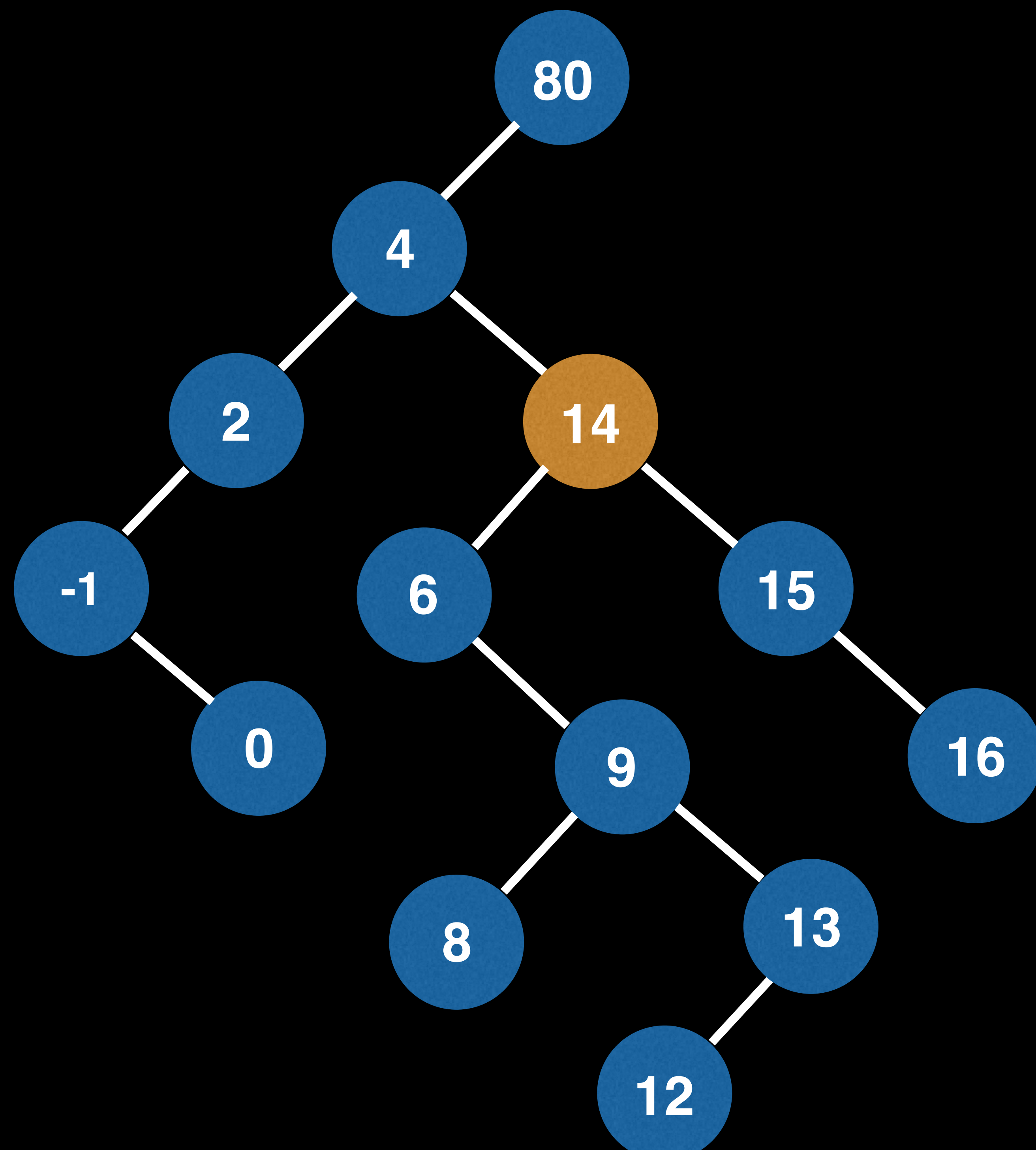
In this example let's remove 14. First begin by finding where 14 is located.



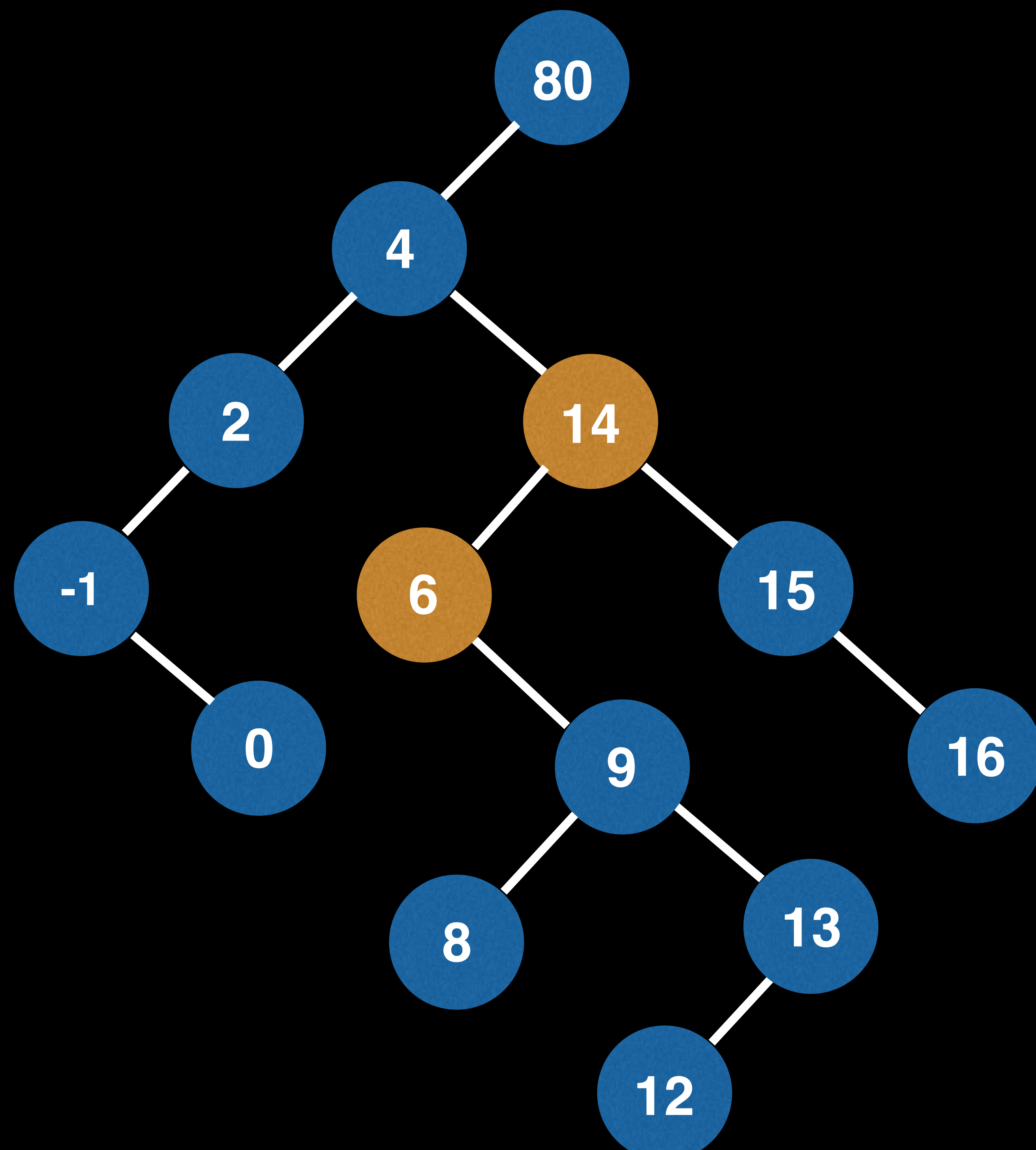
Now find either the smallest value in right subtree or largest in left subtree. Let’s do the latter.



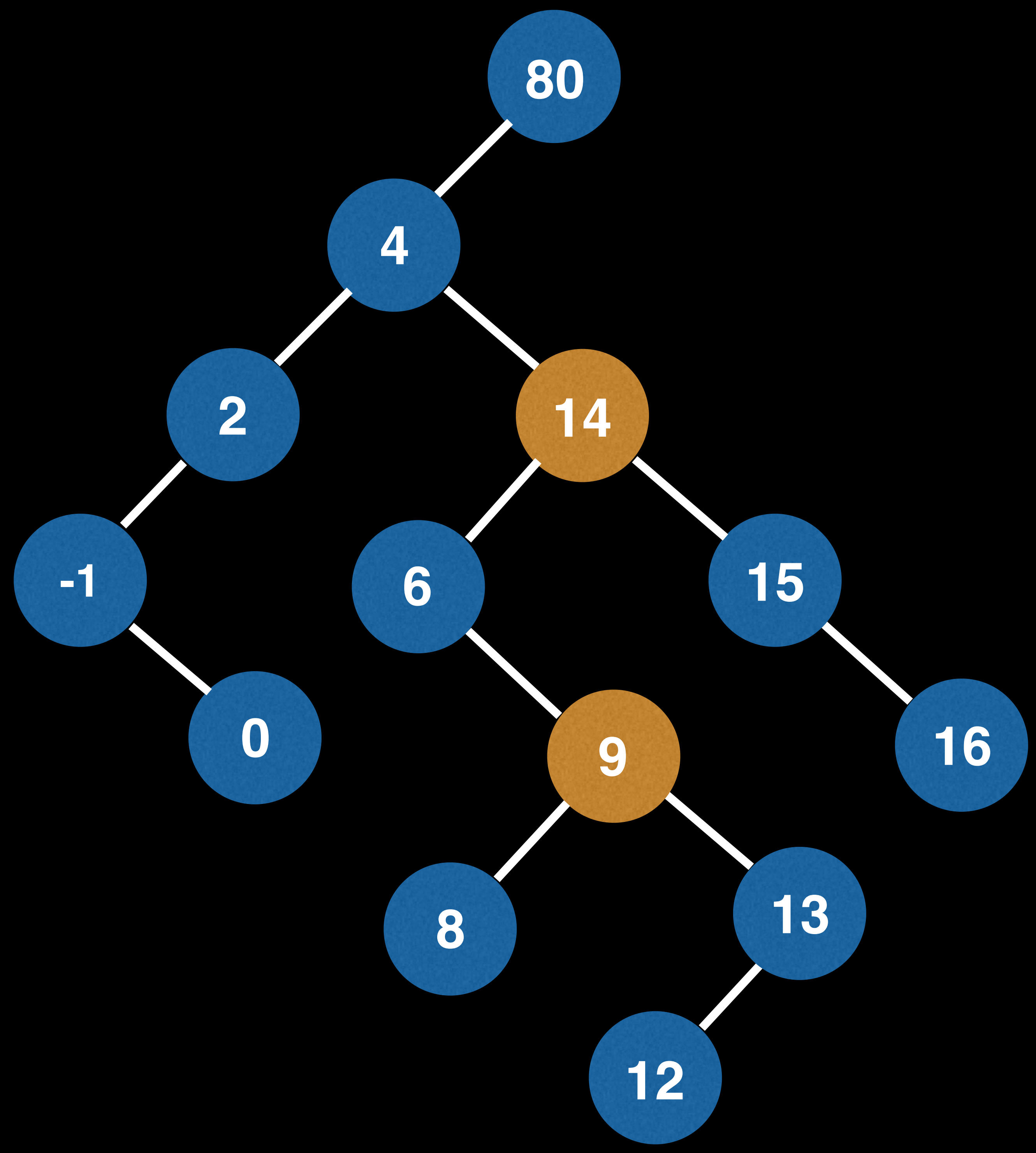
To find the largest value in the left subtree
dig as far right as possible in the left subtree



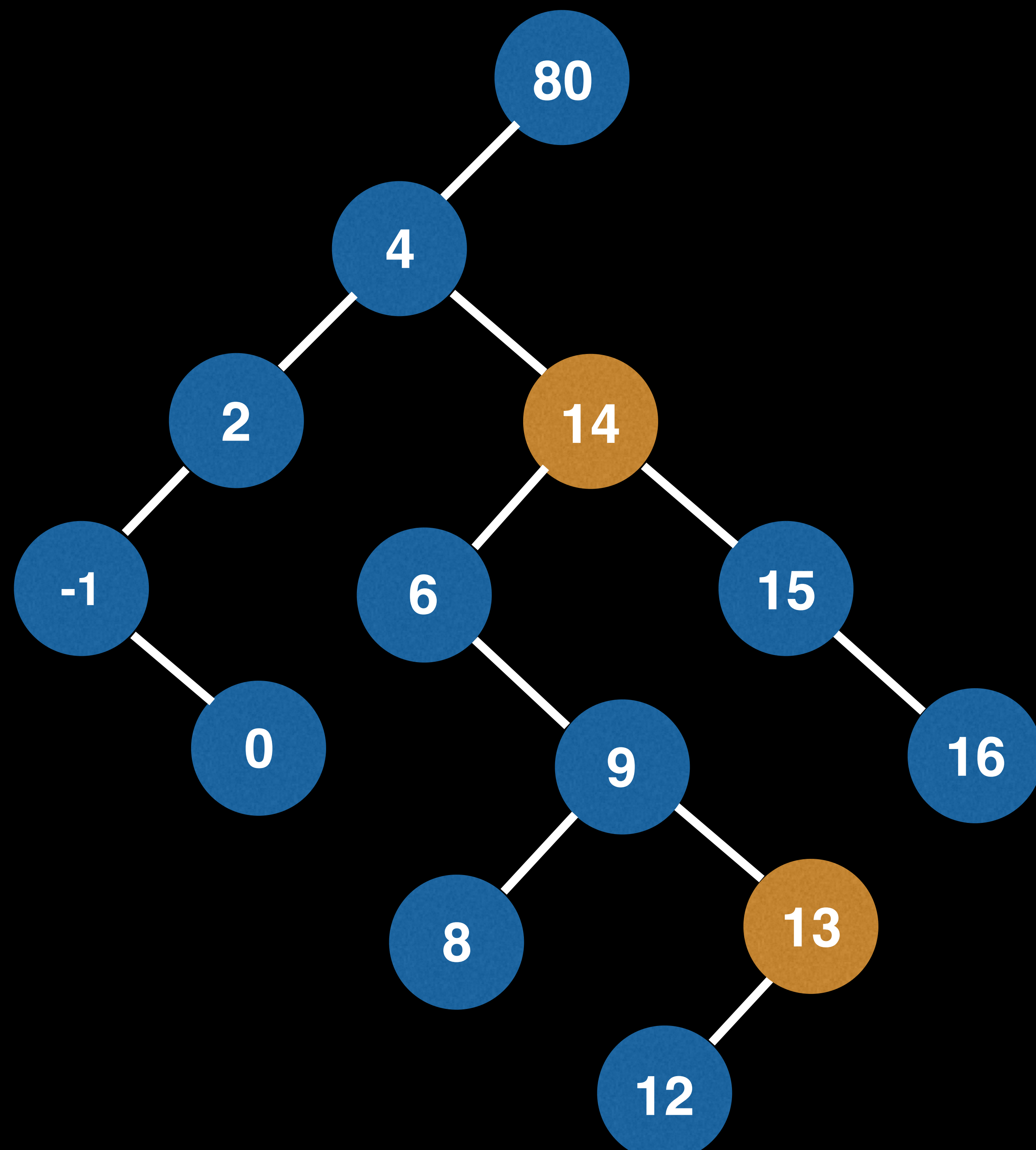
To find the largest value in the left subtree dig as far right as possible in the left subtree



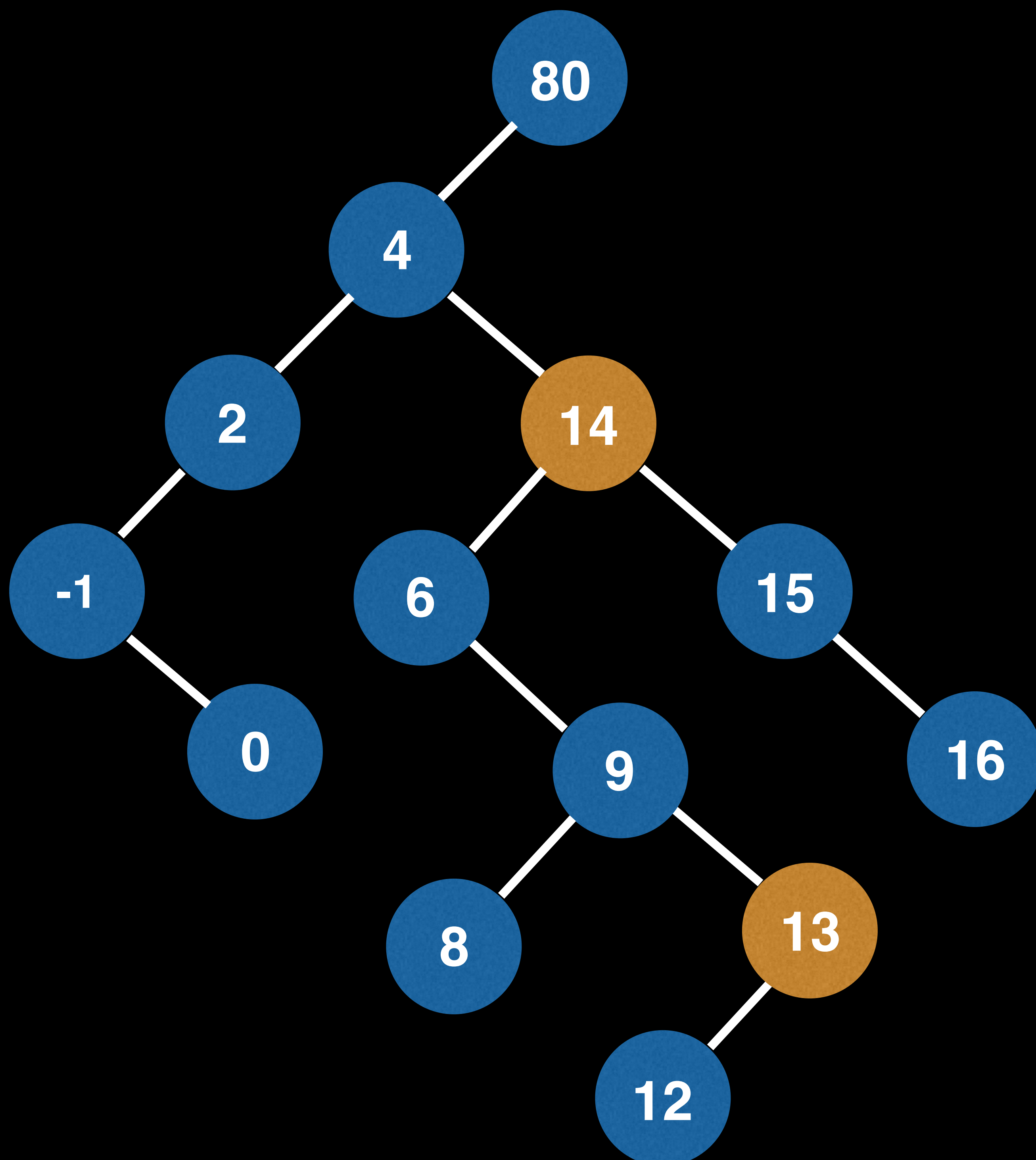
To find the largest value in the left subtree
dig as far right as possible in the left subtree



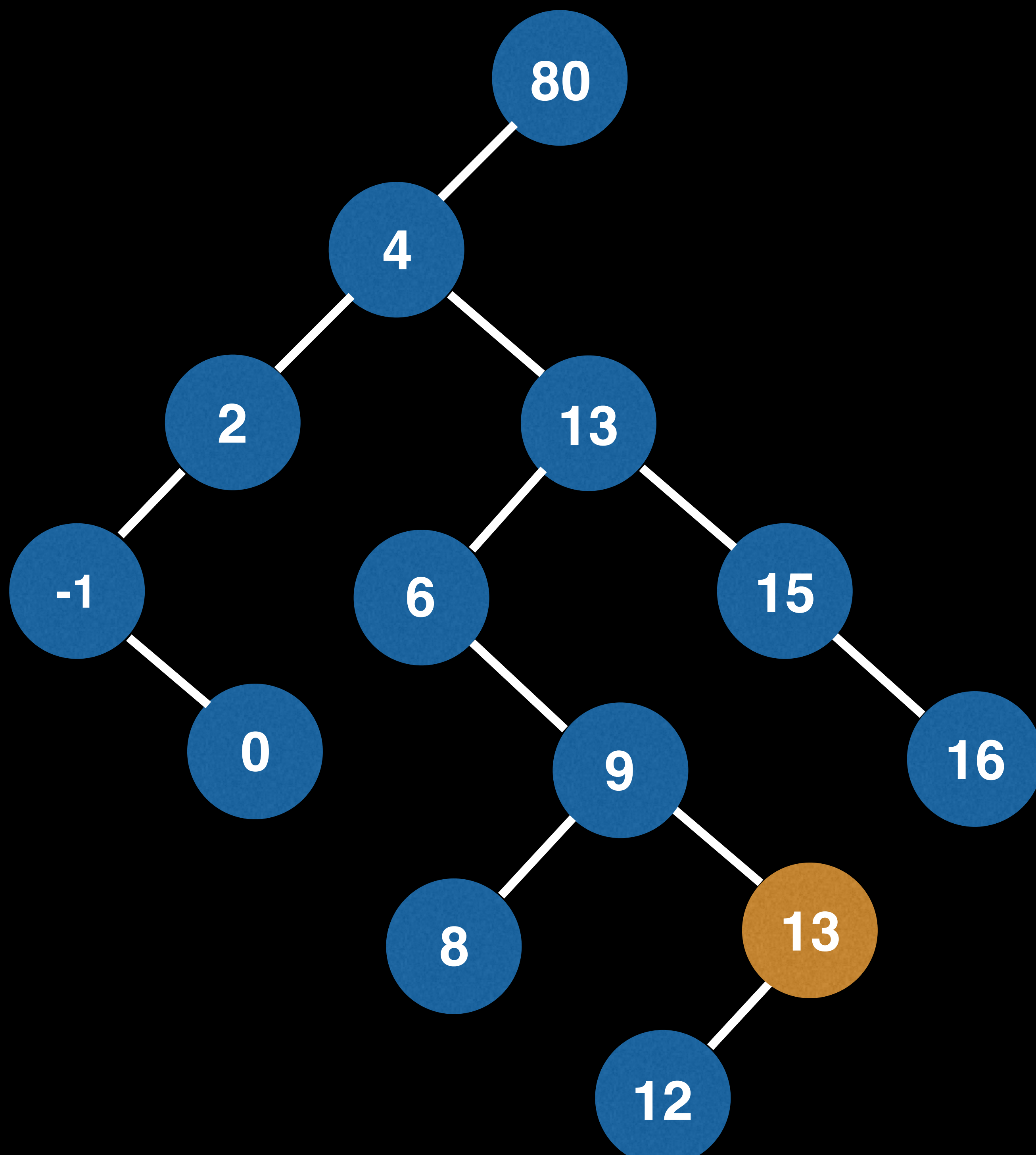
To find the largest value in the left subtree
dig as far right as possible in the left subtree



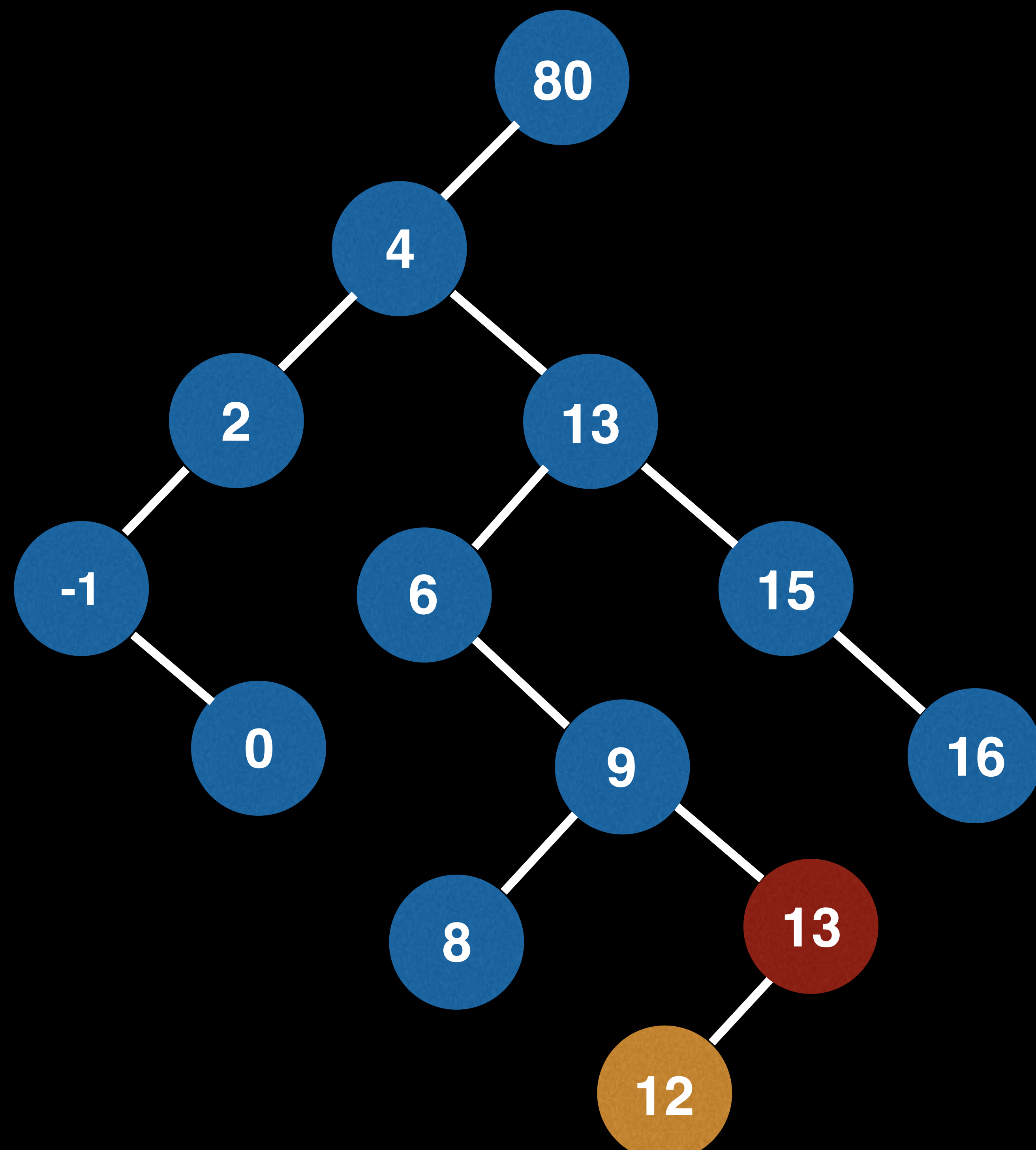
Copy the value found in the successor (13) into the node we wish to remove (14) and remove the successor from the tree.



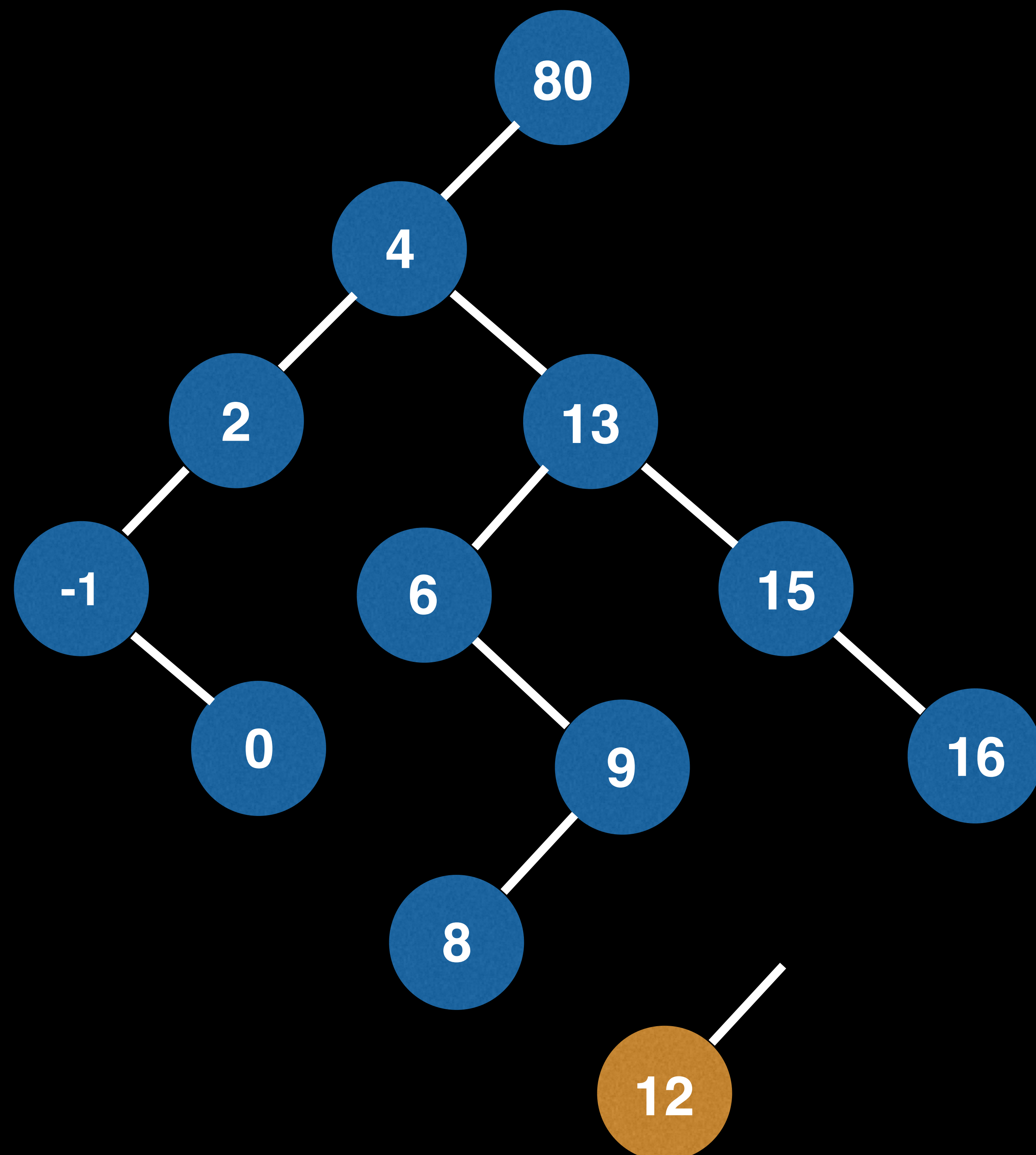
Copy the value found in the successor (13) into the node we wish to remove (14) and remove the successor from the tree.



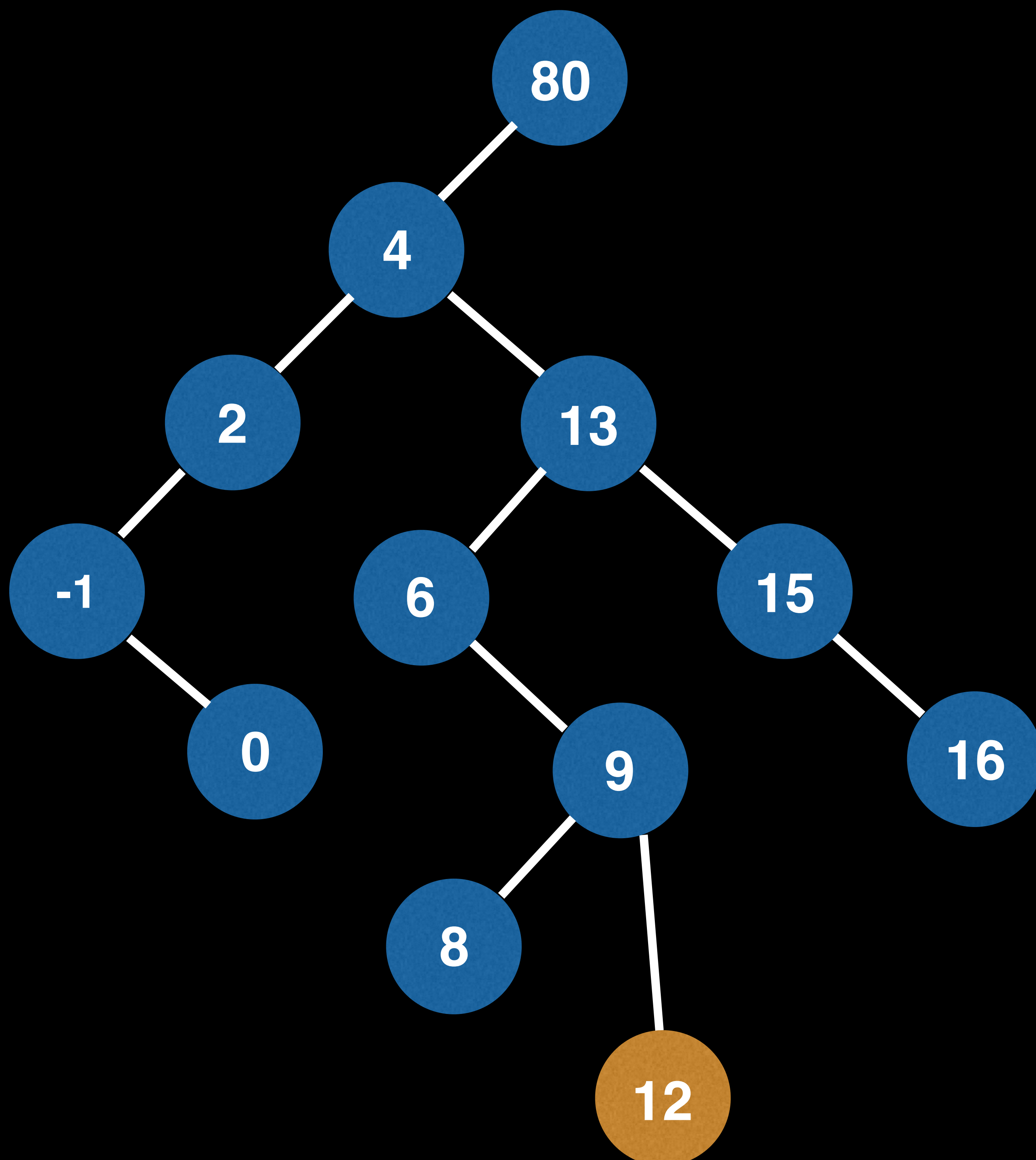
Copy the value found in the successor (13) into the node we wish to remove (14) and remove the successor from the tree.



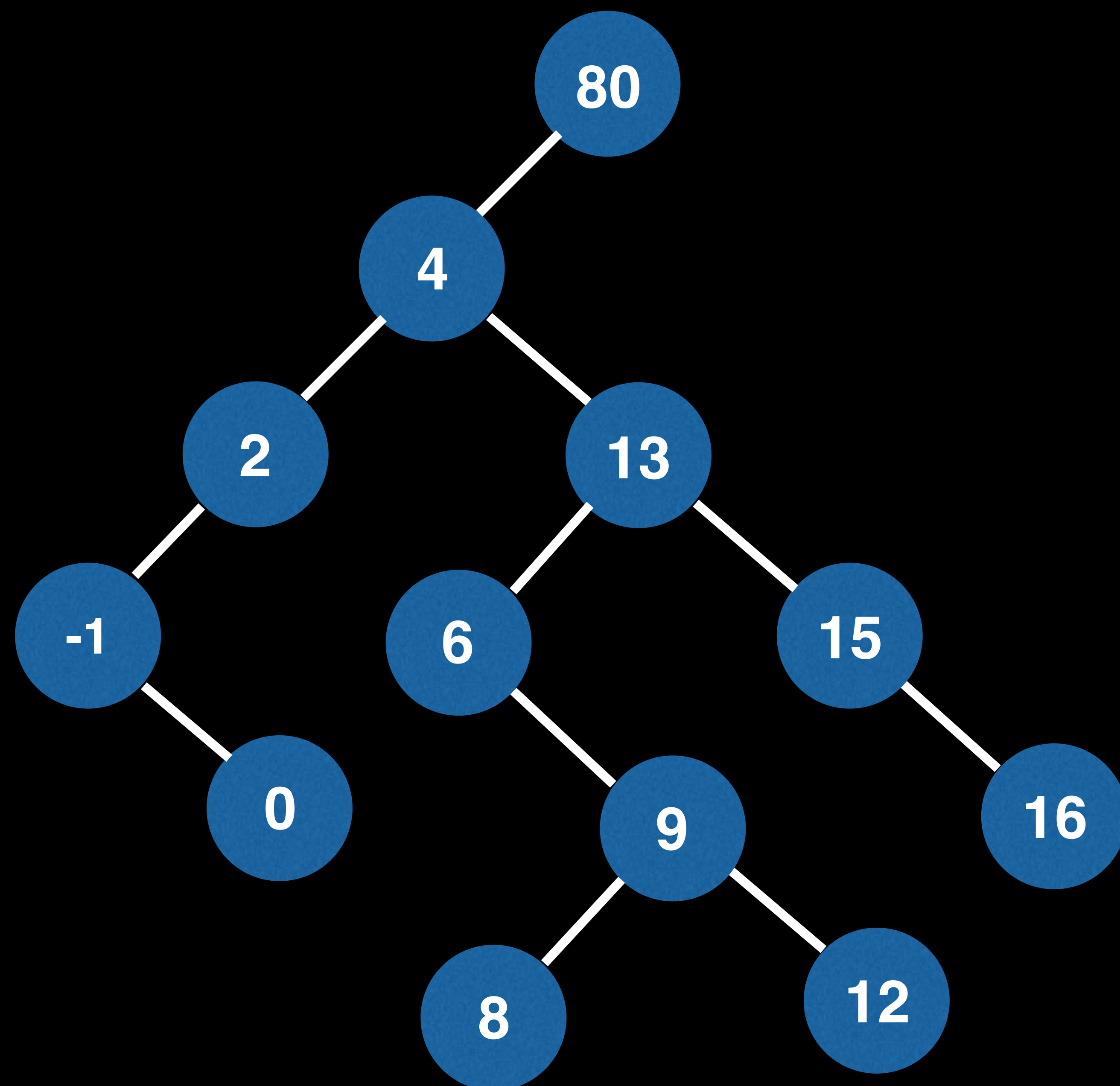
Copy the value found in the successor (13) into the node we wish to remove (14) and remove the successor from the tree.



Copy the value found in the successor (13) into the node we wish to remove (14) and remove the successor from the tree.

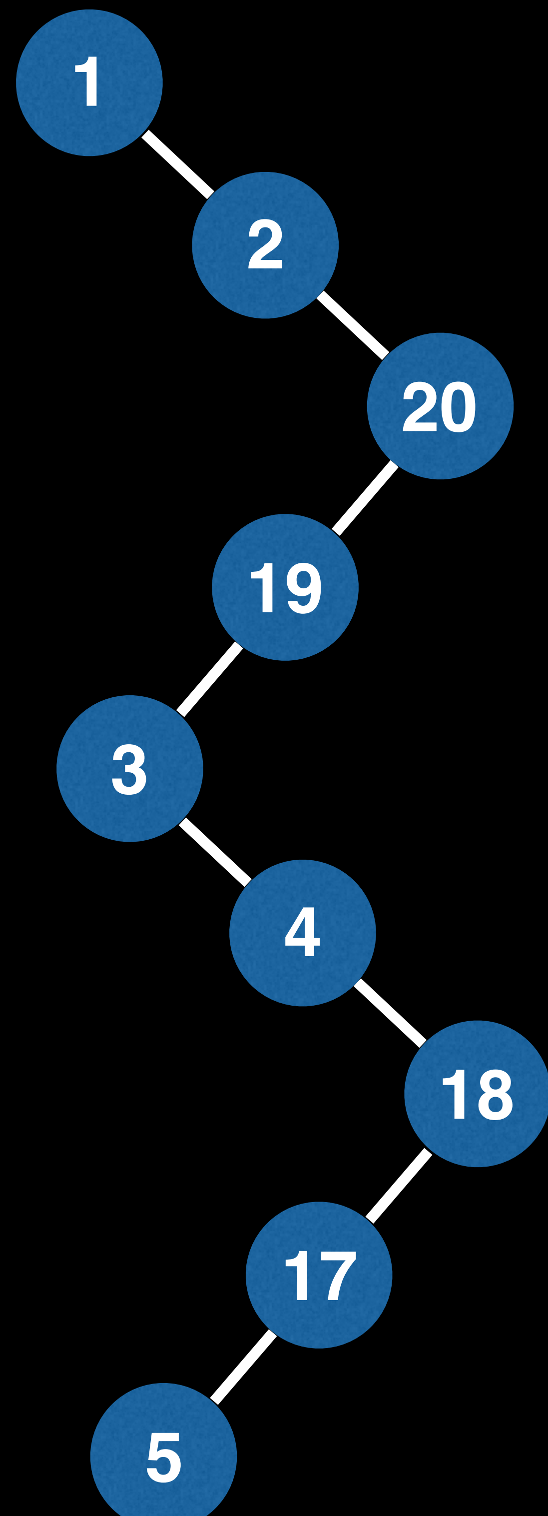


Copy the value found in the successor (13) into the node we wish to remove (14) and remove the successor from the tree.

