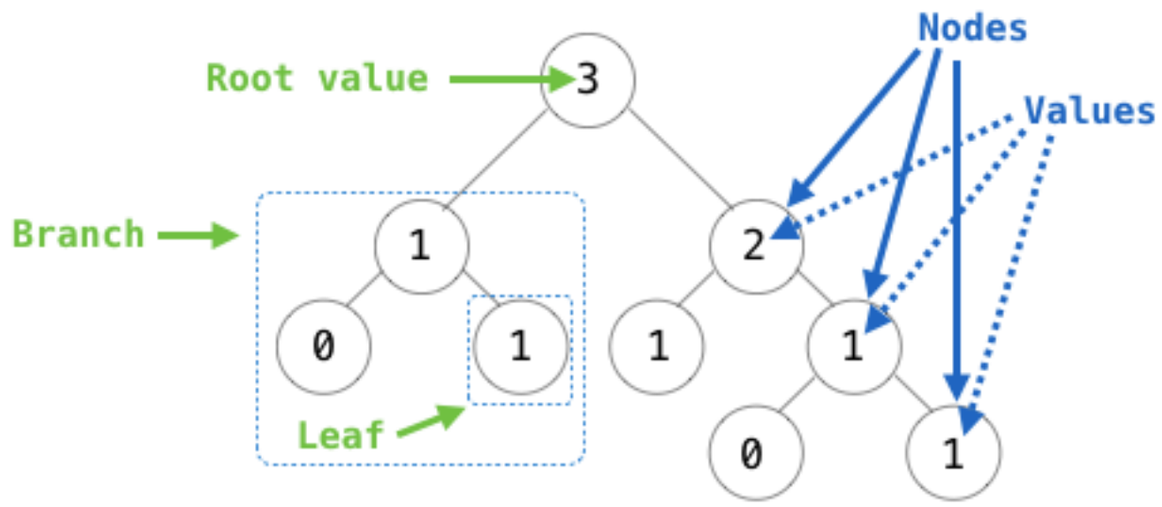
Discussion 03

Trees for Days

What's a tree?



Recursive description (wooden trees):

A **tree** has a **root** value and a list of **branches**

Each branch is a **tree**

A tree with zero branches is called a **leaf**

Relative description (family trees):

Each location in a tree is called a **node**

Each **node** has a **value**

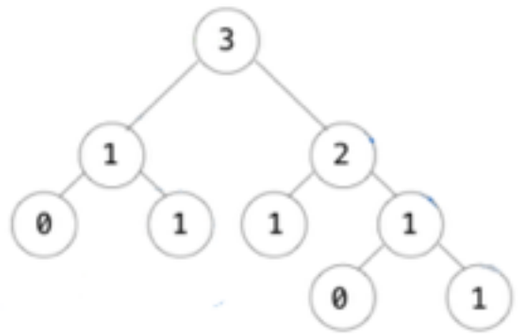
One node can be the **parent**/**child** of another

People often refer to values by their locations: "each parent is the sum of its children"

This slide from lecture covers all of the terminology we use to discuss trees
Memorize this slide.

Python and Trees

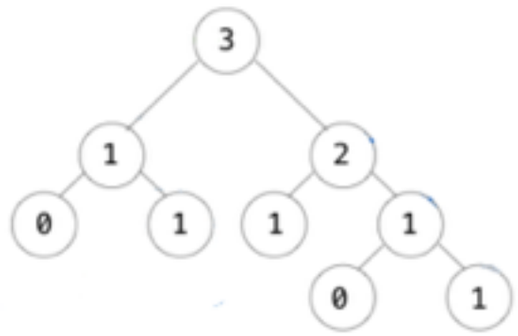
Now we want to translate the tree from the previous slide into Python.
How do we **represent** trees? We don't know a lot of data types yet. So let's use lists!



Python and Trees

Now we want to translate the tree from the previous slide into Python. How do we **represent** trees? We don't know a lot of data types yet. So let's use lists!

```
def tree(root, branches=[]):
    return [root, list(branches)]
```



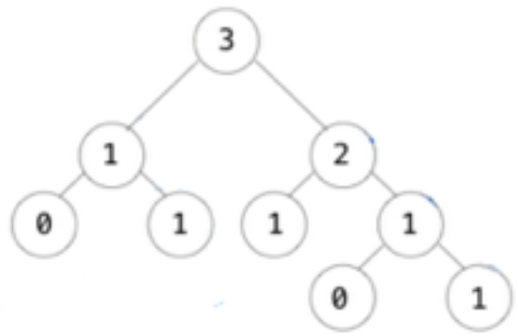
Python and Trees

Now we want to translate the tree from the previous slide into Python. How do we **represent** trees? We don't know a lot of data types yet. So let's use lists!

This is no different from what you have already worked with!
We used lists to represent the latitude and longitude in lab04

```
def tree(root, branches=[]):
    return [root, list(branches)]
```

```
def make_city(name, lat, lon):
    return [name, lat, lon]
```



Python and Trees

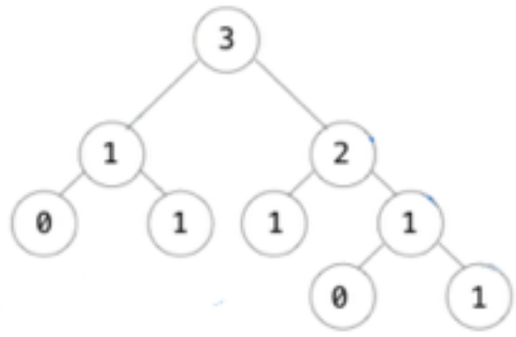
Now we want to translate the tree from the previous slide into Python. How do we **represent** trees? We don't know a lot of data types yet. So let's use lists!