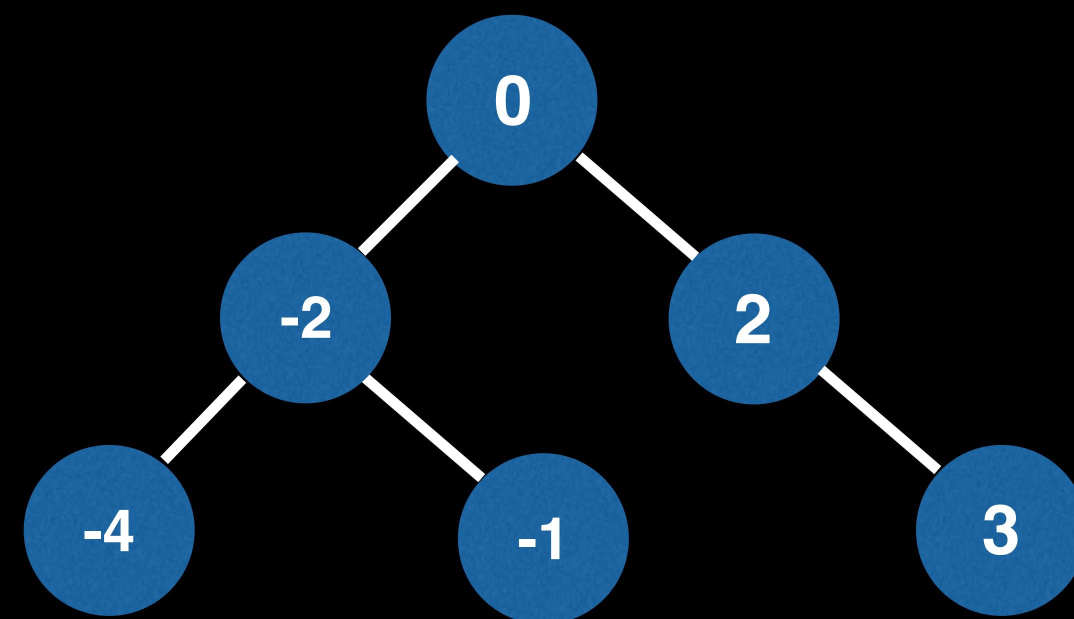


Additional examples

Remove 18

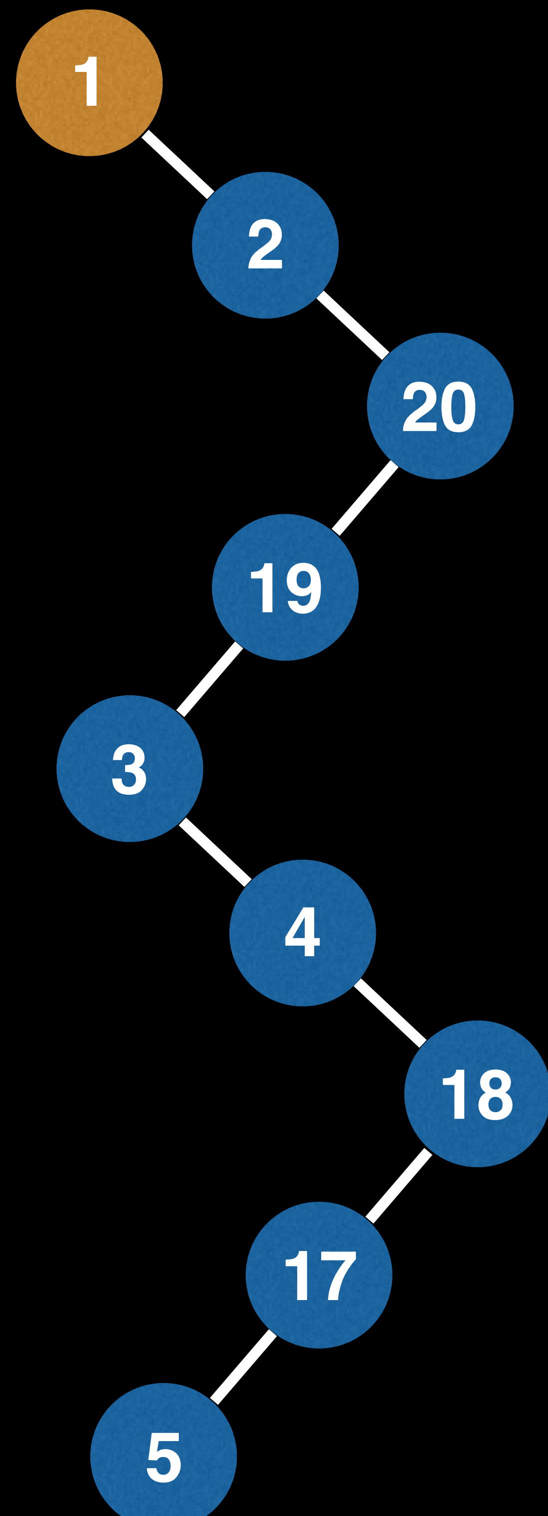


Remove −2

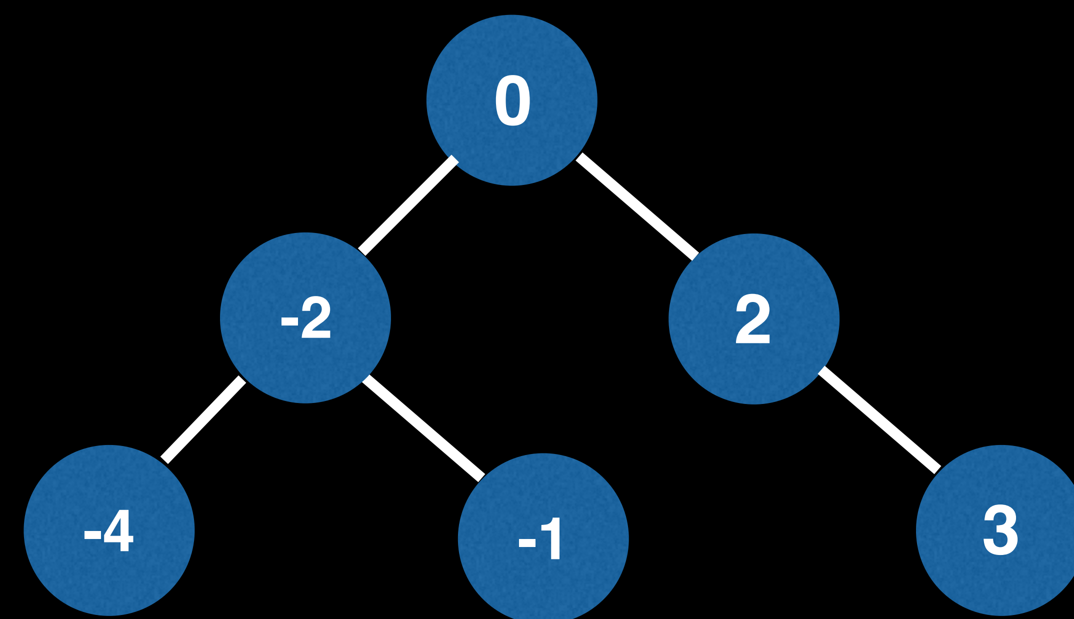


Additional examples

Remove 18

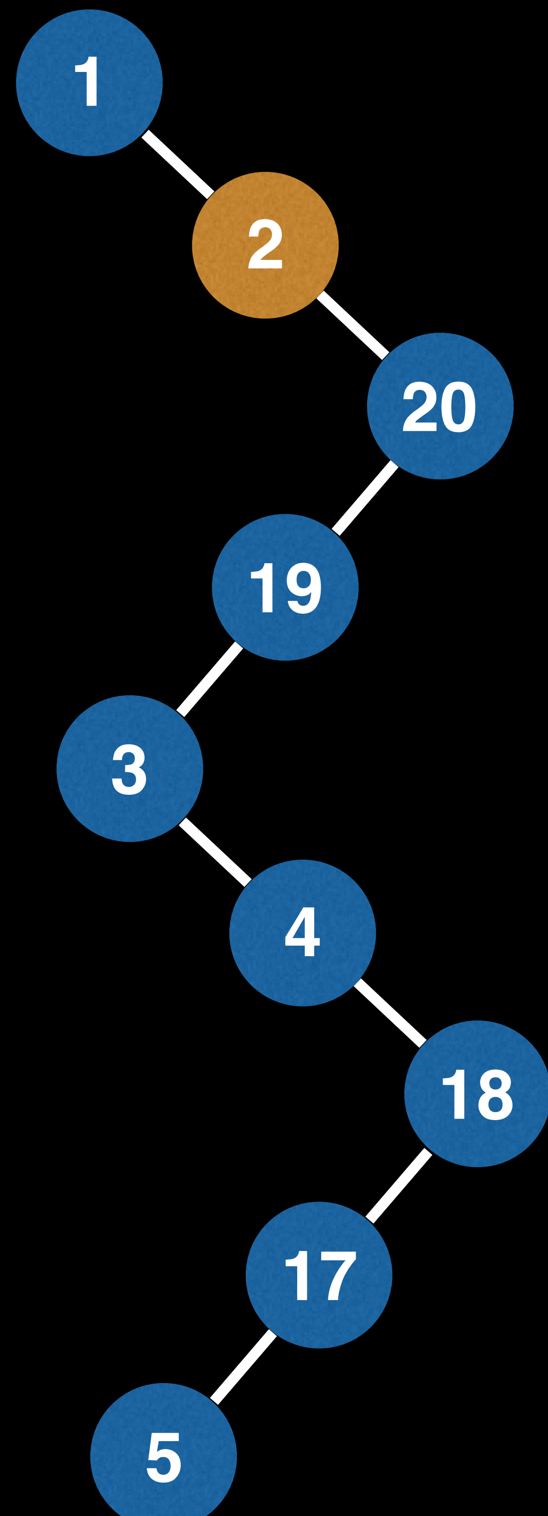


Remove −2

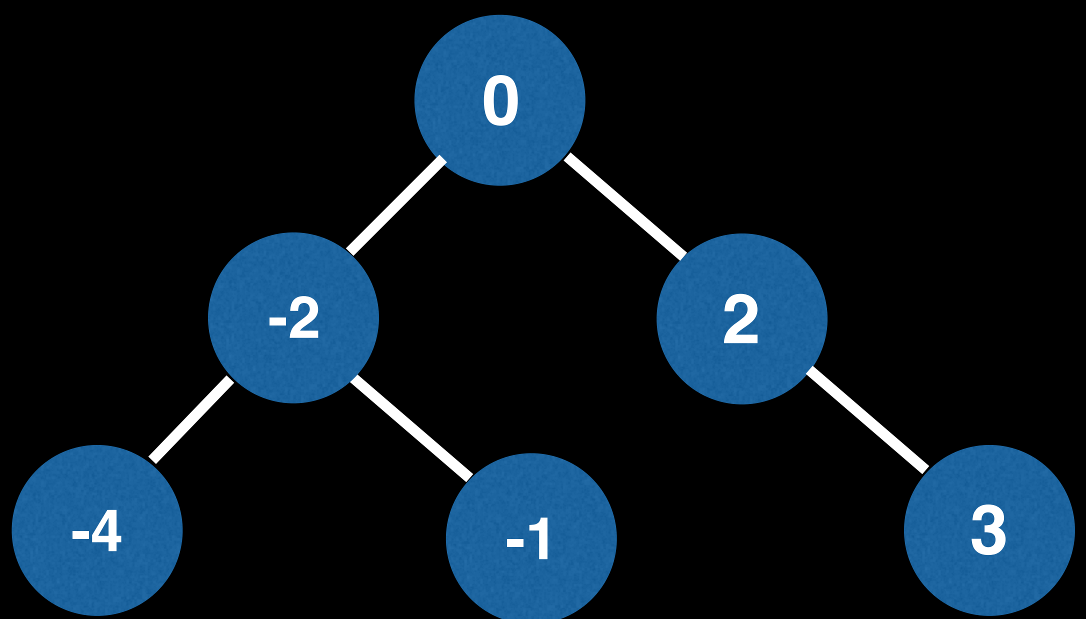


Additional examples

Remove 18

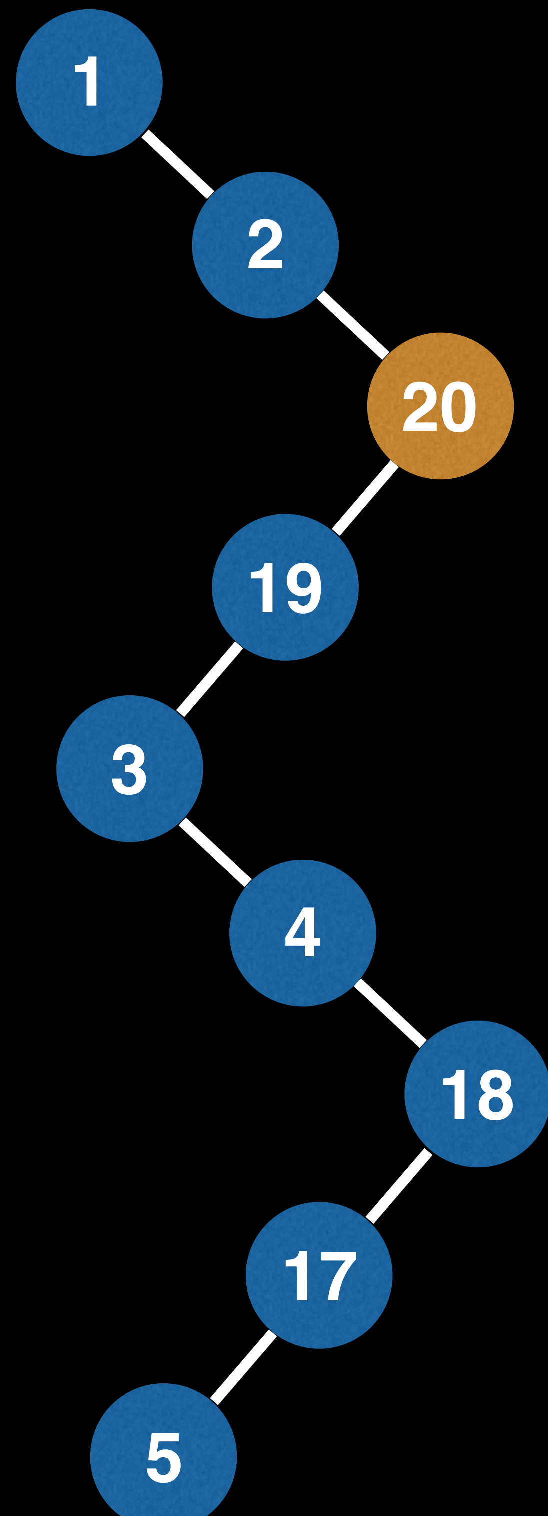


Remove −2

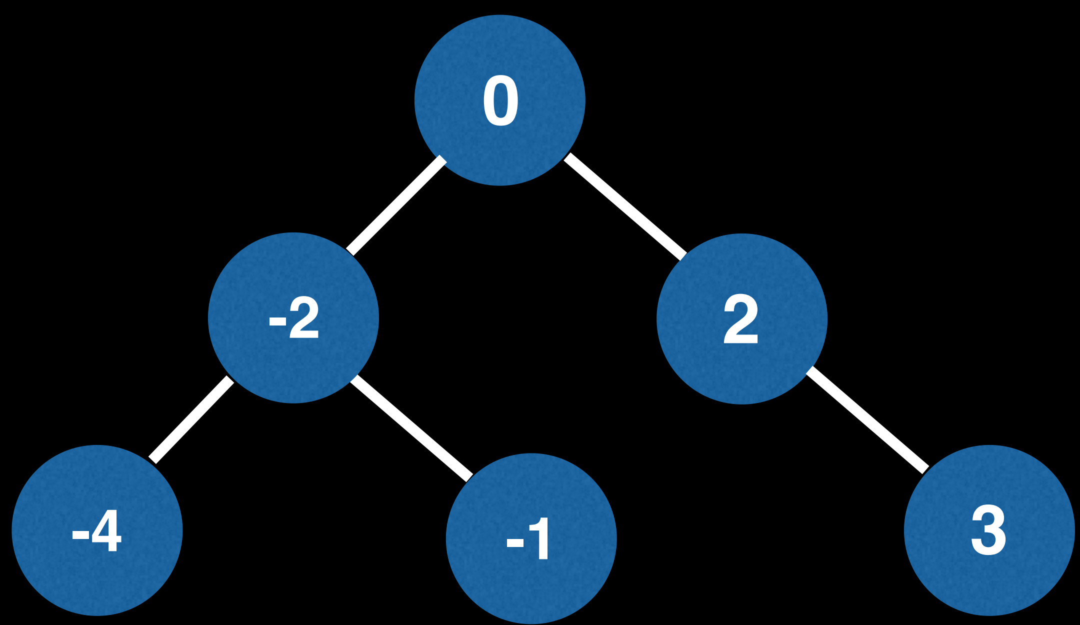


Additional examples

Remove 18

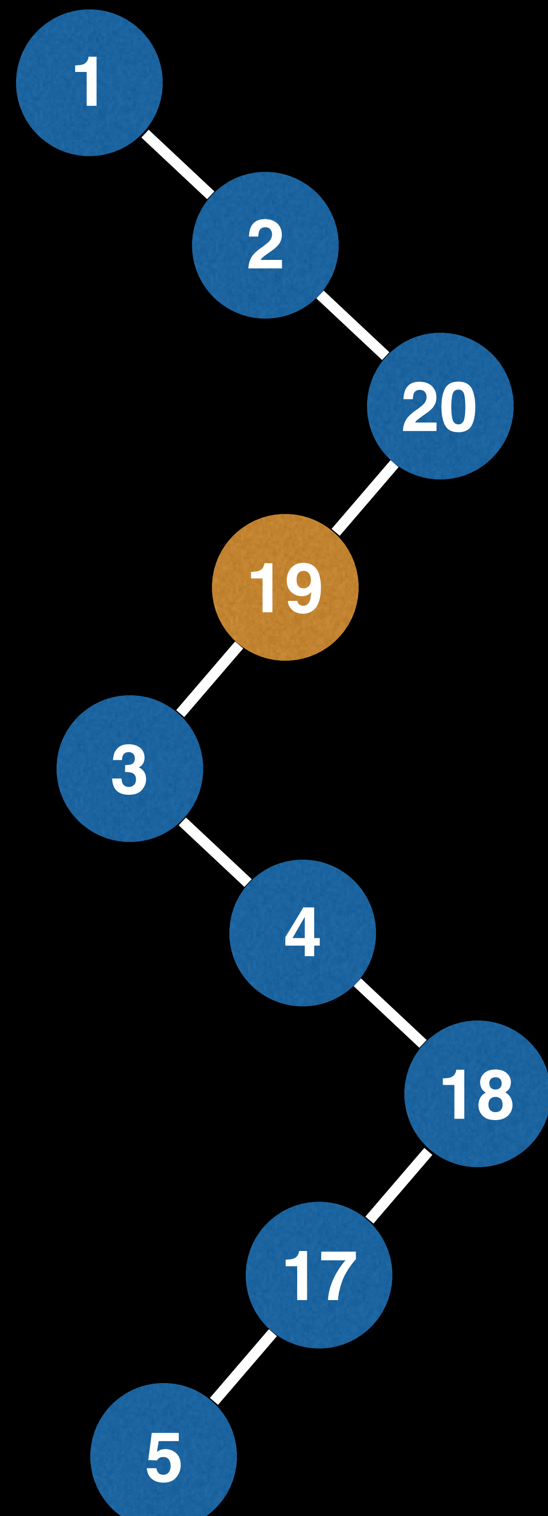


Remove −2

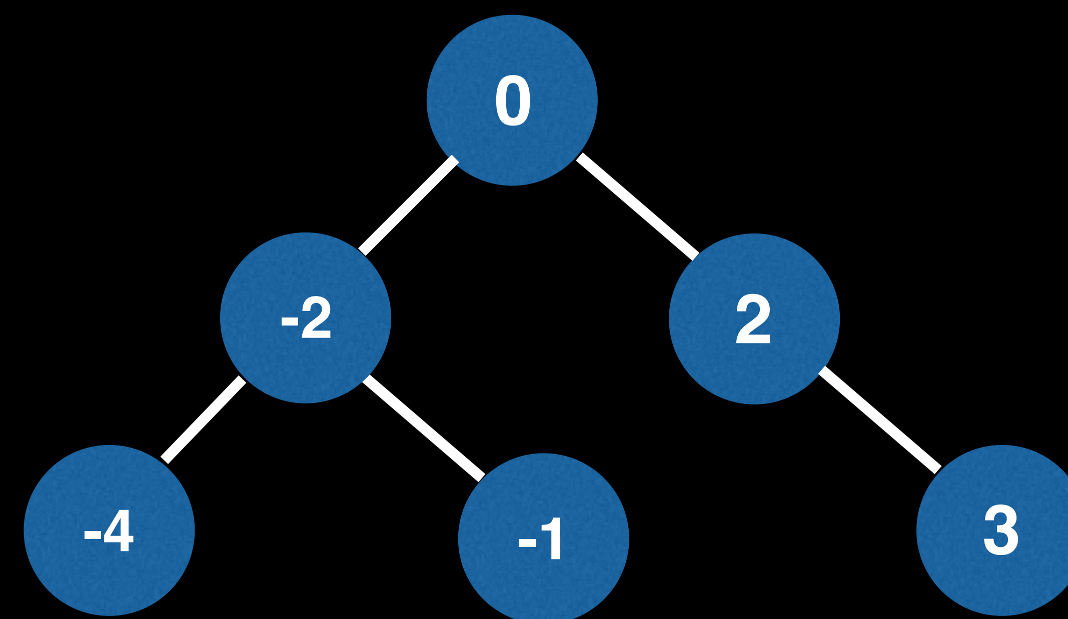


Additional examples

Remove 18

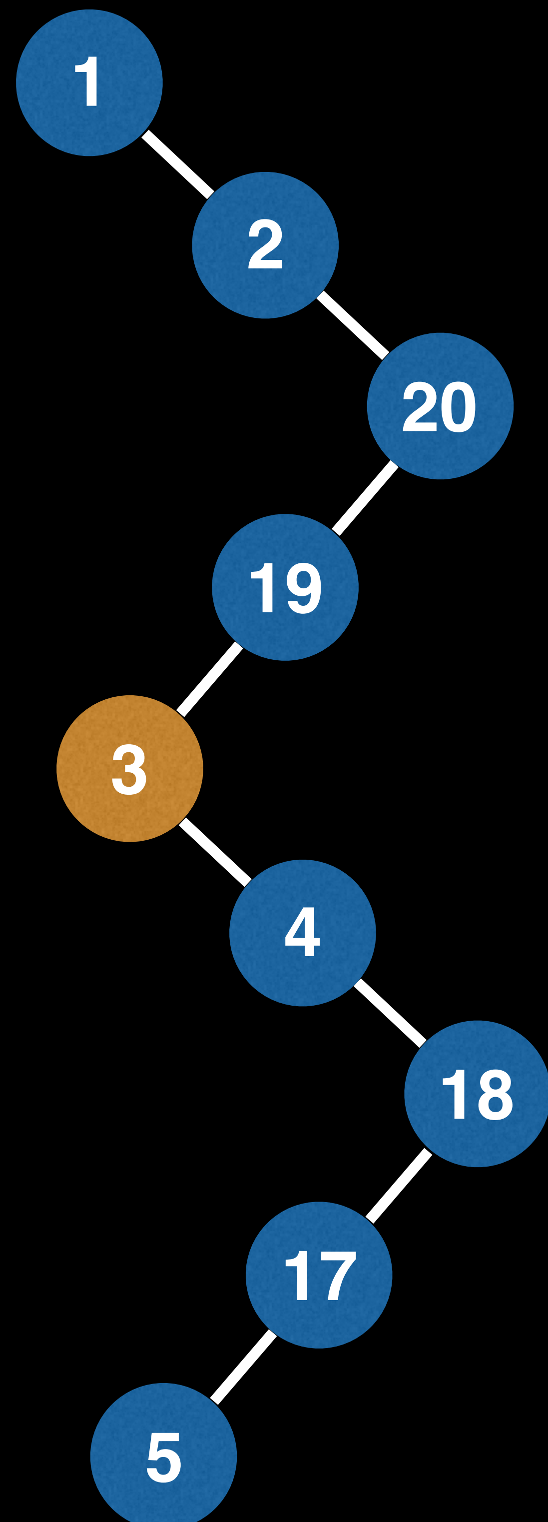


Remove −2

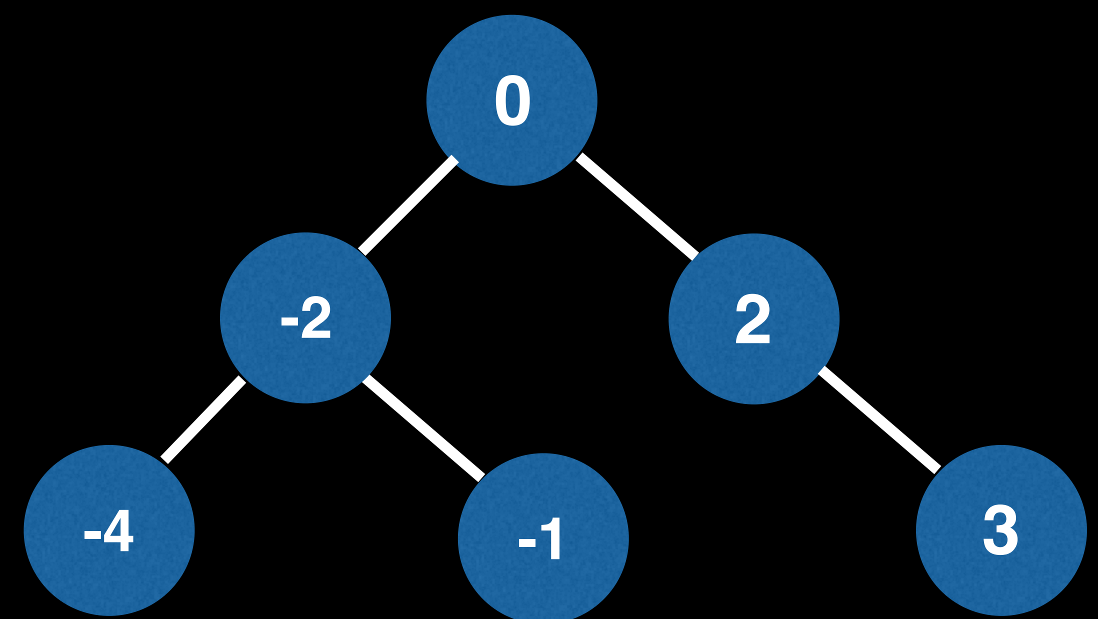


Additional examples

Remove 18

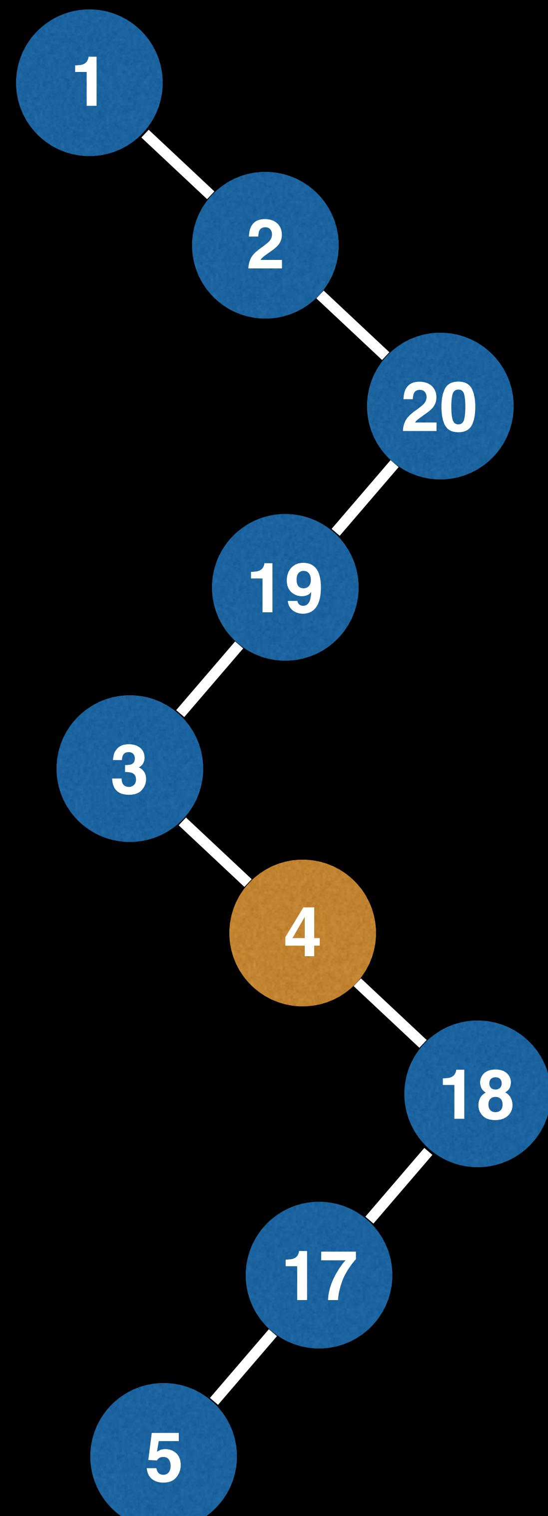


Remove −2

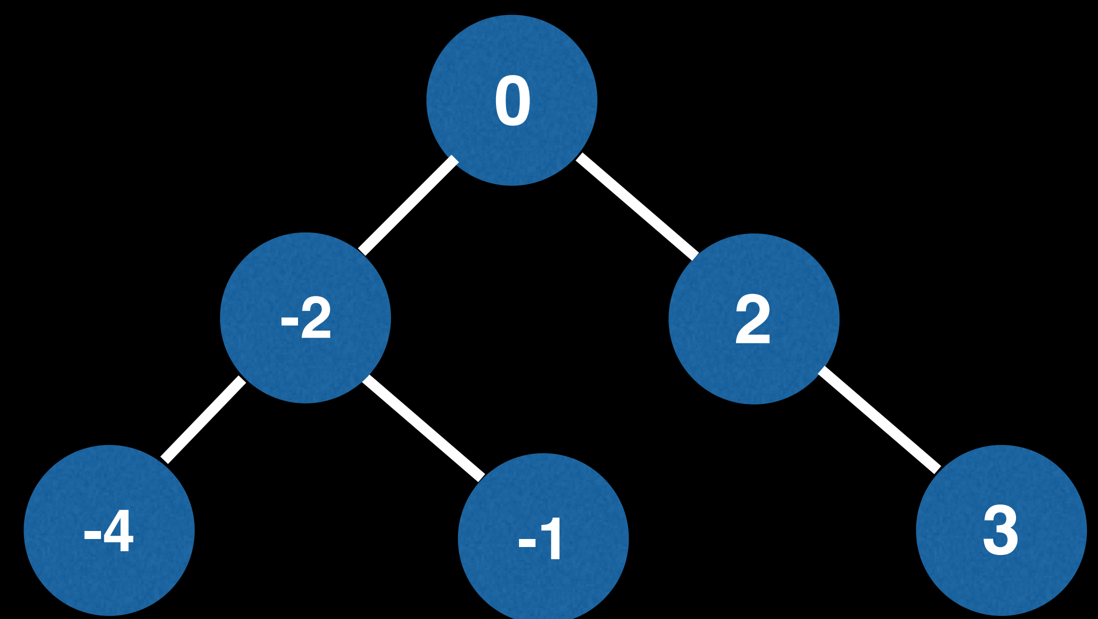


Additional examples

Remove 18

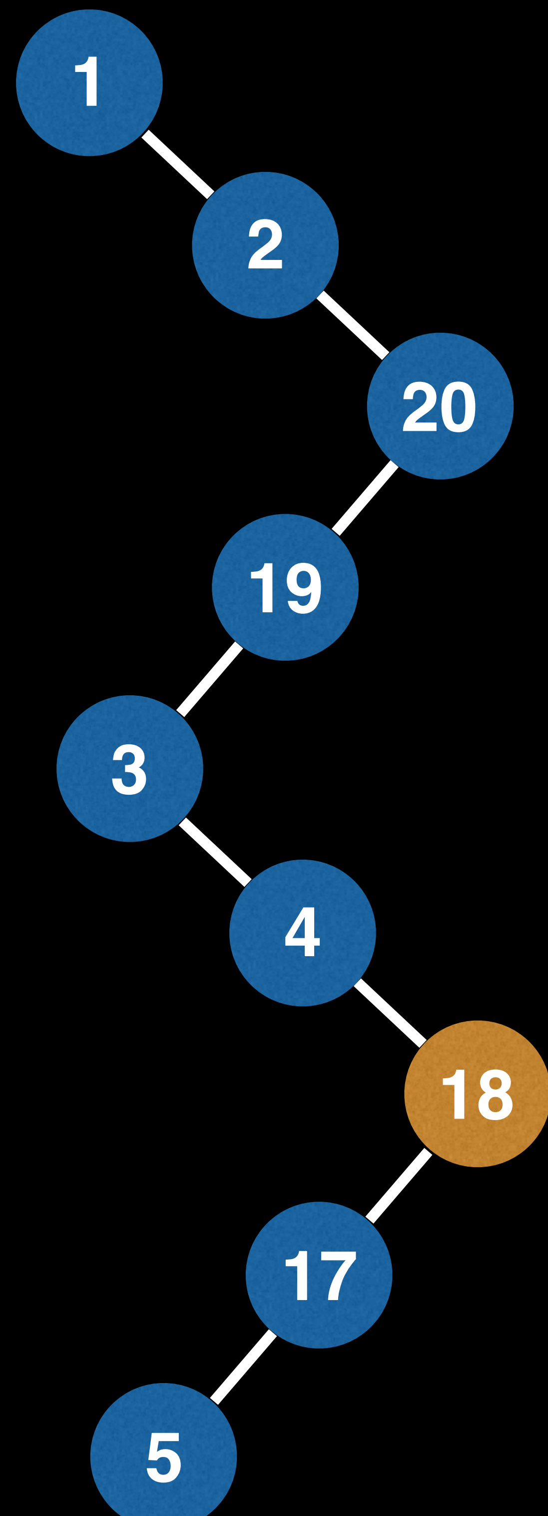


Remove −2

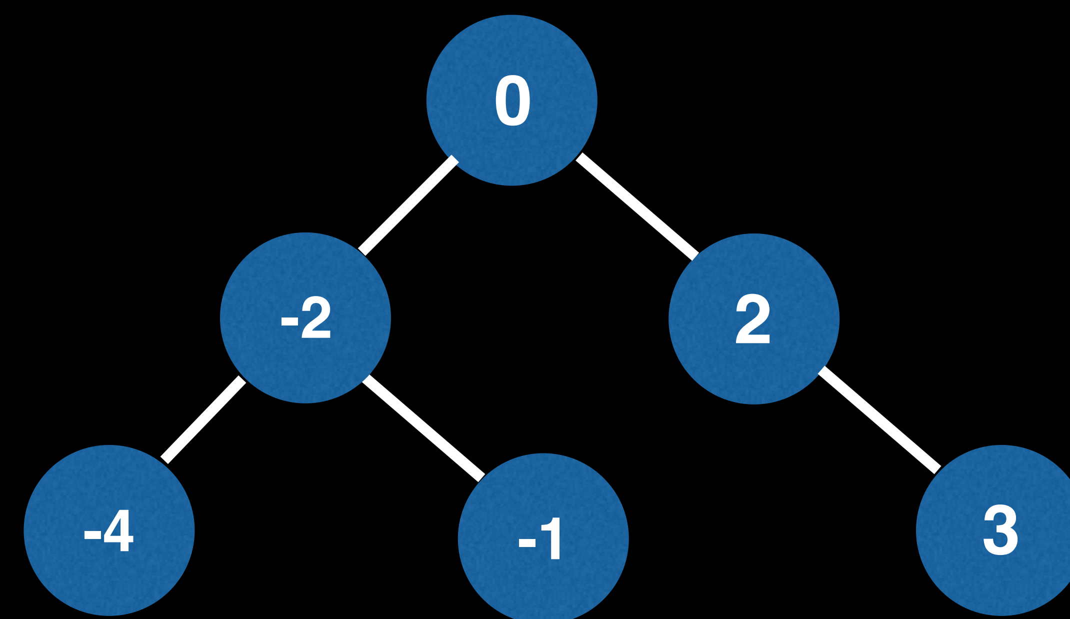


Additional examples

Remove 18

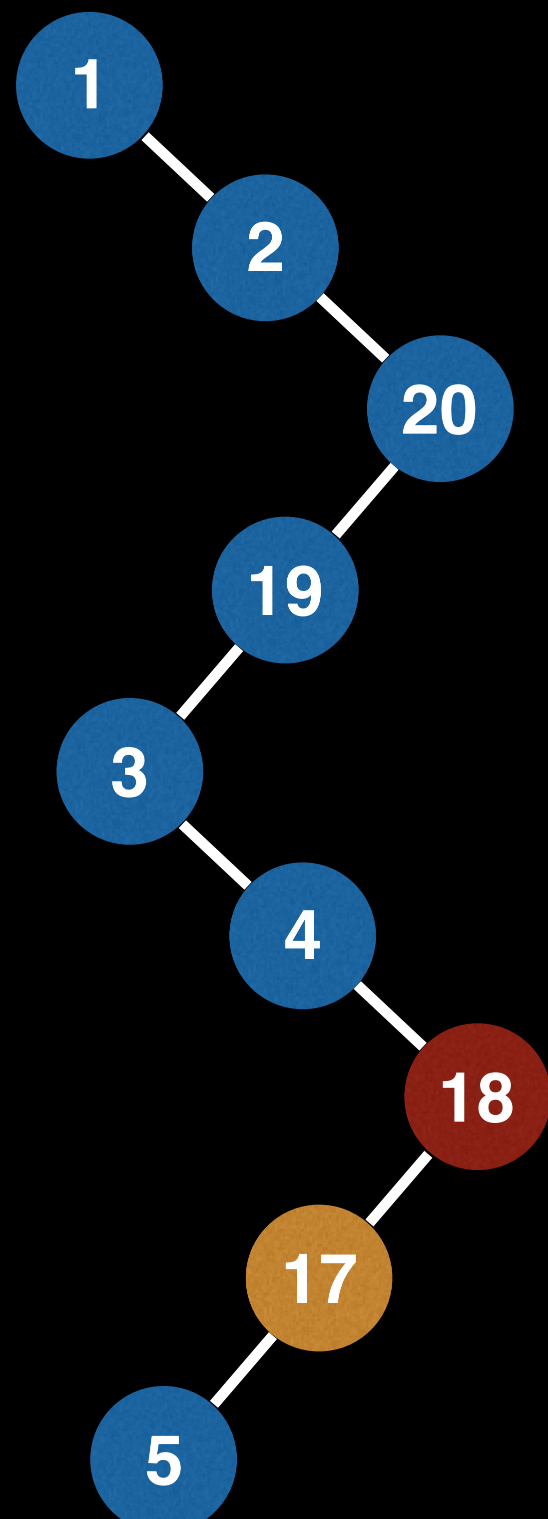


Remove −2

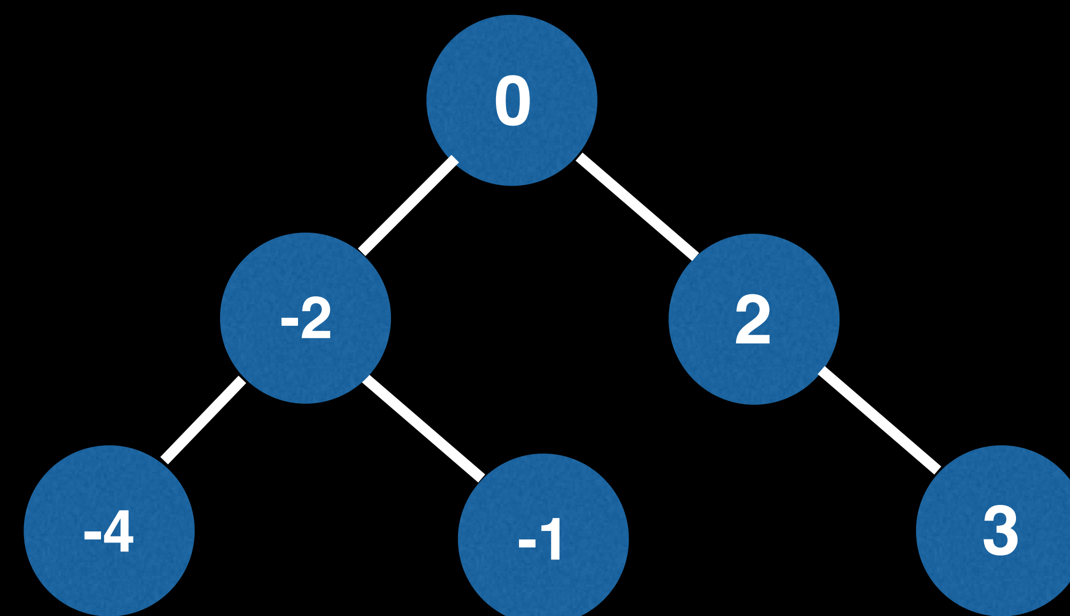


Additional examples

Remove 18

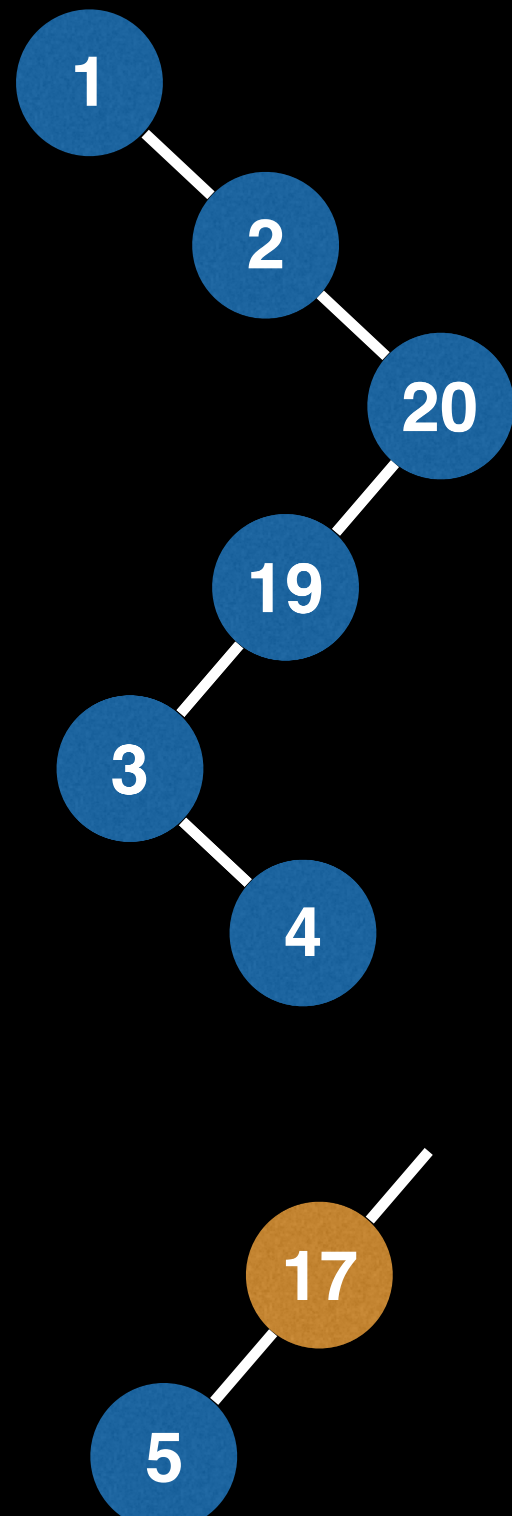


Remove −2

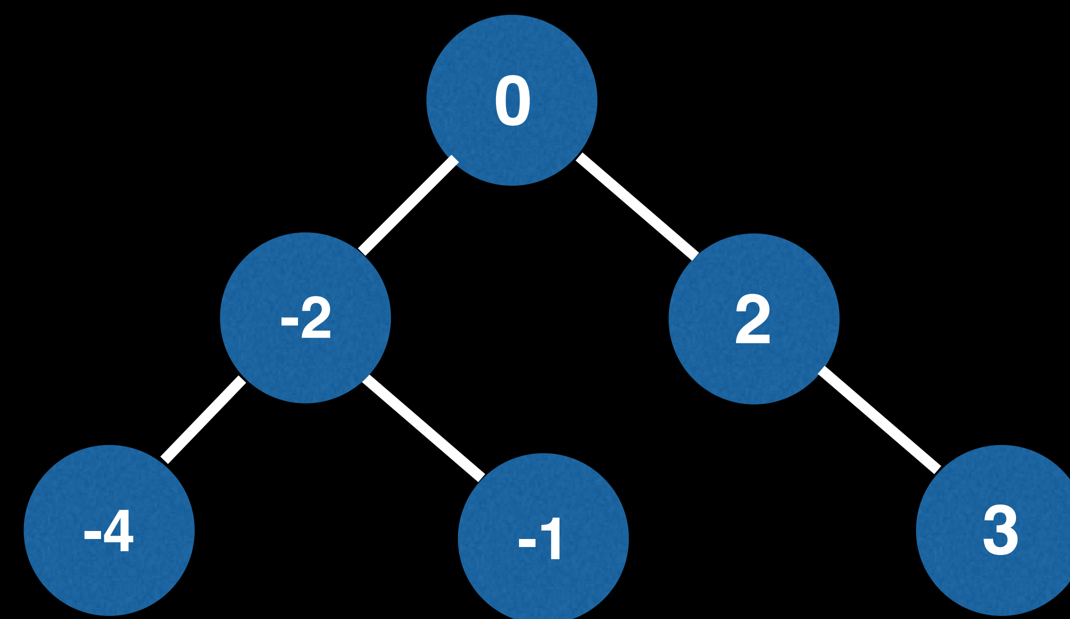


Additional examples

Remove 18

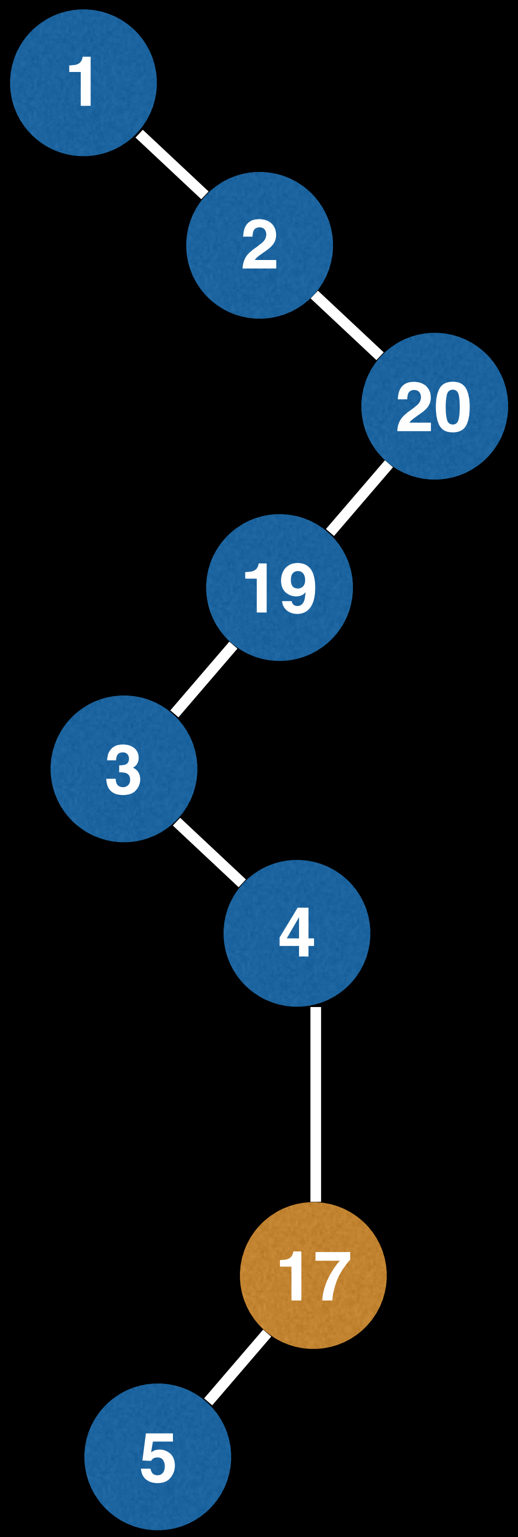


Remove −2

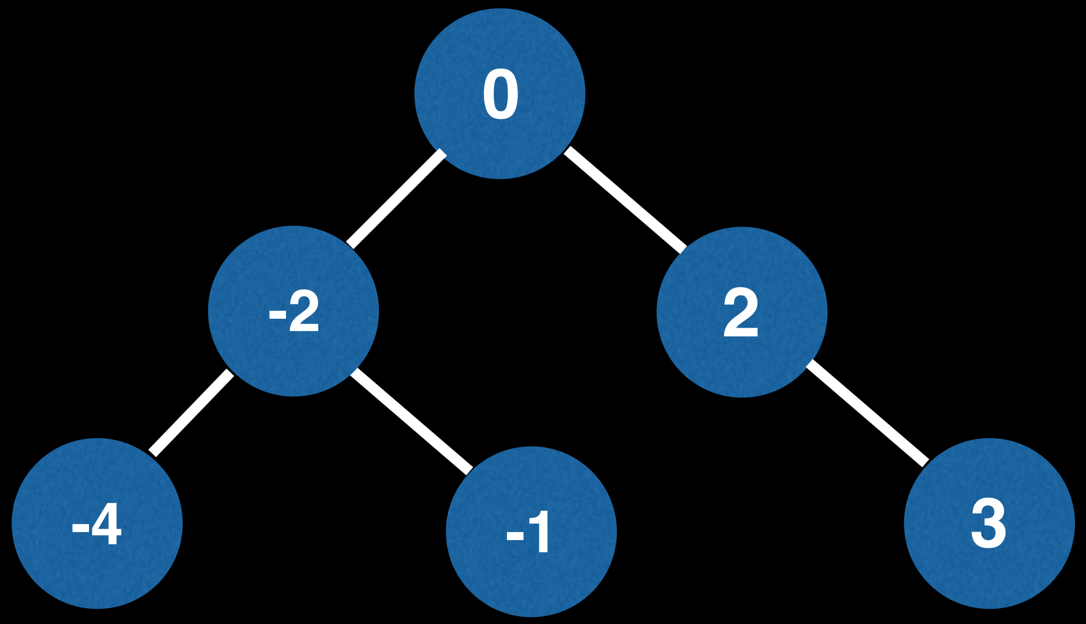


Additional examples

Remove 18

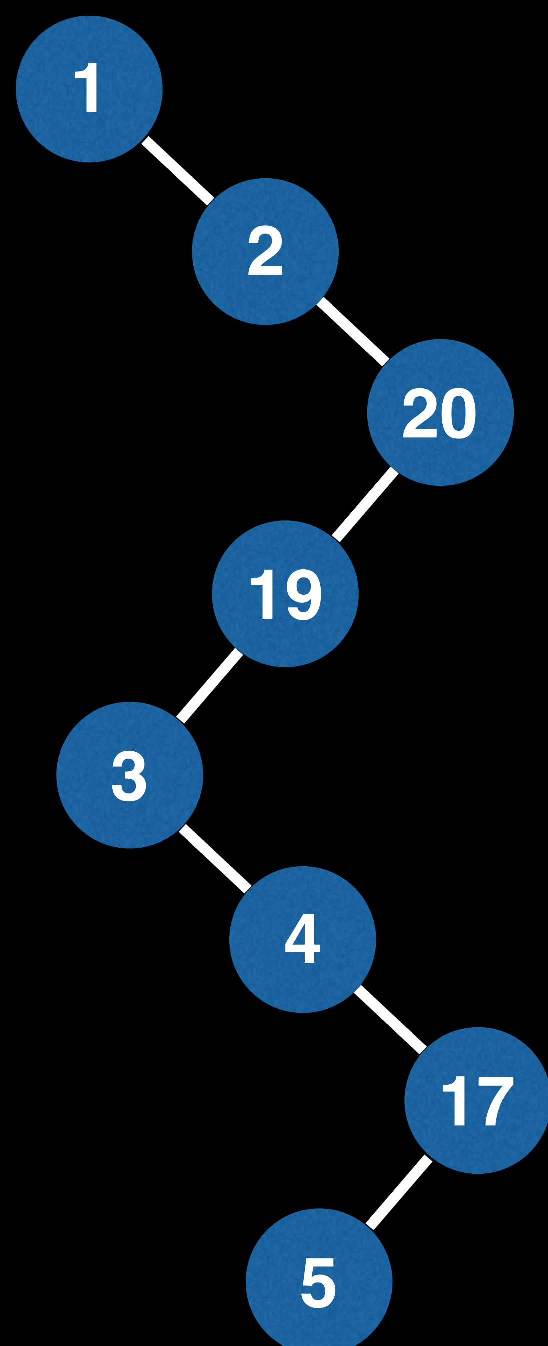


Remove −2

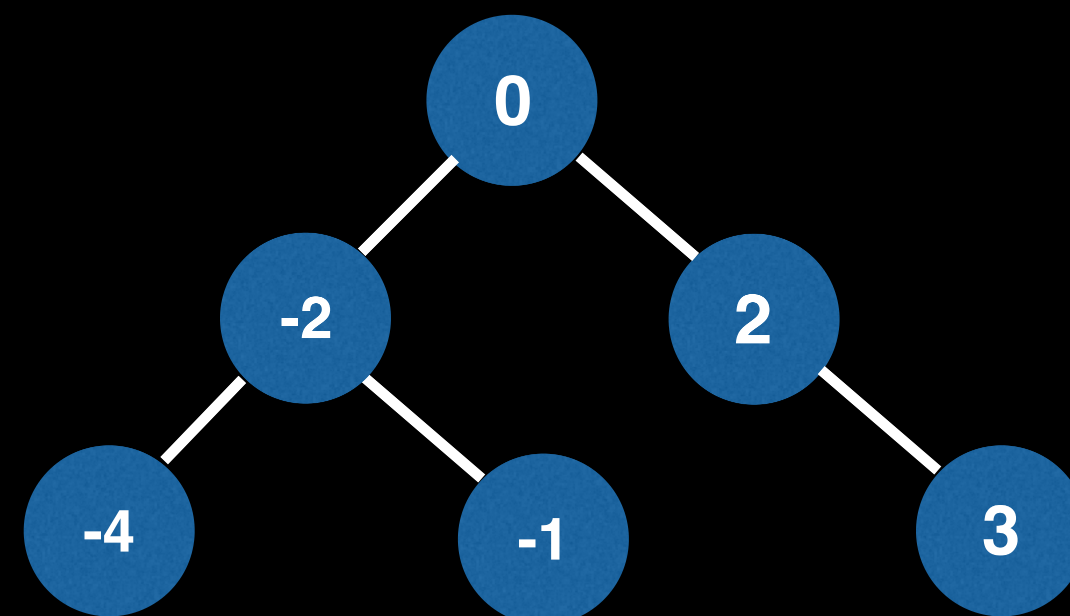


Additional examples

Remove 18

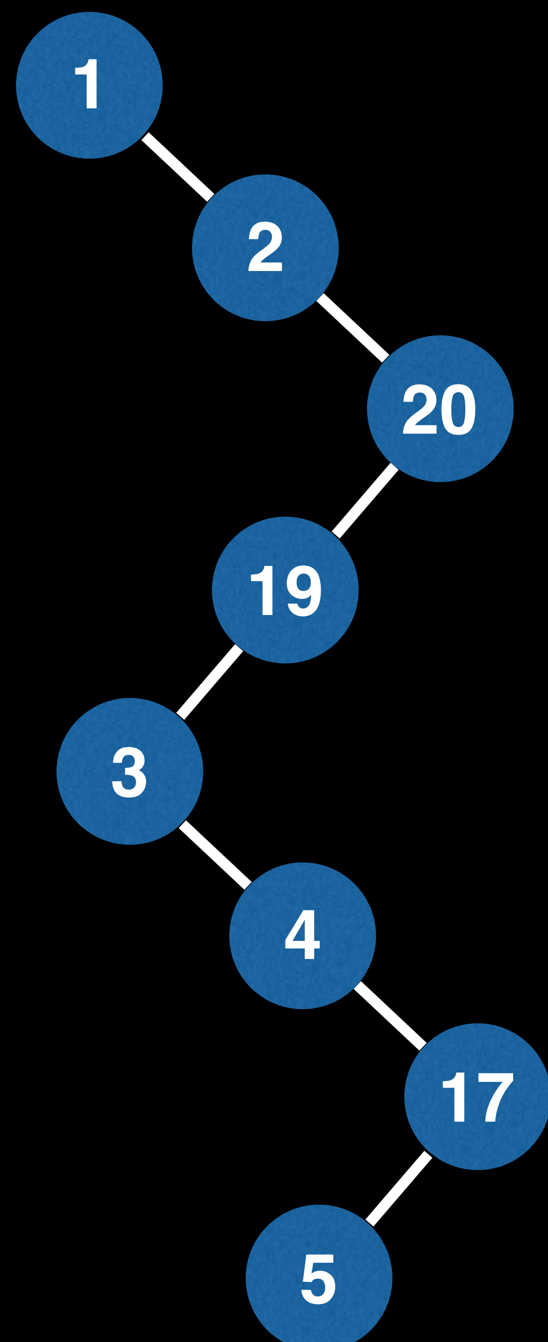


Remove −2

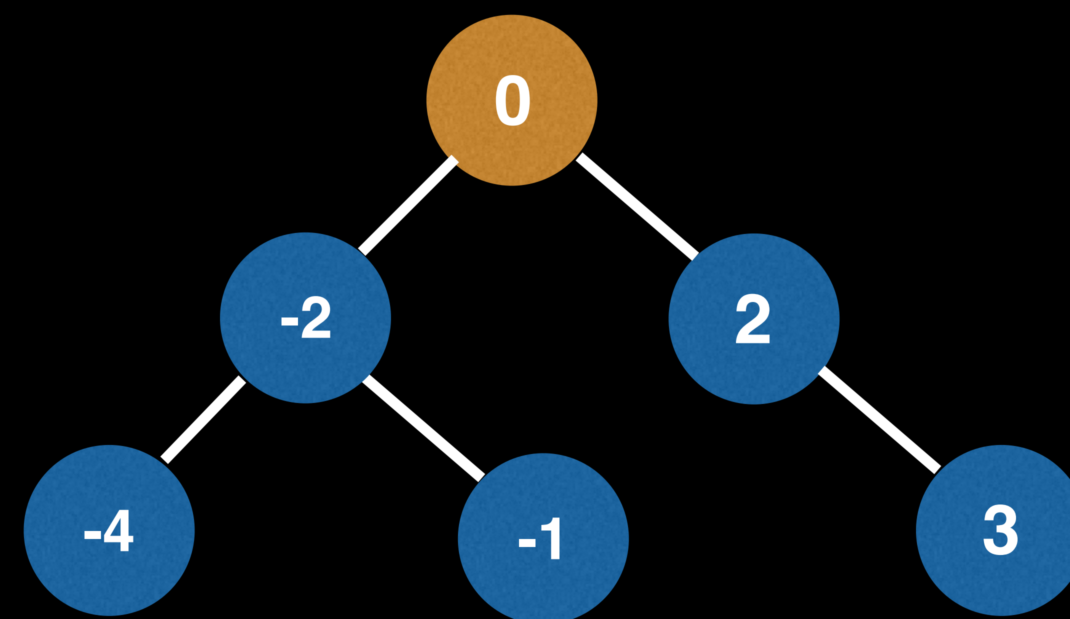


Additional examples

Remove 18



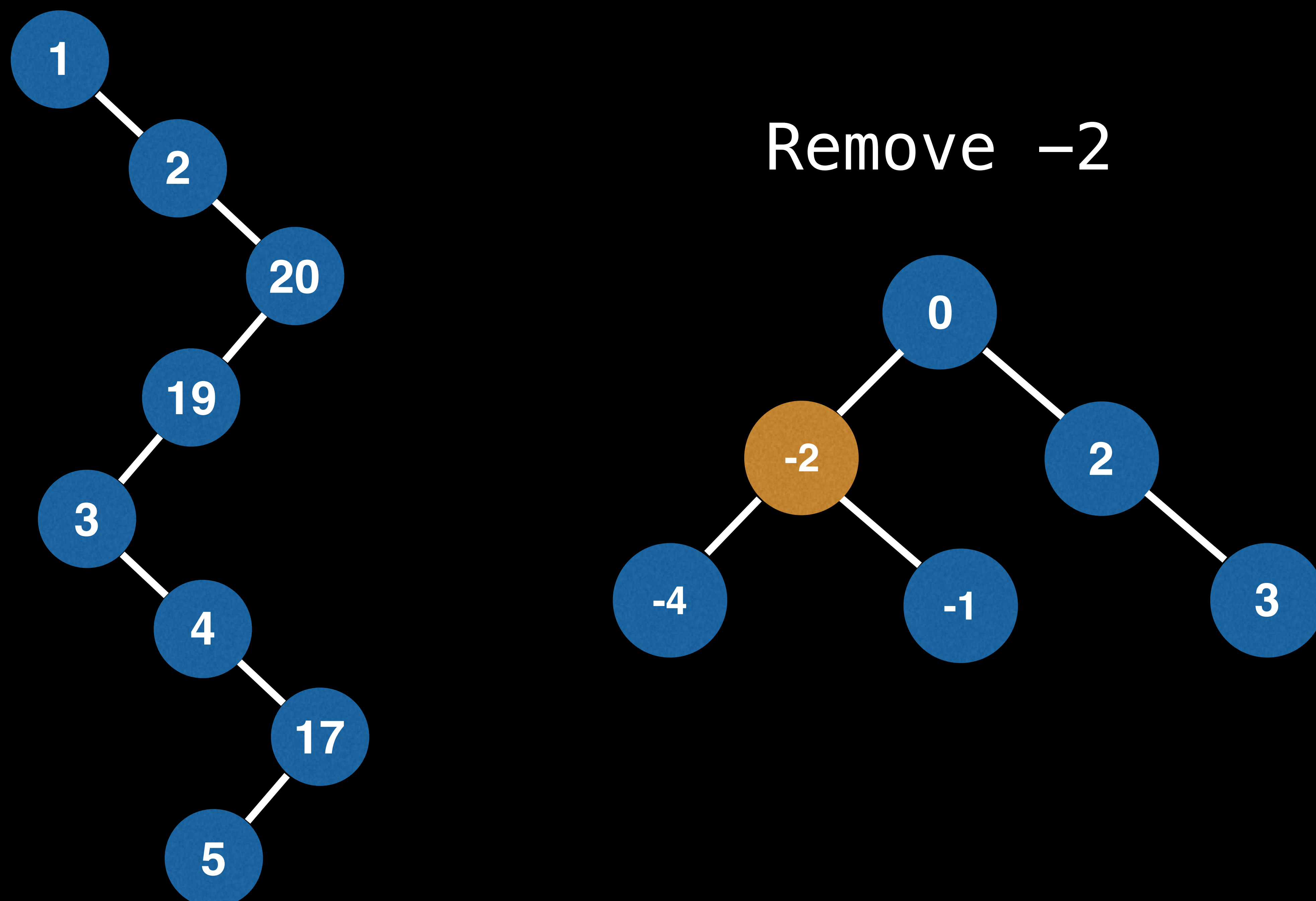
Remove −2



Additional examples

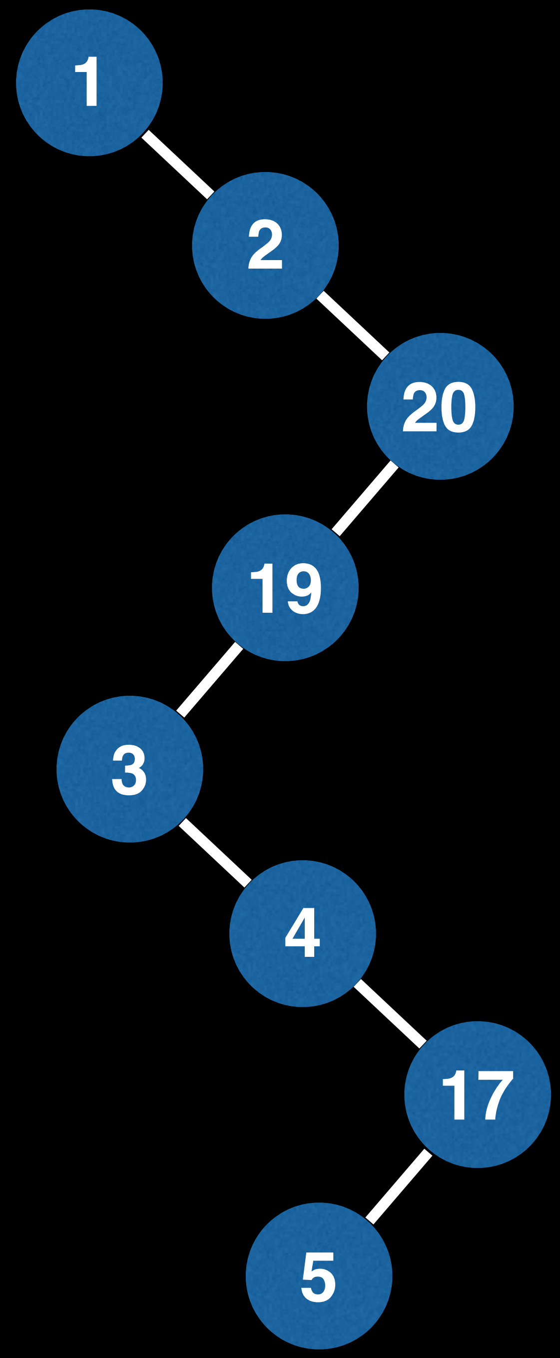
Remove 18

Remove −2

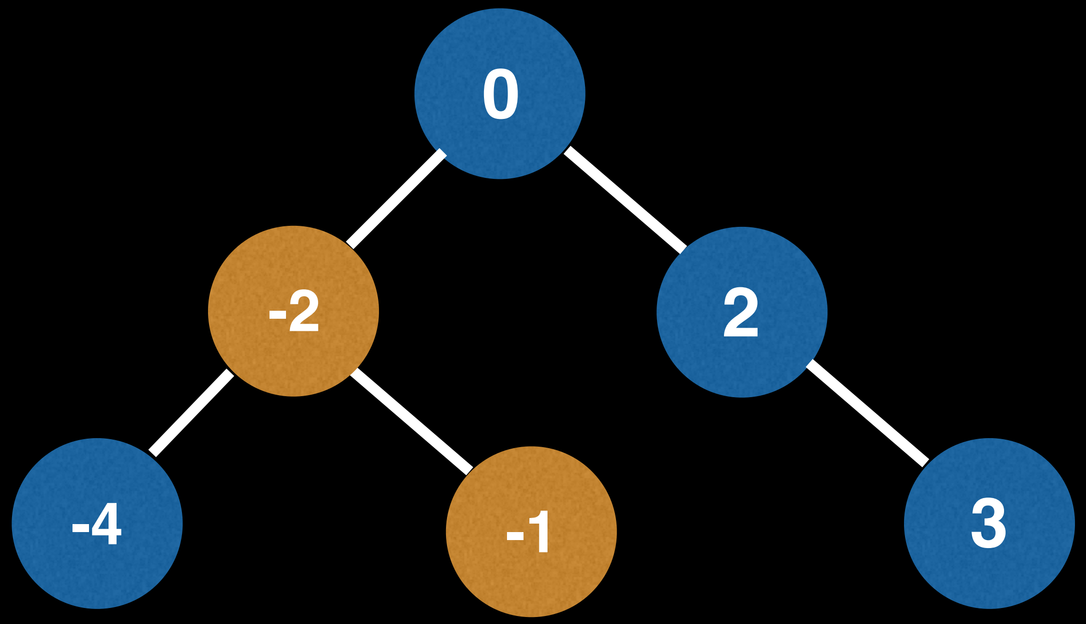


Additional examples

Remove 18

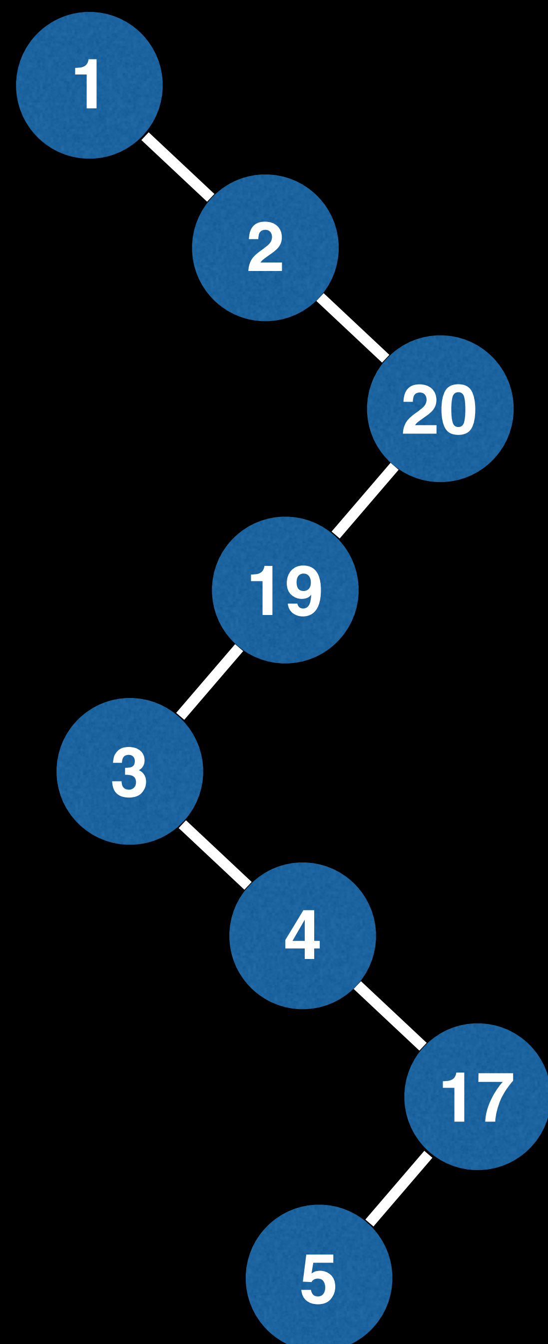


Remove −2

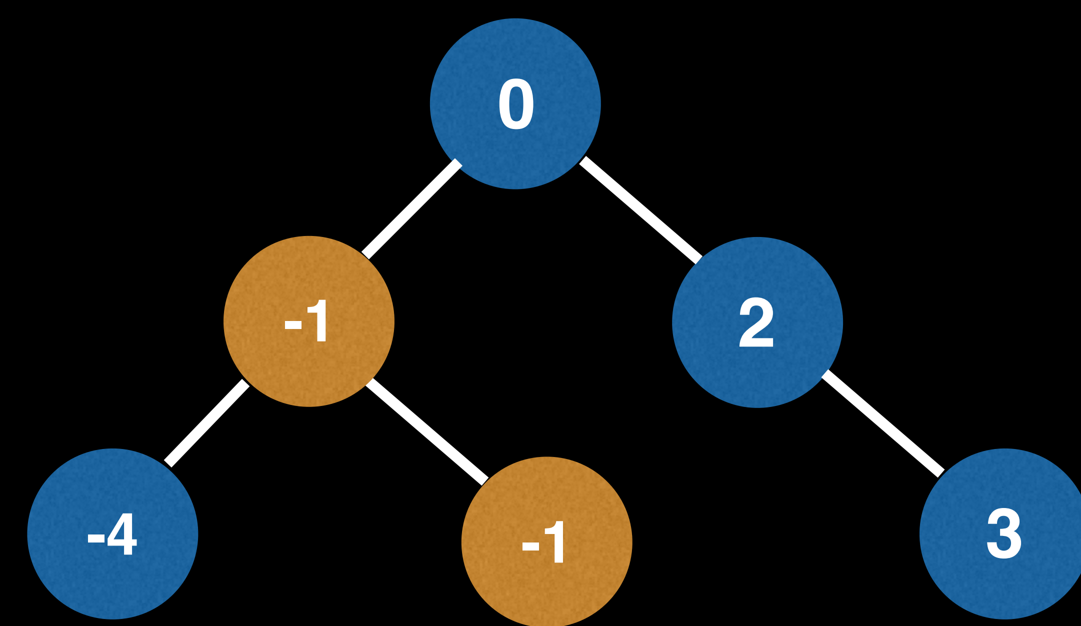


Additional examples

Remove 18

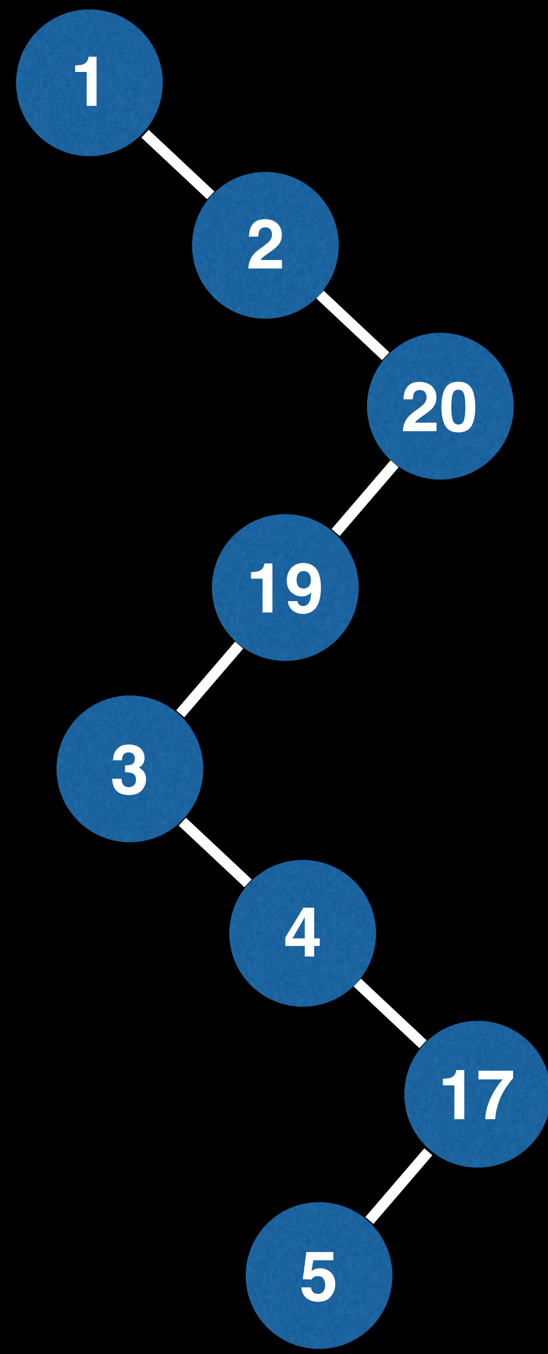


Remove −2

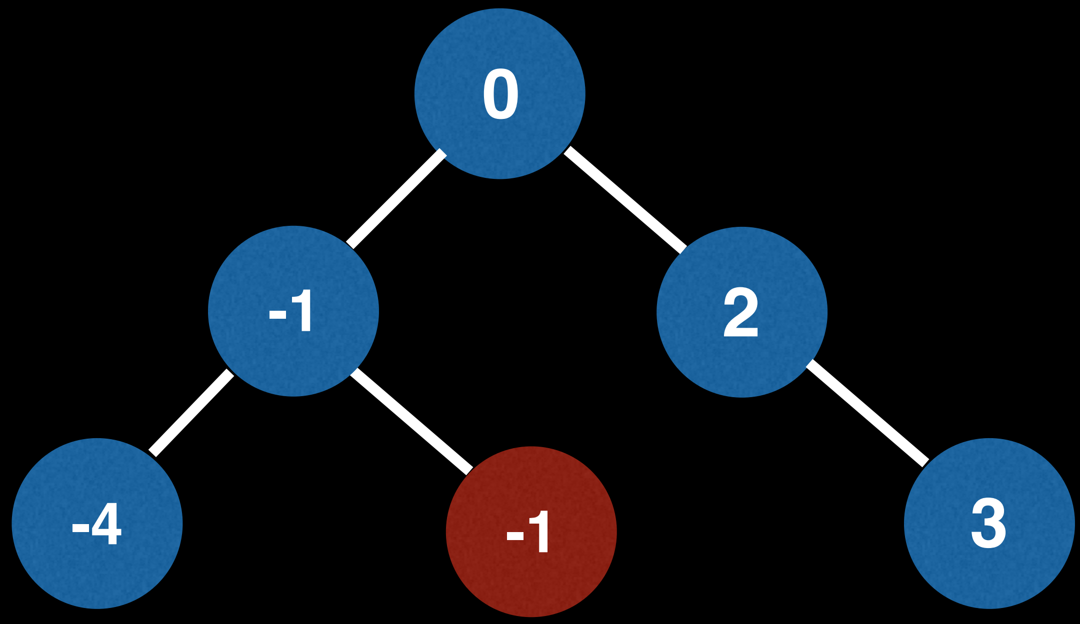


Additional examples

Remove 18

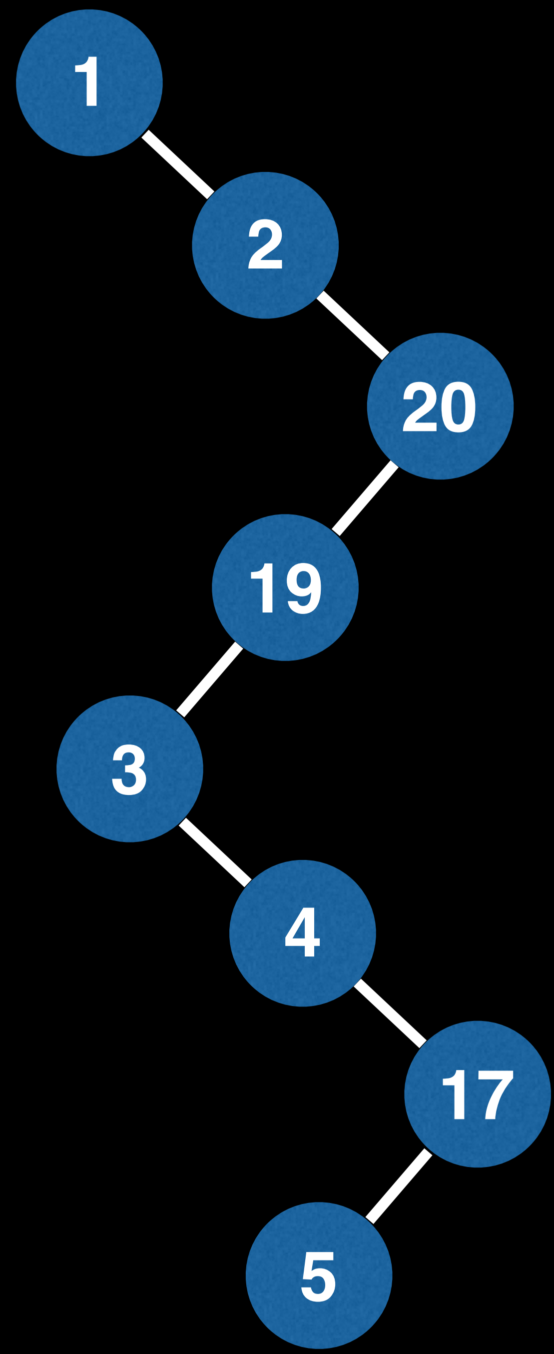


Remove −2

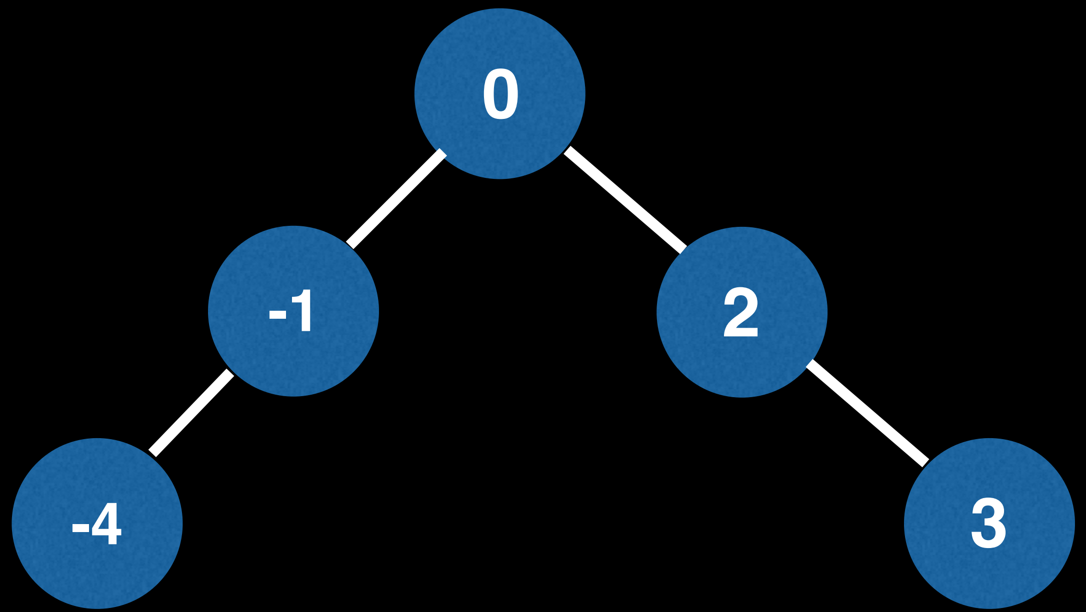


Additional examples

Remove 18



Remove −2



Tree Traversals

(Preorder, Inorder, Postorder & Level order)

Preorder, Inorder & PostOrder

These three types of traversals are naturally defined recursively:

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```

preorder prints before the recursive calls

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```

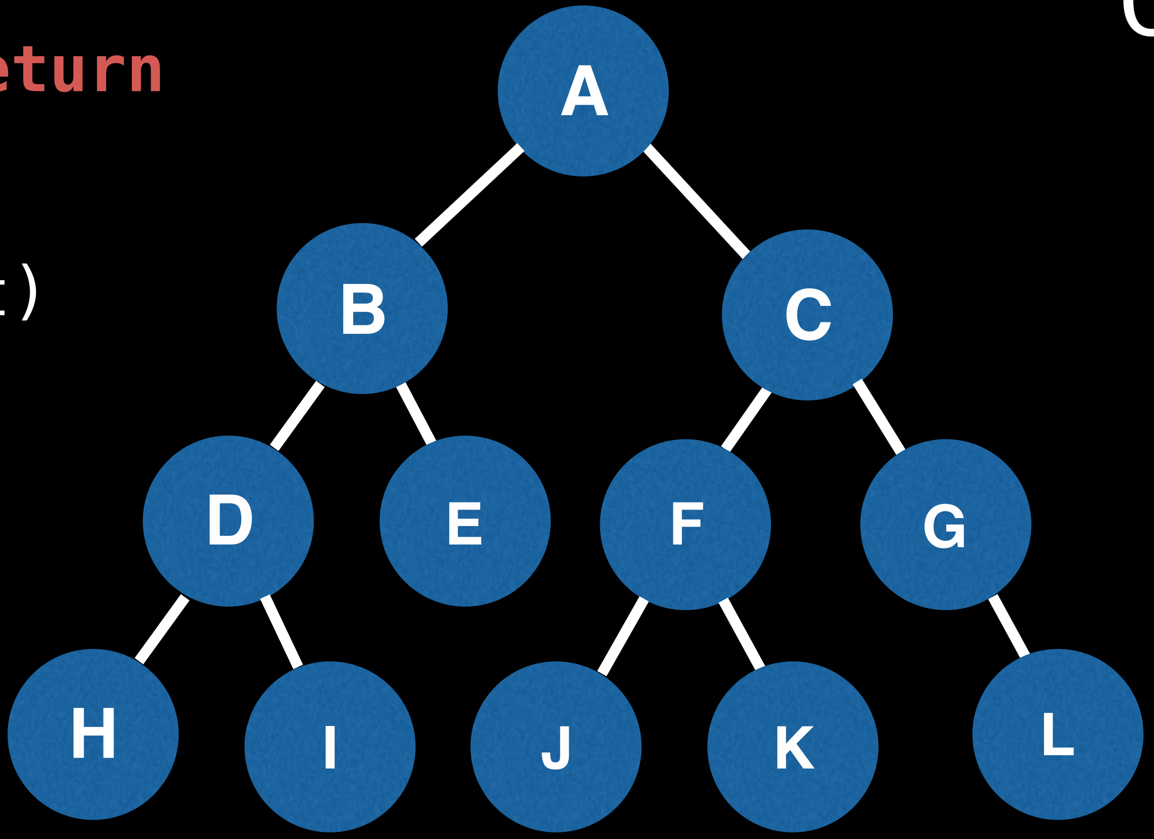
inorder prints between the recursive calls

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```

postorder prints after the recursive calls

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```

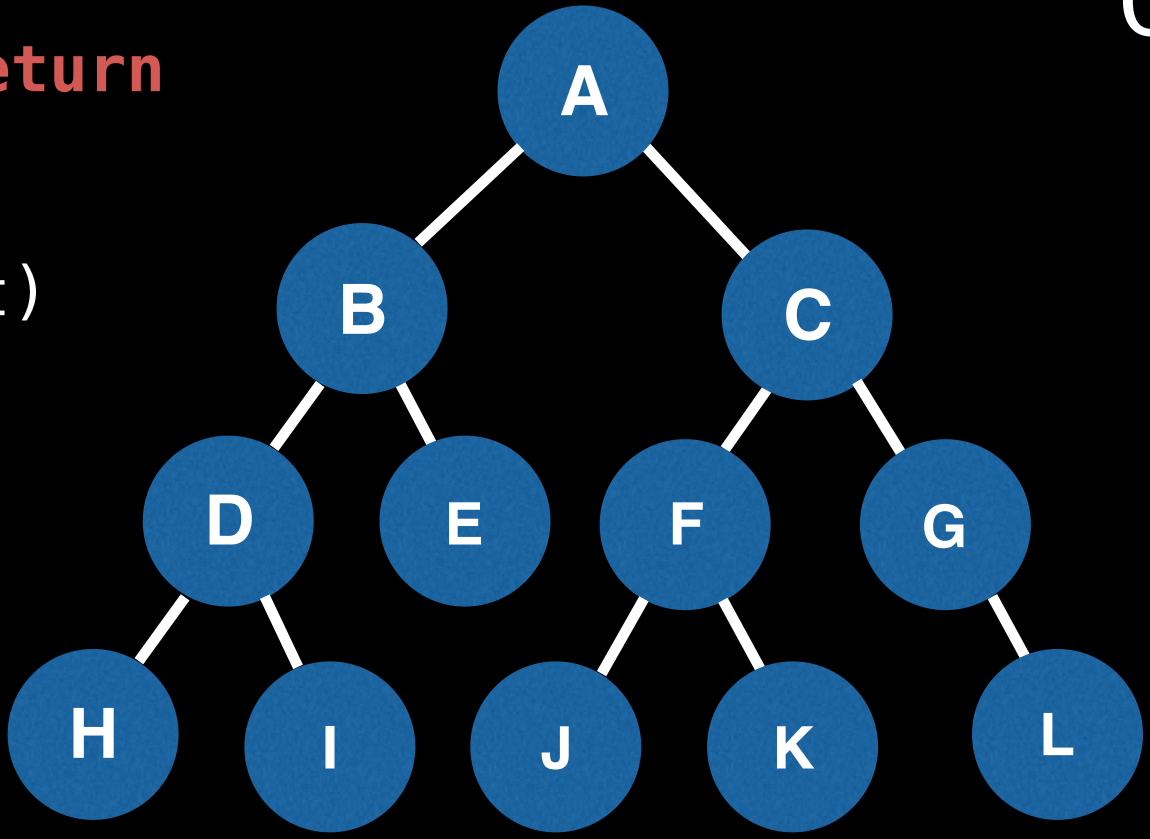


Call Stack:

Print the value of the current node
then traverse the left subtree
followed by the right subtree.

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```

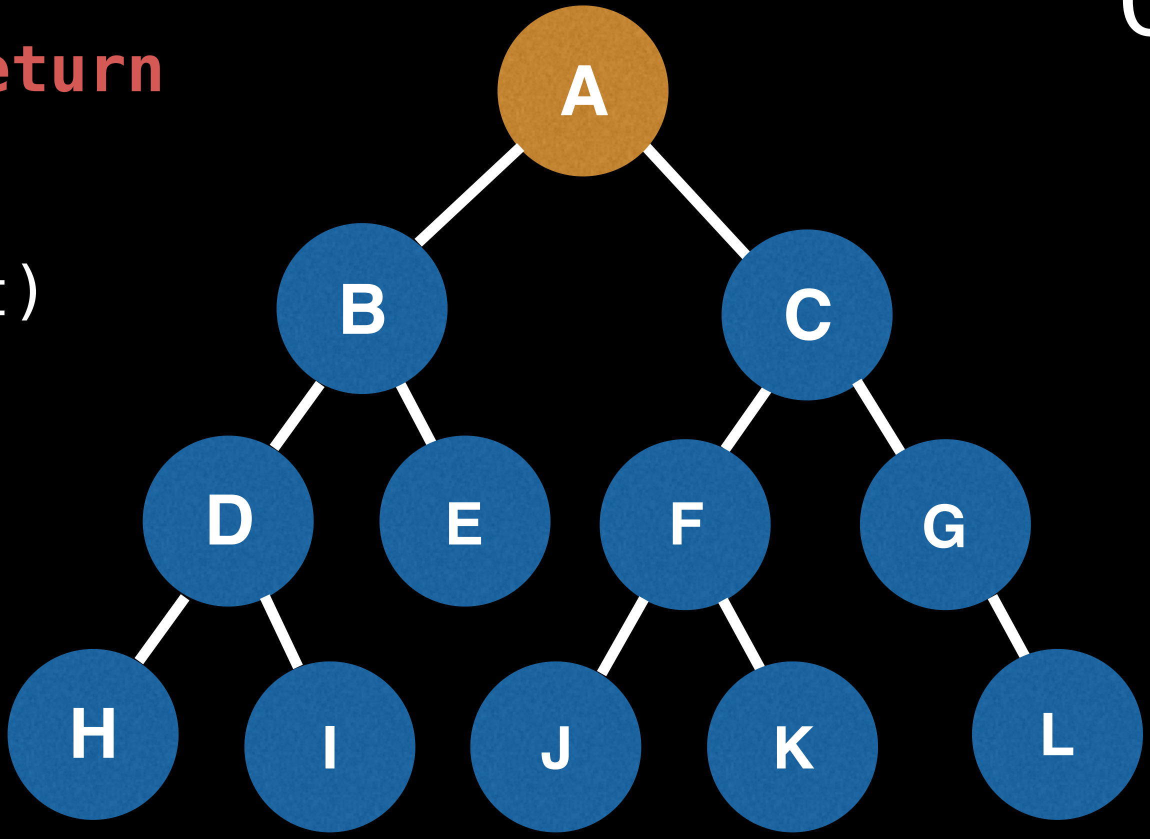


Call Stack:

Order:

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



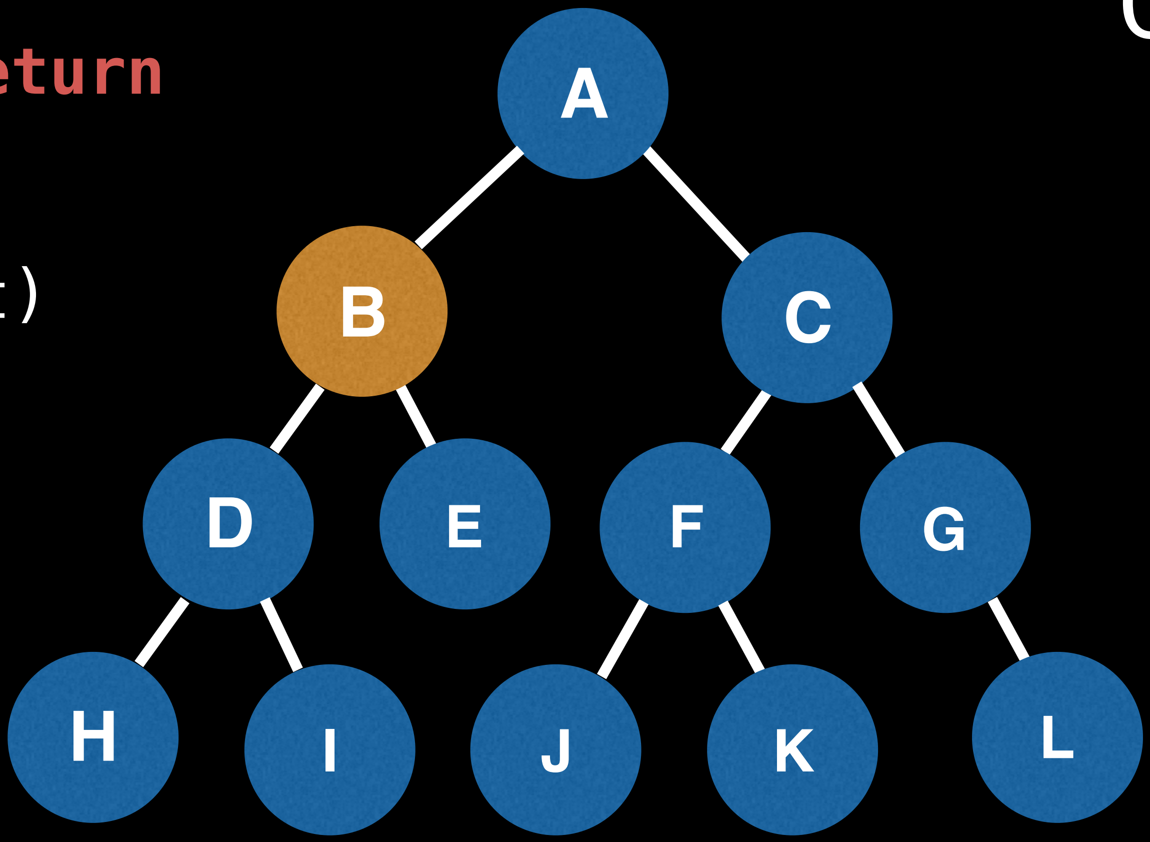
Call Stack:

node A

Order: A

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



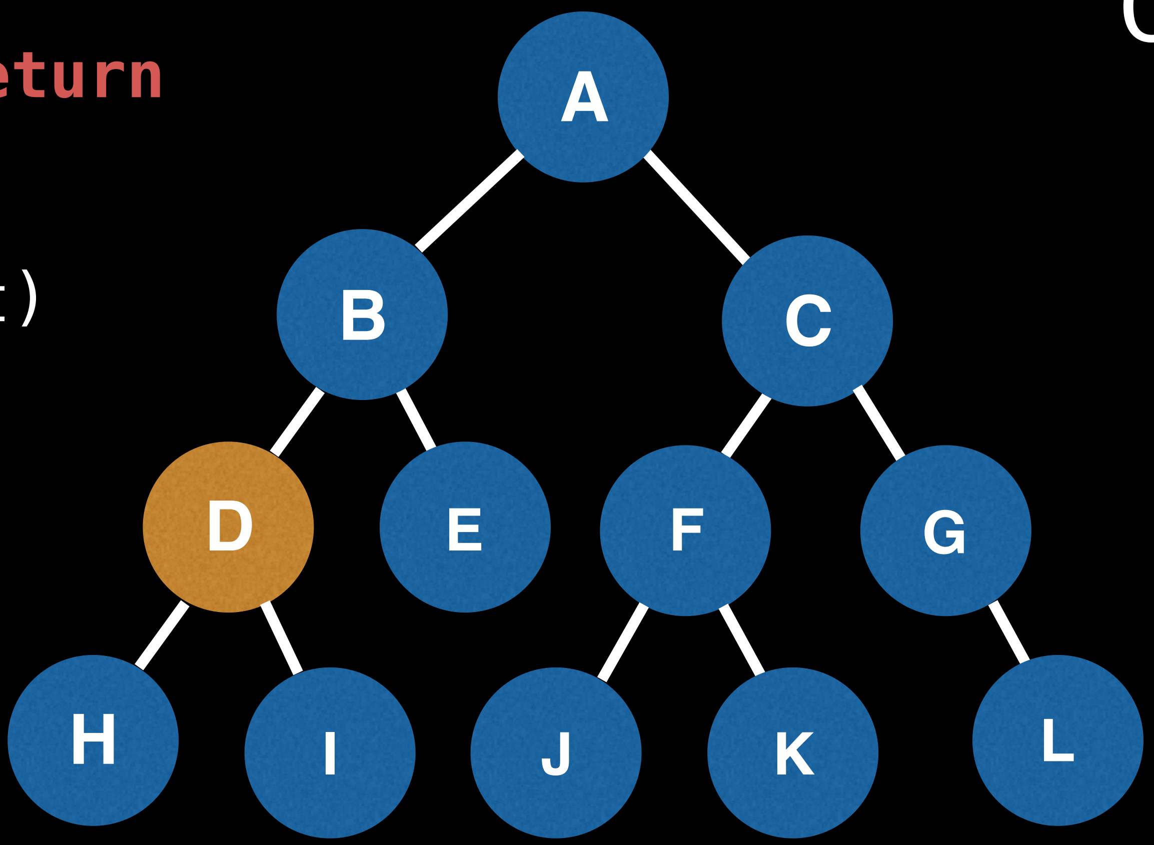
Call Stack:

node A
node B

Order: A,B

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



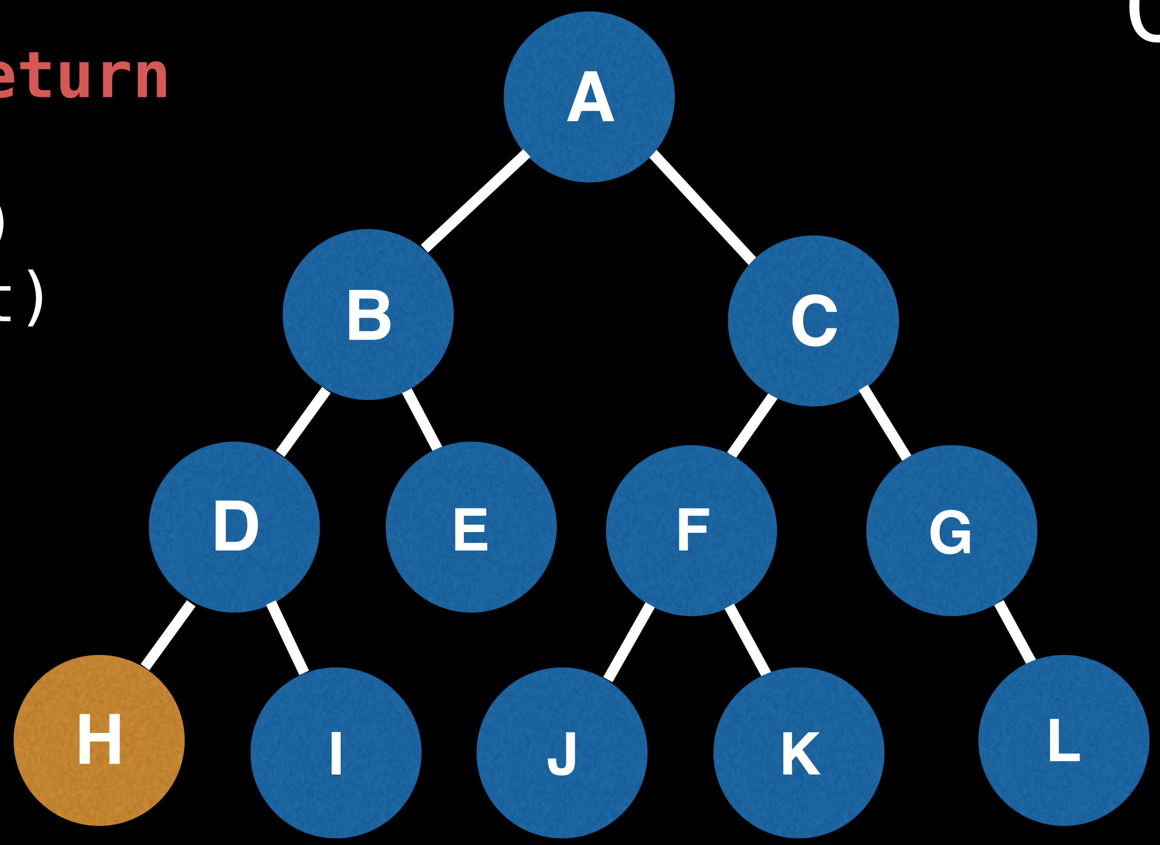
Call Stack:

node A
node B
node D

Order: A,B,D

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



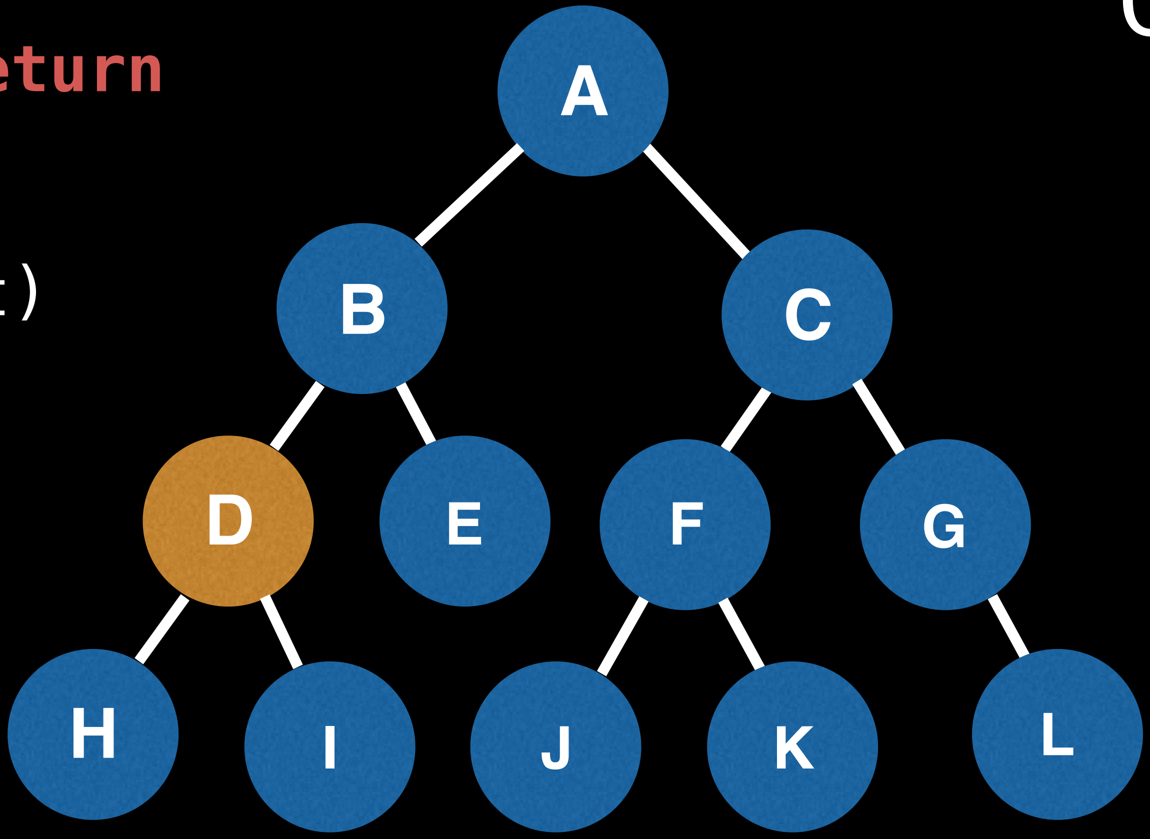
Call Stack:

node A
node B
node D
node H

Order: A,B,D,H

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



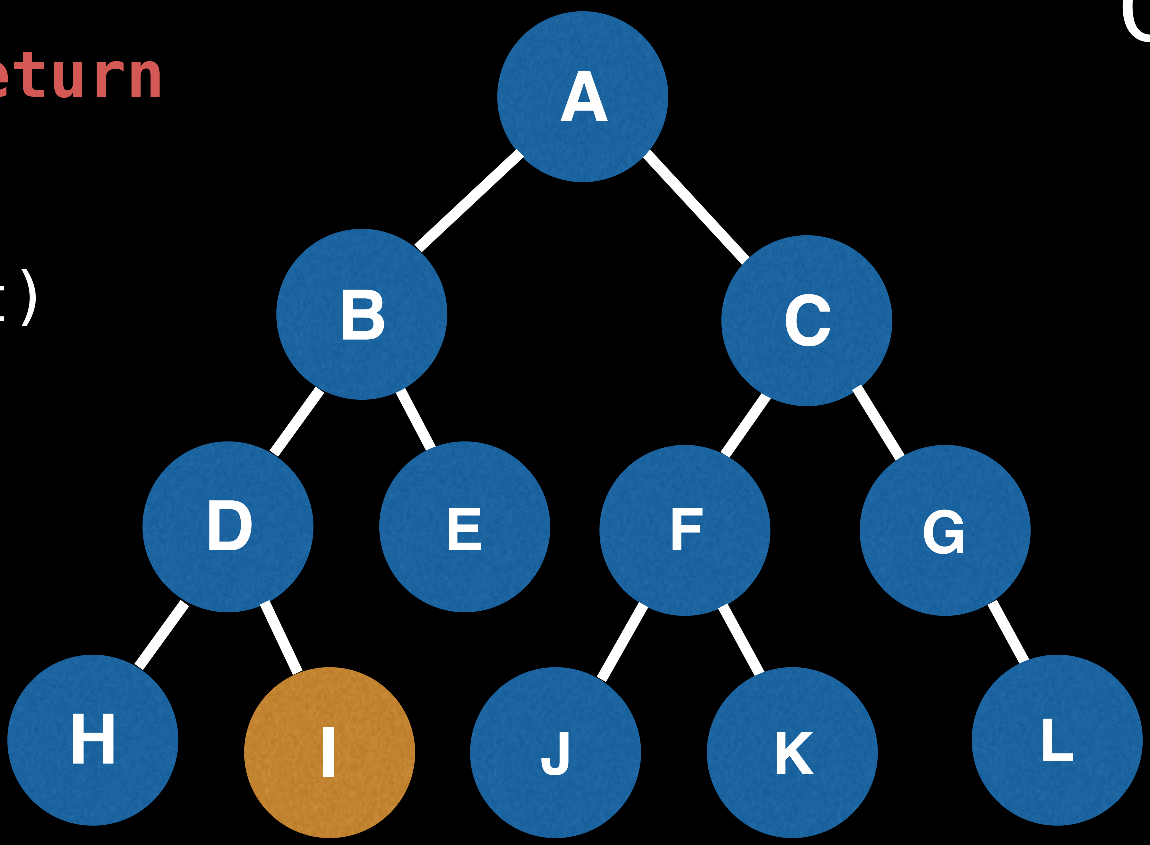
Call Stack:

node A
node B
node D

Order: A,B,D,H

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



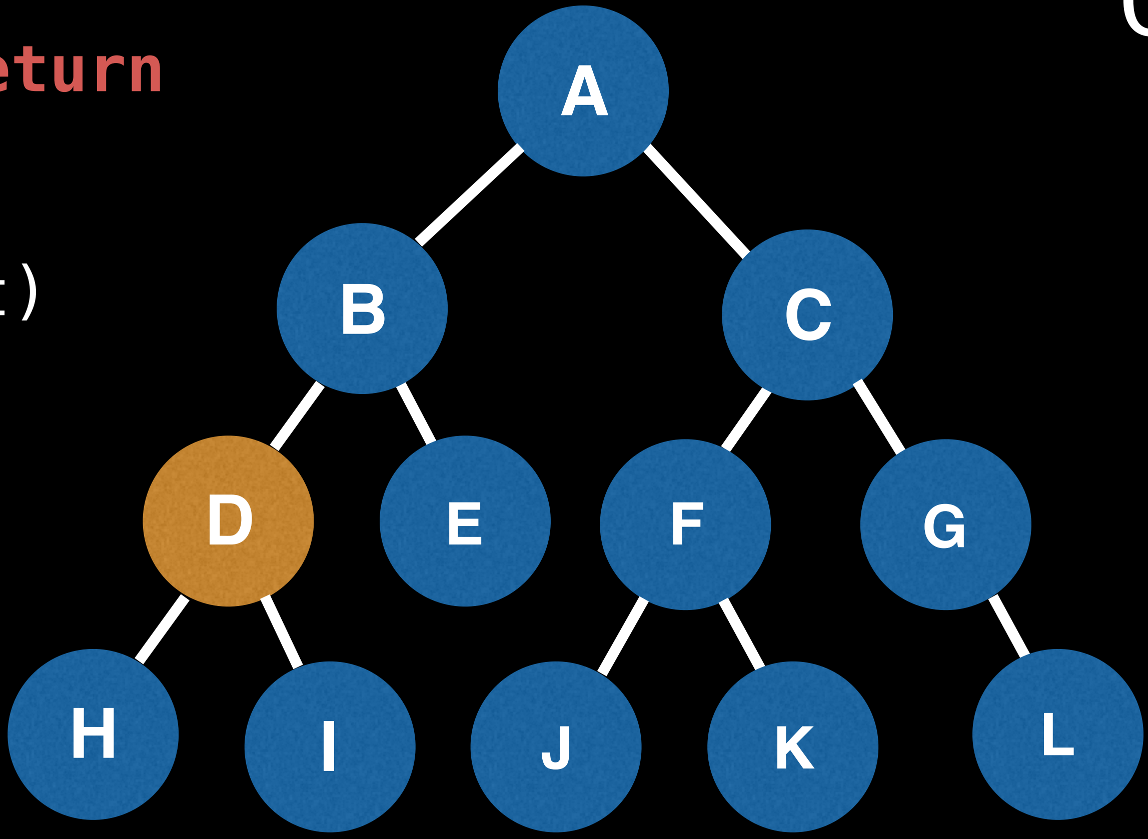
Call Stack:

node A
node B
node D
node I

Order: A,B,D,H,I

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



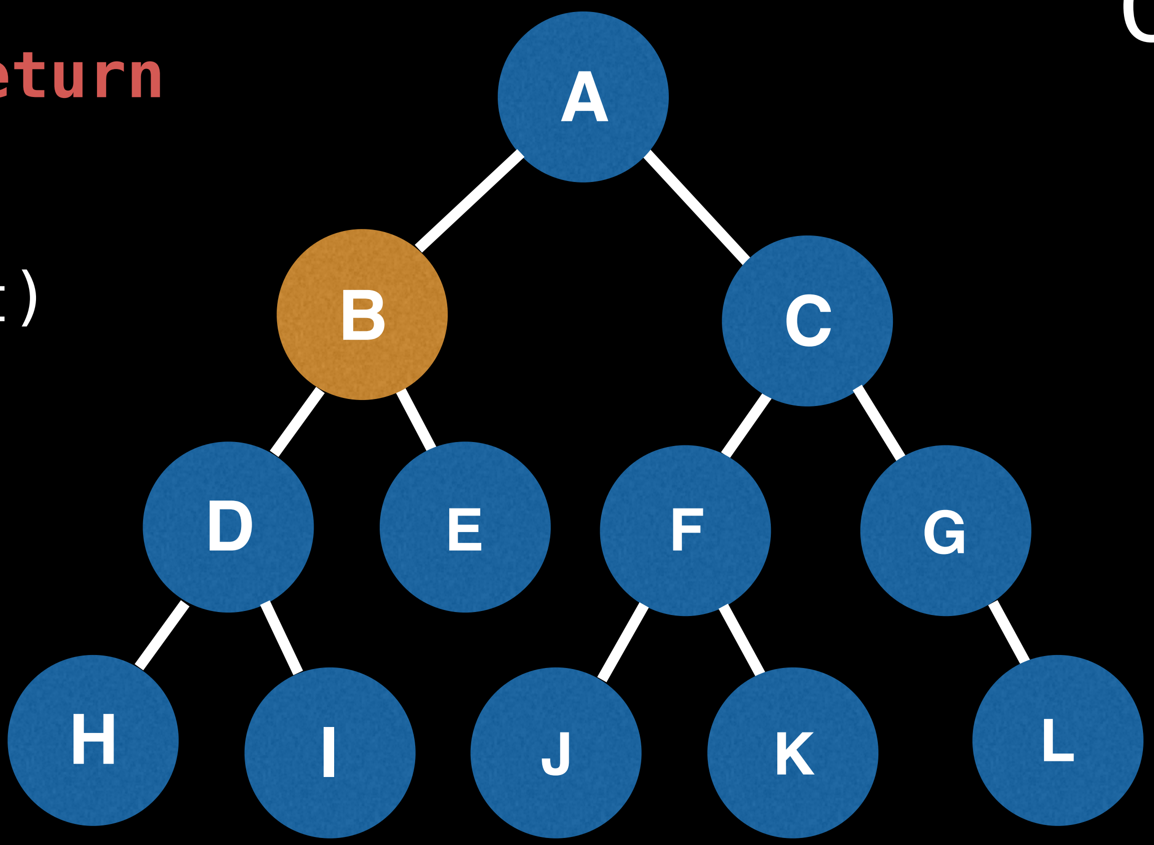
Call Stack:

node A
node B
node D

Order: A,B,D,H,I

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



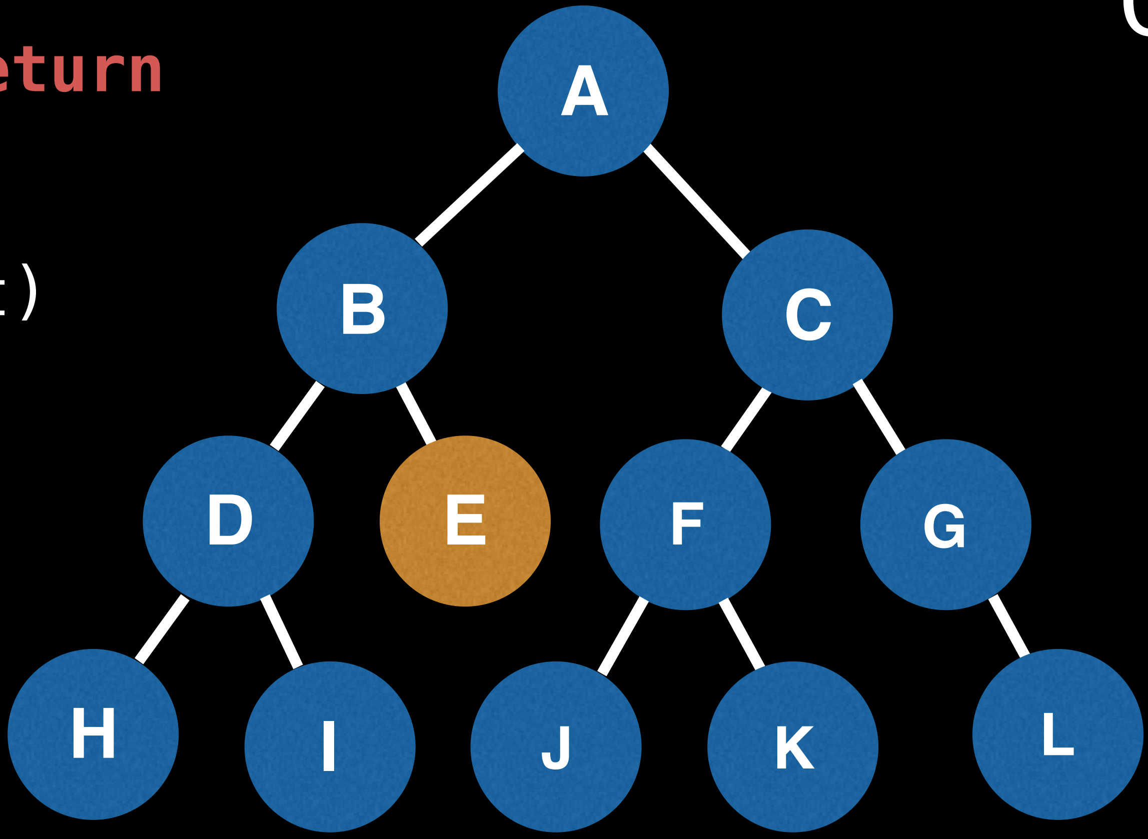
Call Stack:

node A
node B

Order: A,B,D,H,I

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



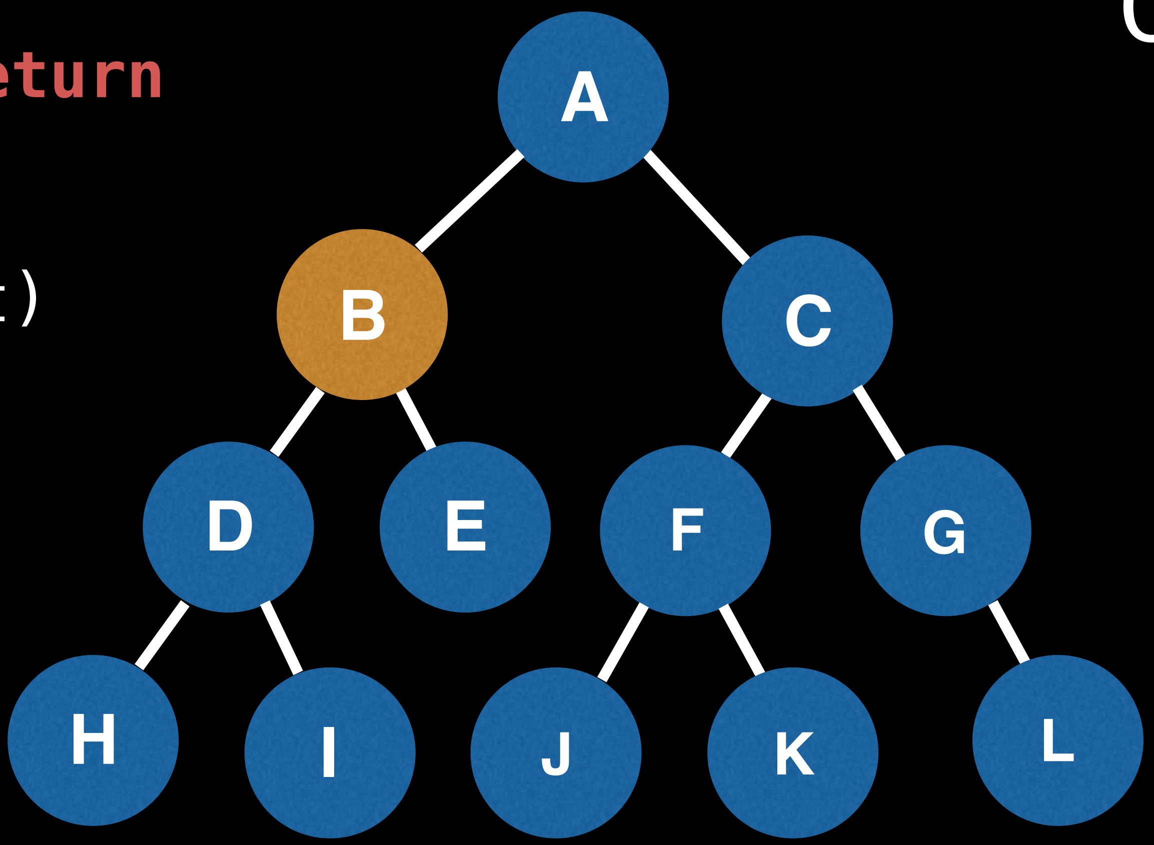
Call Stack:

node A
node B
node E

Order: A,B,D,H,I,E

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



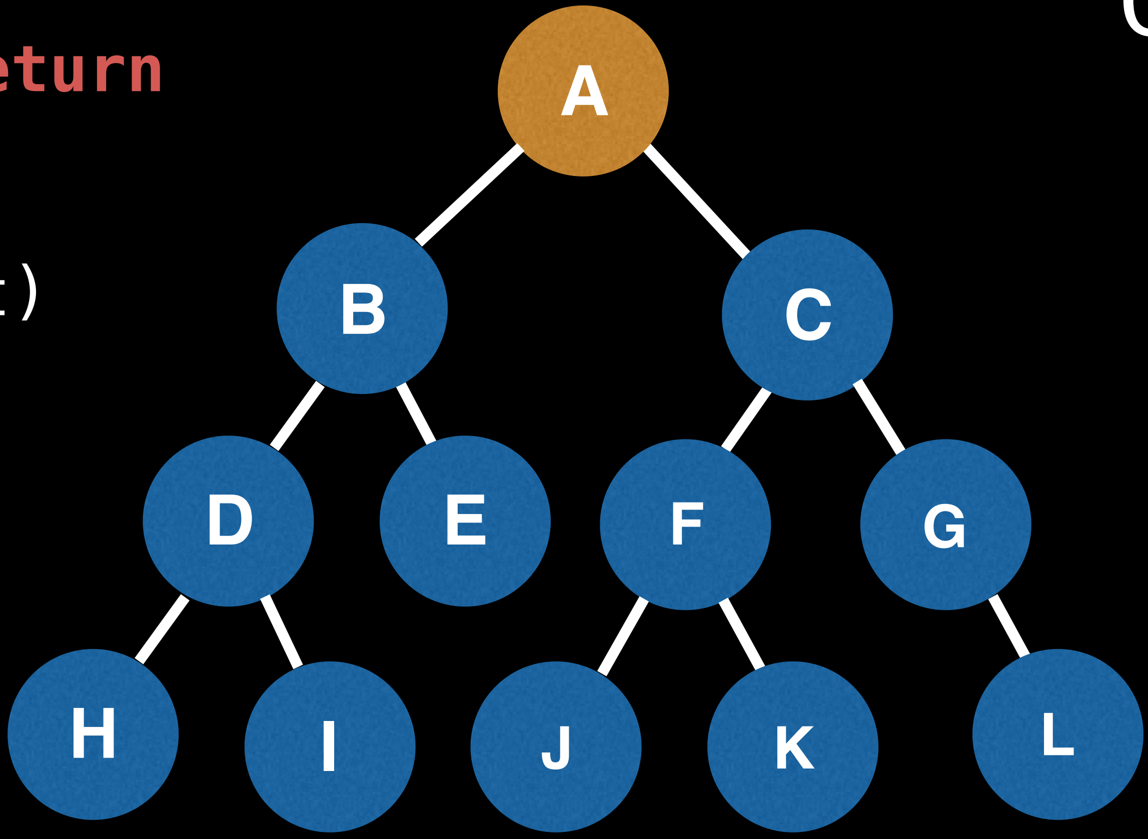
Call Stack:

node A
node B

Order: A,B,D,H,I,E

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



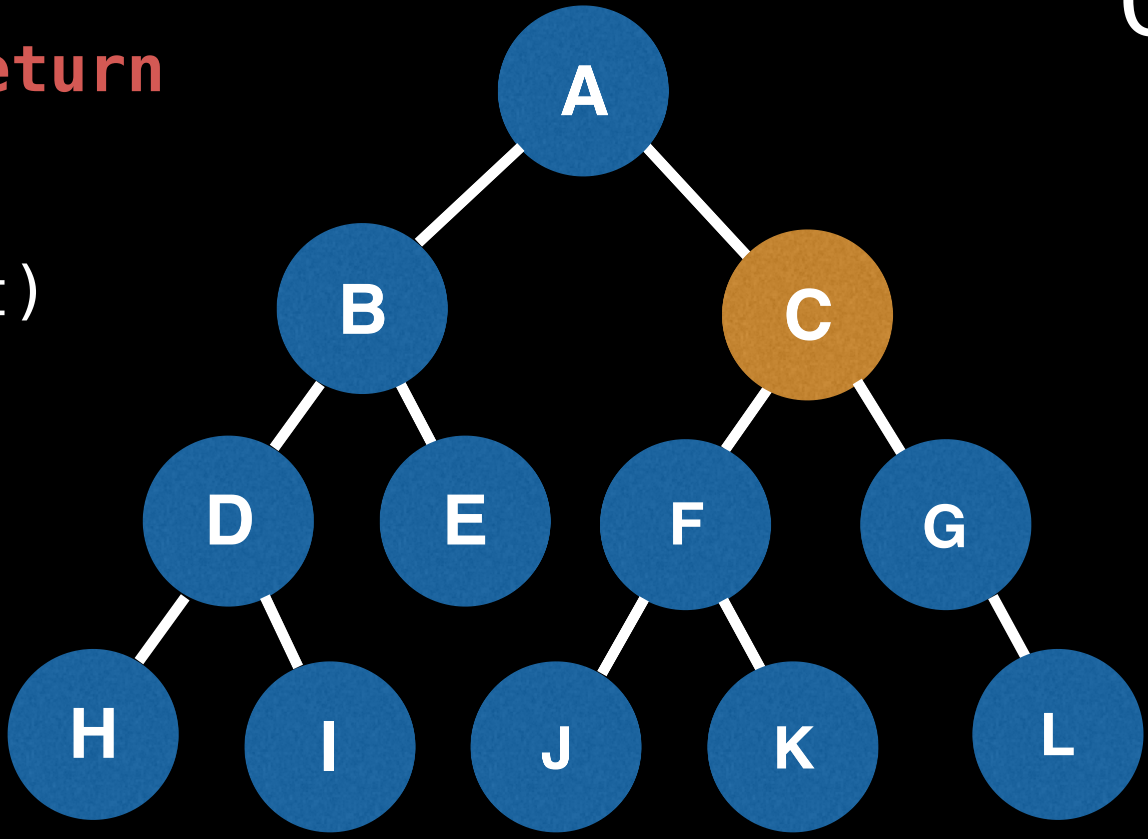
Call Stack:

node A

Order: A,B,D,H,I,E

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



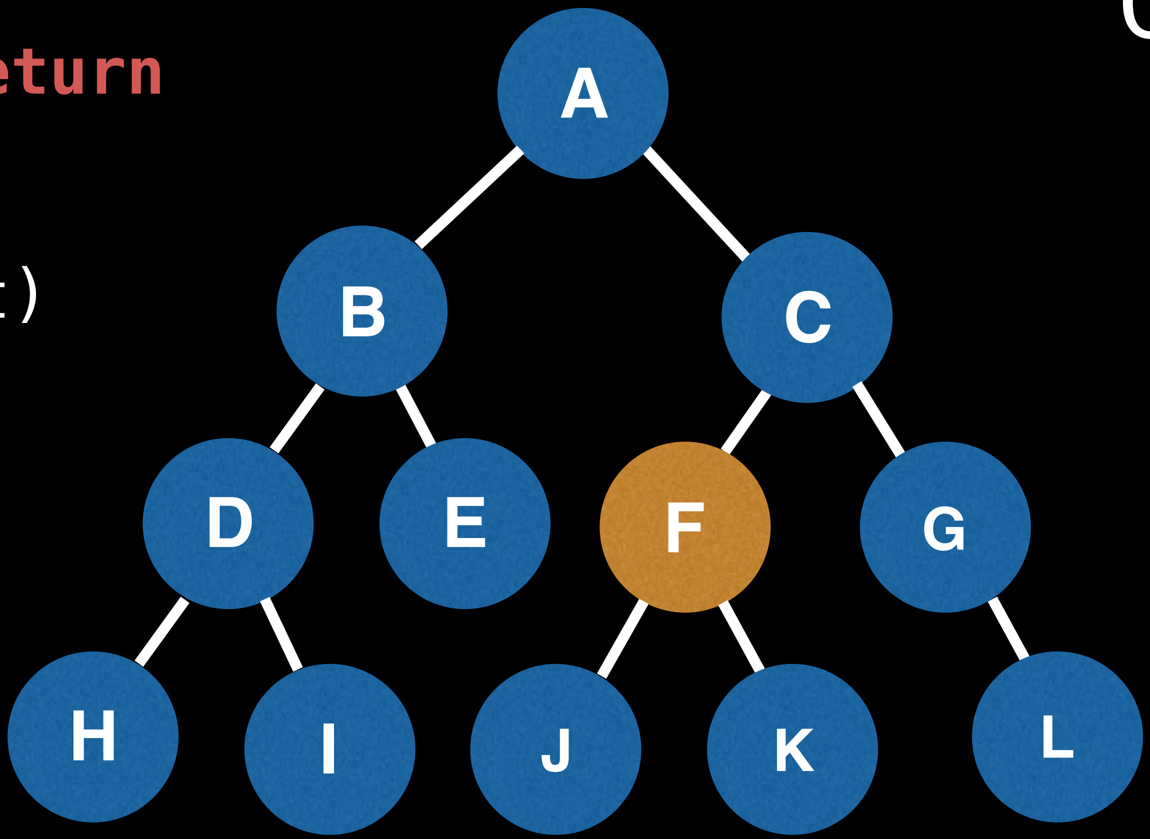
Call Stack:

node A
node C

Order: A,B,D,H,I,E,C

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



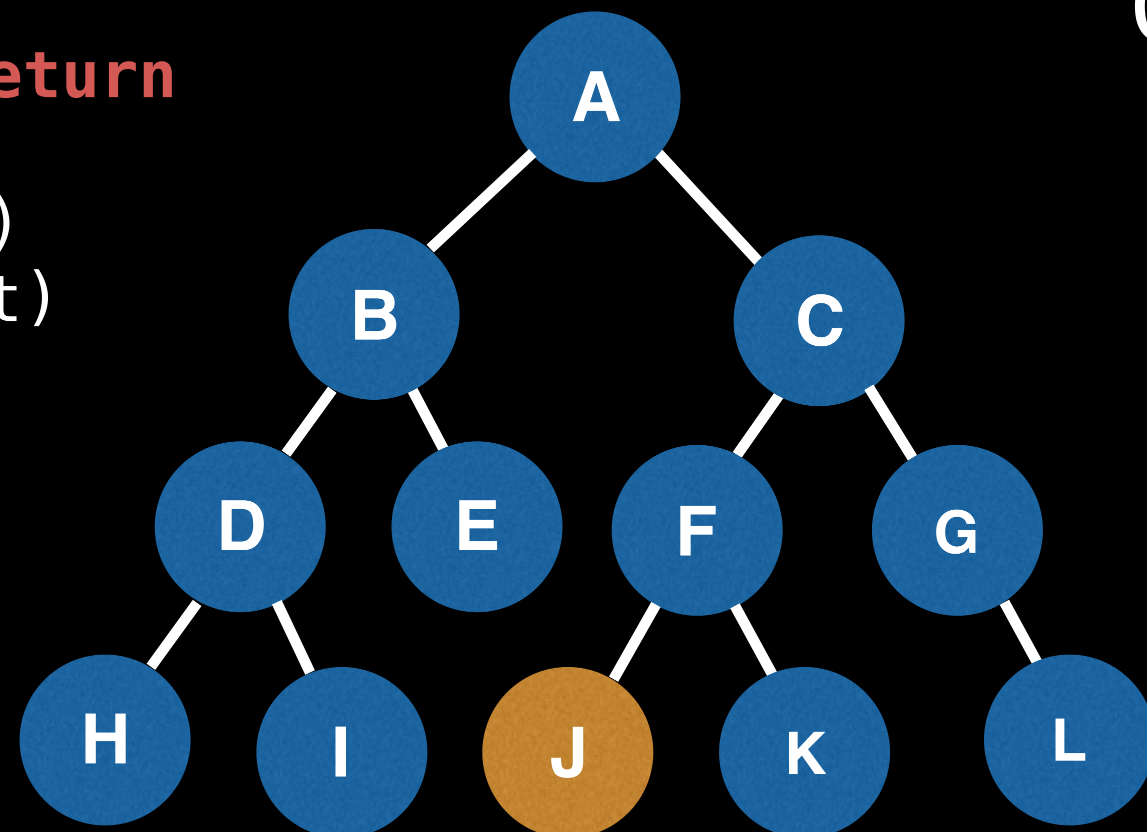
Call Stack:

node A
node C
node F

Order: A,B,D,H,I,E,C,F

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



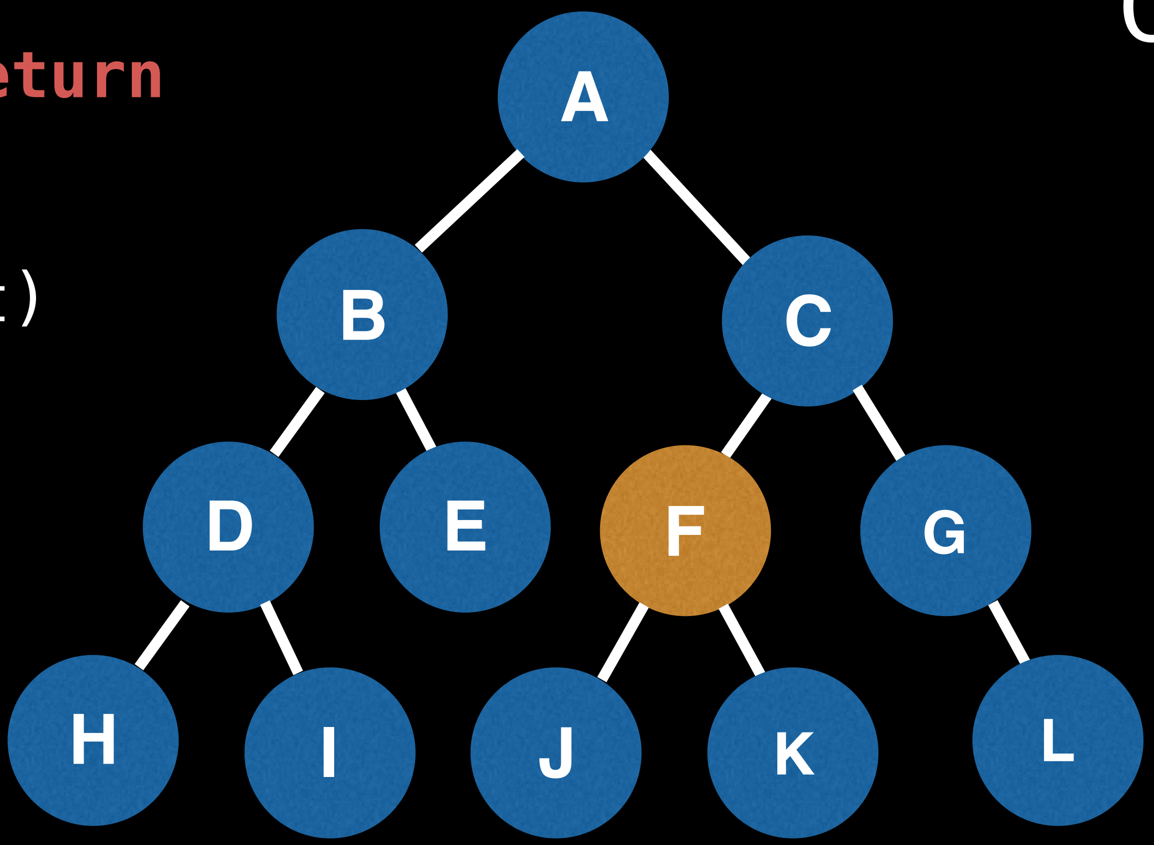
Call Stack:

node A
node C
node F
node J

Order: A,B,D,H,I,E,C,F,J

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



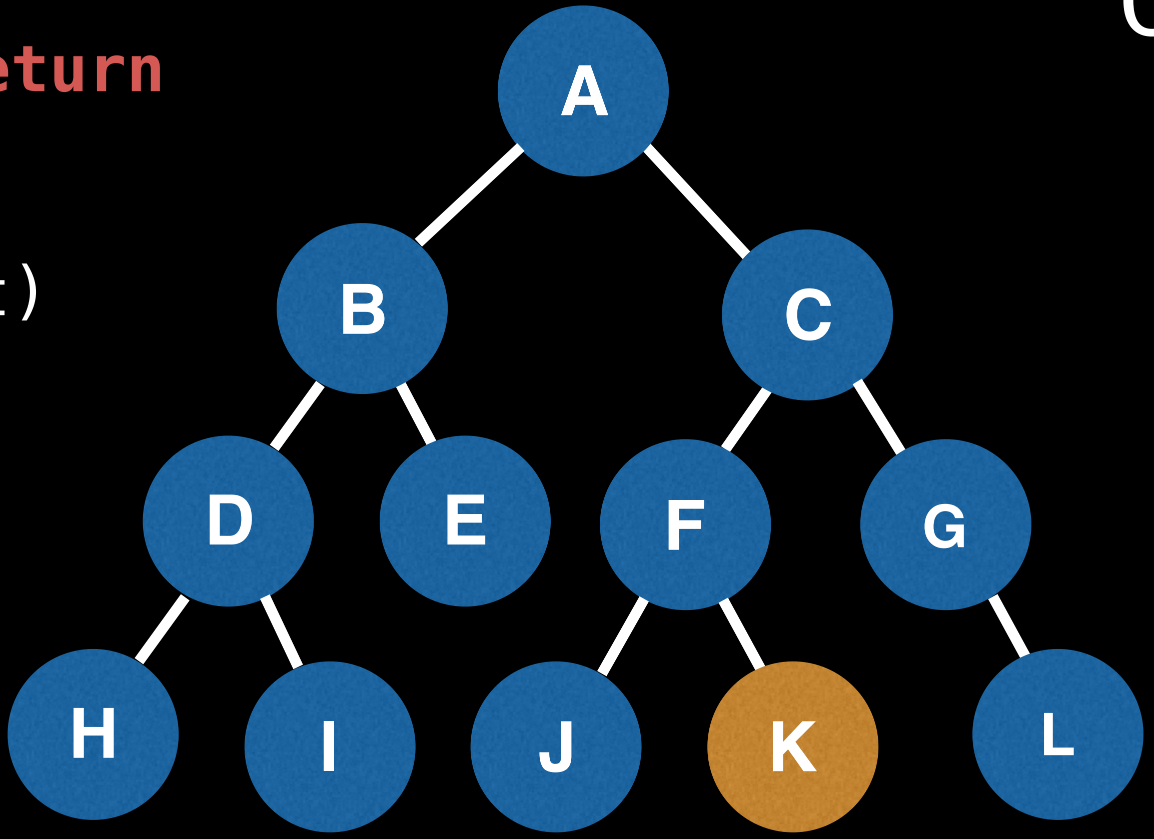
Call Stack:

node A
node C
node F

Order: A,B,D,H,I,E,C,F,J

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



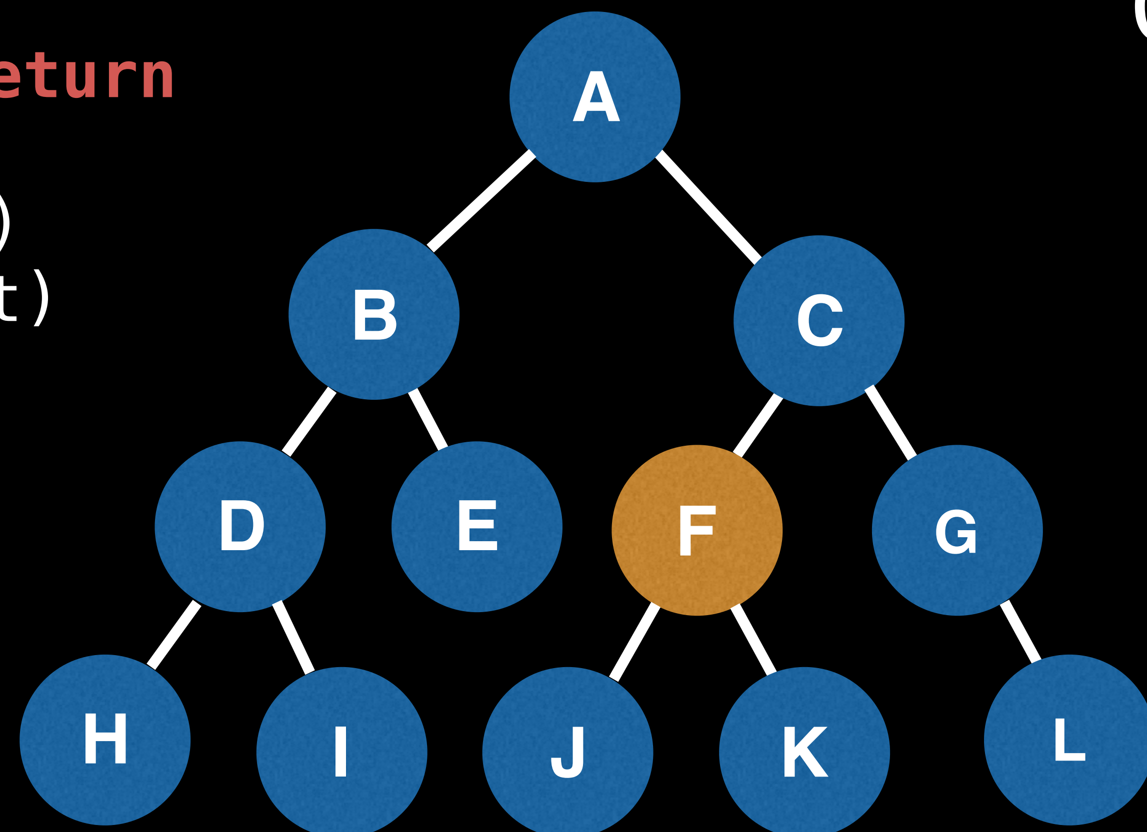
Call Stack:

node A
node C
node F
node K

Order: A,B,D,H,I,E,C,F,J,K

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



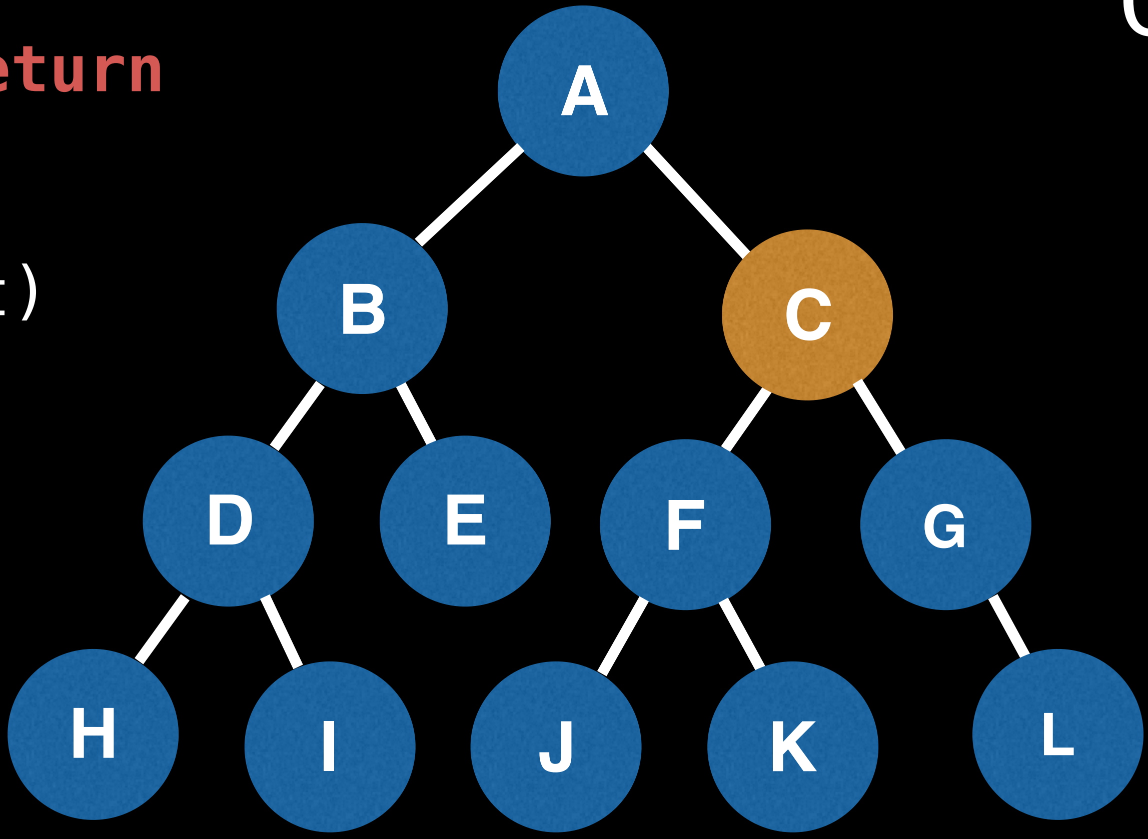
Call Stack:

node A
node C
node F

Order: A,B,D,H,I,E,C,F,J,K

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



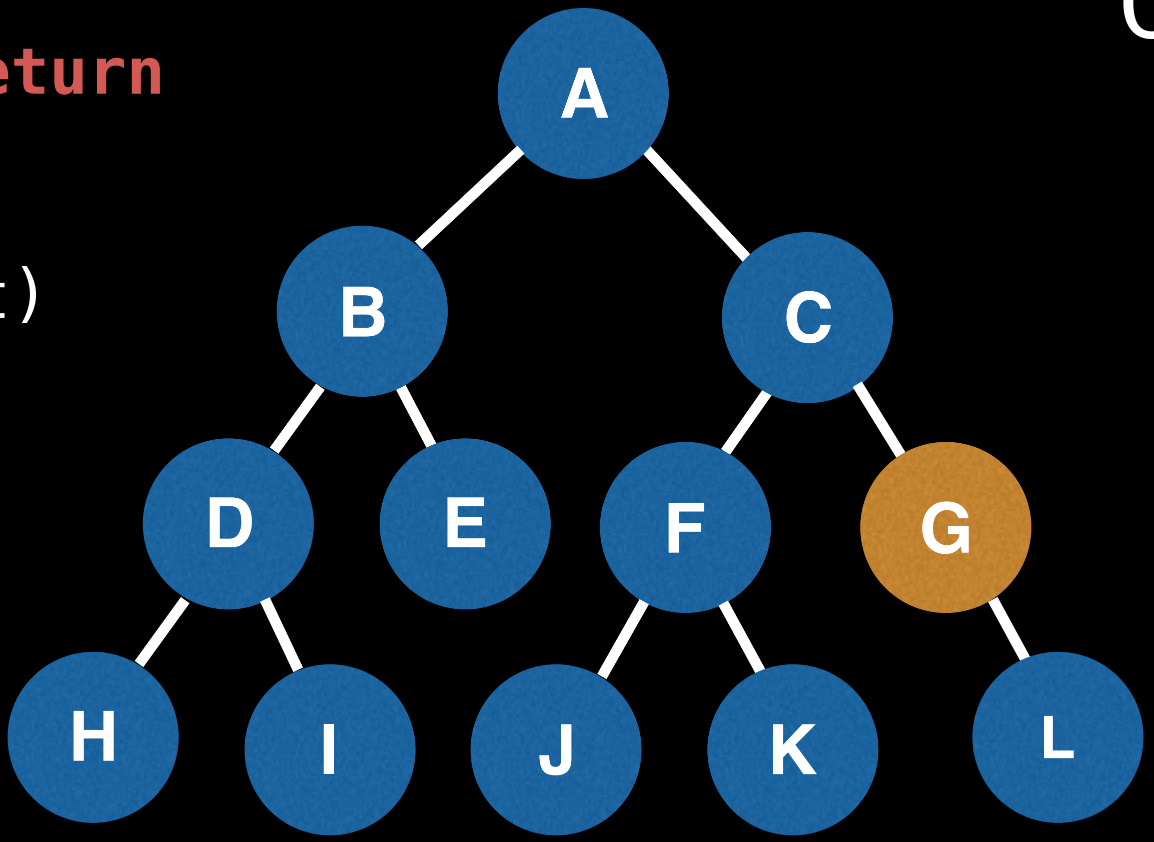
Call Stack:

node A
node C

Order: A,B,D,H,I,E,C,F,J,K

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



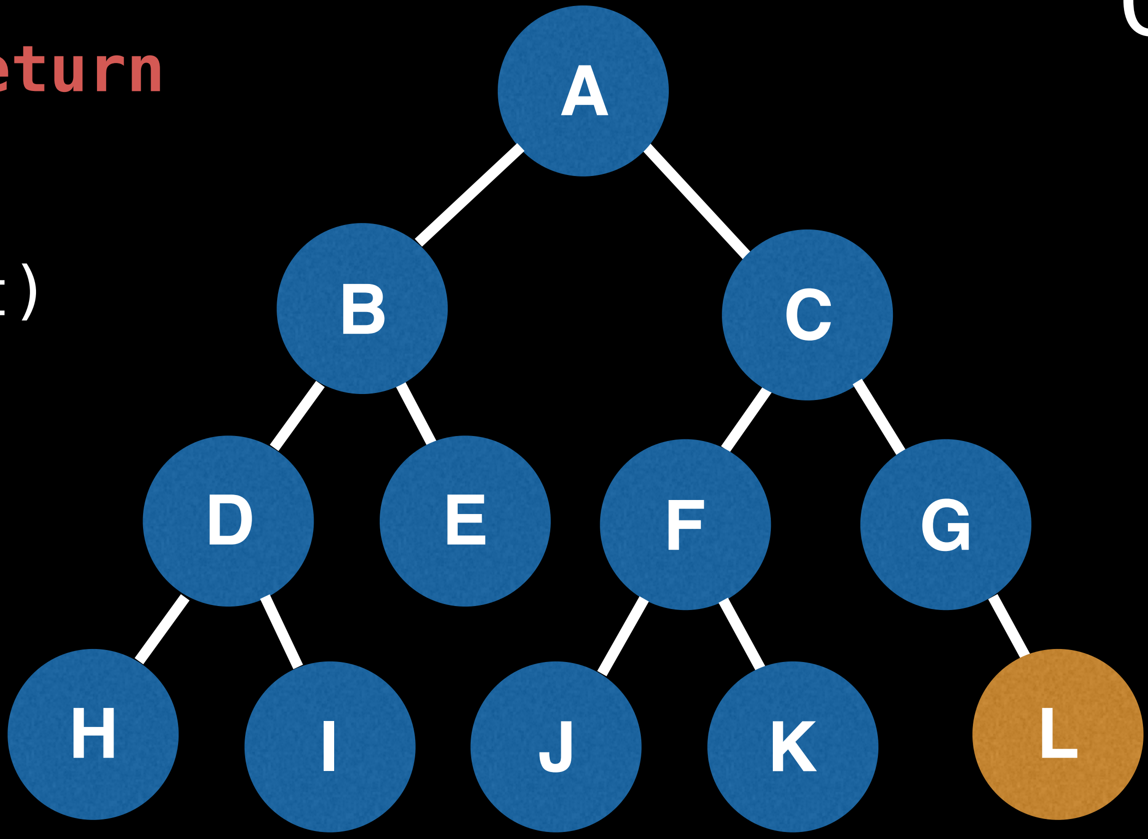
Call Stack:

node A
node C
node G

Order: A,B,D,H,I,E,C,F,J,K,G

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



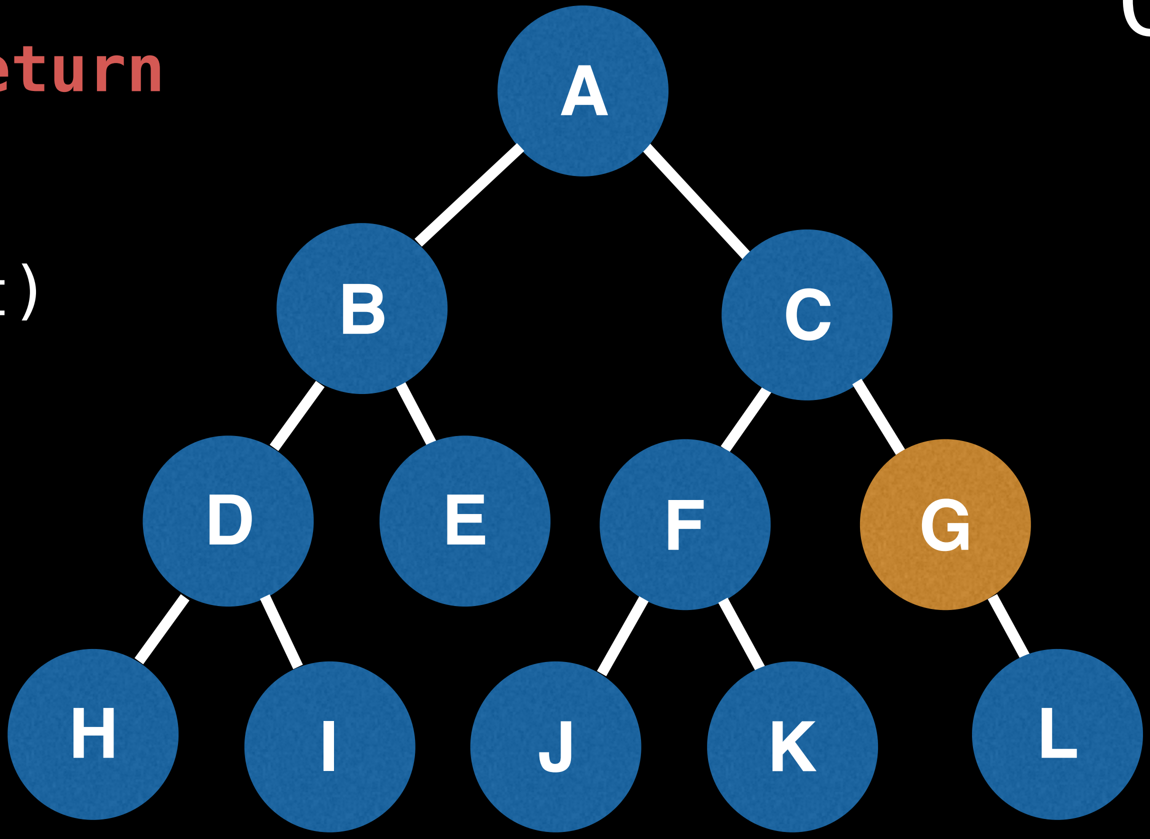
Call Stack:

node A
node C
node G
node L

Order: A,B,D,H,I,E,C,F,J,K,G,L

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



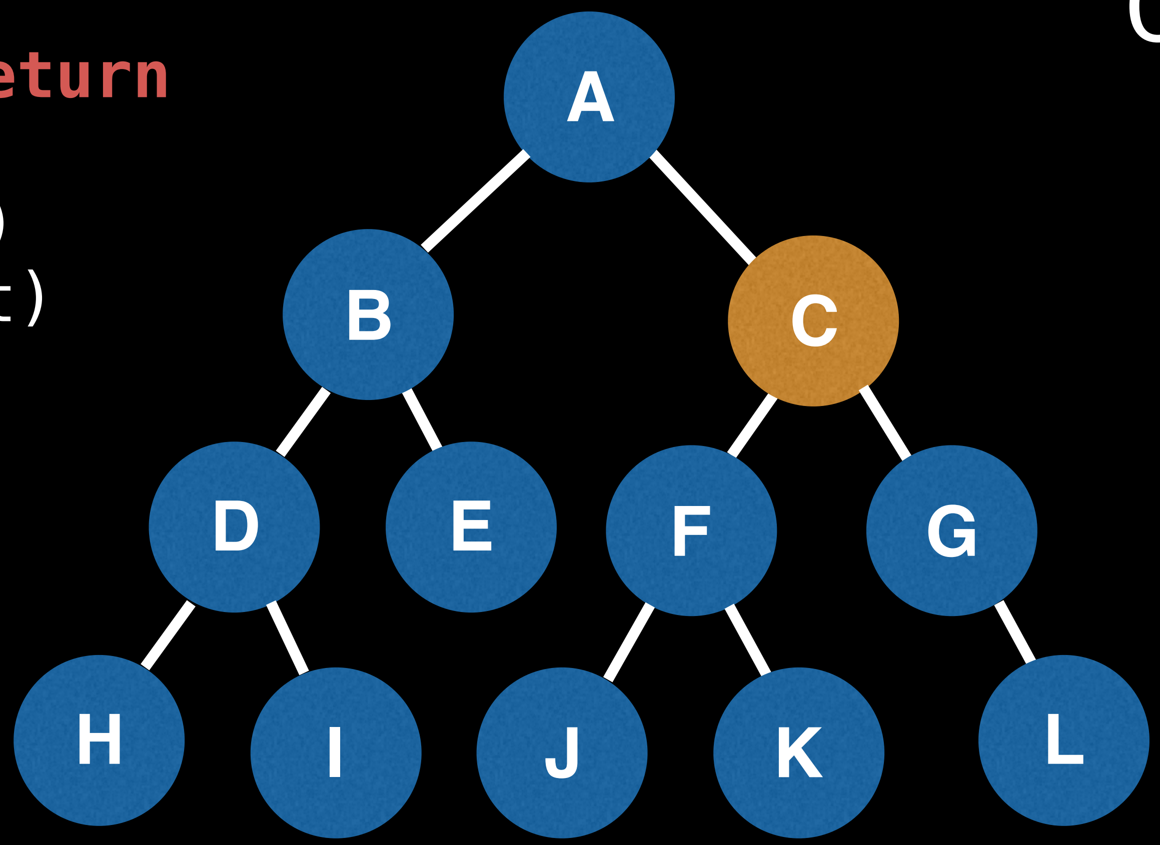
Call Stack:

node A
node C
node G

Order: A,B,D,H,I,E,C,F,J,K,G,L

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



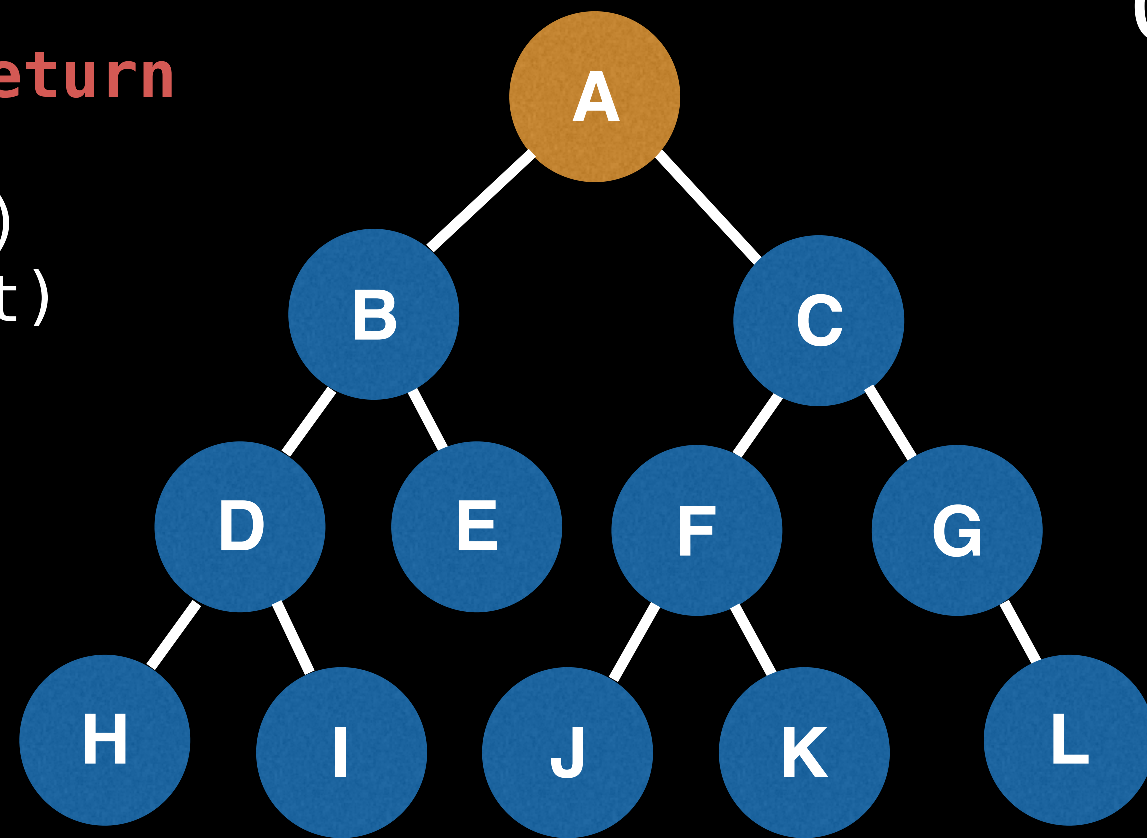
Call Stack:

node A
node C

Order: A,B,D,H,I,E,C,F,J,K,G,L

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```



Call Stack:

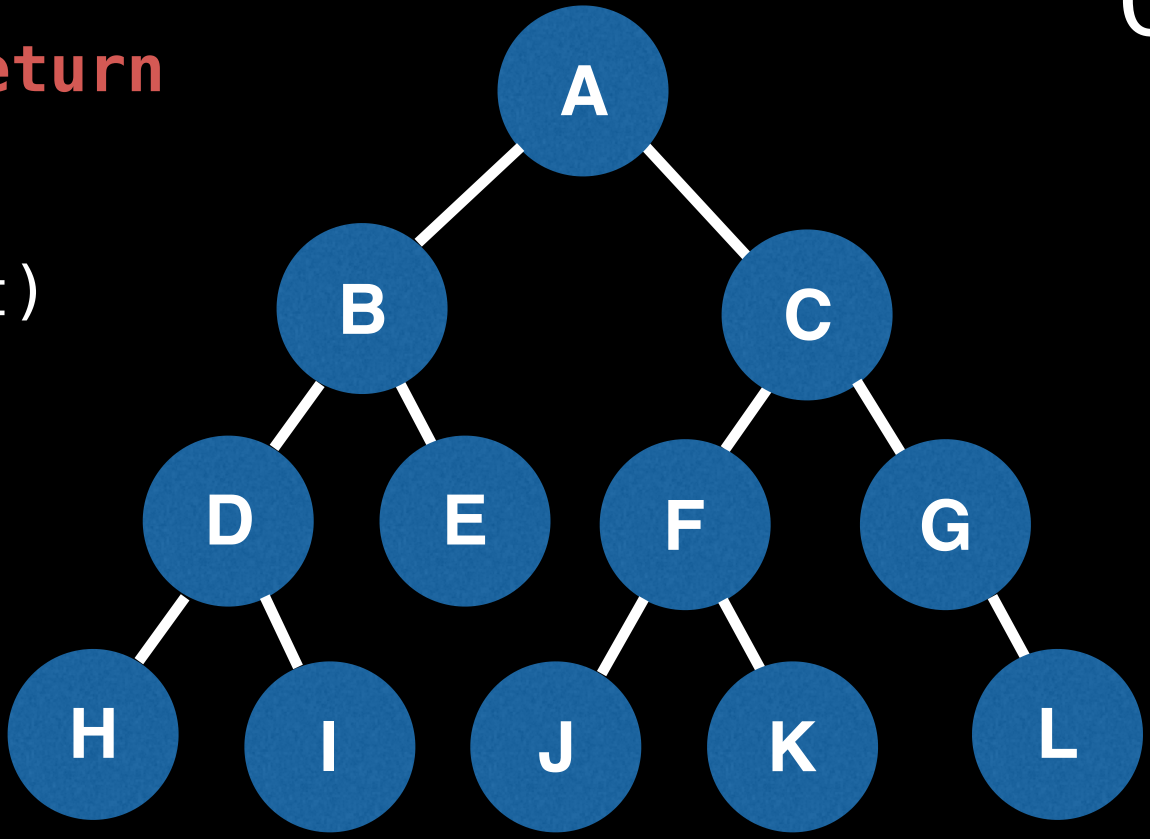
node A

Order: A,B,D,H,I,E,C,F,J,K,G,L

Preorder Traversal

```
preorder(node):
  if node == null: return
  print(node.value)
  preorder(node.left)
  preorder(node.right)
```

Call Stack:

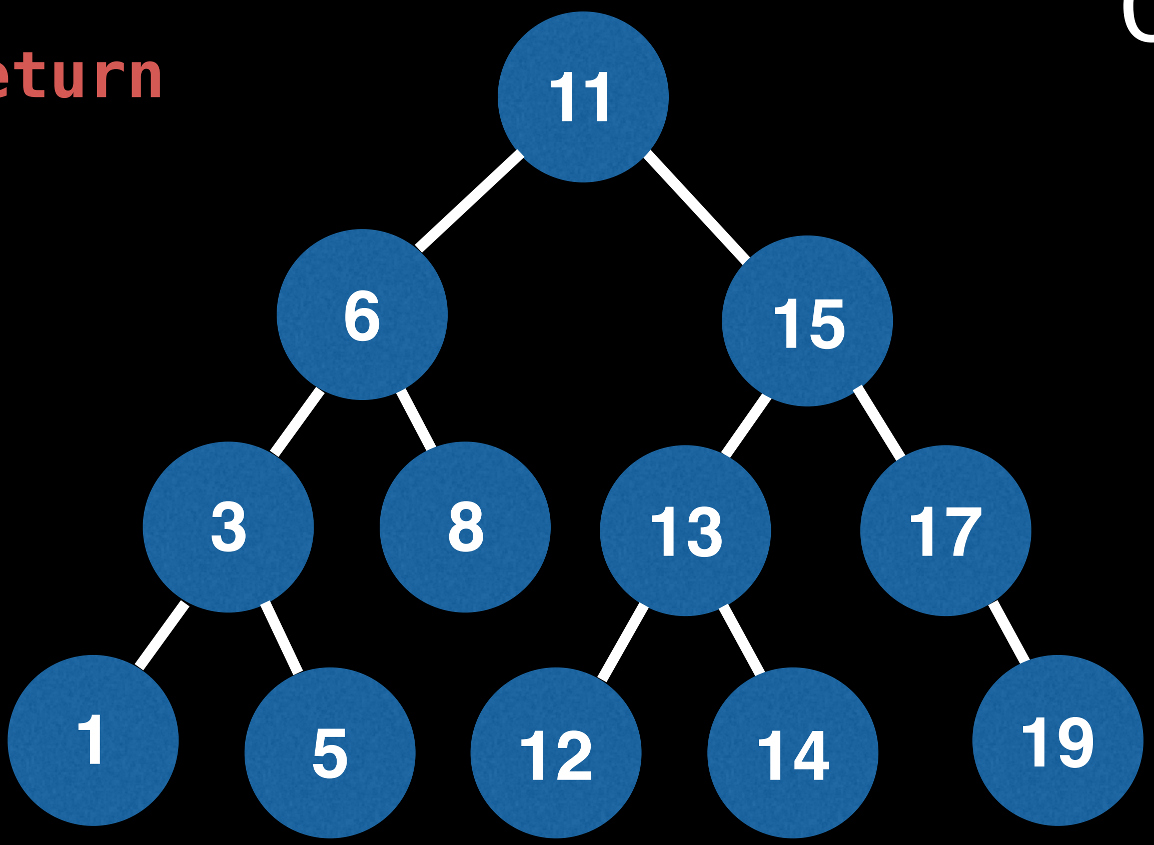


Order: A,B,D,H,I,E,C,F,J,K,G,L

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```

Call Stack:

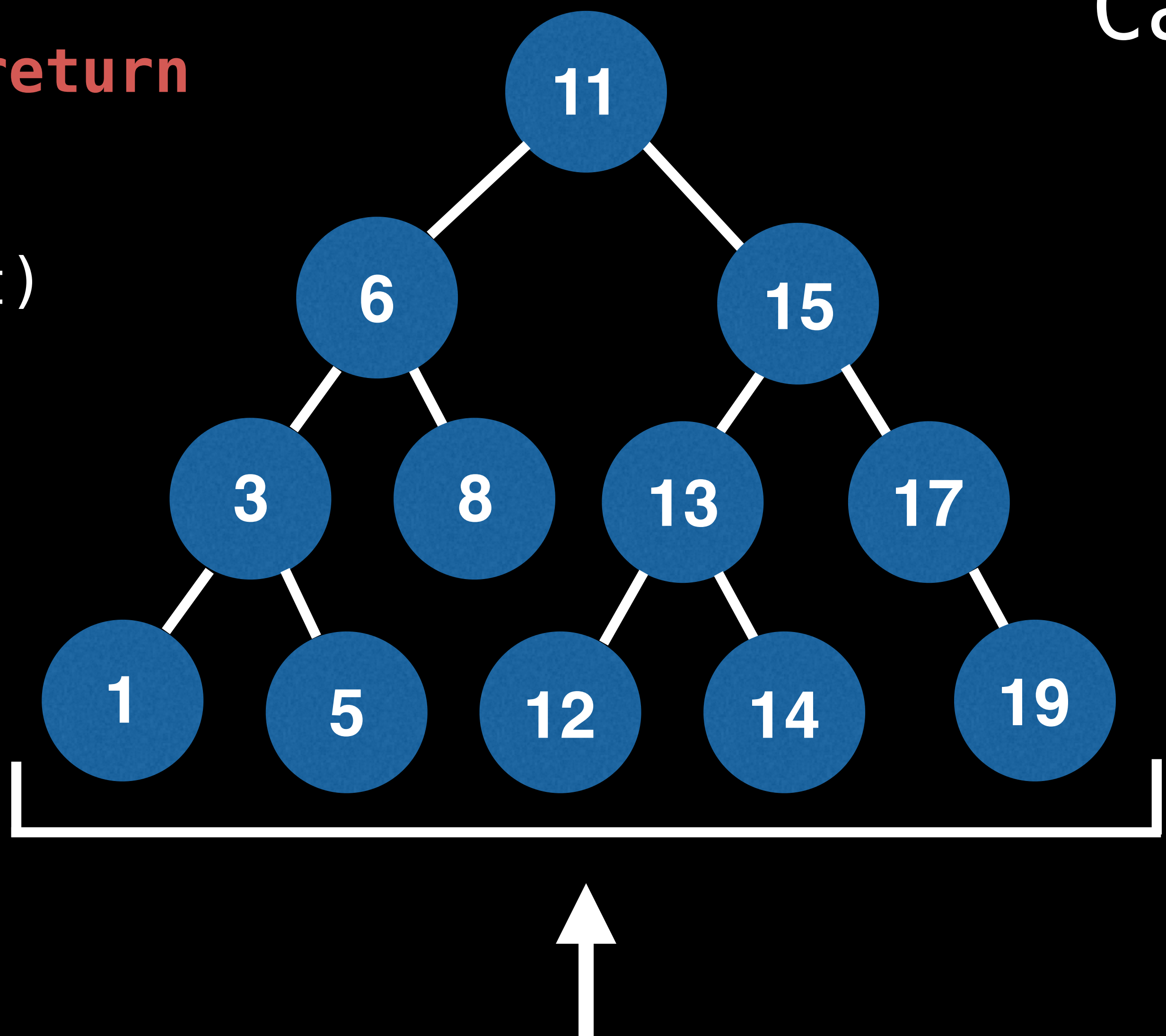


Traverse the left subtree, then print the value of the node and continue traversing the right subtree.

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```

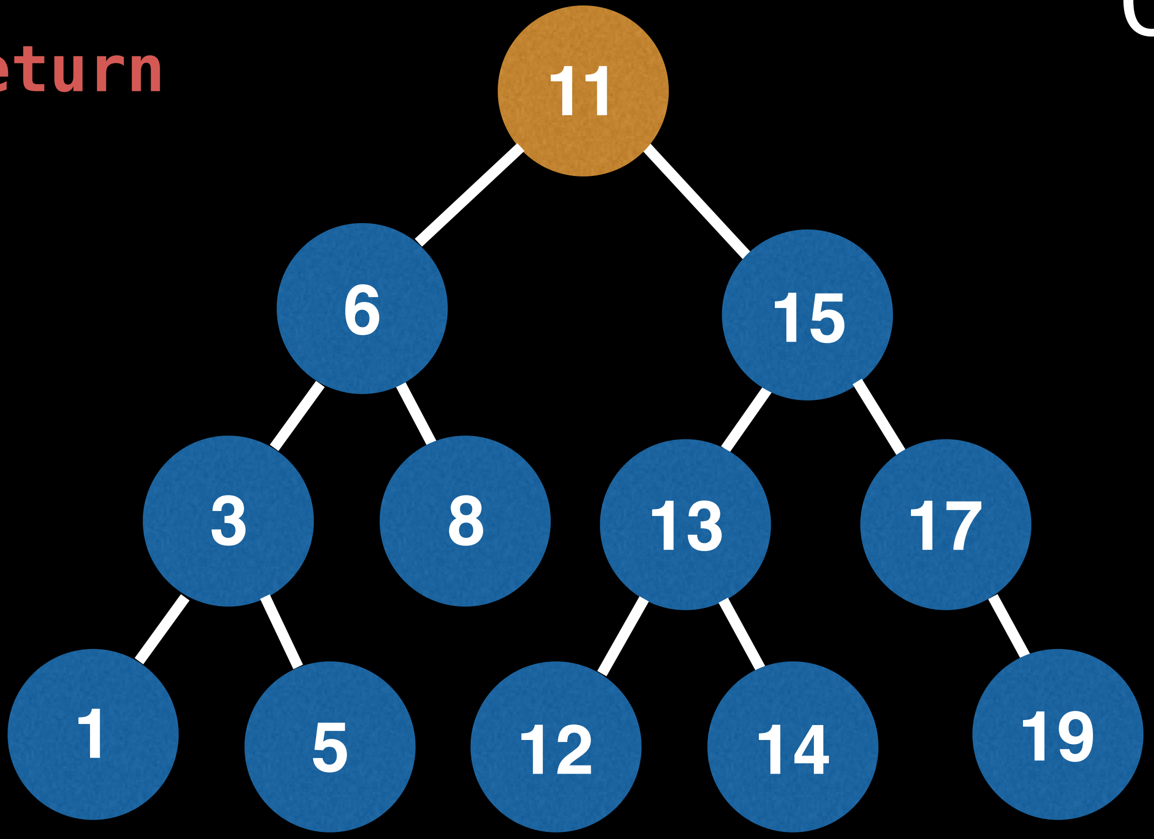
Call Stack:



In this example our tree is
a Binary Search Tree.

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



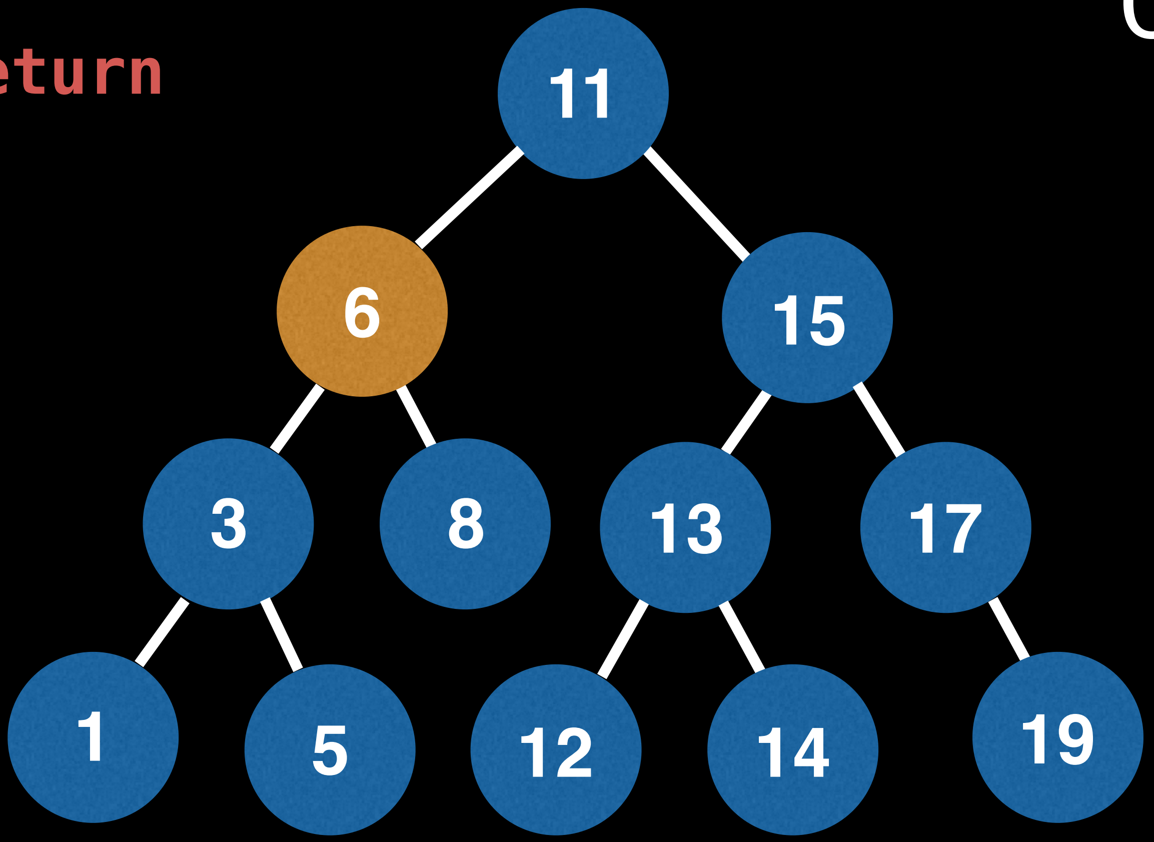
Call Stack:

node 11

Order:

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



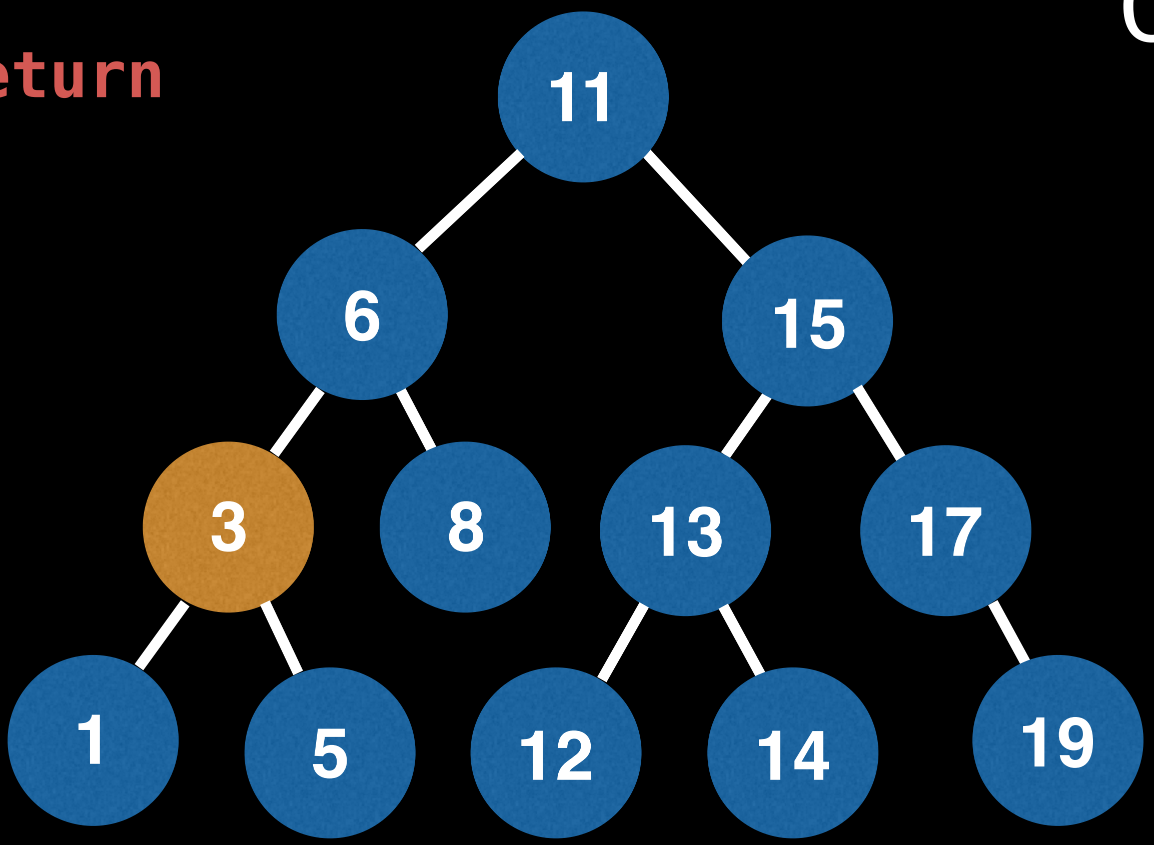
Call Stack:

node 11
node 6

Order:

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



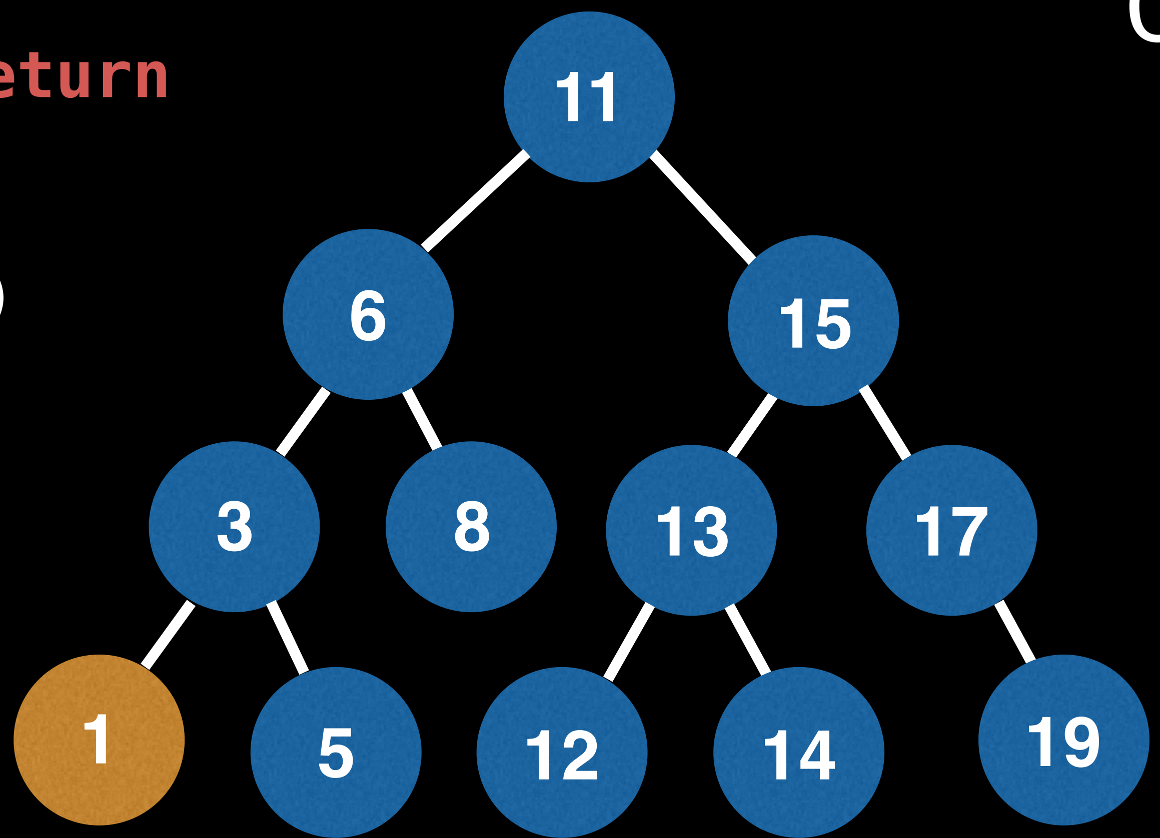
Call Stack:

node 11
node 6
node 3

Order:

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



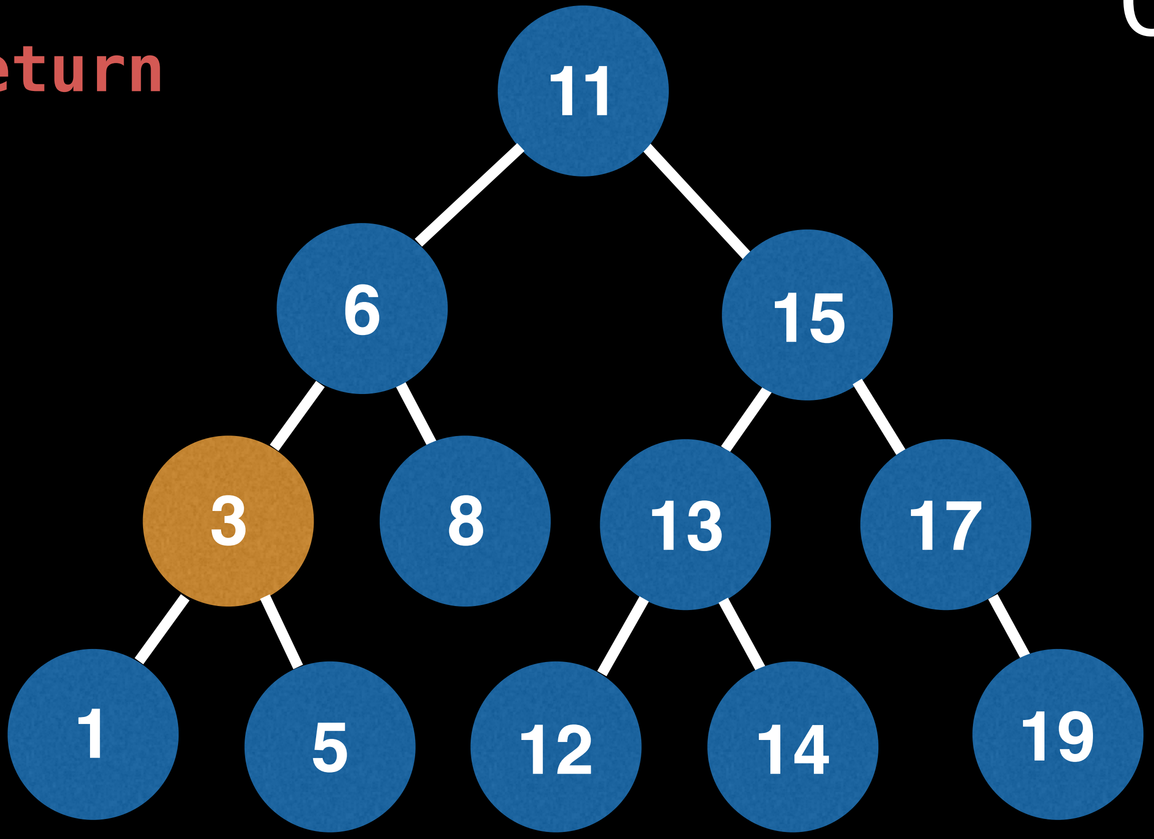
Call Stack:

node 11
node 6
node 3
node 1

Order: 1

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



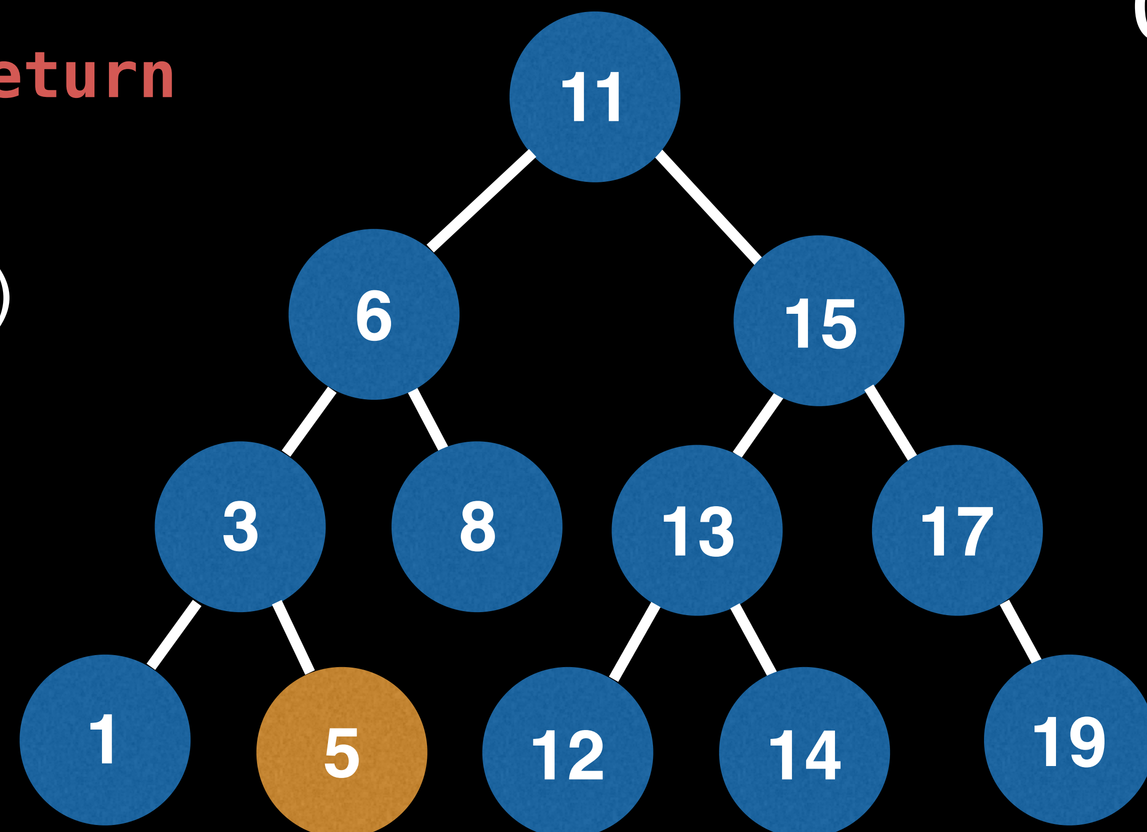
Call Stack:

node 11
node 6
node 3

Order: 1,3

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



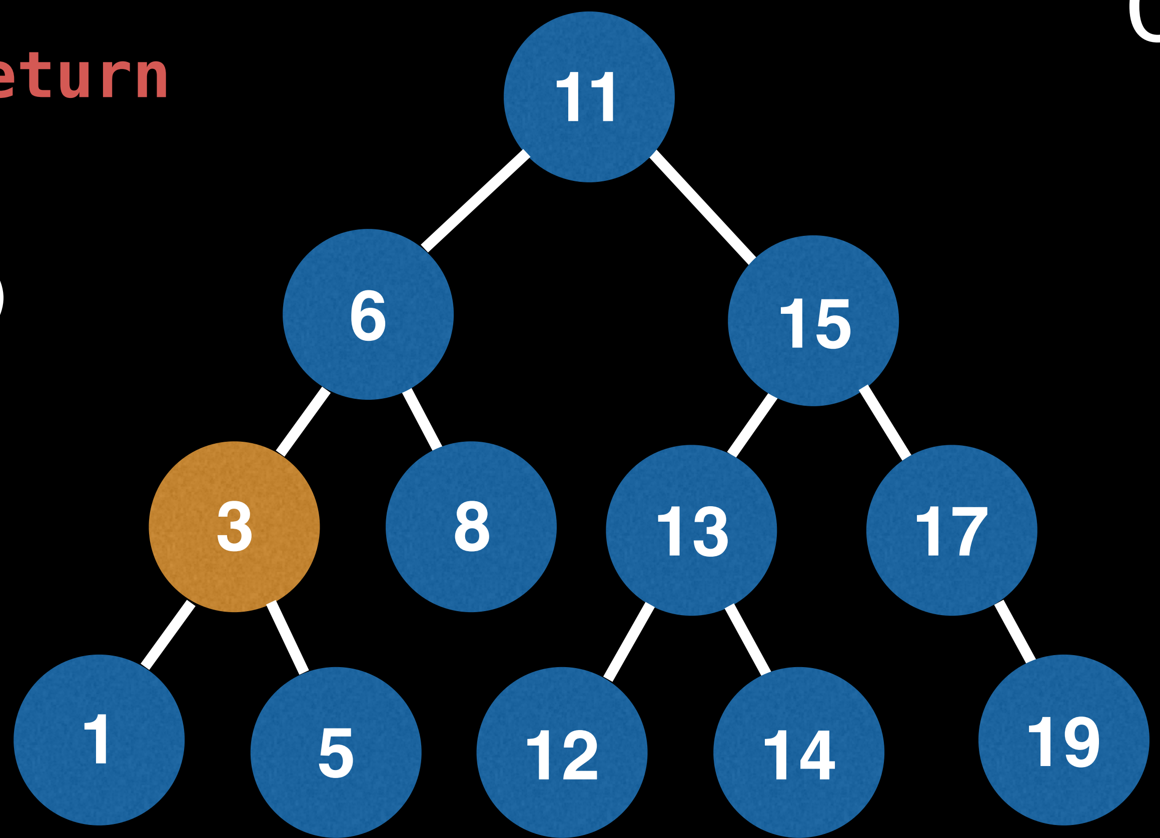
Call Stack:

node 11
node 6
node 3
node 5

Order: 1,3,5

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



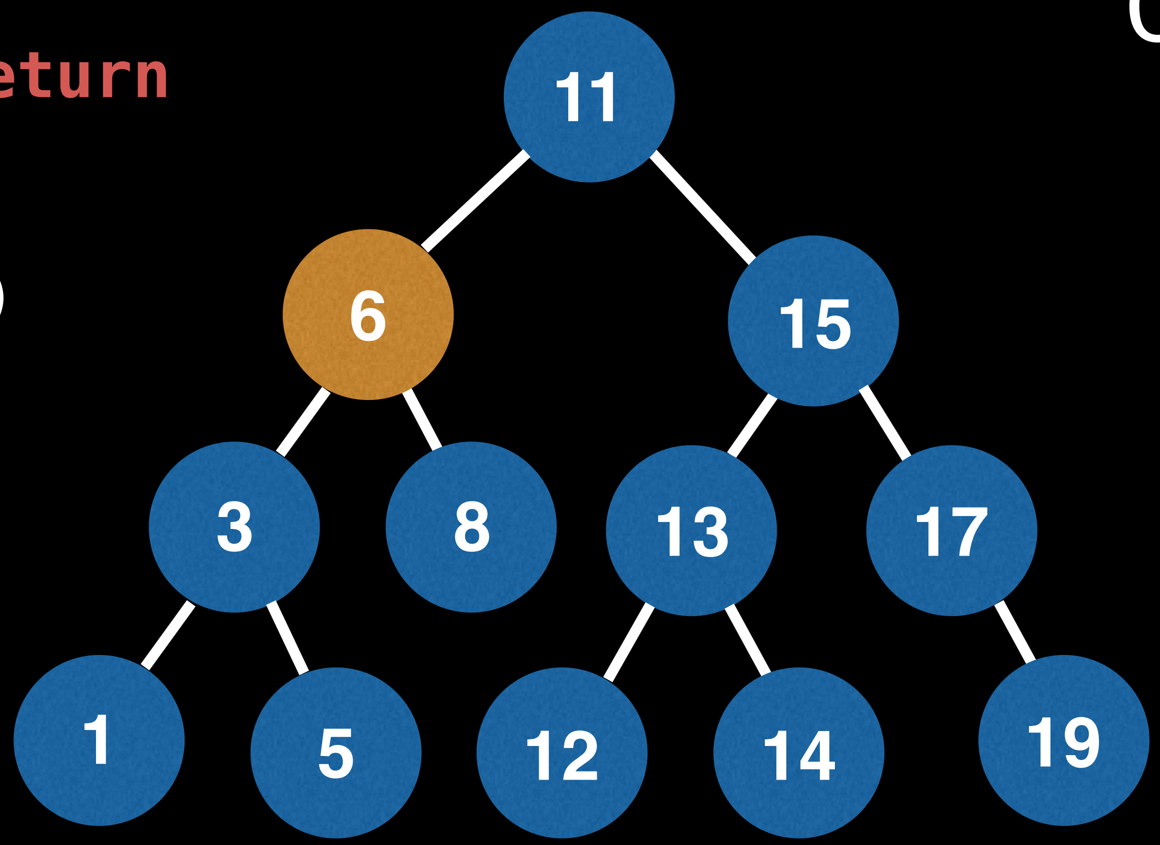
Call Stack:

node 11
node 6
node 3

Order: 1,3,5

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



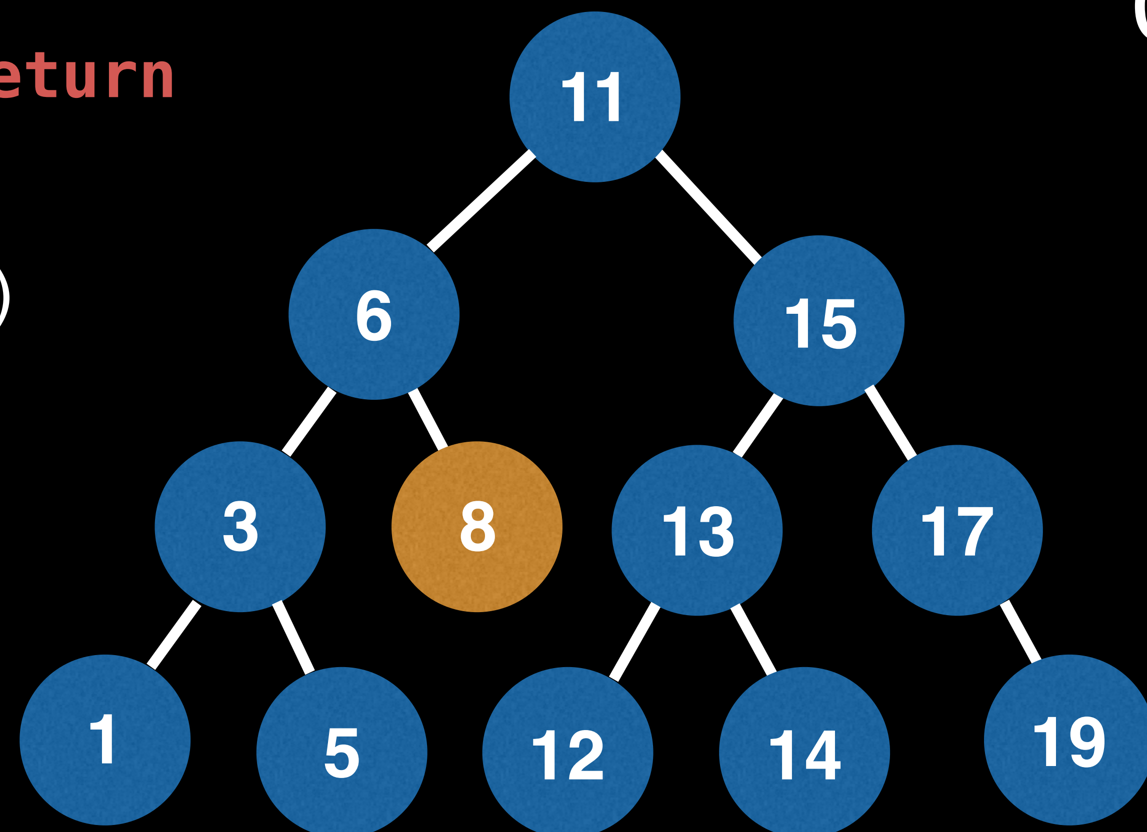
Call Stack:

node 11
node 6

Order: 1,3,5,6

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



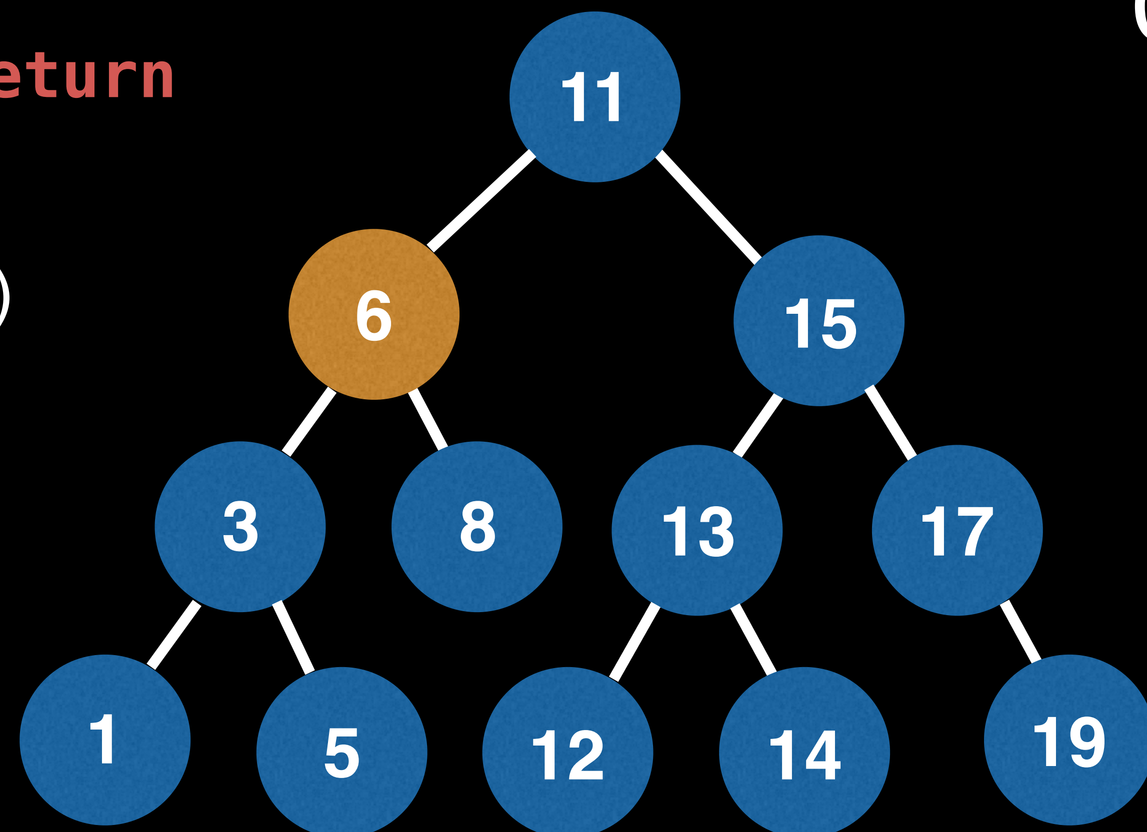
Call Stack:

node 11
node 6
node 8

Order: 1,3,5,6,8

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



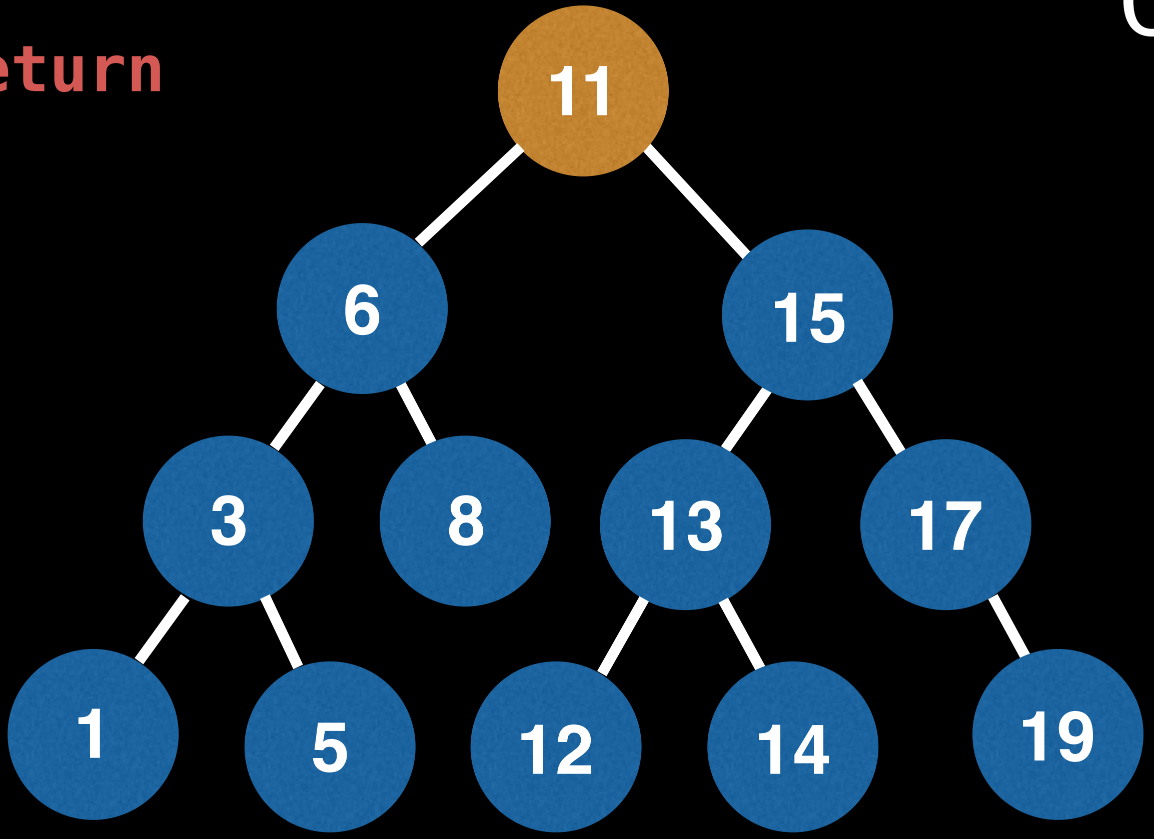
Call Stack:

node 11
node 6

Order: 1,3,5,6,8

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



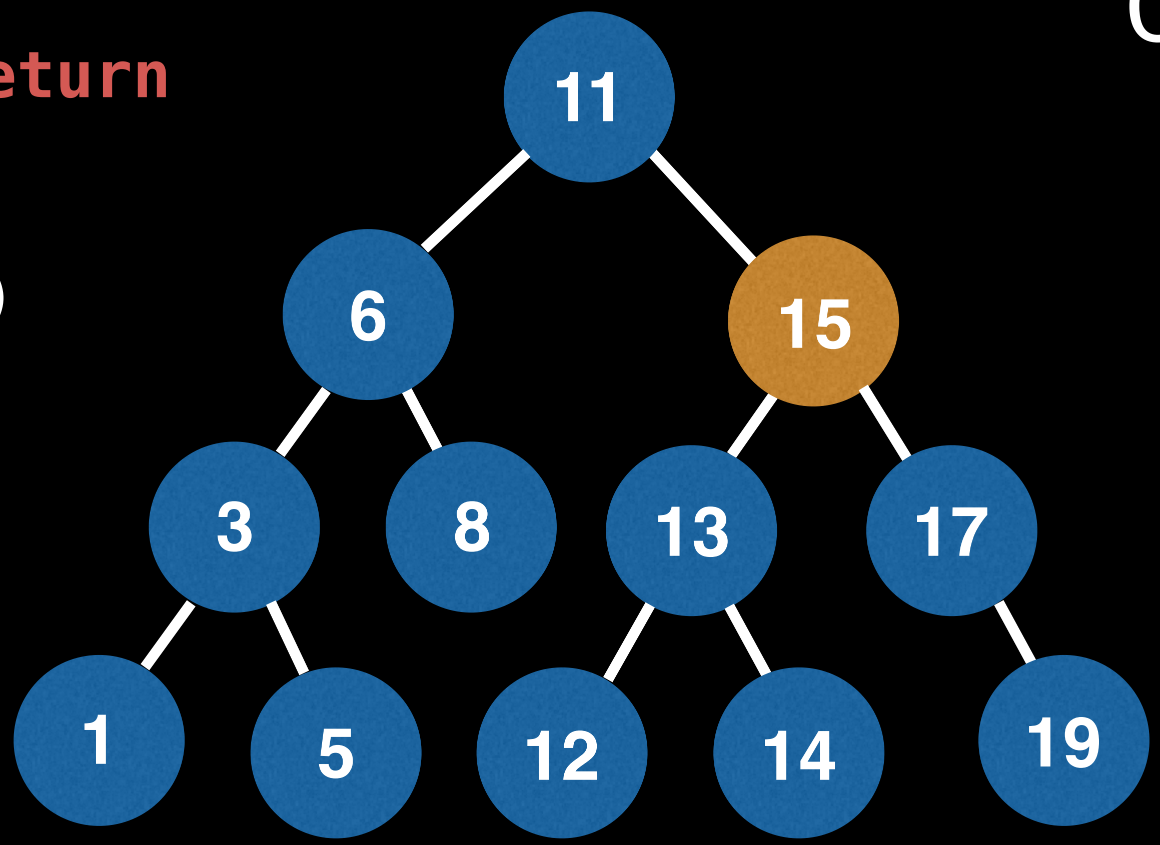
Call Stack:

node 11

Order: 1,3,5,6,8,11

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



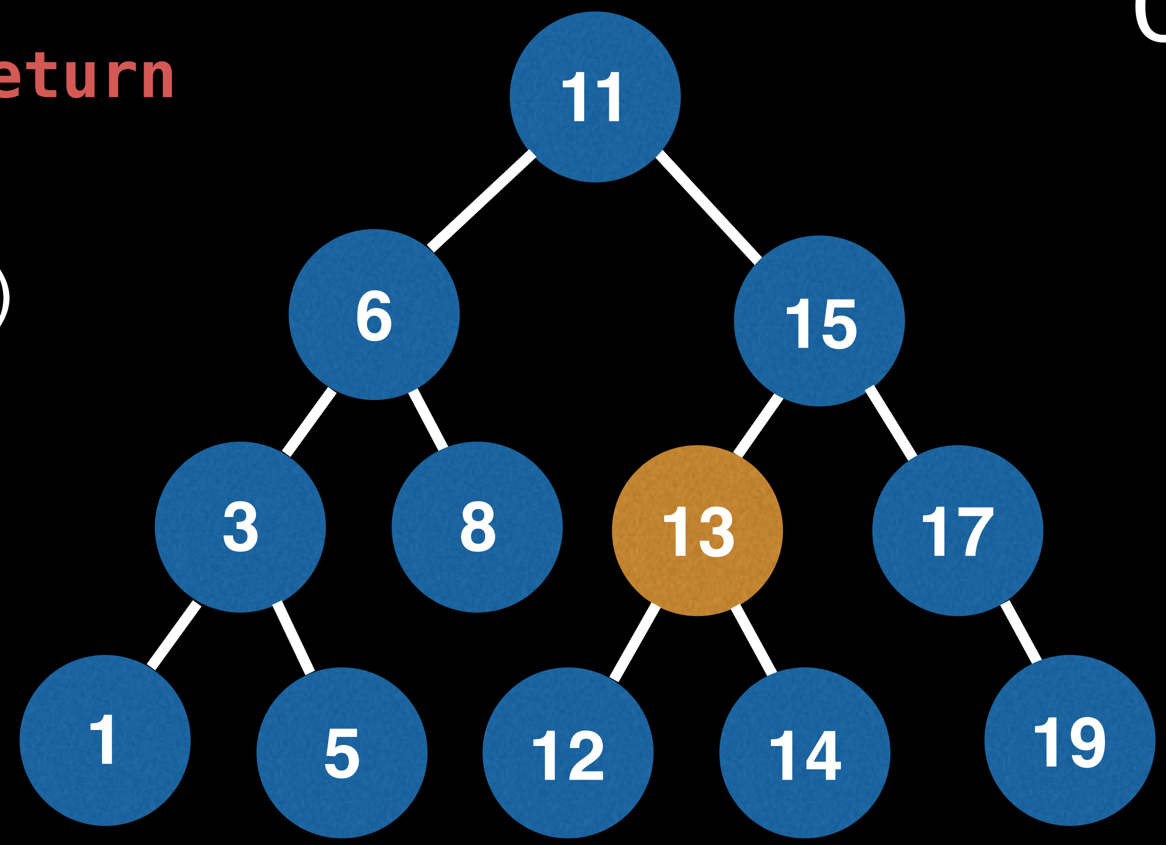
Call Stack:

node 11
node 15

Order: 1,3,5,6,8,11

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



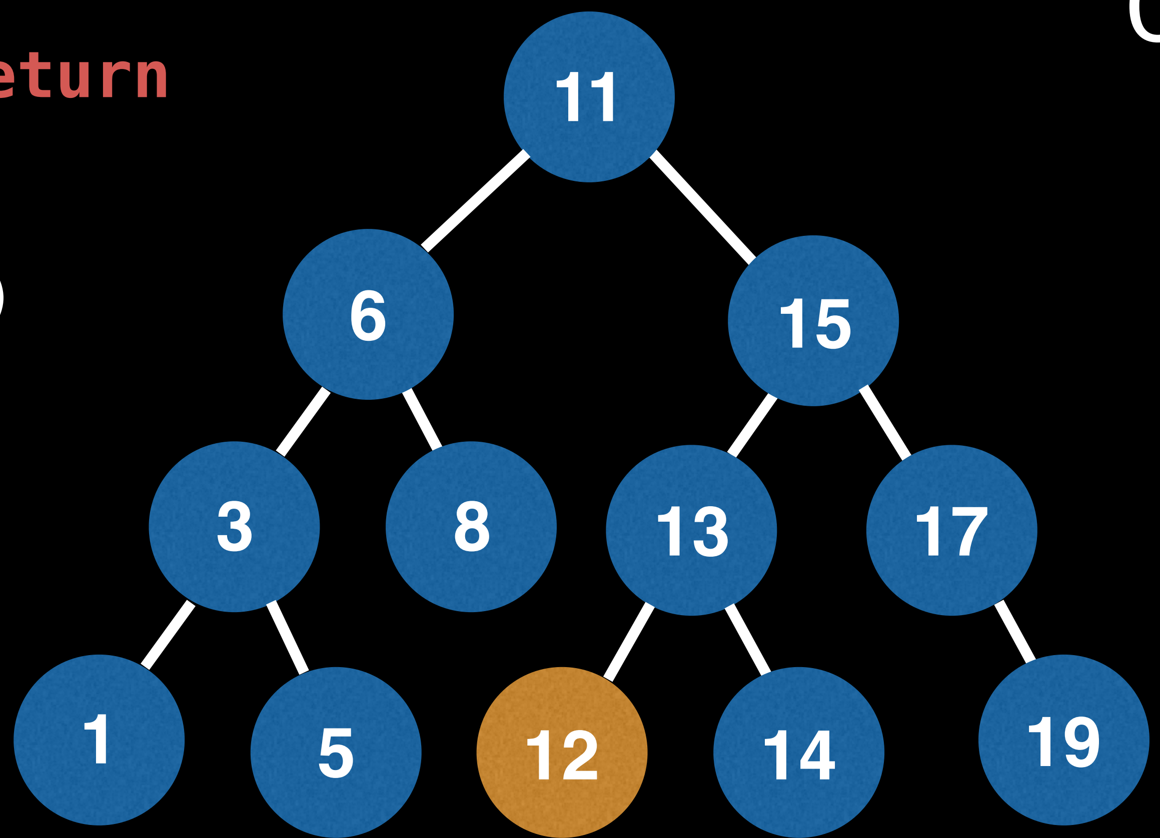
Call Stack:

node 11
node 15
node 13

Order: 1,3,5,6,8,11

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



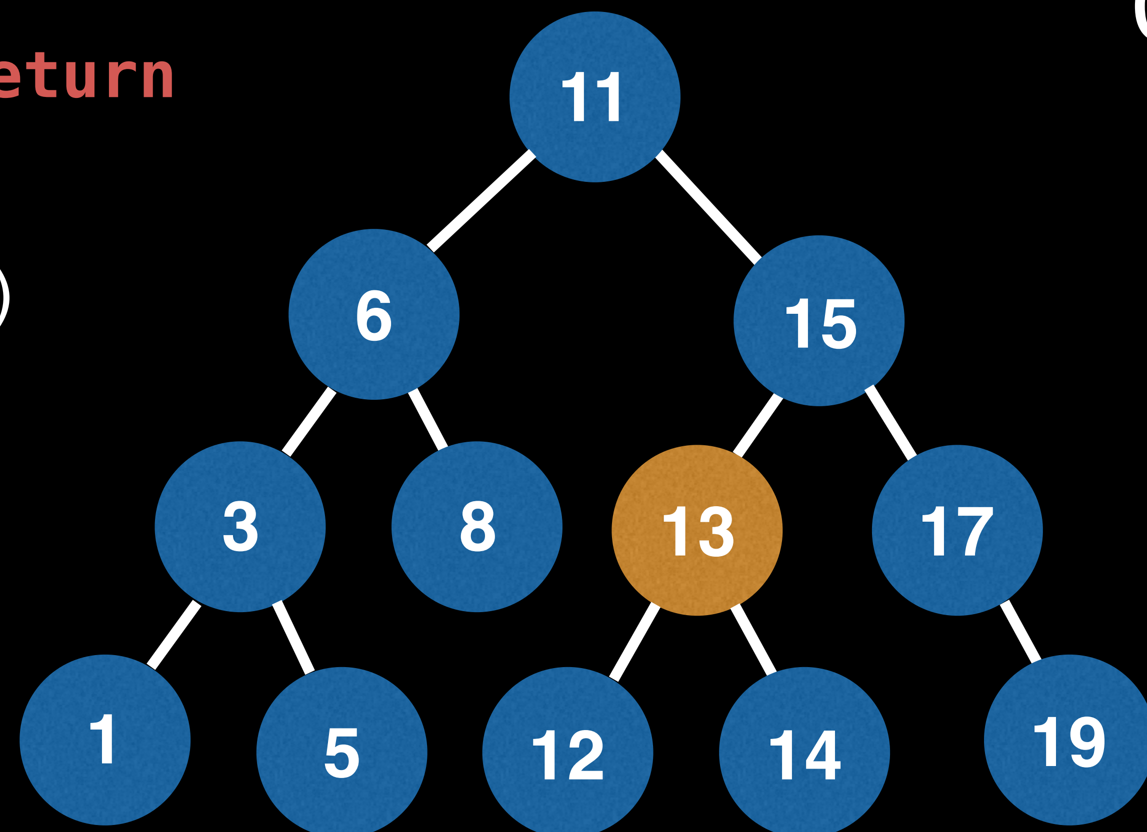
Call Stack:

node 11
node 15
node 13
node 12

Order: 1,3,5,6,8,11,12

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



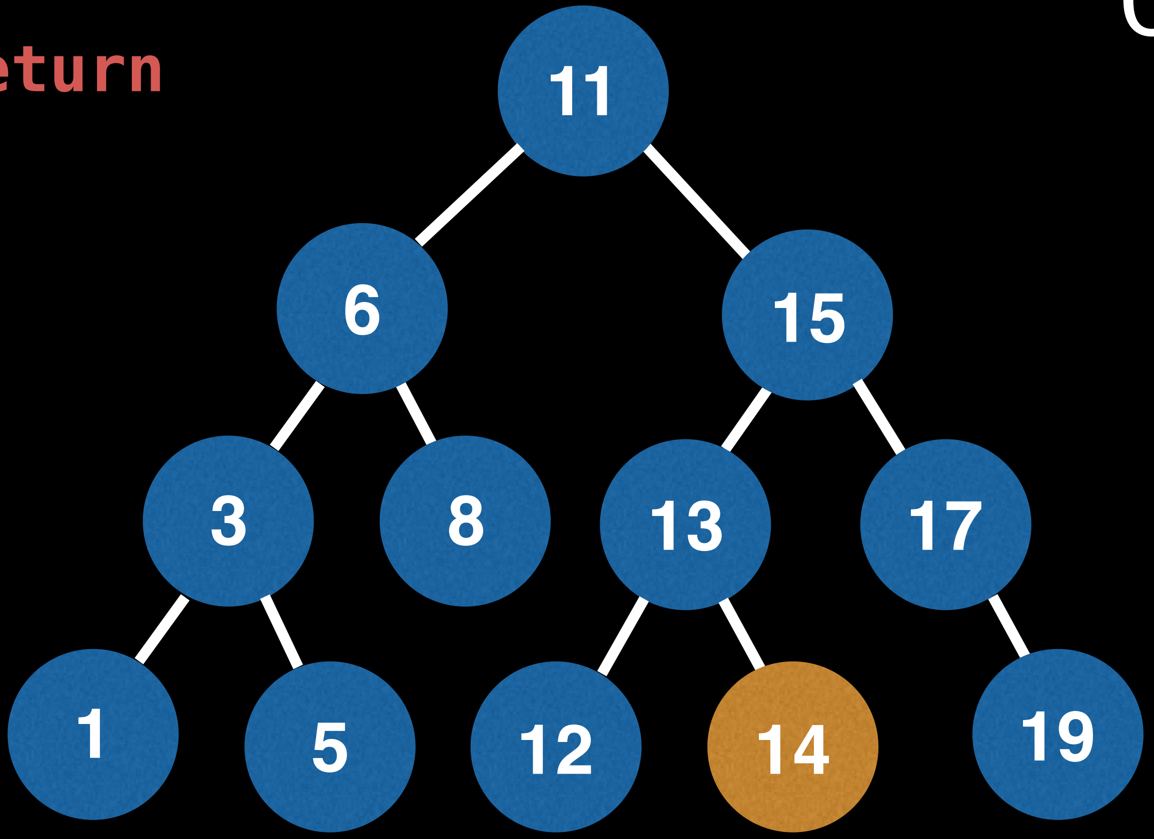
Call Stack:

node 11
node 15
node 13

Order: 1,3,5,6,8,11,12,13

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



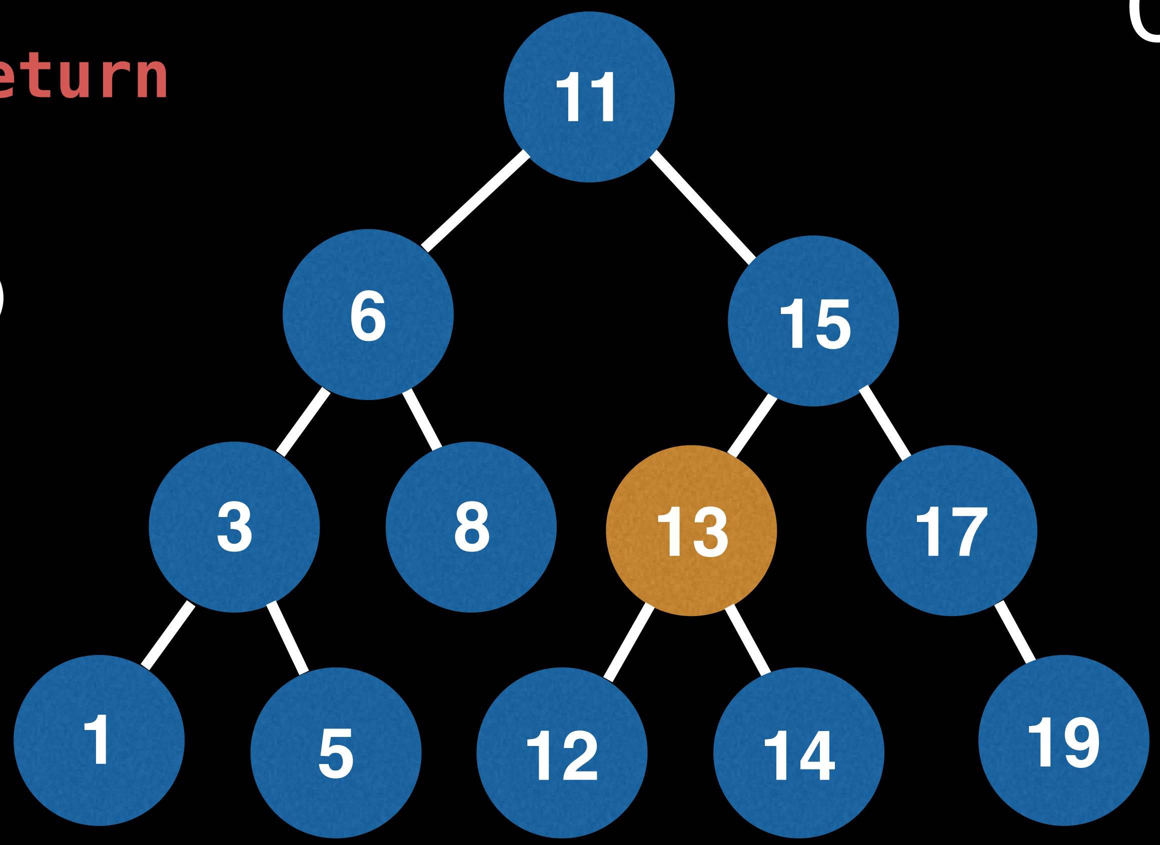
Call Stack:

node 11
node 15
node 13
node 14

Order: 1,3,5,6,8,11,12,13,14

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



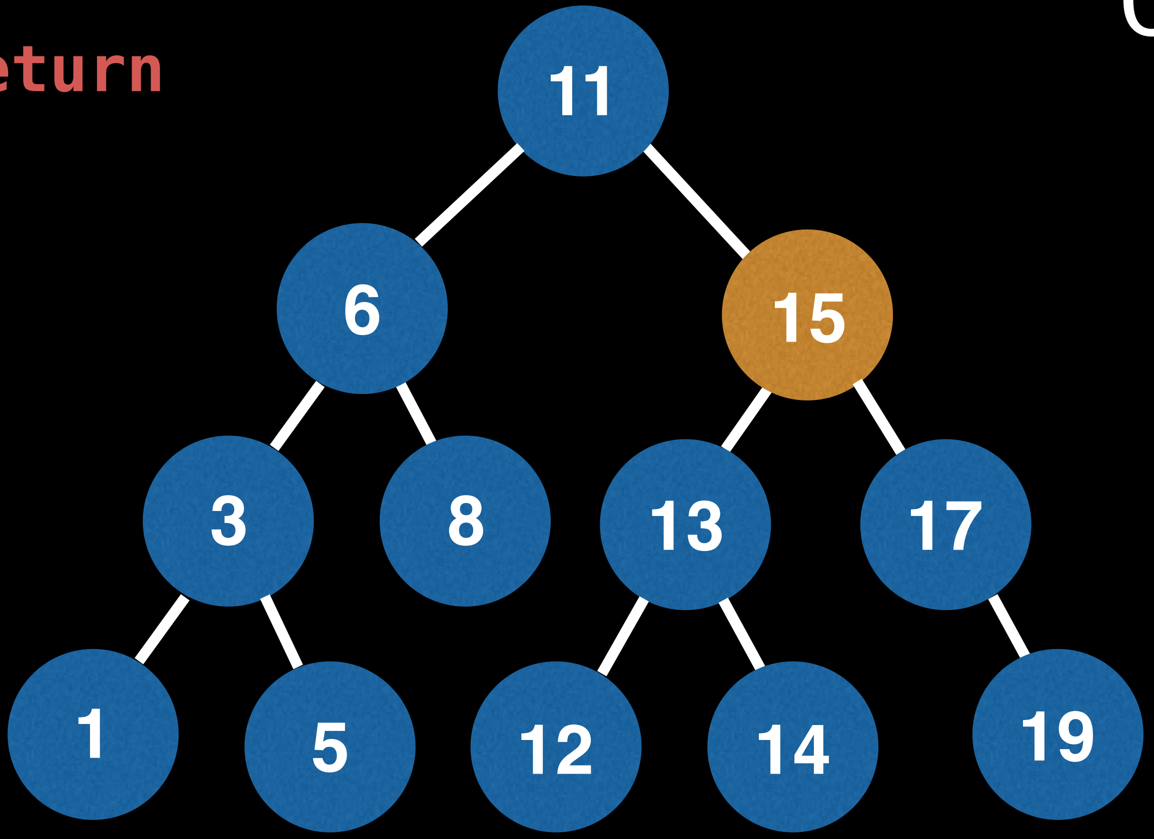
Call Stack:

node 11
node 15
node 13

Order: 1,3,5,6,8,11,12,13,14

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



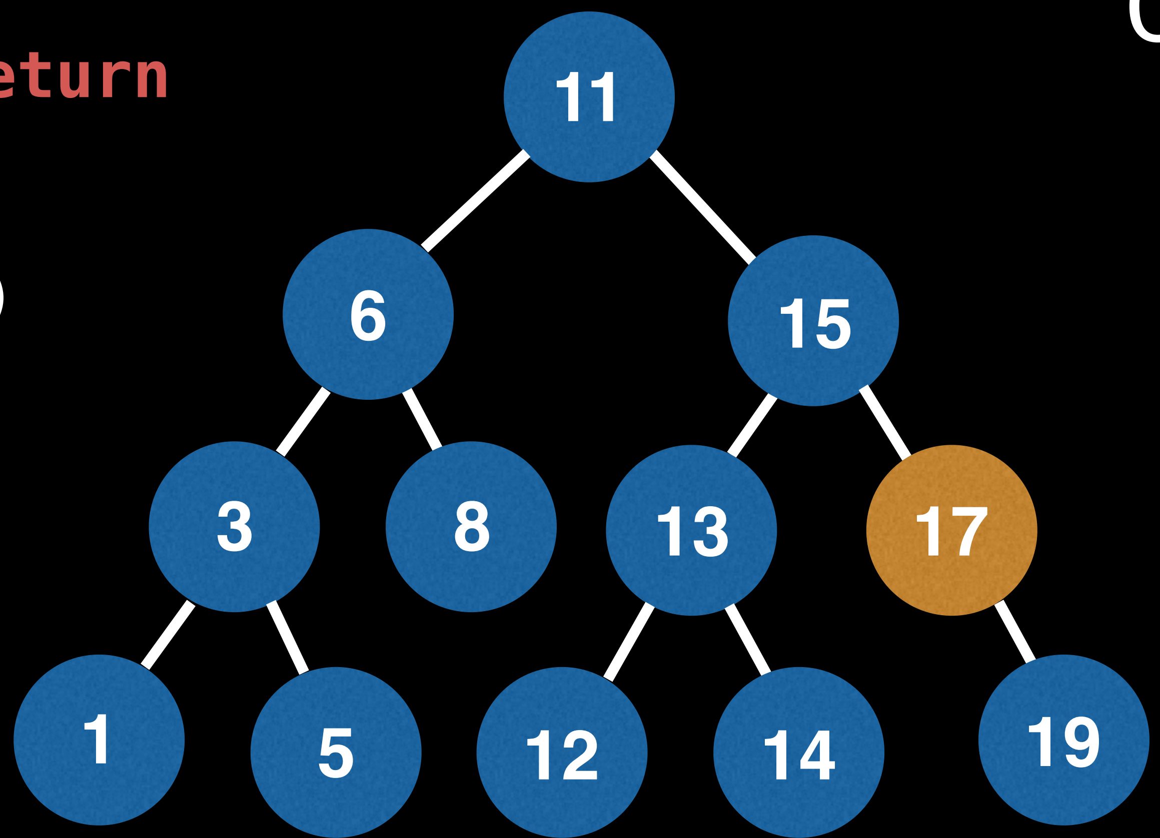
Call Stack:

node 11
node 15

Order: 1,3,5,6,8,11,12,13,14,15

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



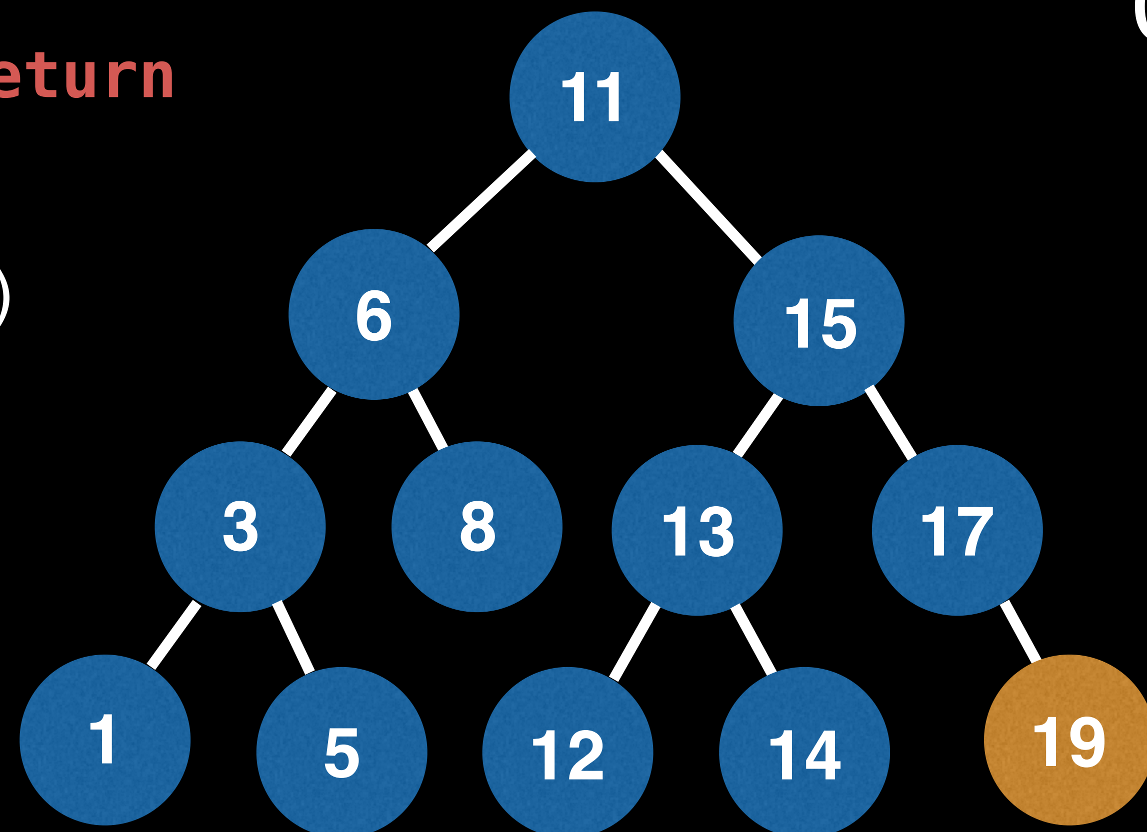
Call Stack:

node 11
node 15
node 17

Order: 1,3,5,6,8,11,12,13,14,15,17

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



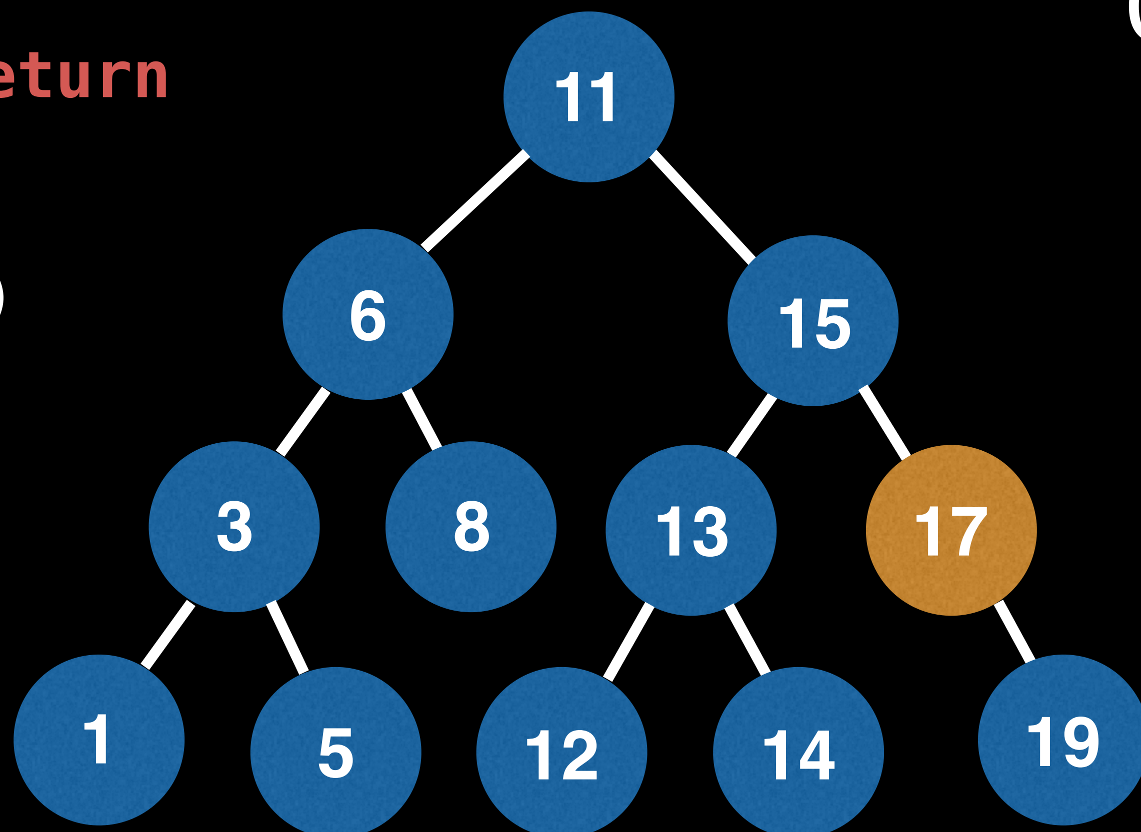
Call Stack:

node 11
node 15
node 17
node 19

Order: 1,3,5,6,8,11,12,13,14,15,17,19

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



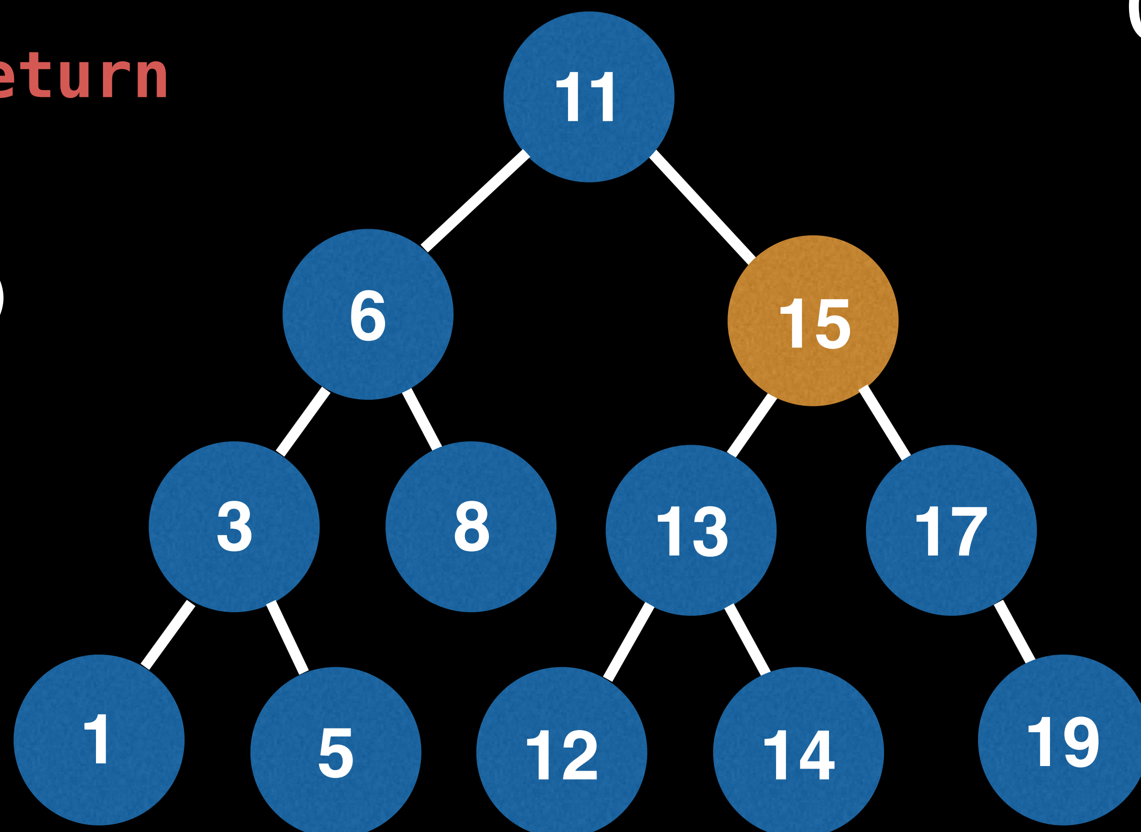
Call Stack:

node 11
node 15
node 17

Order: 1,3,5,6,8,11,12,13,14,15,17,19

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



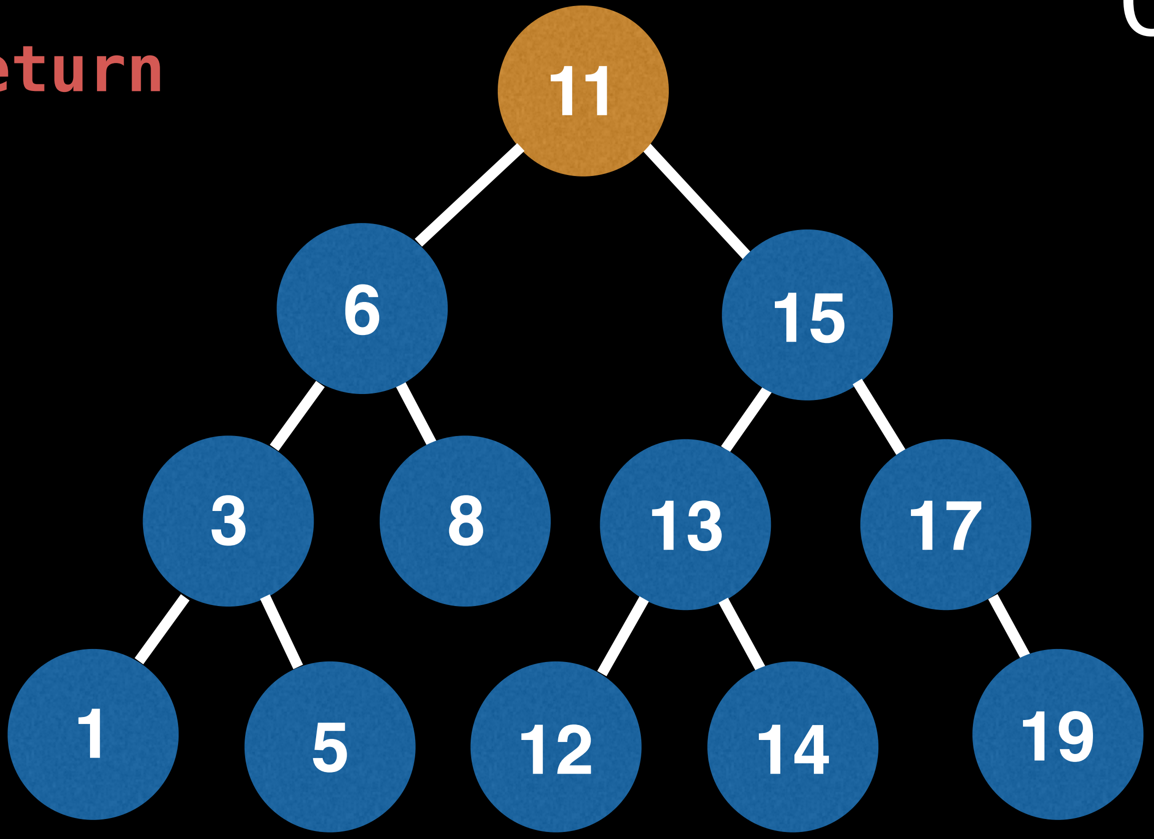
Call Stack:

node 11
node 15

Order: 1,3,5,6,8,11,12,13,14,15,17,19

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```



Call Stack:

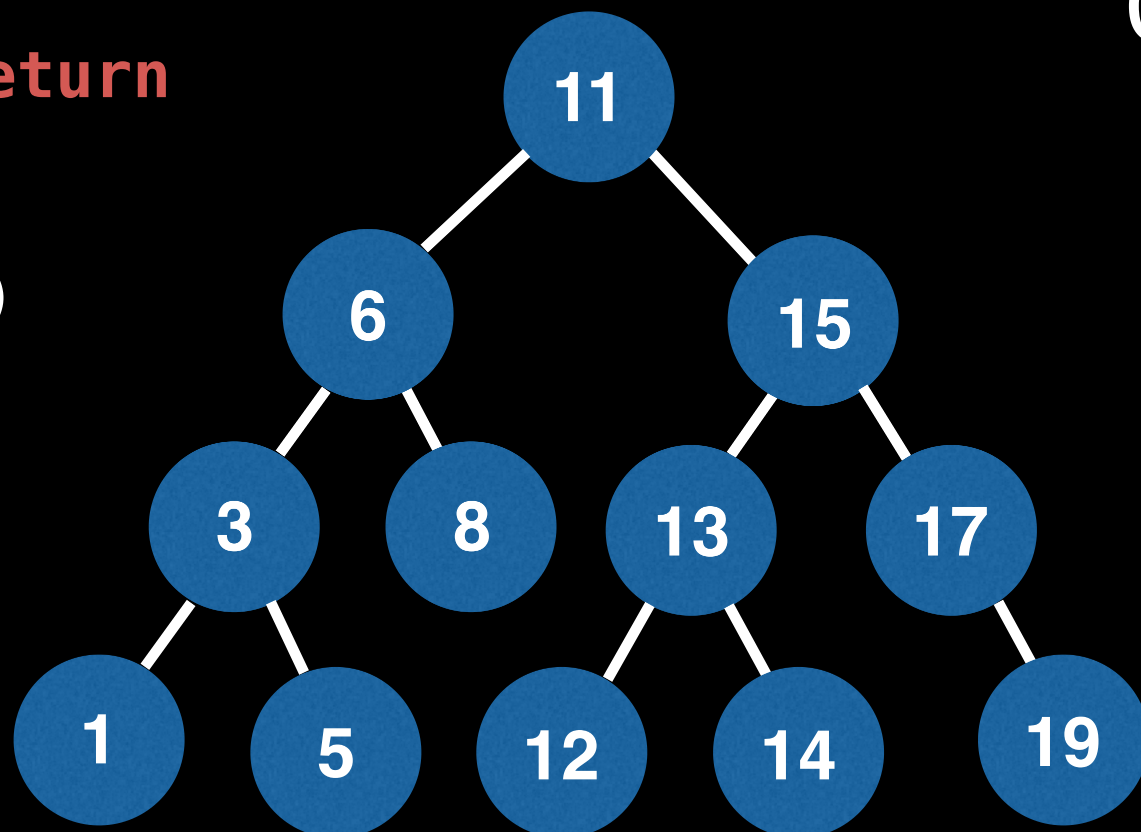
node 11

Order: 1,3,5,6,8,11,12,13,14,15,17,19

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```

Call Stack:

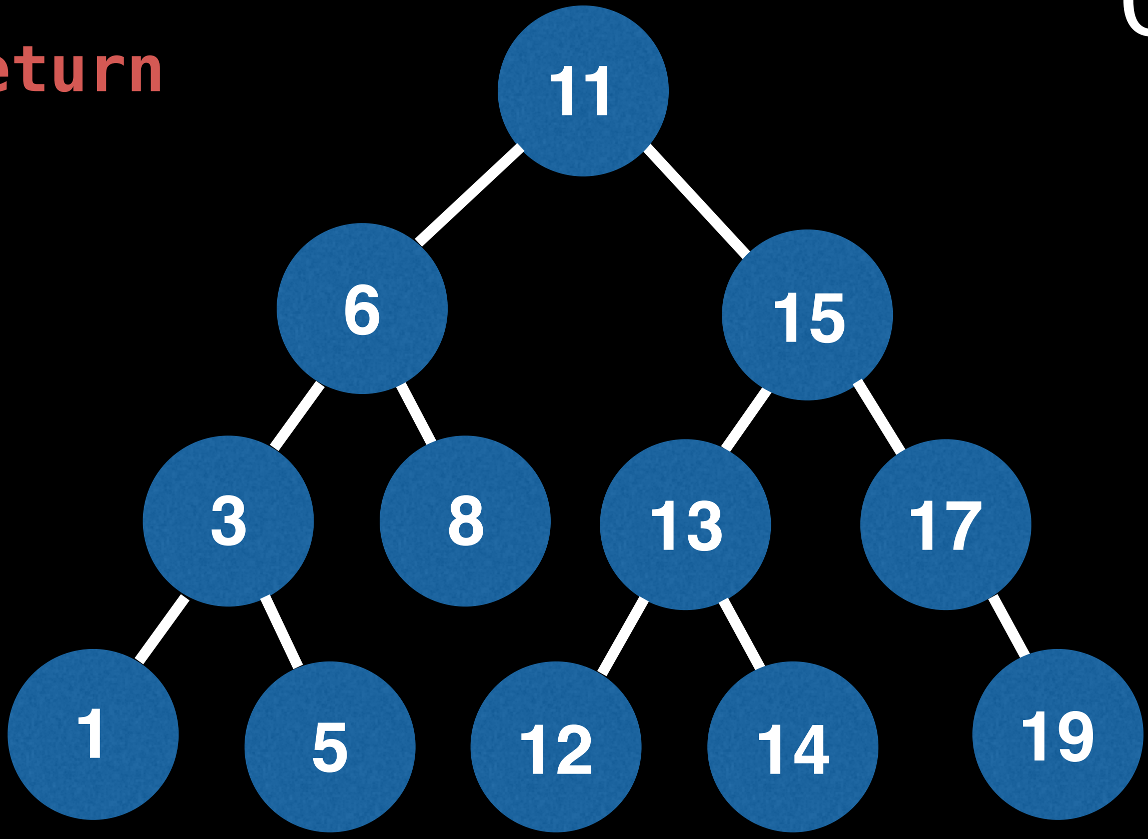


Order: 1,3,5,6,8,11,12,13,14,15,17,19

Inorder Traversal

```
inorder(node):
  if node == null: return
  inorder(node.left)
  print(node.value)
  inorder(node.right)
```

Call Stack:



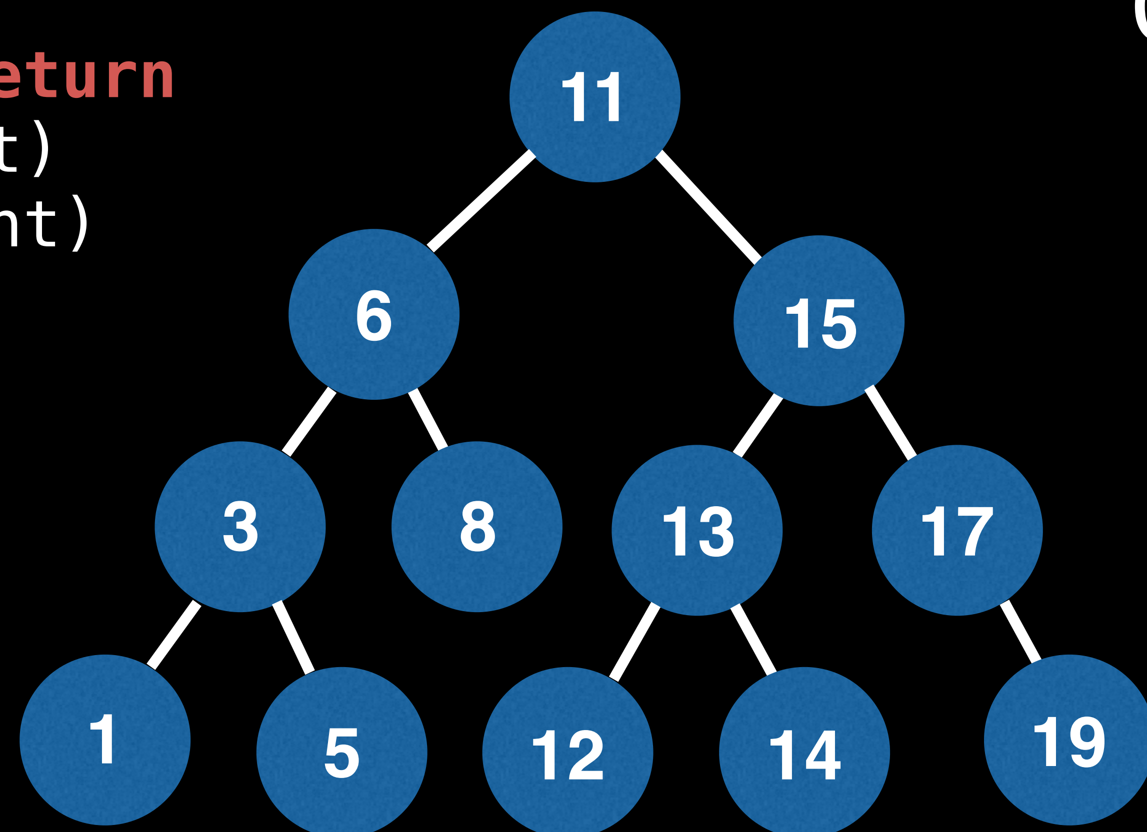
Order: 1,3,5,6,8,11,12,13,14,15,17,19

Notice that with a BST the values printed by the inorder traversal are in increasing order!

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```

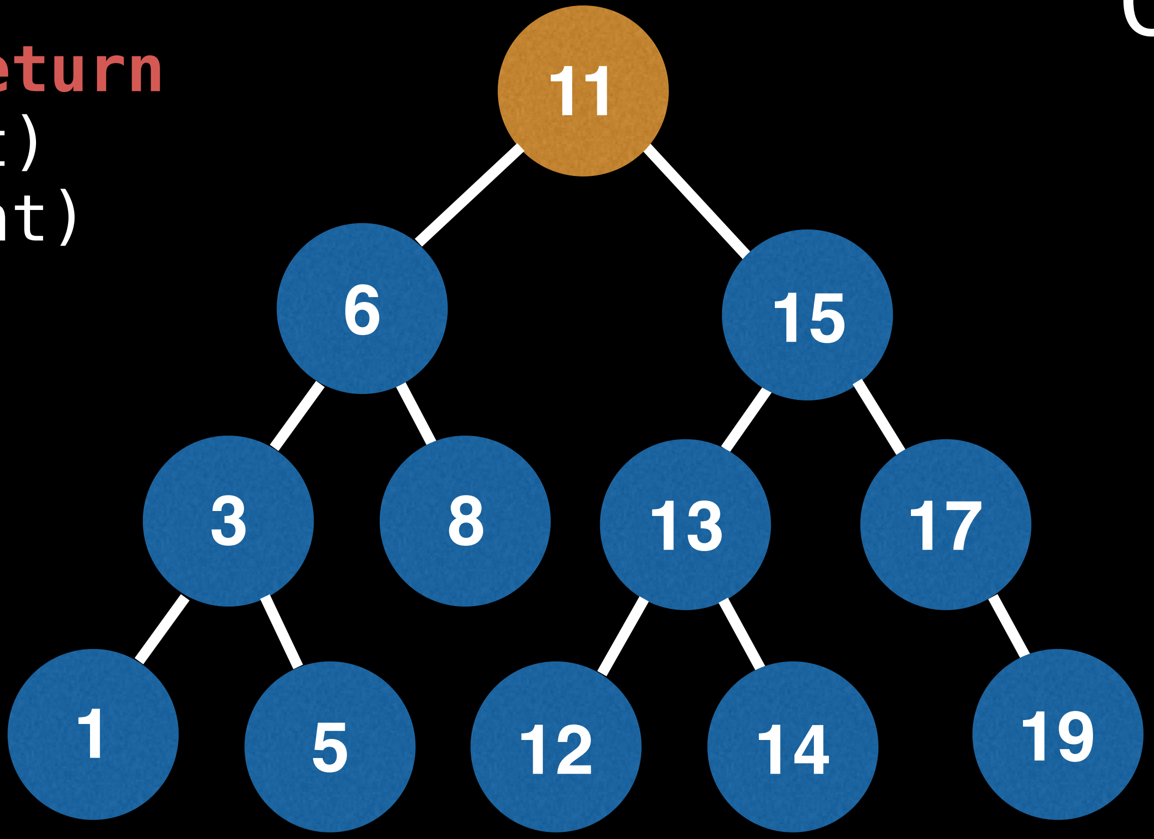
Call Stack:



Traverse the left subtree followed by the right subtree then print the value of the node

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



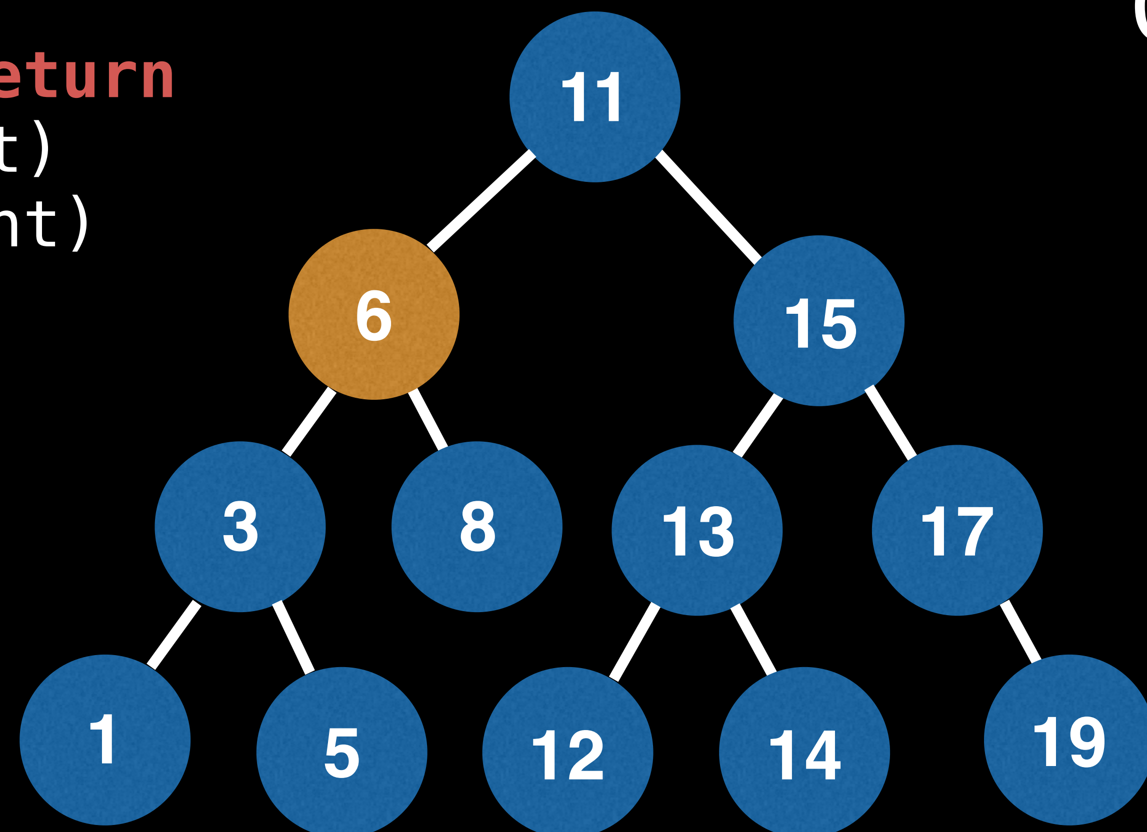
Call Stack:

node 11

Order:

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



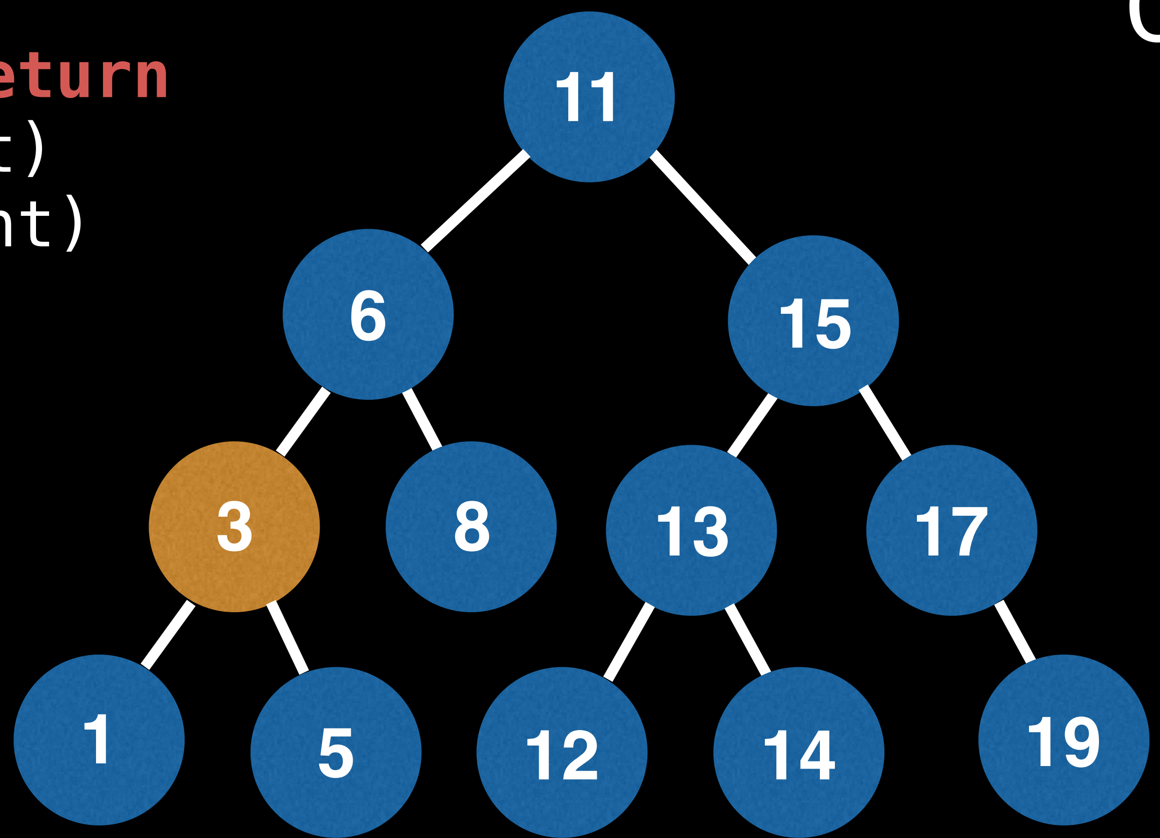
Call Stack:

node 11
node 6

Order:

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



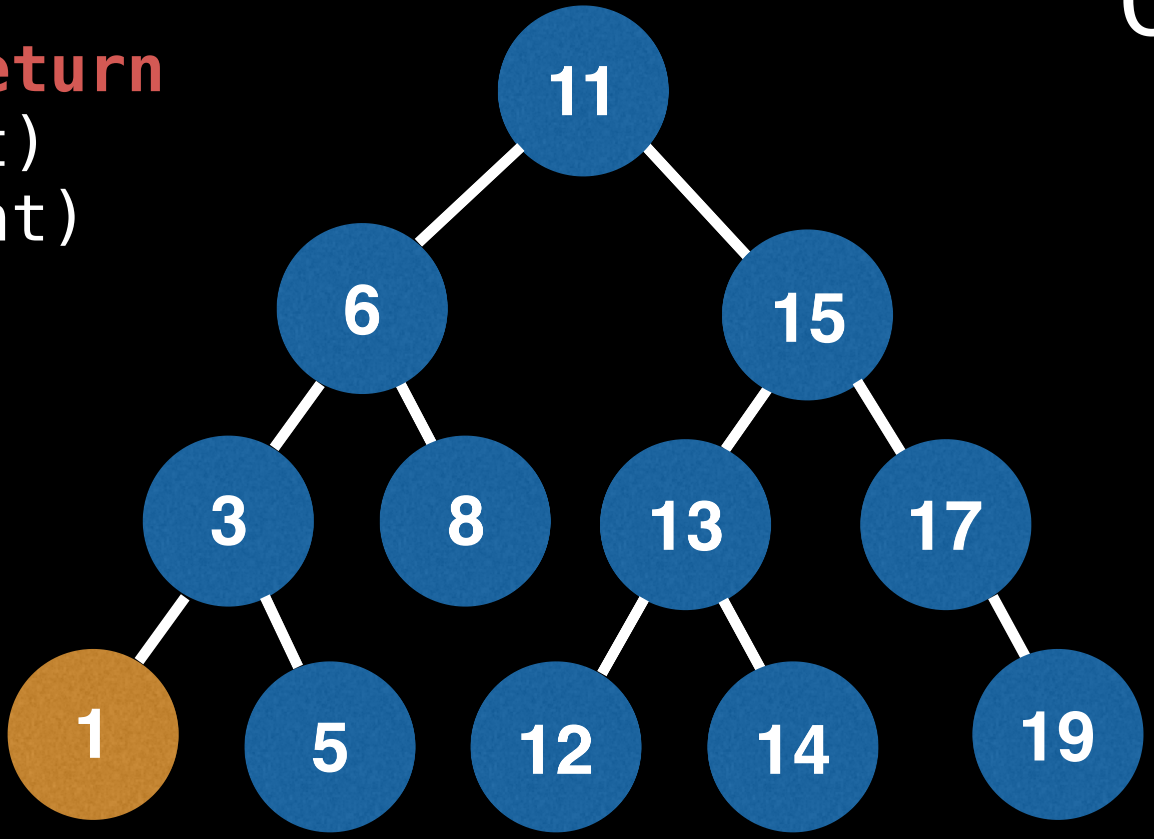
Call Stack:

node 11
node 6
node 3

Order:

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



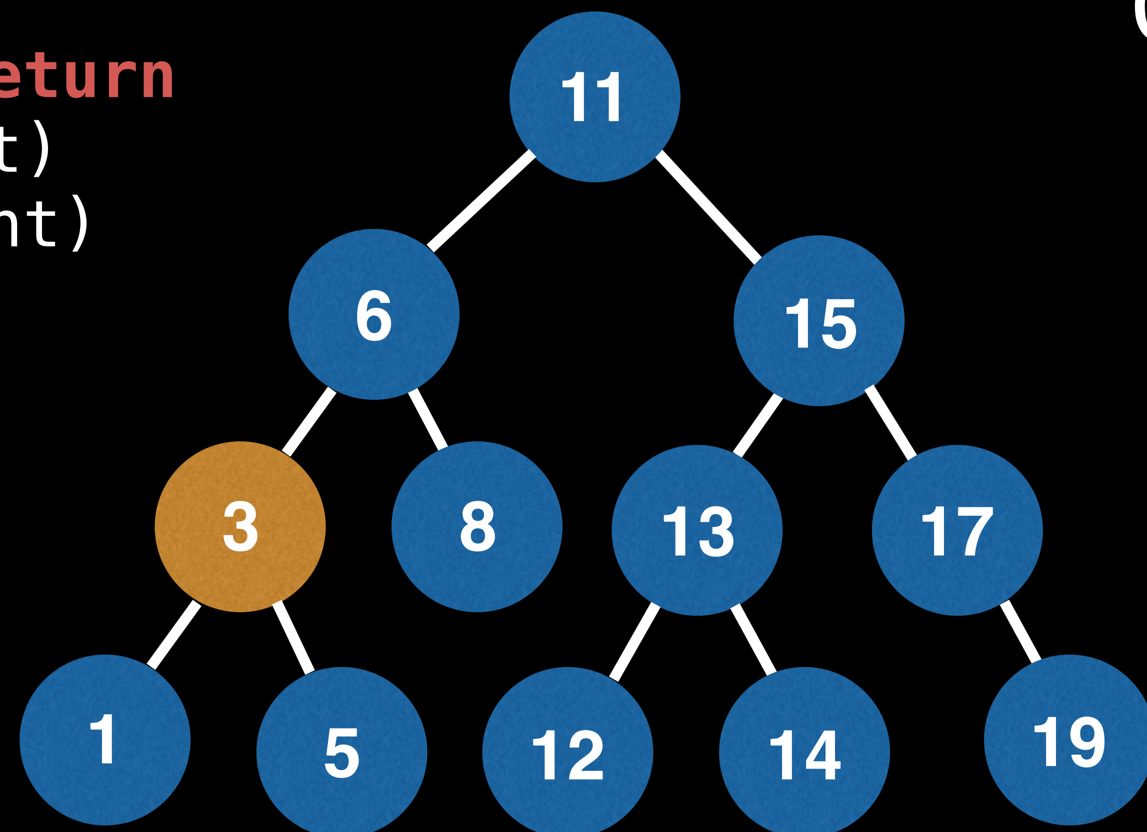
Call Stack:

node 11
node 6
node 3
node 1

Order: 1

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



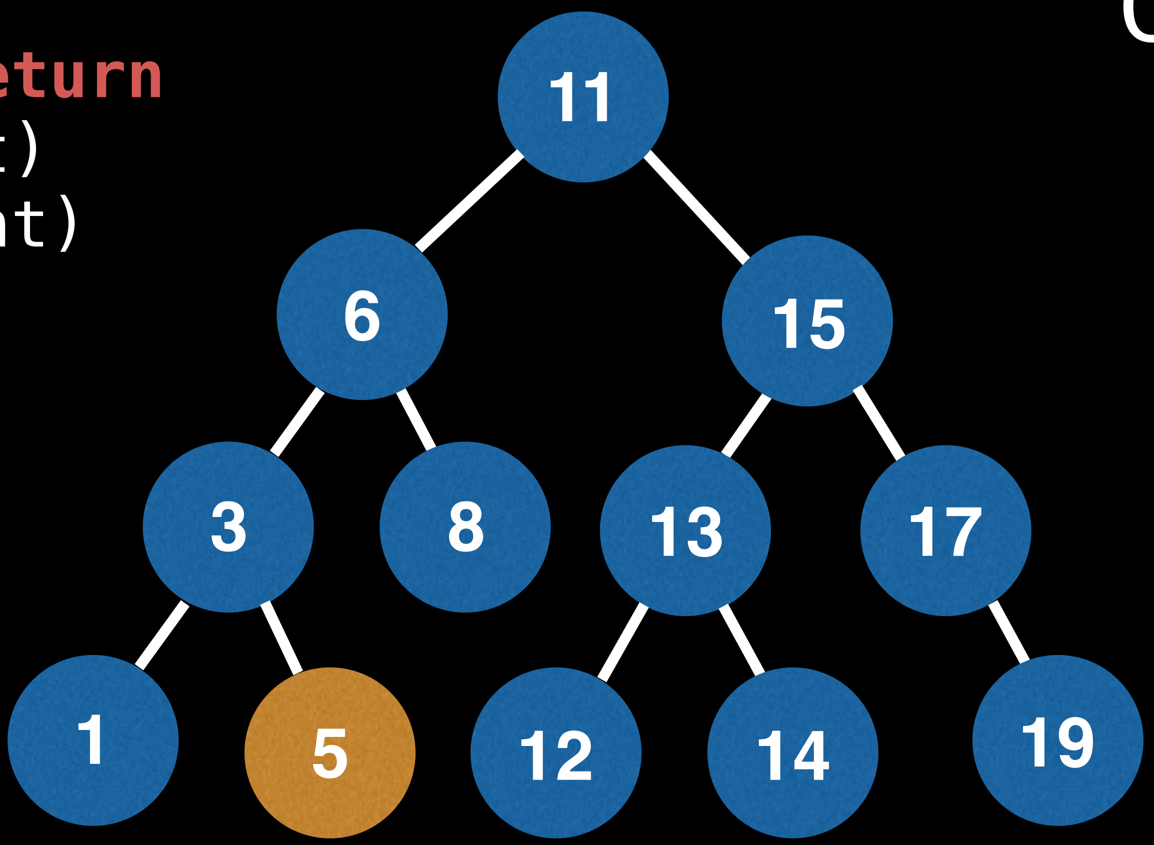
Call Stack:

node 11
node 6
node 3

Order: 1

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



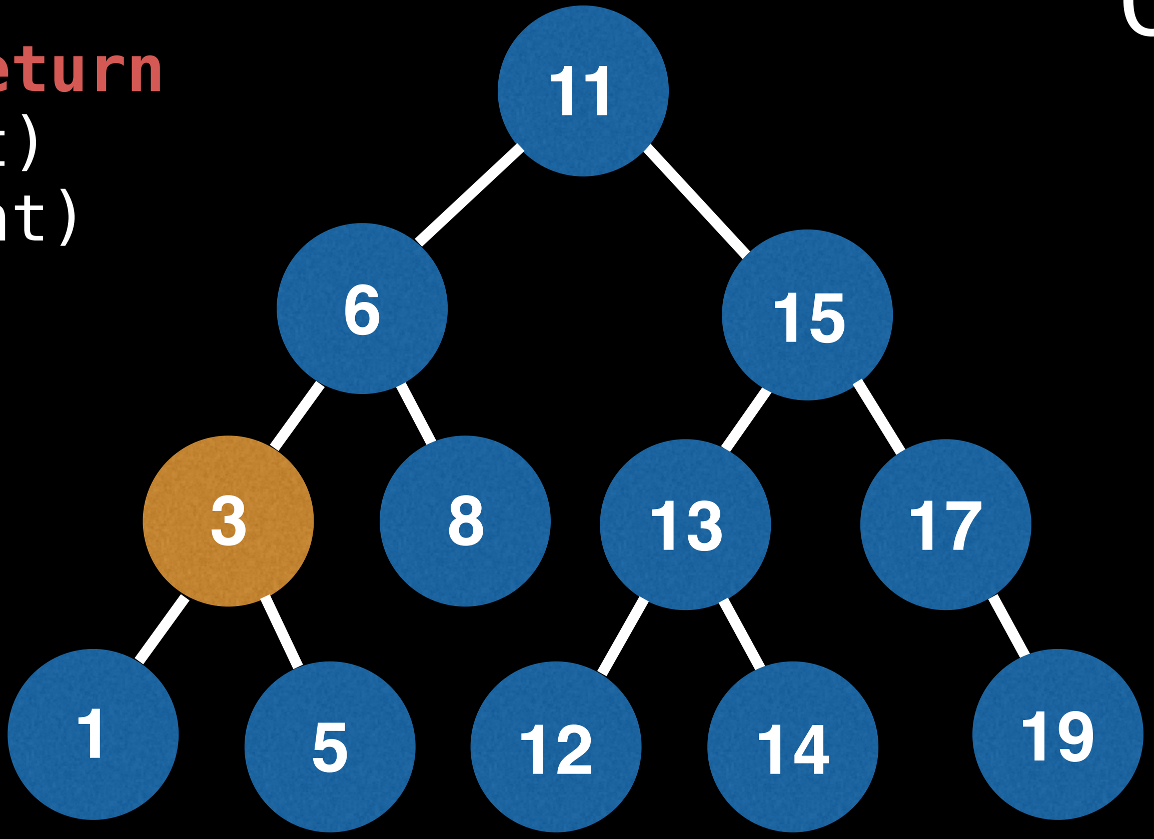
Call Stack:

node 11
node 6
node 3
node 5

Order: 1,5

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



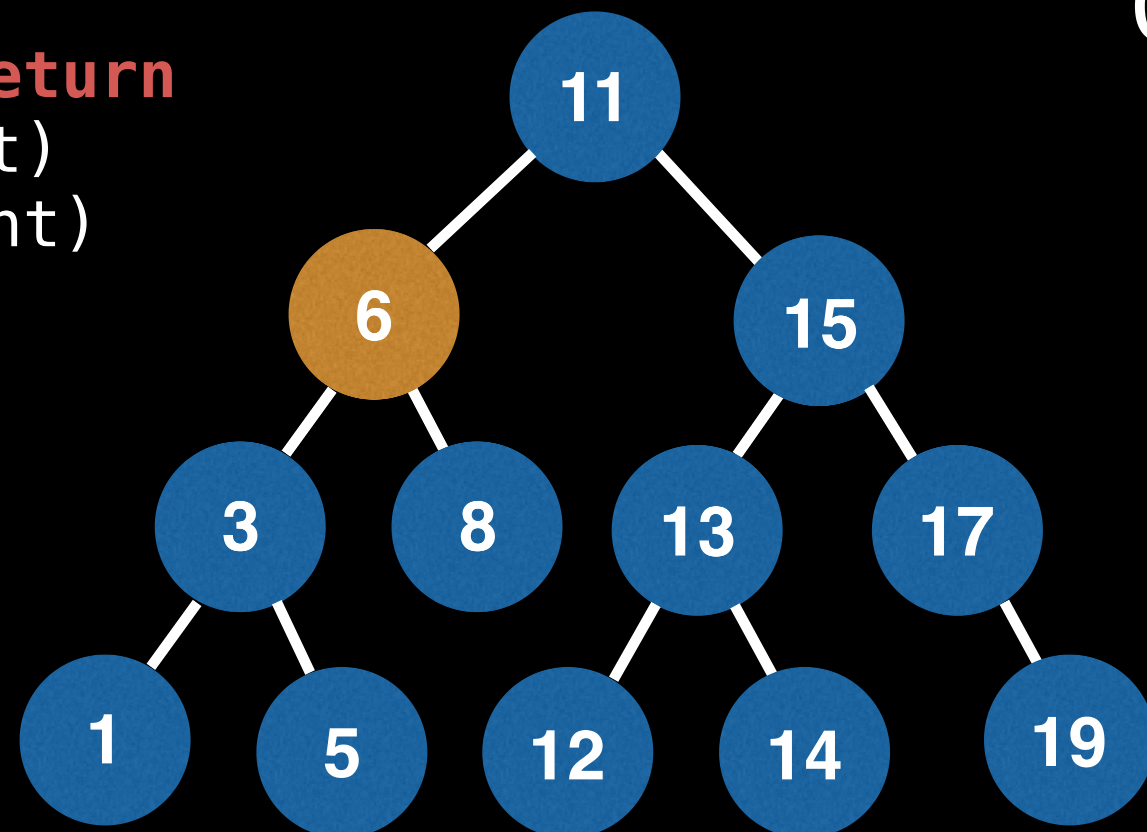
Call Stack:

node 11
node 6
node 3

Order: 1,5,3

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



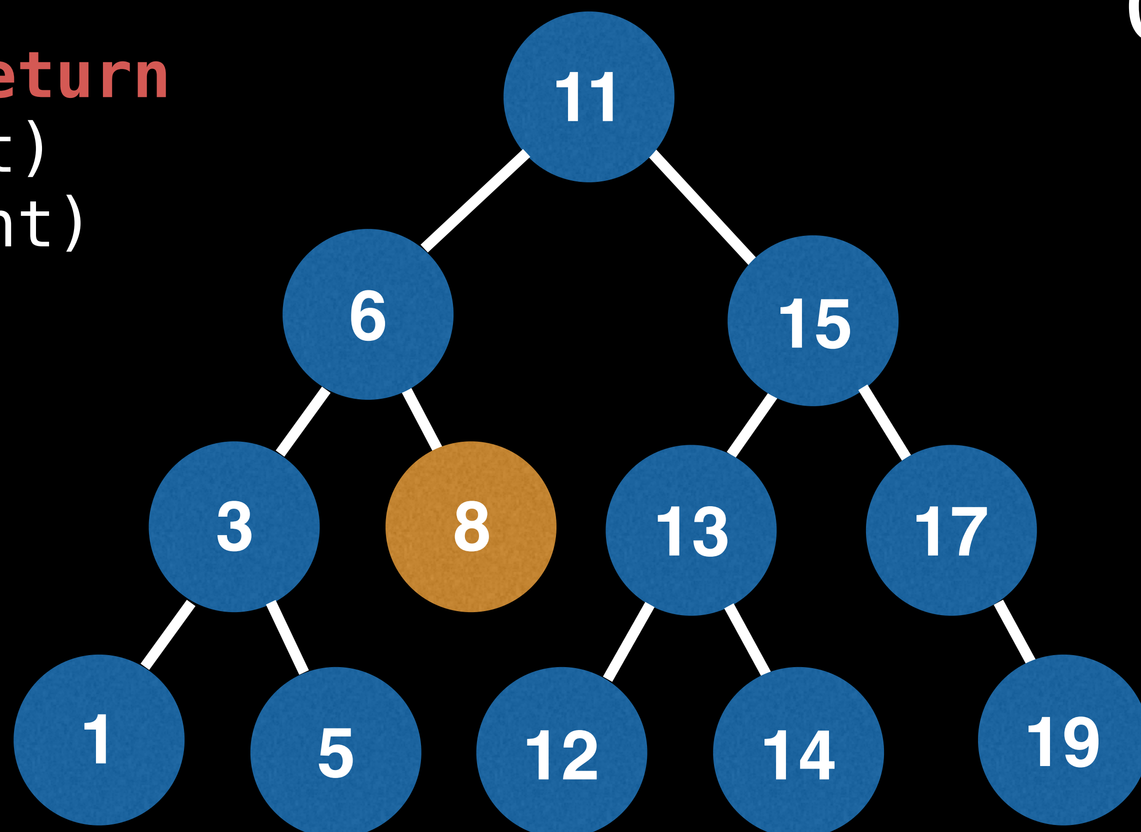
Call Stack:

node 11
node 6

Order: 1,5,3

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



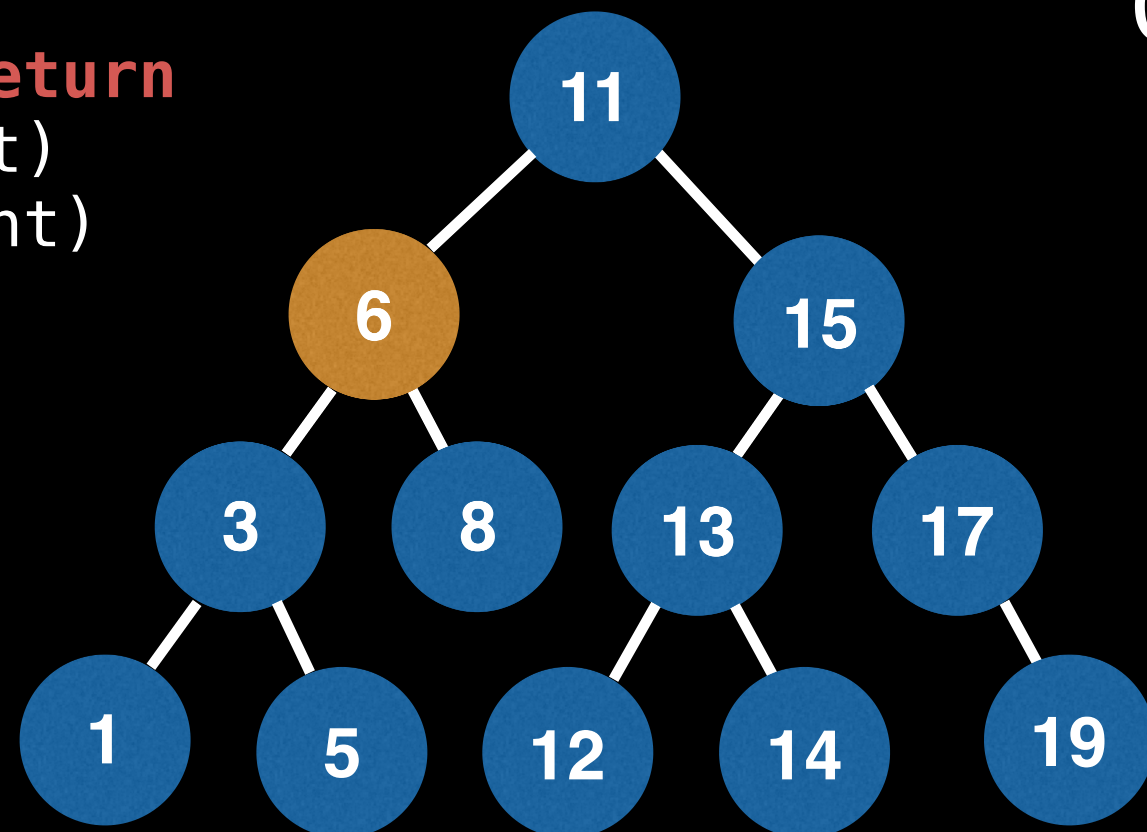
Call Stack:

node 11
node 6
node 8

Order: 1,5,3,8

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



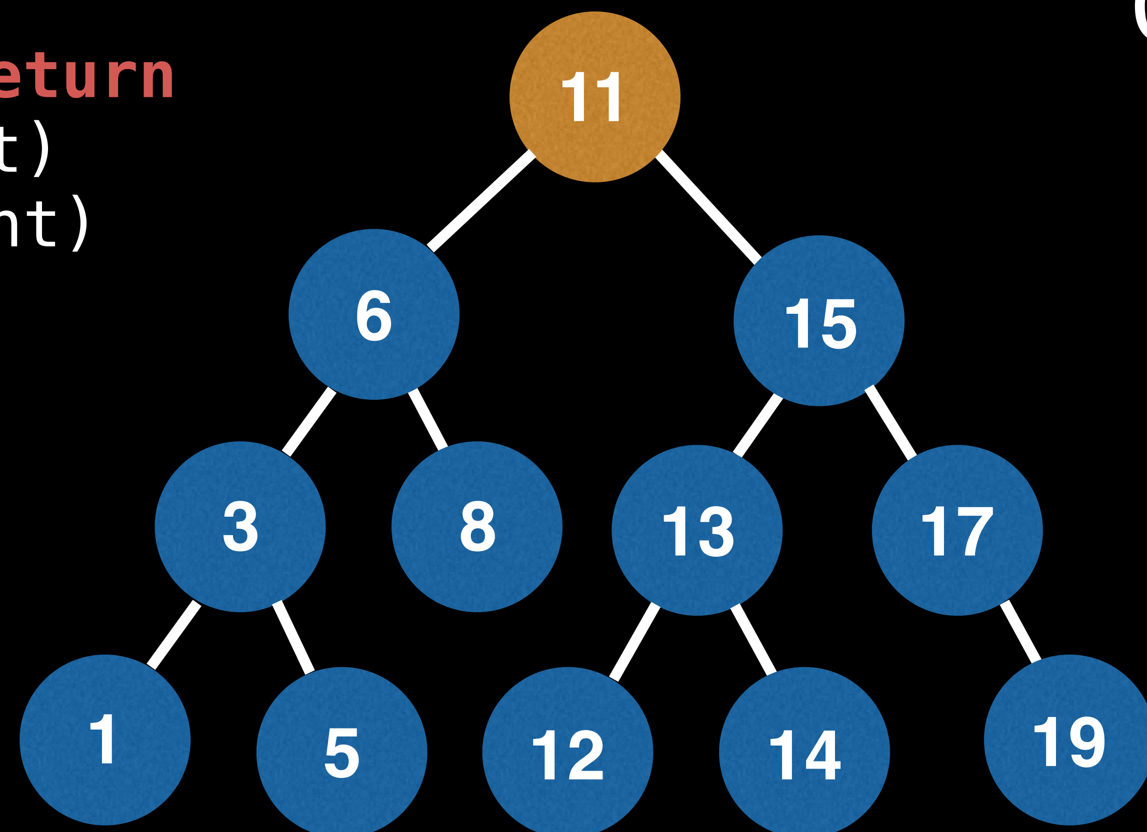
Call Stack:

node 11
node 6

Order: 1,5,3,8,6

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



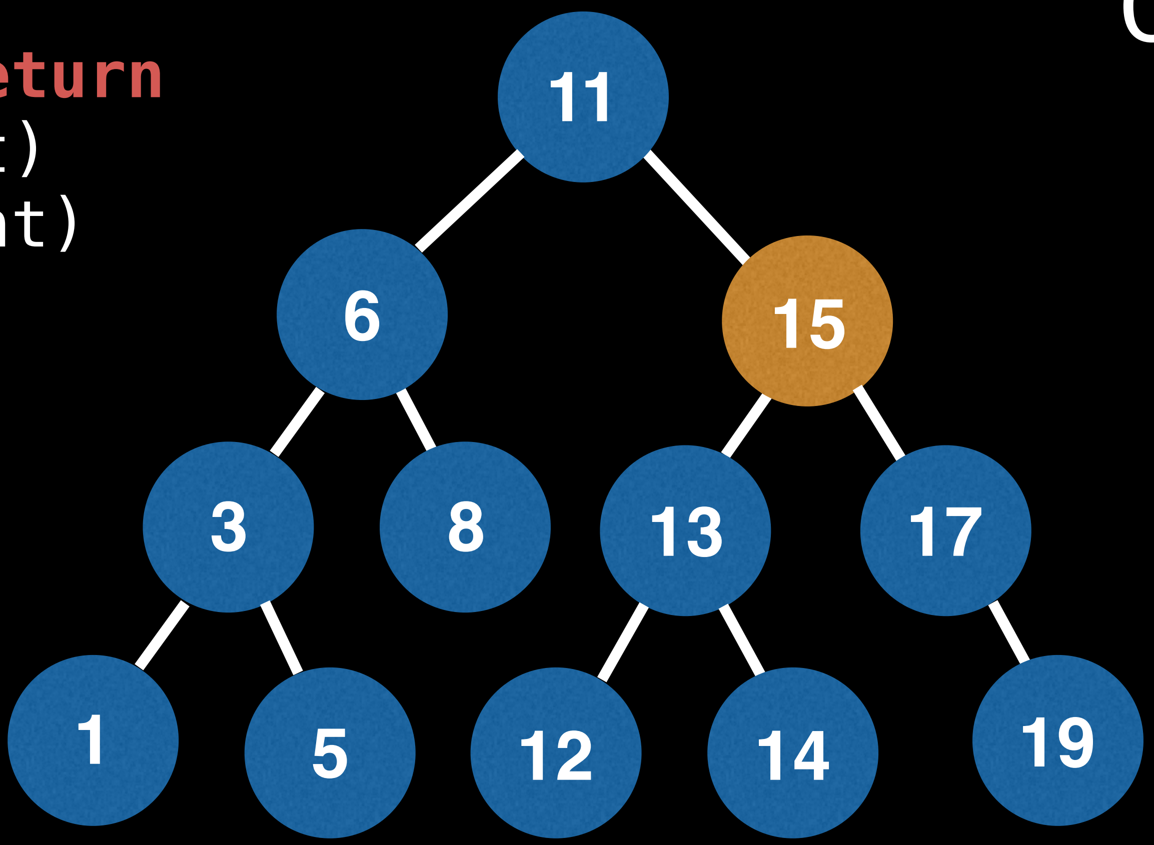
Call Stack:

node 11

Order: 1,5,3,8,6

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



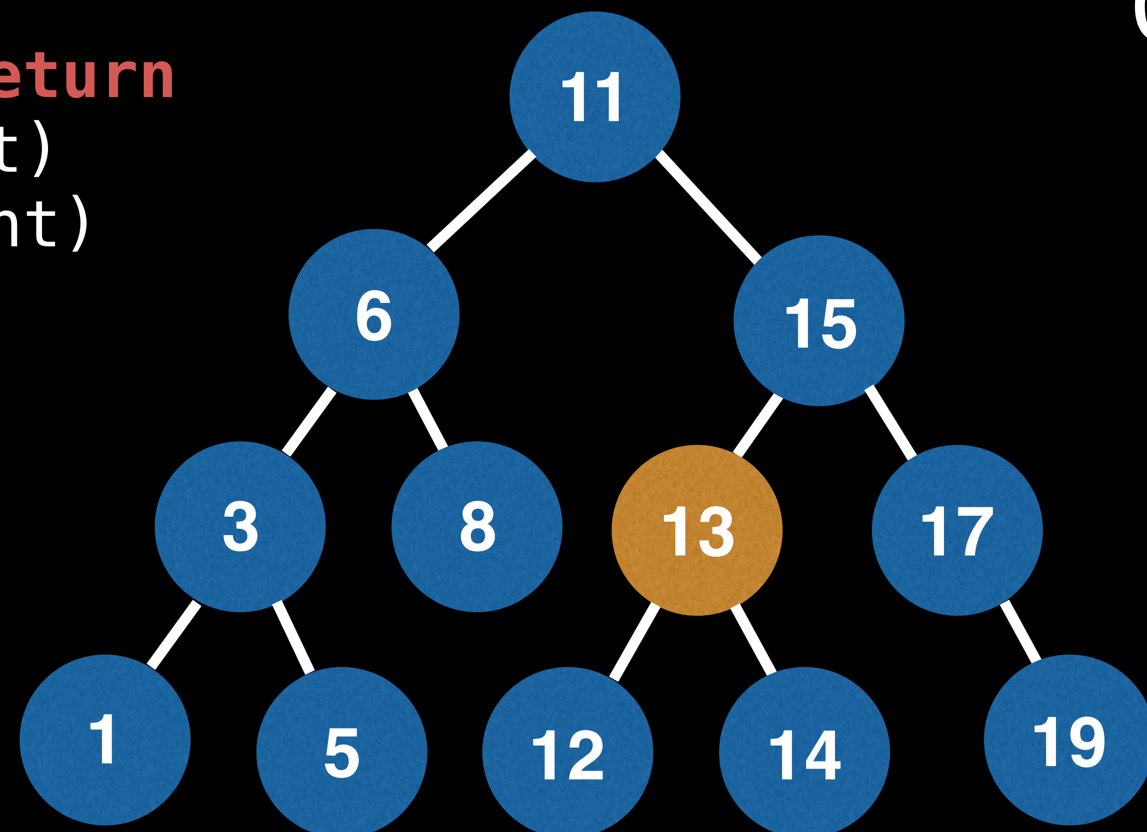
Call Stack:

node 11
node 15

Order: 1,5,3,8,6

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



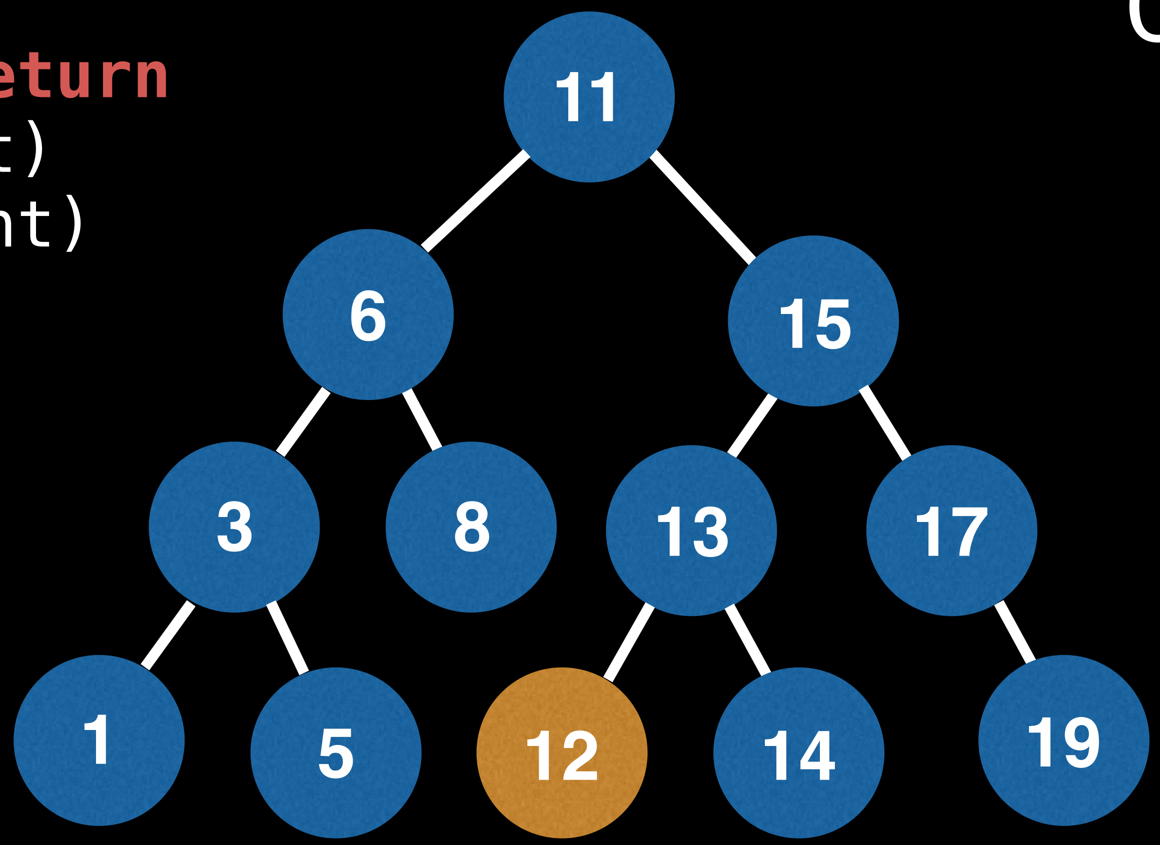
Call Stack:

node 11
node 15
node 13

Order: 1,5,3,8,6

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



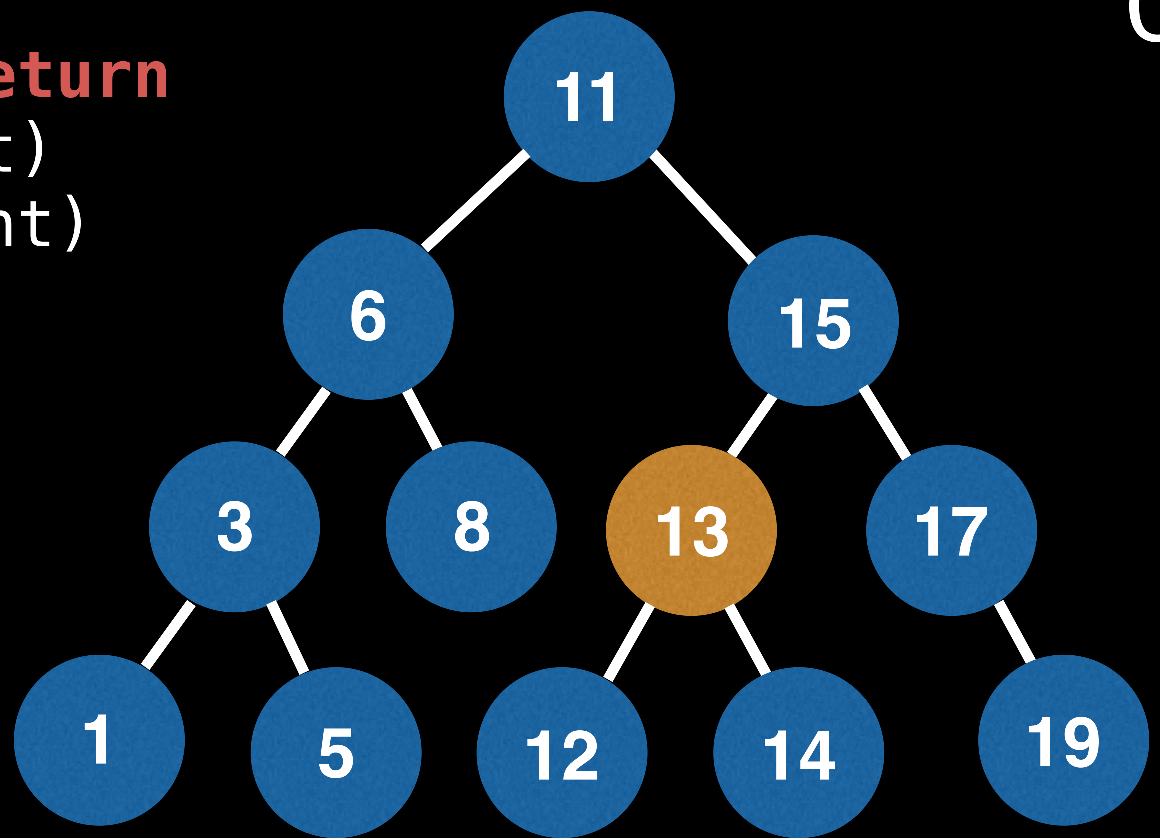
Call Stack:

node 11
node 15
node 13
node 12

Order: 1,5,3,8,6,12

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



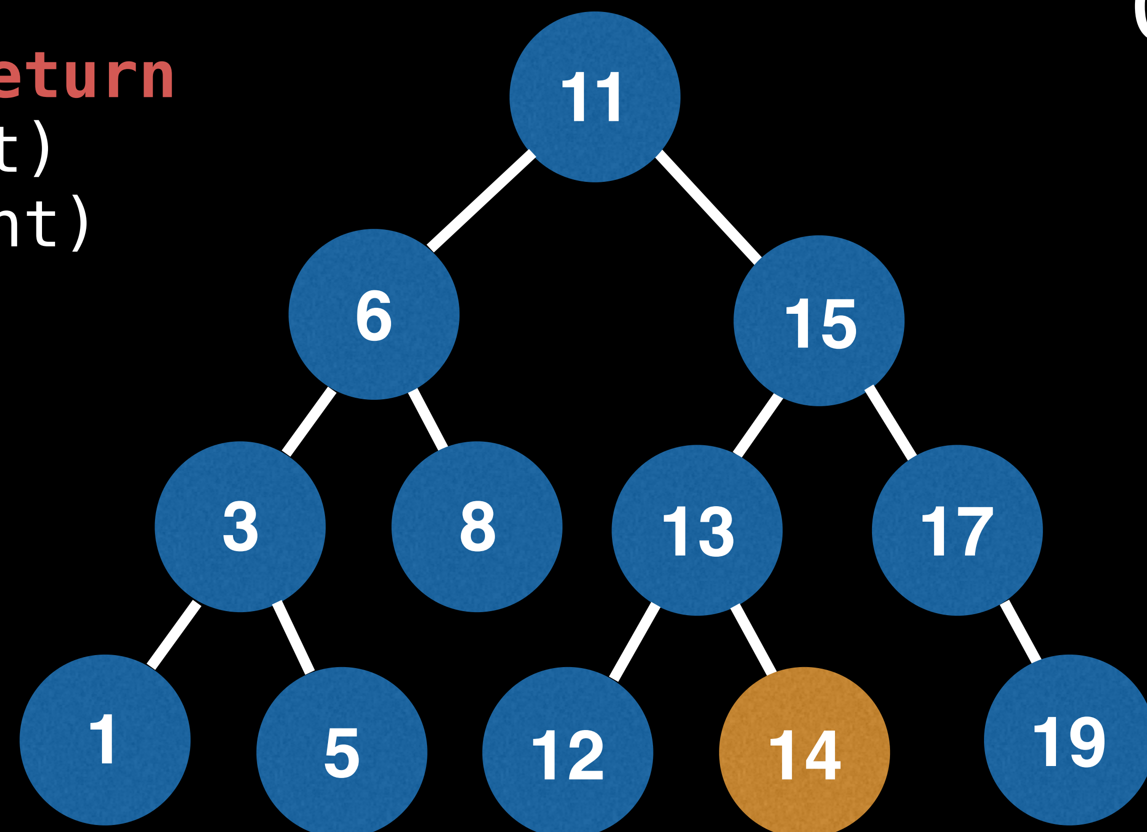
Call Stack:

node 11
node 15
node 13

Order: 1,5,3,8,6,12

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



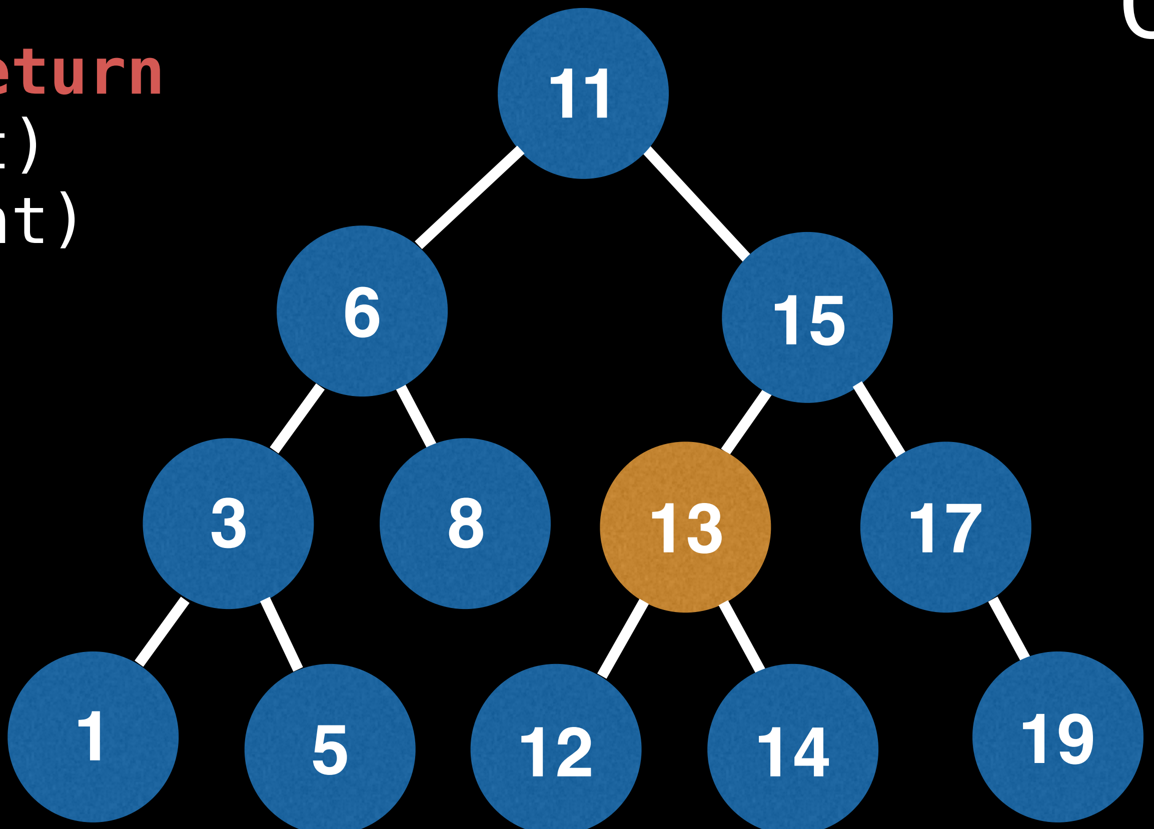
Call Stack:

node 11
node 15
node 13
node 14

Order: 1,5,3,8,6,12,14

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



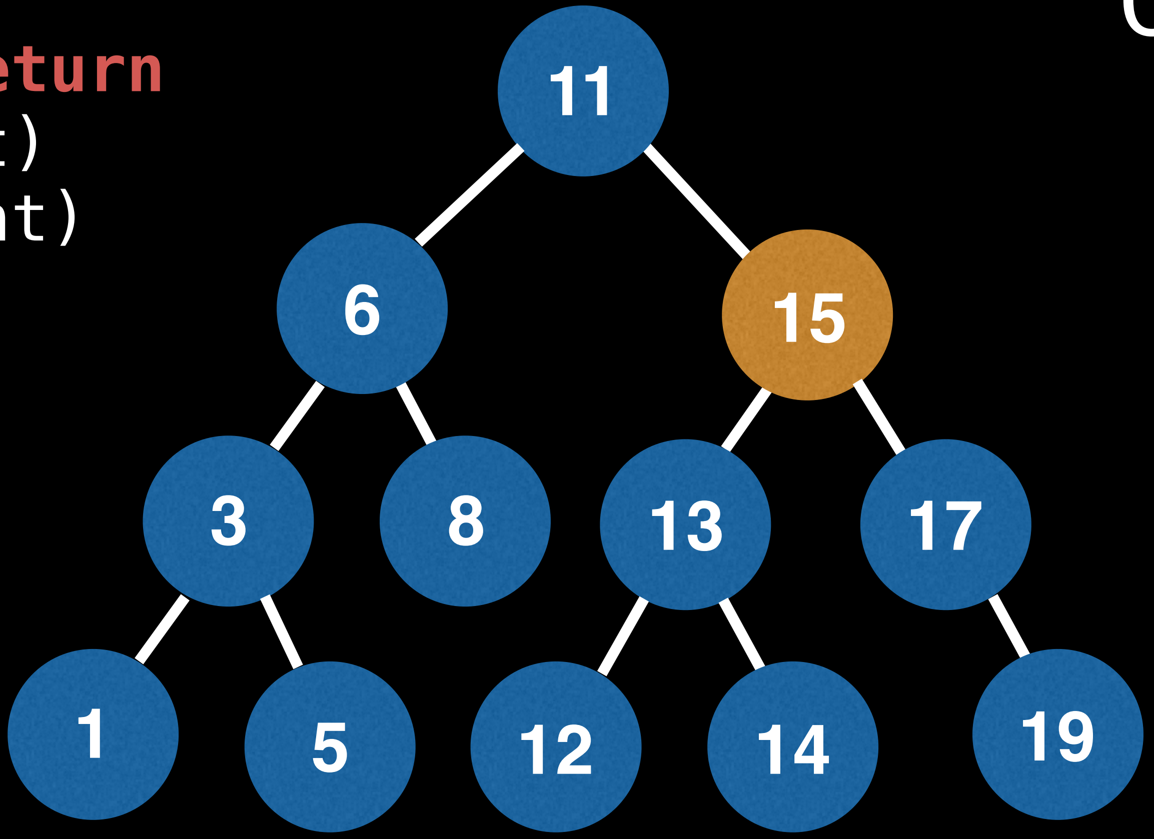
Call Stack:

node 11
node 15
node 13

Order: 1,5,3,8,6,12,14,13

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



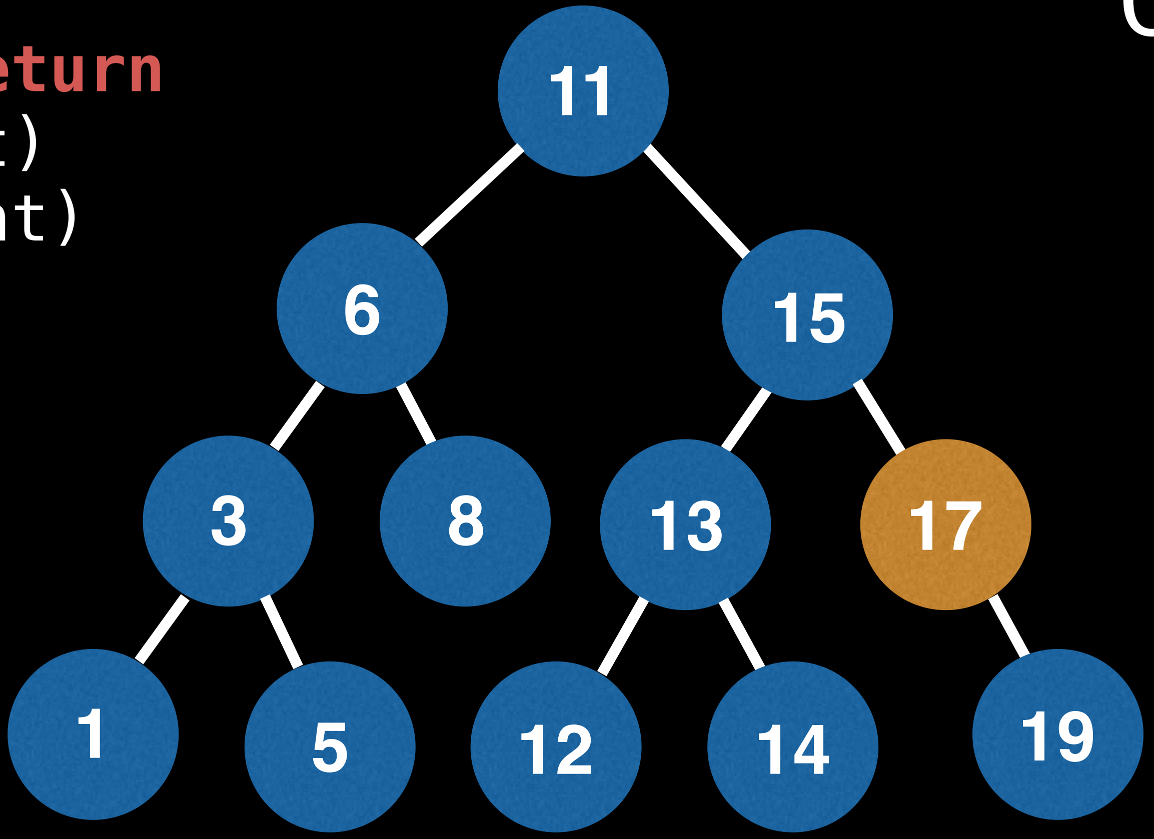
Call Stack:

node 11
node 15

Order: 1,5,3,8,6,12,14,13

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



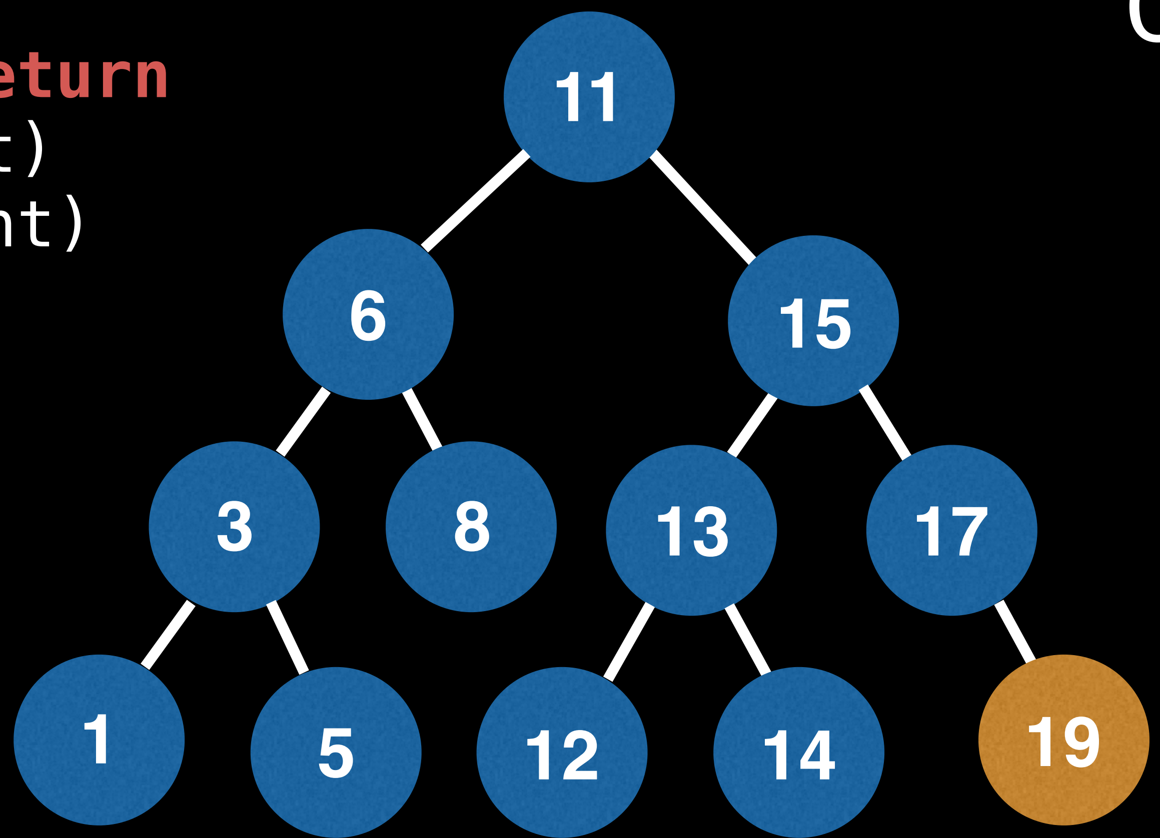
Call Stack:

node 11
node 15
node 17

Order: 1,5,3,8,6,12,14,13

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



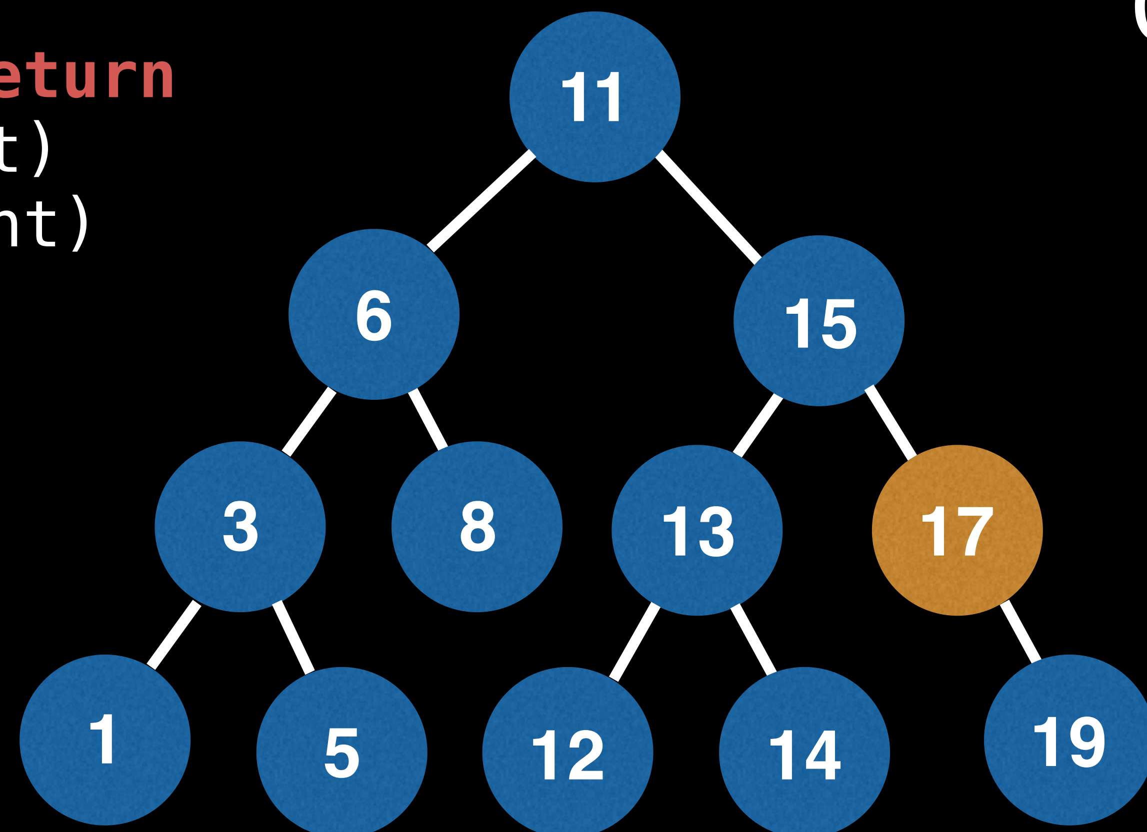
Call Stack:

node 11
node 15
node 17
node 19

Order: 1,5,3,8,6,12,14,13,19

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



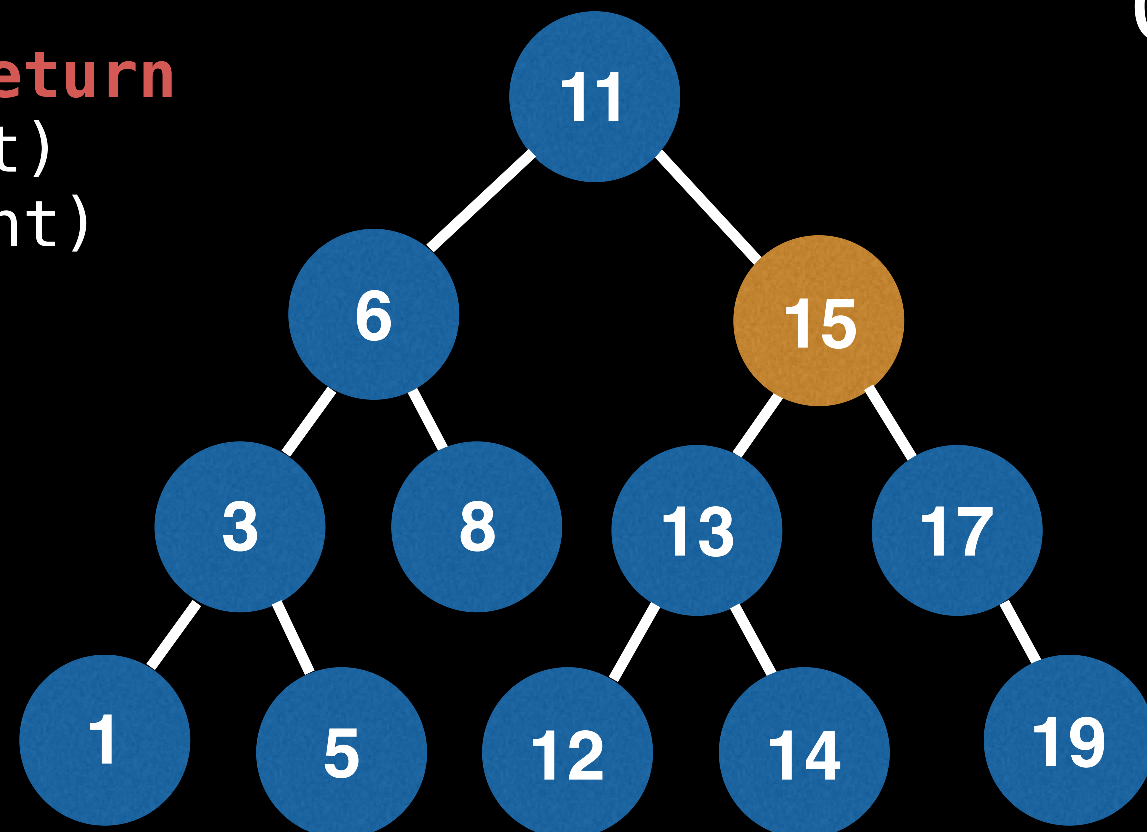
Call Stack:

node 11
node 15
node 17

Order: 1,5,3,8,6,12,14,13,19,17

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```



Call Stack:

node 11
node 15

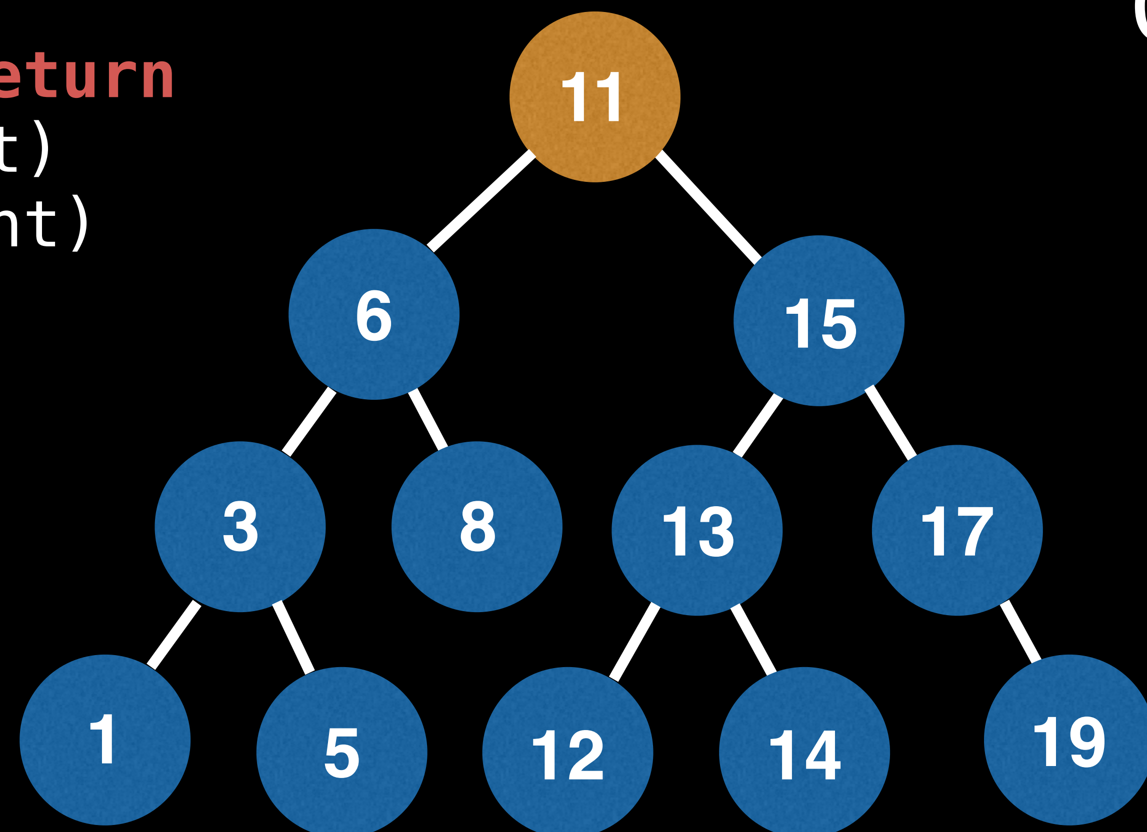
Order: 1,5,3,8,6,12,14,13,19,17,15

Postorder Traversal

```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```

Call Stack:

node 11

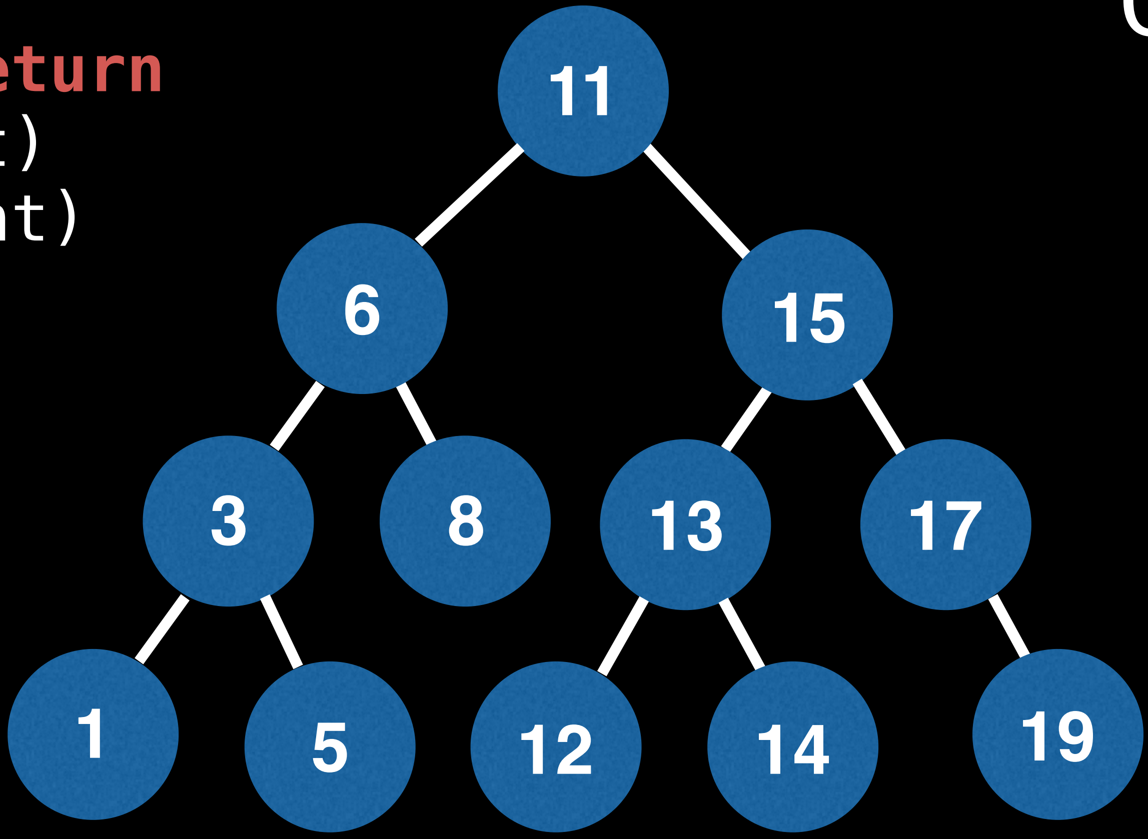


Order: 1,5,3,8,6,12,14,13,19,17,15,11

Postorder Traversal

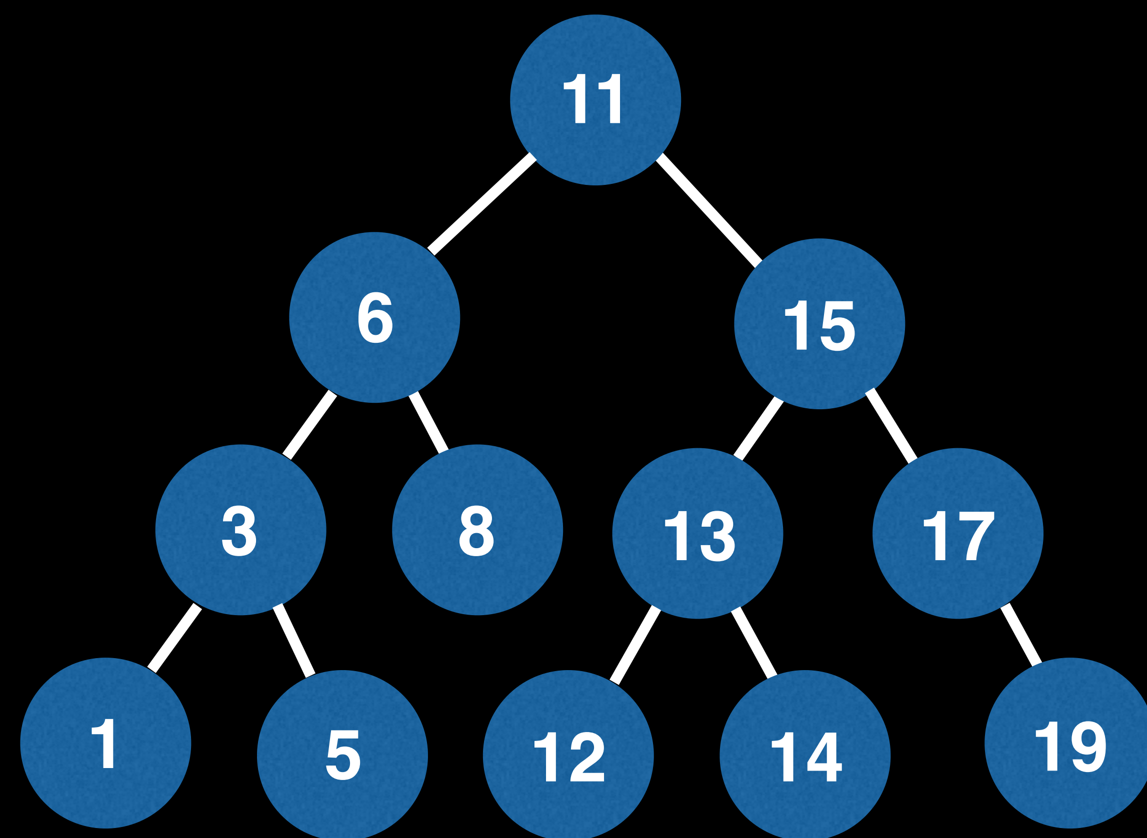
```
postorder(node):
  if node == null: return
  postorder(node.left)
  postorder(node.right)
  print(node.value)
```

Call Stack:



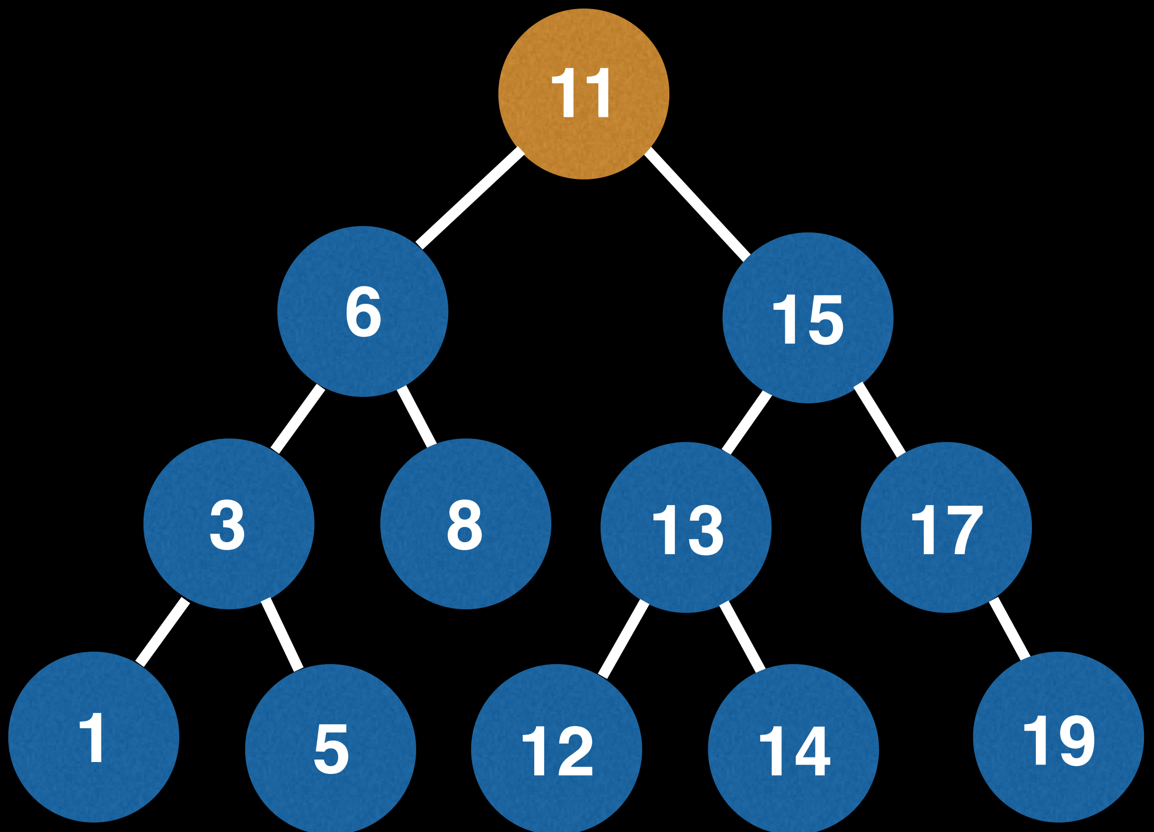
Order: 1,5,3,8,6,12,14,13,19,17,15,11

Level order Traversal



In a level order traversal we want to print the nodes as they appear one layer at a time.

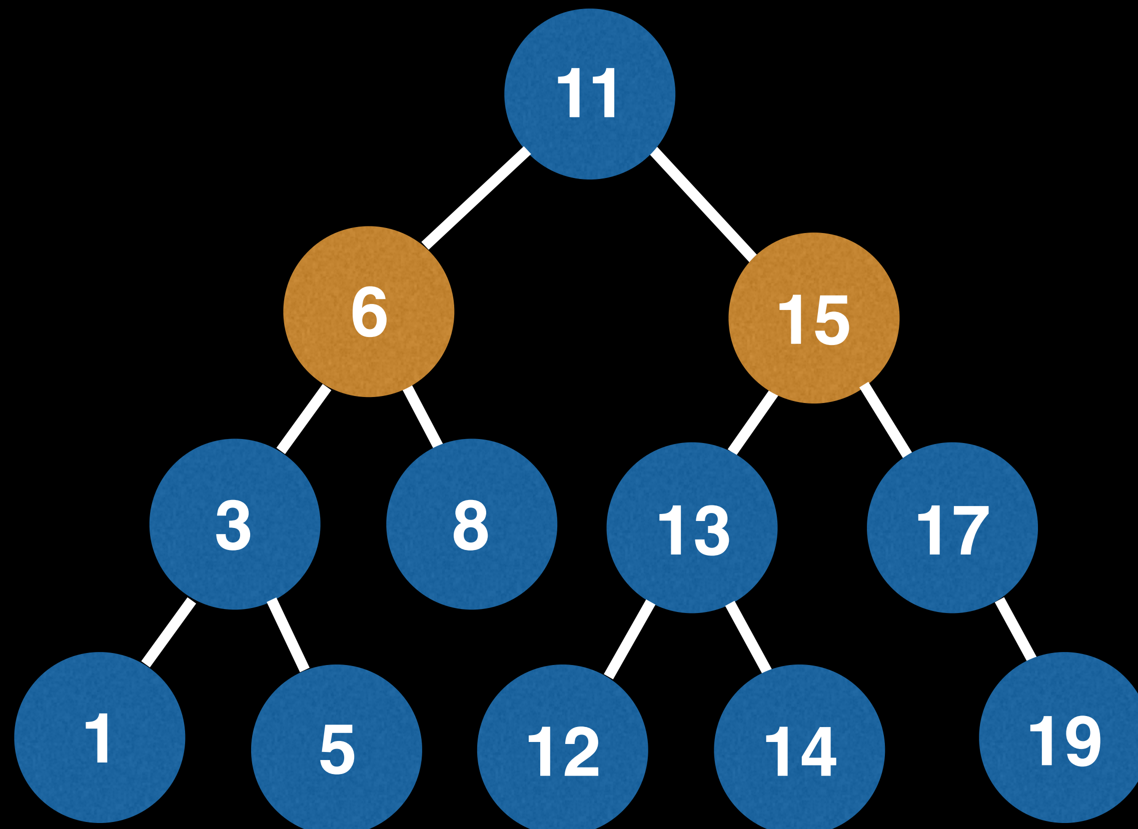
Level order Traversal



In a level order traversal we want to print the nodes as they appear one layer at a time.

Order: 11

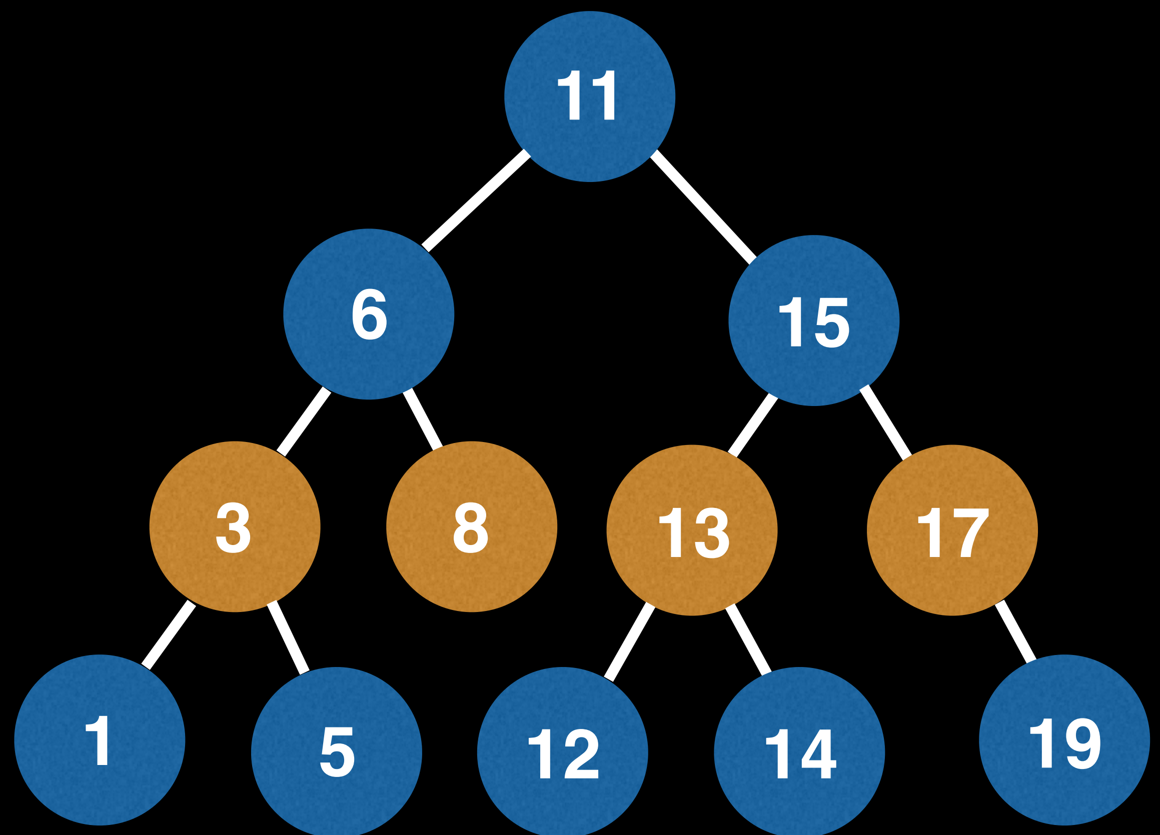
Level order Traversal



In a level order traversal we want to print the nodes as they appear one layer at a time.

Order: 11,6,15

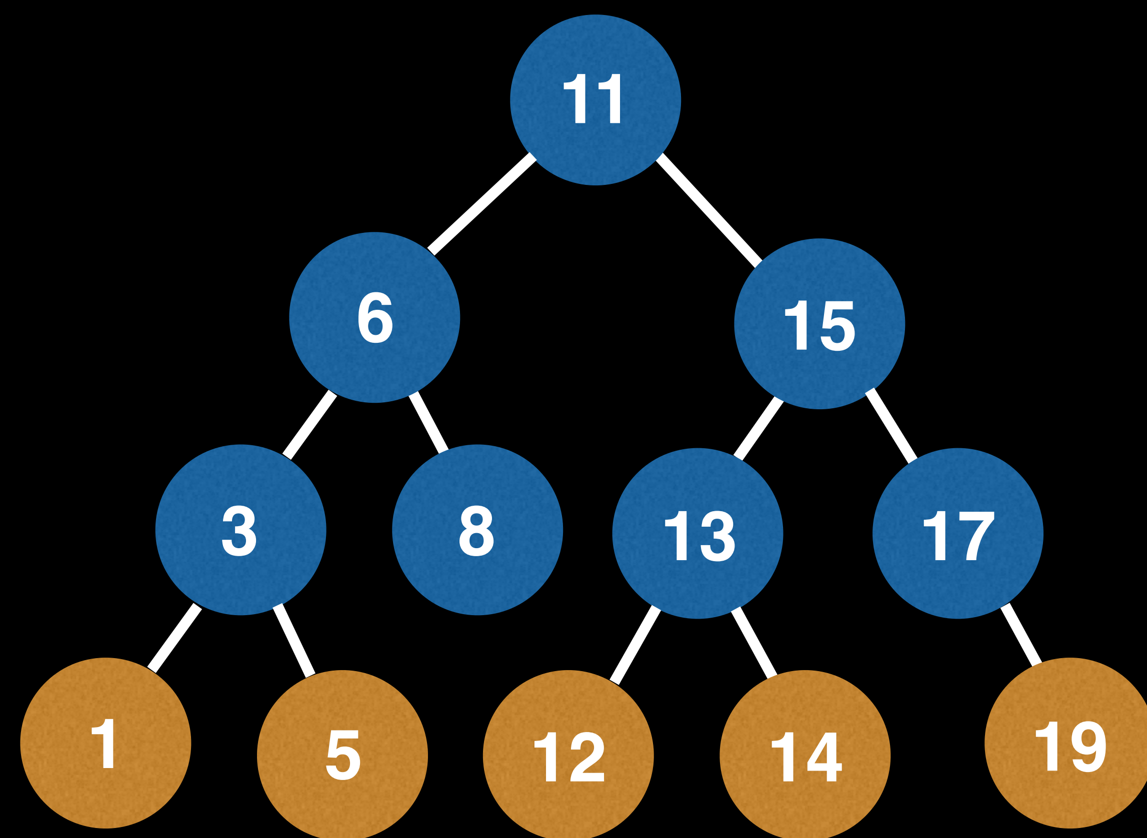
Level order Traversal



In a level order traversal we want to print the nodes as they appear one layer at a time.

Order: 11,6,15,3,8,13,17

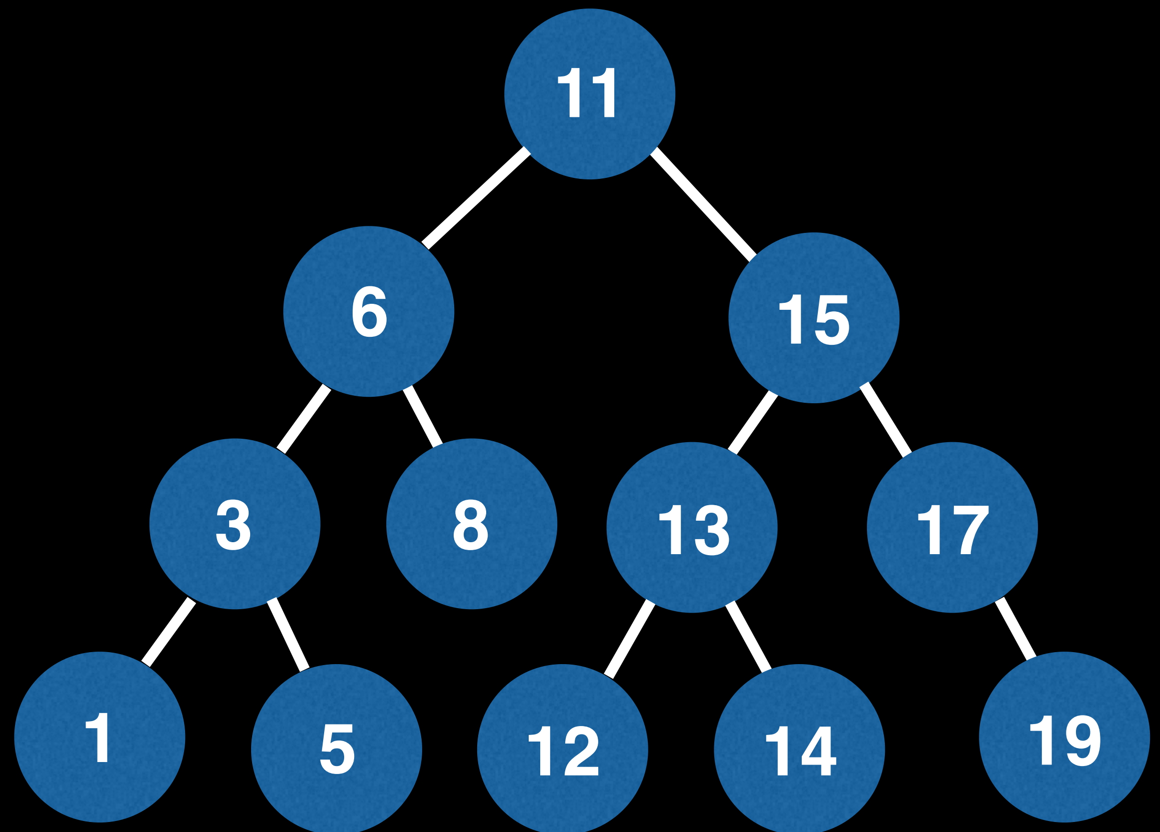
Level order Traversal



In a level order traversal we want to print the nodes as they appear one layer at a time.

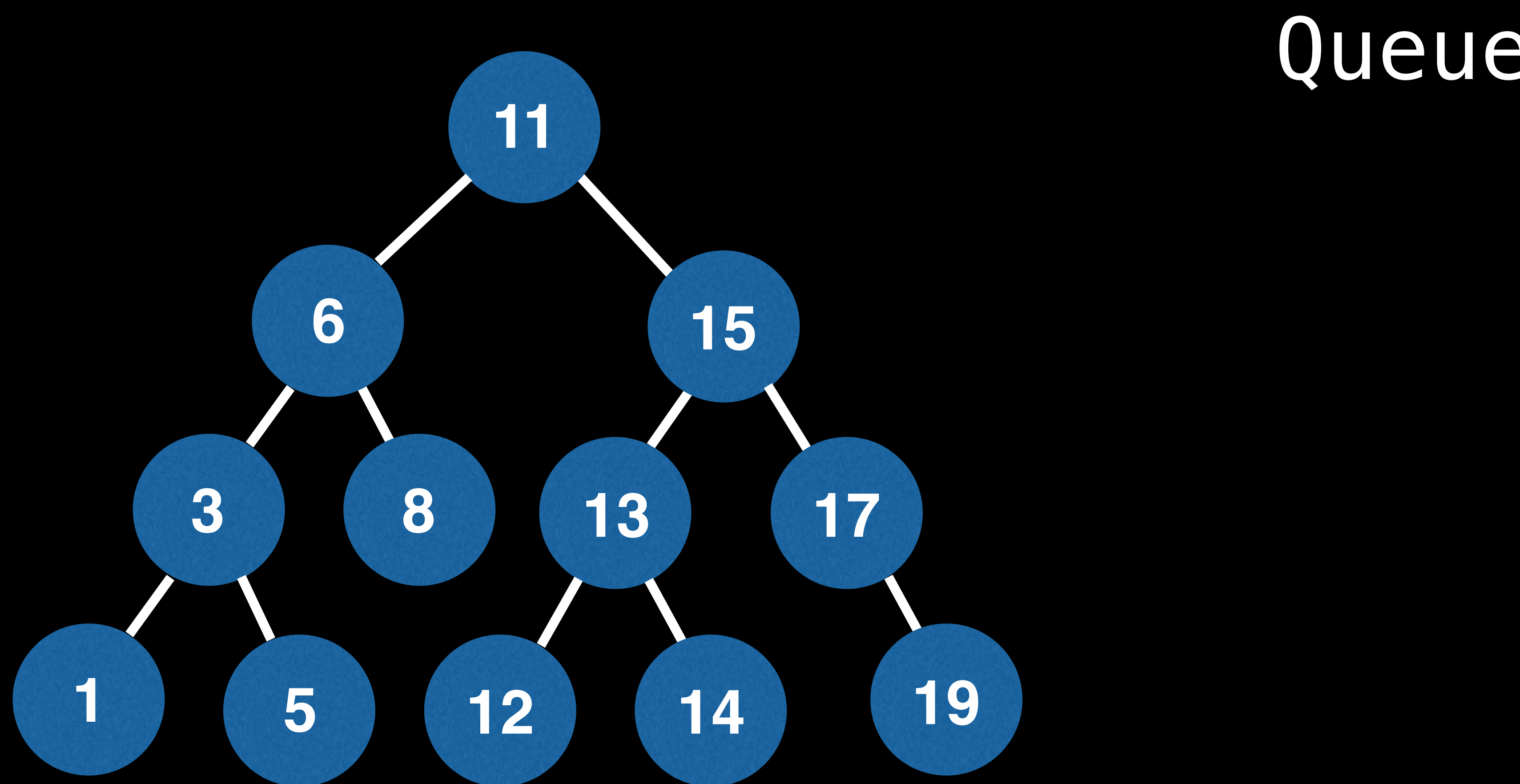
Order: 11,6,15,3,8,13,17,1,5,12,14,19

Level order Traversal



To obtain this ordering we want to do a **Breadth First Search** (BFS) from the root node down to the leaf nodes.

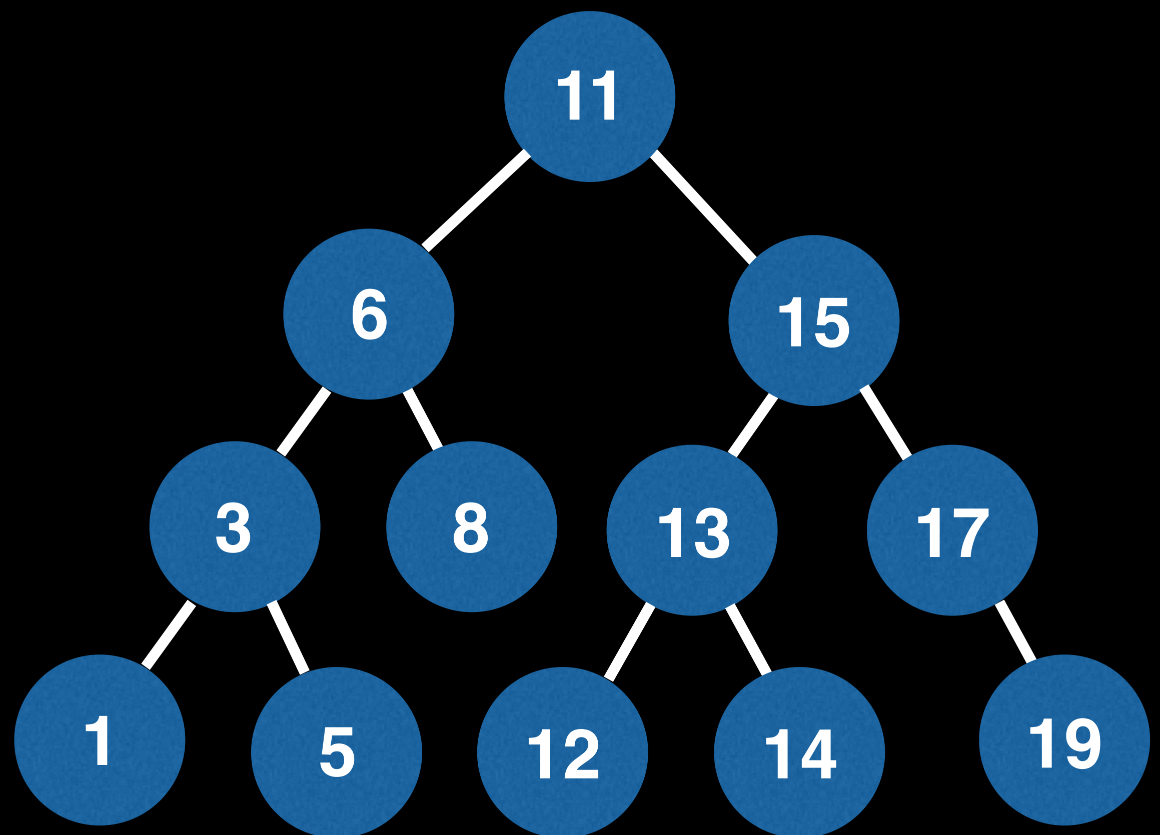
Level order Traversal



To do a BFS we will need to maintain a **Queue** of the nodes left to explore.

Begin with the root inside of the queue and finish when the queue is empty.

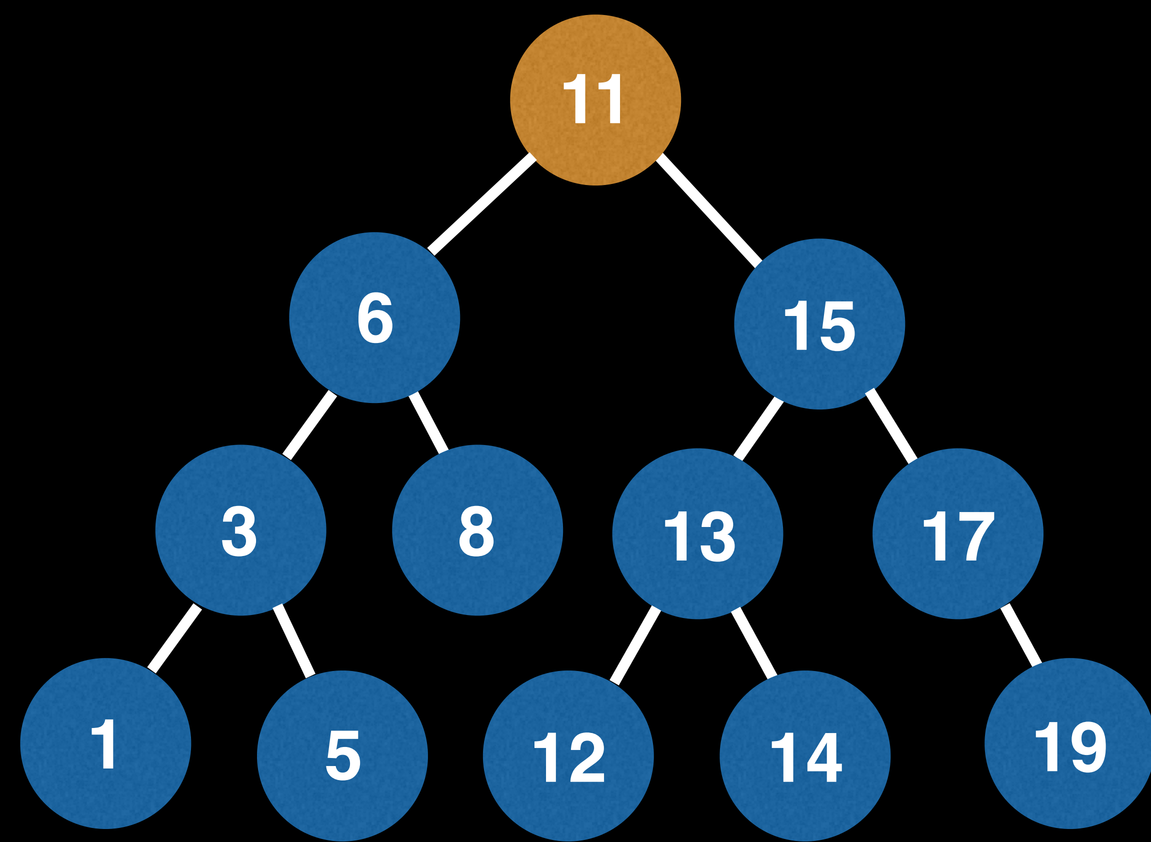
Level order Traversal



Queue
node 11

At each iteration we add the left child and then the right child of the current node to our Queue.

Level order Traversal

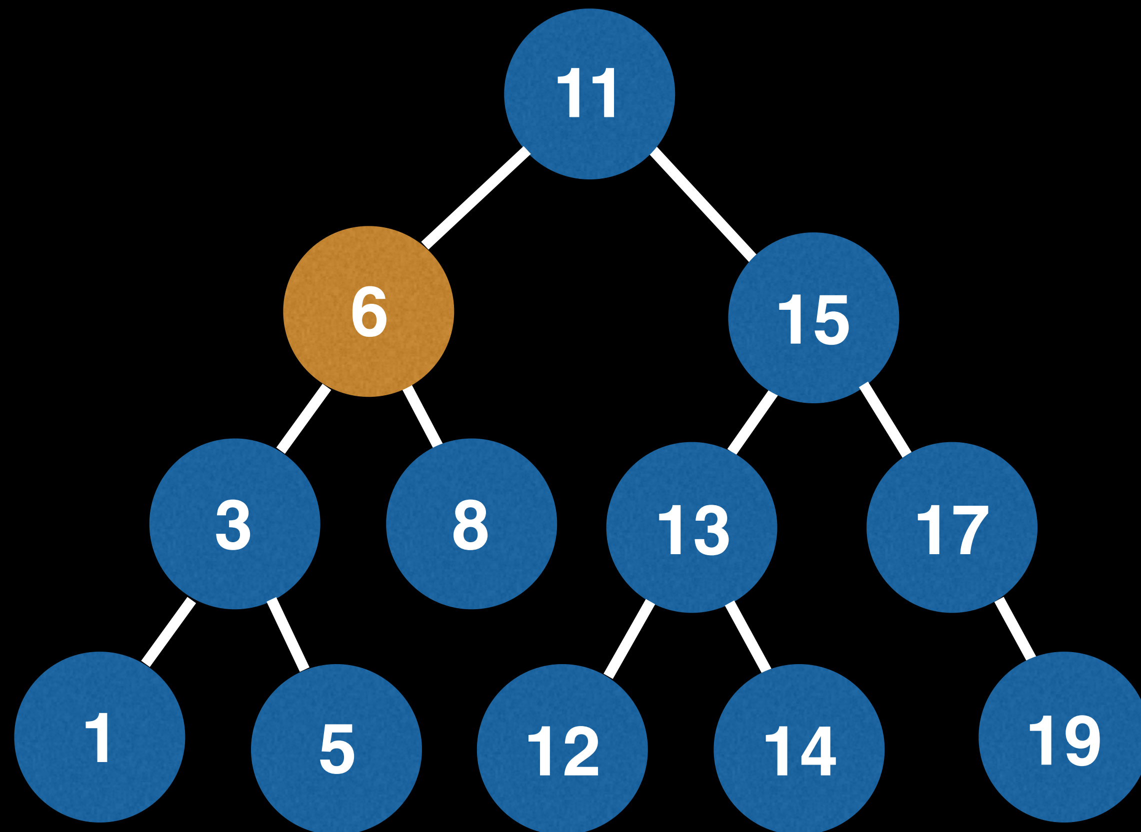


Queue

node 6
node 15

Order: 11

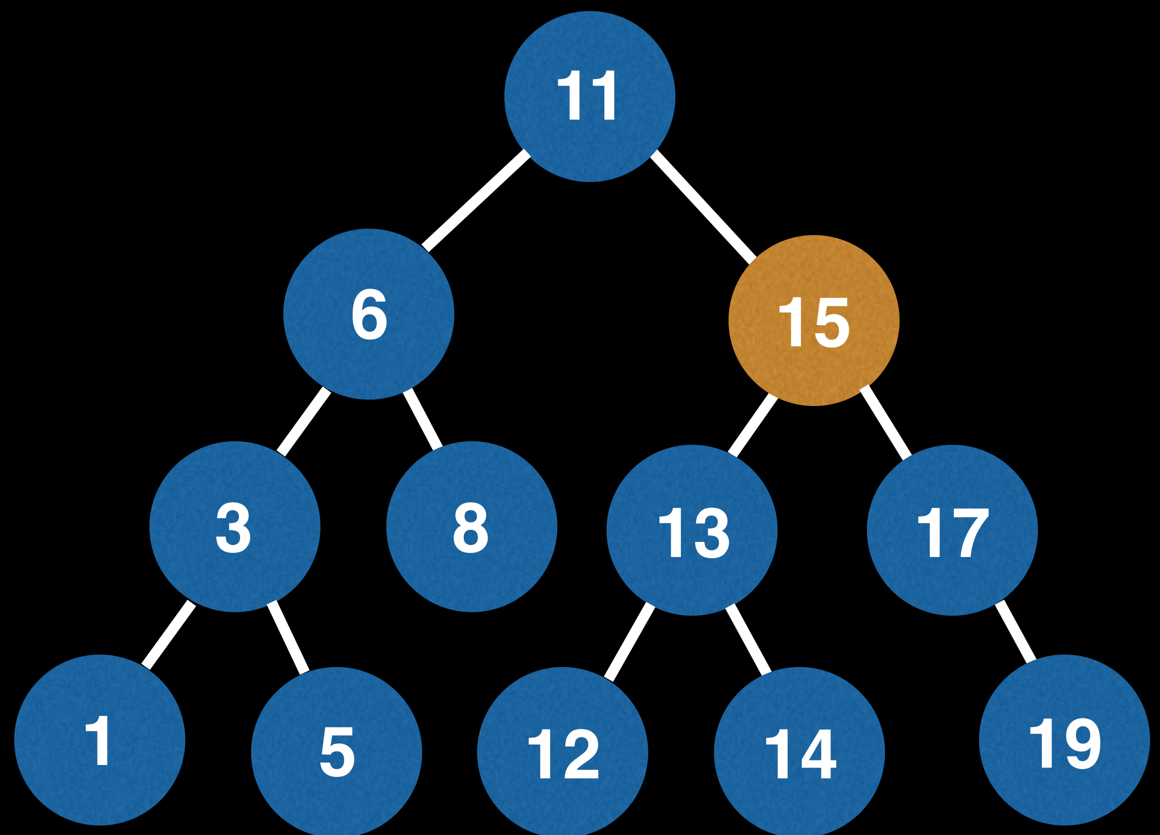
Level order Traversal



Queue
node 15
node 3
node 8

Order: 11,6

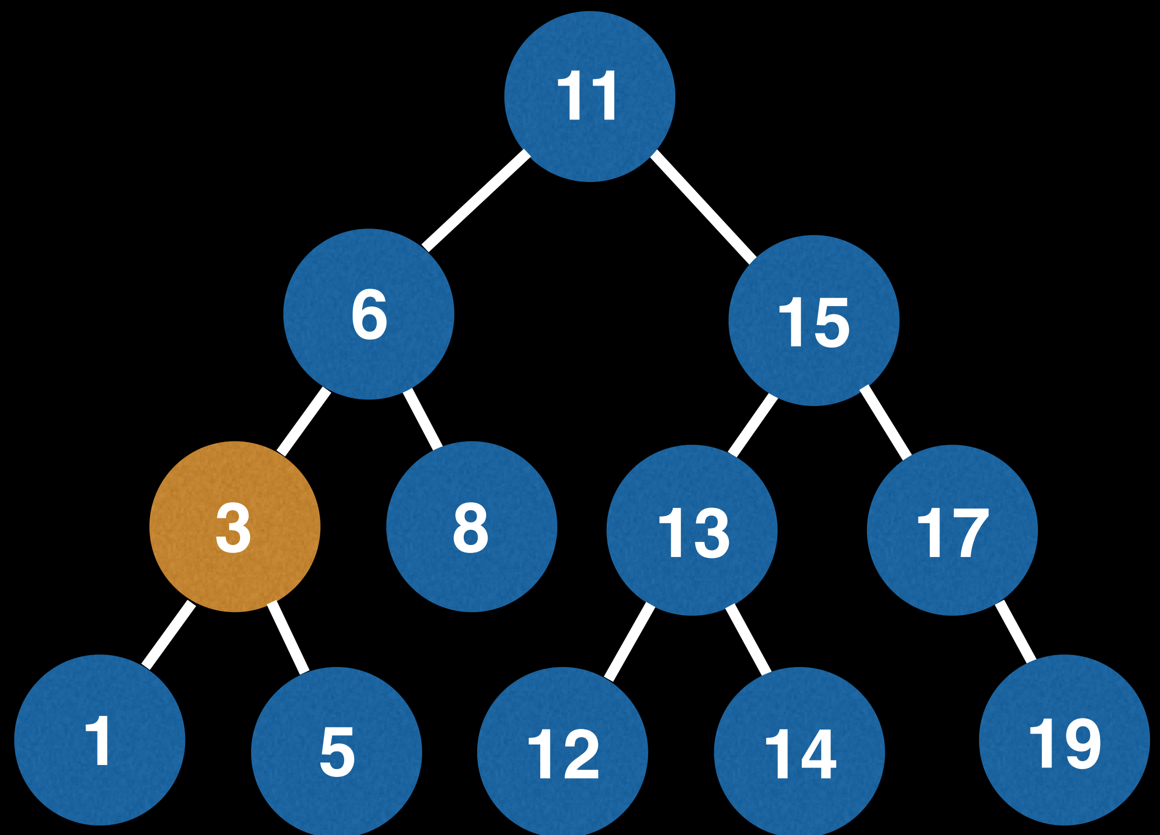
Level order Traversal



Queue
node 3
node 8
node 13
node 17

Order: 11,6,15

Level order Traversal

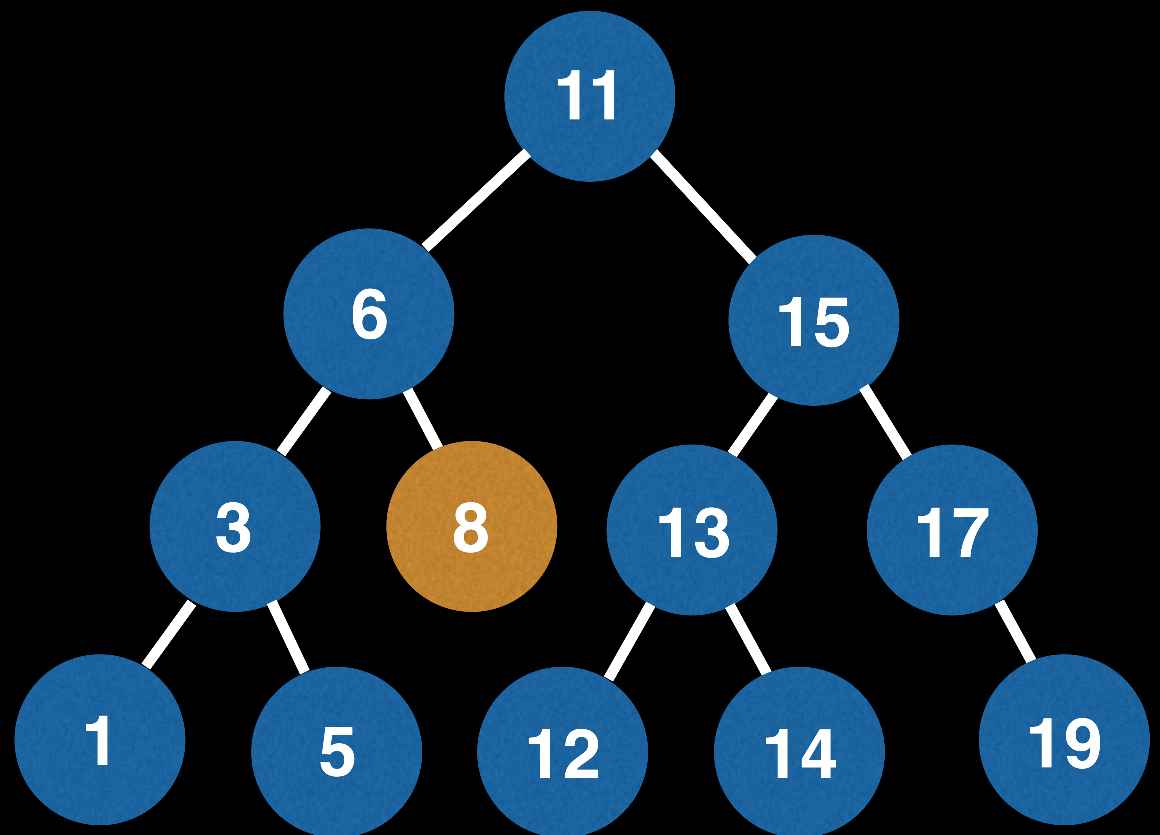


Queue

node 8
node 13
node 17
node 1
node 5

Order: 11,6,15,3

Level order Traversal

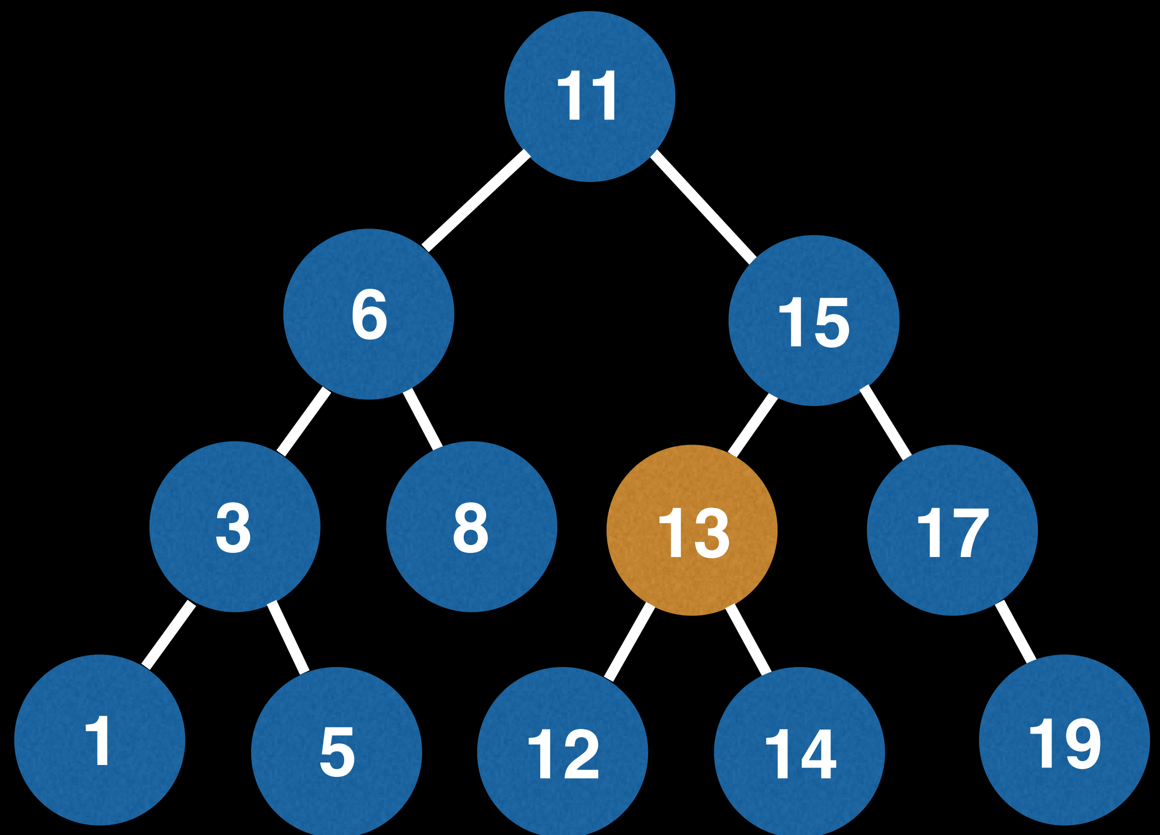


Queue

node 13
node 17
node 1
node 5

Order: 11,6,15,3,8

Level order Traversal

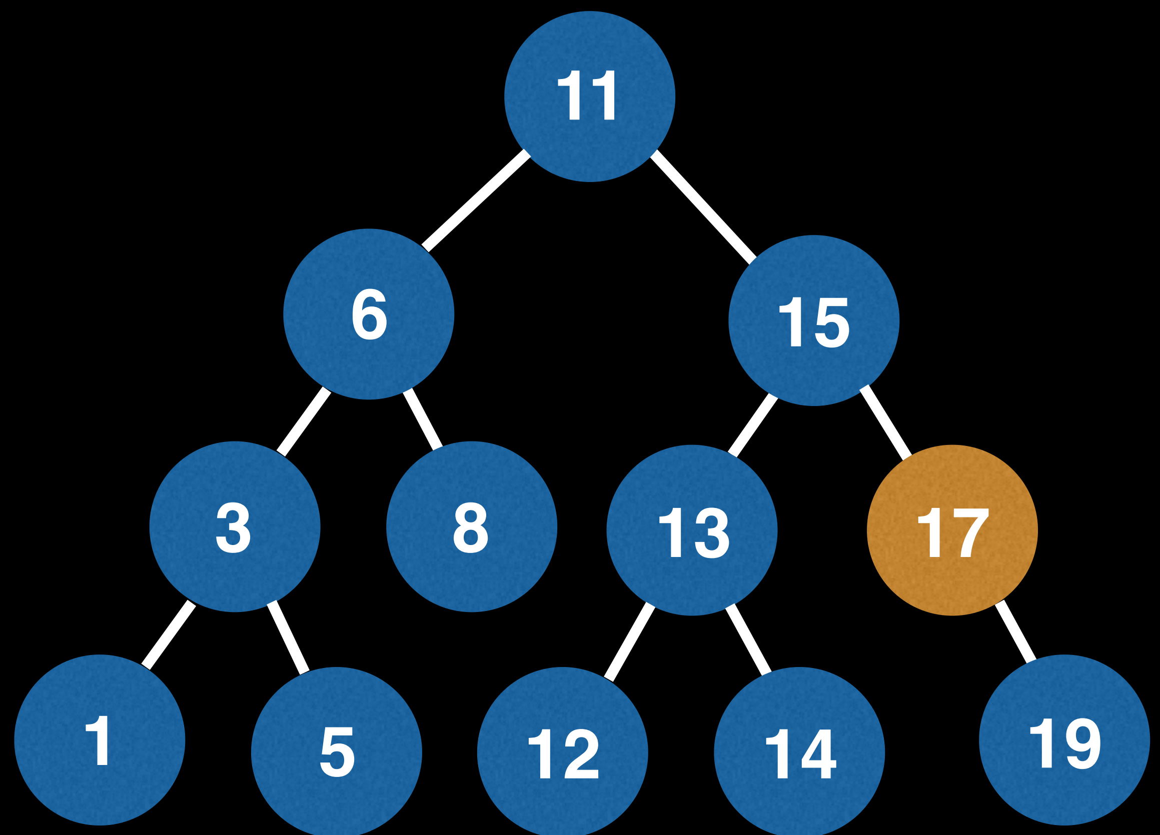


Queue

node 17
node 1
node 5
node 12
node 14

Order: 11,6,15,3,8,13

Level order Traversal



Queue

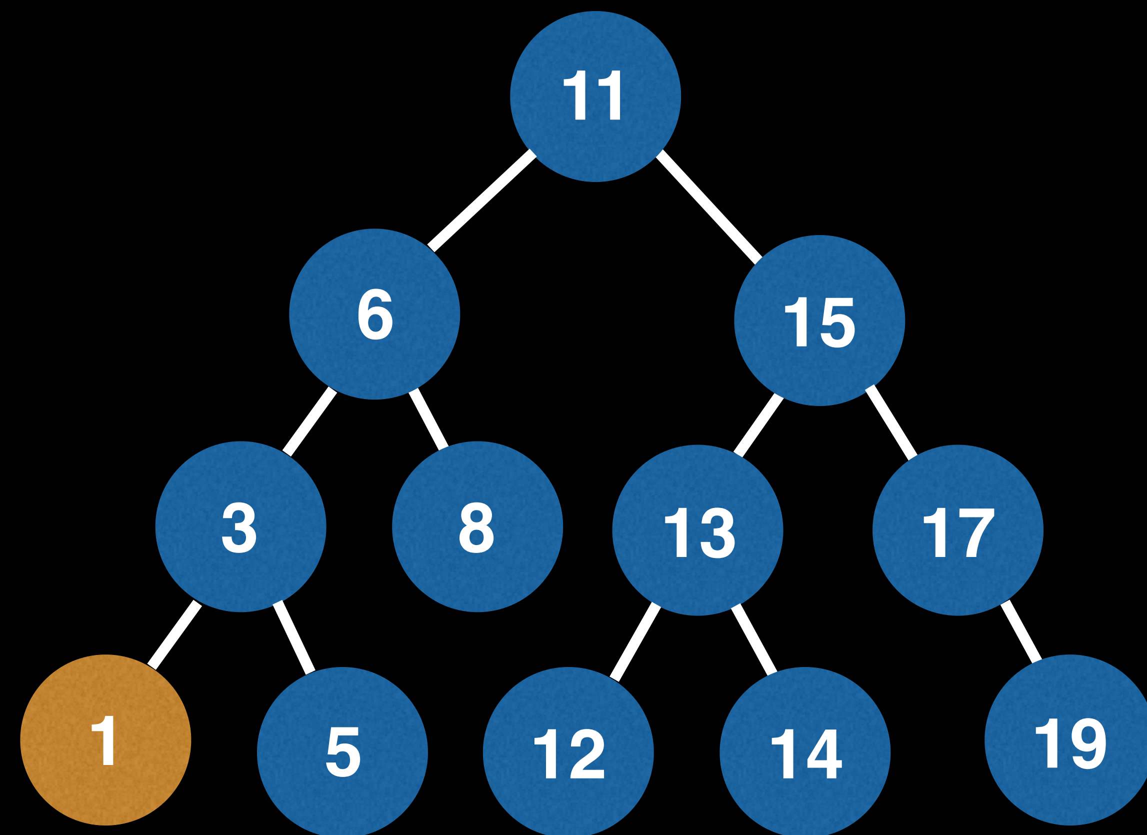
node 1
node 5
node 12
node 14
node 19

Order: 11,6,15,3,8,13,17

Level order Traversal

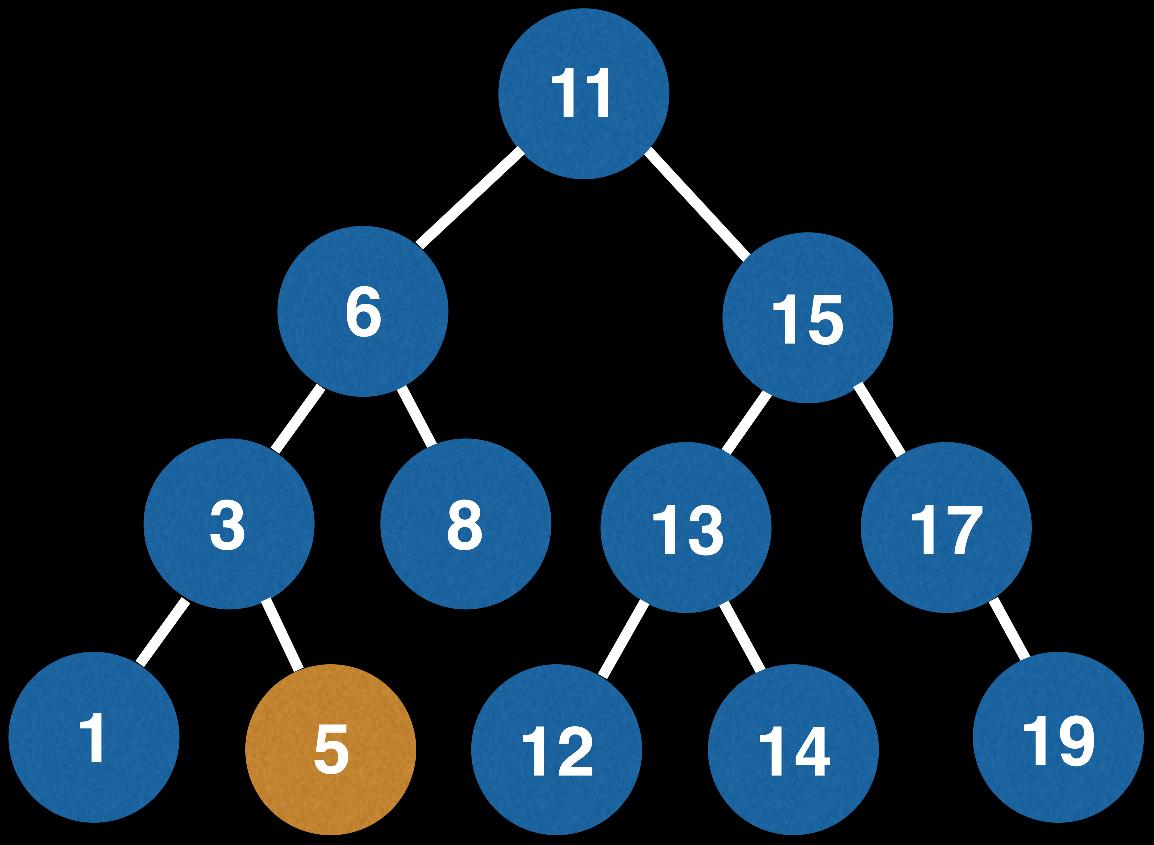
Queue

node 5
node 12
node 14
node 19



Order: 11,6,15,3,8,13,17,1

Level order Traversal



Queue

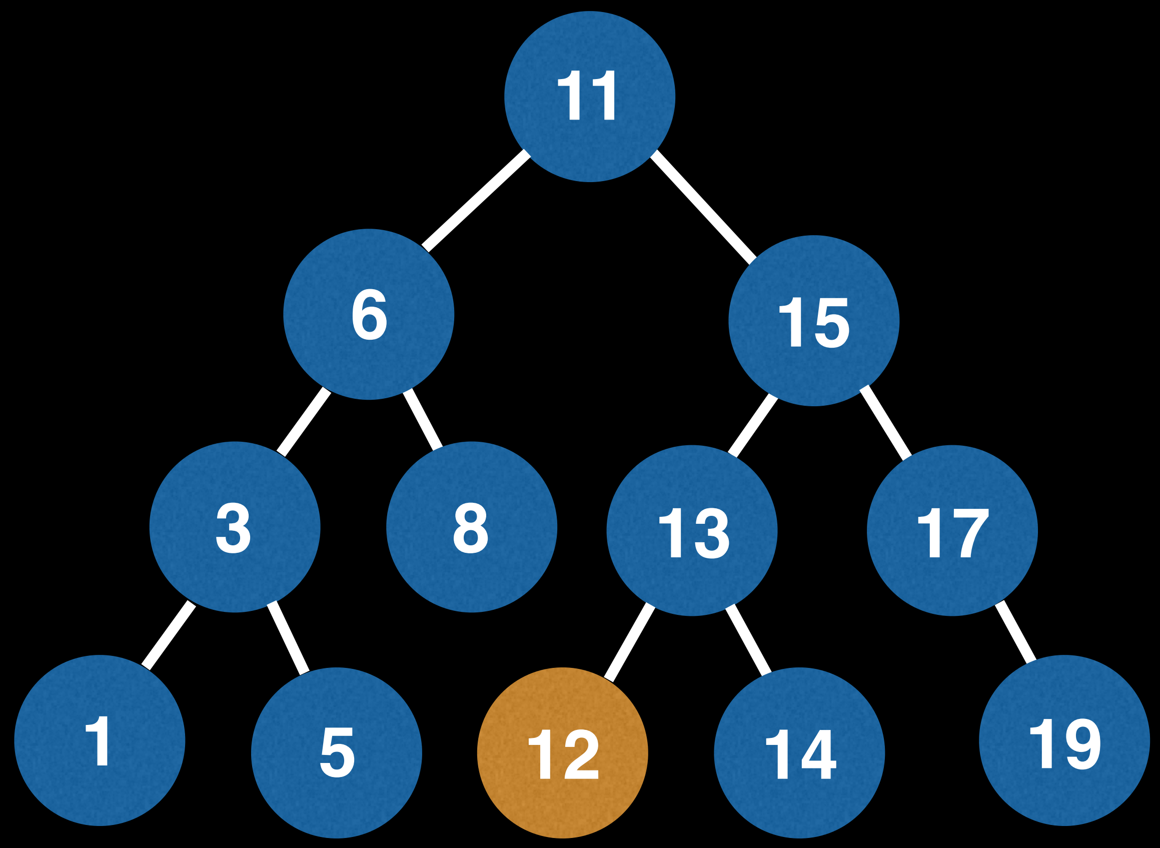
node 12
node 14
node 19

Order: 11,6,15,3,8,13,17,1,5

Level order Traversal

Queue

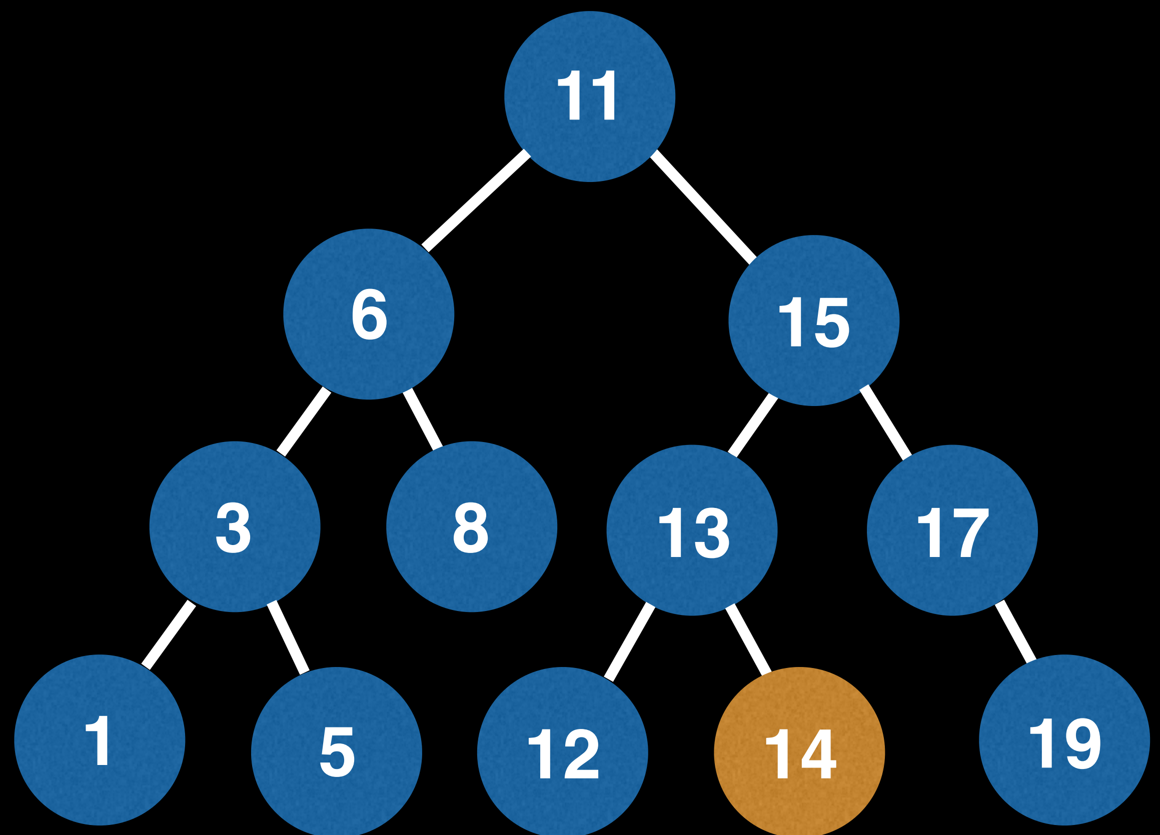
node 14
node 19



Order: 11,6,15,3,8,13,17,1,5,12

Level order Traversal

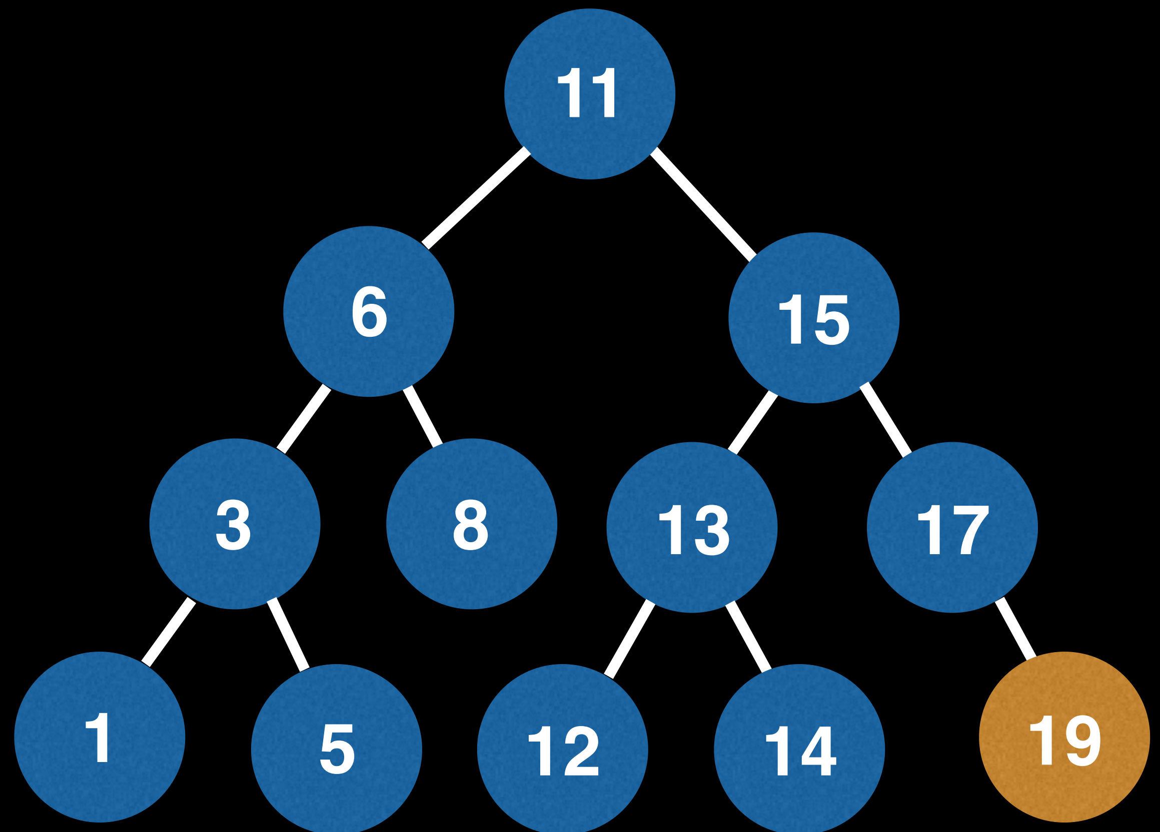
Queue
node 19



Order: 11,6,15,3,8,13,17,1,5,12,14

Level order Traversal

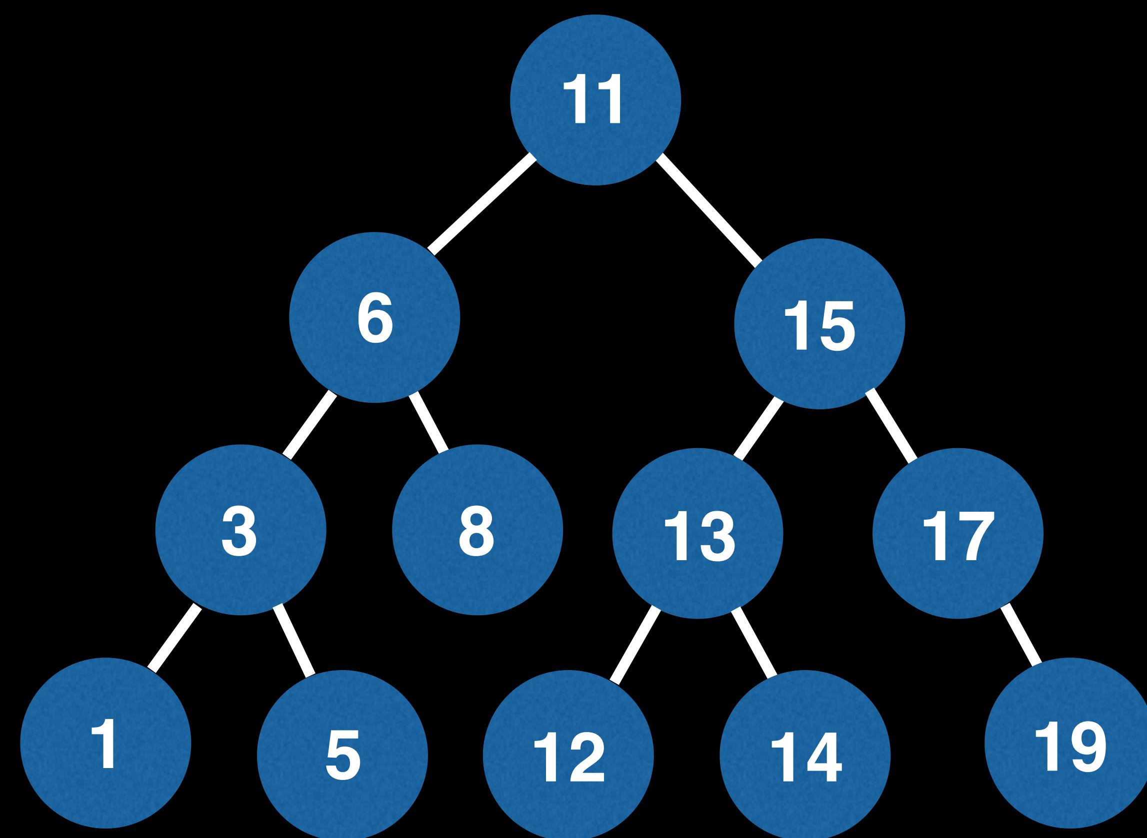
Queue



Order: 11,6,15,3,8,13,17,1,5,12,14,19

Level order Traversal

Queue



Order: 11,6,15,3,8,13,17,1,5,12,14,19