This is no different from what you have already worked with!
We used lists to represent the latitude and longitude in lab04

```
def tree(root, branches=[]):
    return [root, list(branches)]
```

```
def make_city(name, lat, lon):
    return [name, lat, lon]
```

This is a value.
In the tree on the previous page, it is 3



Python and Trees

Now we want to translate the tree from the previous slide into Python. How do we **represent** trees? We don't know a lot of data types yet. So let's use lists!

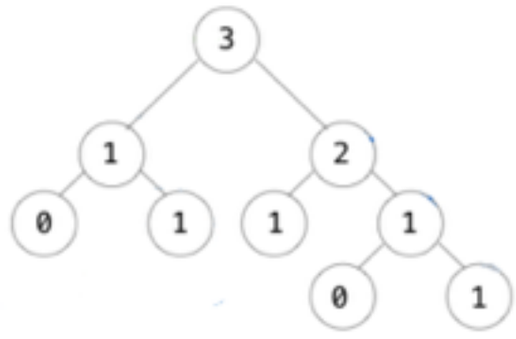
This is no different from what you have already worked with!
We used lists to represent the latitude and longitude in lab04

```
def tree(root, branches=[]):
    return [root, list(branches)]
```

```
def make_city(name, lat, lon):
    return [name, lat, lon]
```

This is a value. In the tree on the previous page, it is 3

These are all of the branches. Each branch is itself a tree. So every element after the first element is a **list**.



Python and Trees

Now we want to translate the tree from the previous slide into Python. How do we **represent** trees? We don't know a lot of data types yet. So let's use lists!

This is no different from what you have already worked with!
We used lists to represent the latitude and longitude in lab04

```
def tree(root, branches=[]):
    return [root, list(branches)]
```

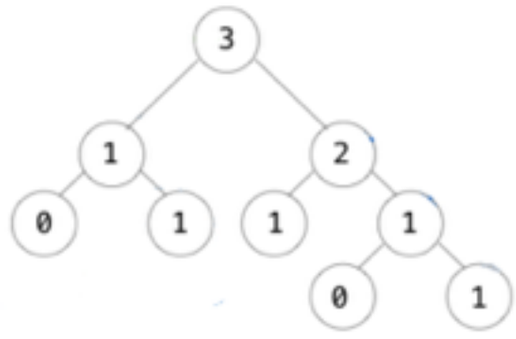
```
def make_city(name, lat, lon):
    return [name, lat, lon]
```

This is a value. In the tree on the previous page, it is 3

These are all of the branches. Each branch is itself a tree. So every element after the first element is a **list**.

```
def root(t):
    return t[0]
```

return the first element of the list that we constructed in the tree function above



Python and Trees

Now we want to translate the tree from the previous slide into Python. How do we **represent** trees? We don't know a lot of data types yet. So let's use lists!

This is no different from what you have already worked with!
We used lists to represent the latitude and longitude in lab04

```
def tree(root, branches=[]):
    return [root, list(branches)]
```

```
def make_city(name, lat, lon):
    return [name, lat, lon]
```

This is a value. In the tree on the previous page, it is 3

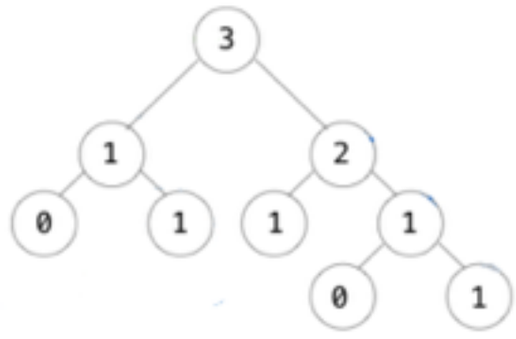
These are all of the branches. Each branch is itself a tree. So every element after the first element is a **list**.

```
def root(t):
    return t[0]
```

return the first element of the list that we constructed in the tree function above

```
def branches(t):
    return t[1:]
```

return the rest of the elements, which are all trees



Python and Trees

Now we want to translate the tree from the previous slide into Python. How do we **represent** trees? We don't know a lot of data types yet. So let's use lists!

This is no different from what you have already worked with!
We used lists to represent the latitude and longitude in lab04

```
def tree(root, branches=[]):
    return [root, list(branches)]
```

```
def make_city(name, lat, lon):
    return [name, lat, lon]
```

This is a value. In the tree on the previous page, it is 3

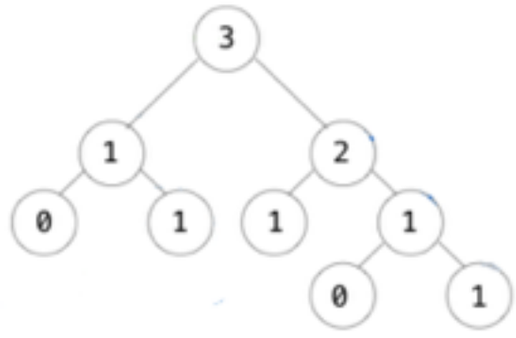
These are all of the branches. Each branch is itself a tree. So every element after the first element is a **list**.

```
def root(t):
    return t[0]
```

return the first element of the list that we constructed in the tree function above

```
def branches(t):
    return t[1:]
```

return the rest of the elements, which are all trees




```
def is_leaf(t):
    return not branches(t)
```

if a tree has no branches, it is a leaf!
not [] —> not False —> True

Tree Function Example

Write a function that returns a new tree and increments every value by 1. This is the example from lecture.

Plan:

— “return a new tree” —> call tree()

— “increment by 1” —> root(t)+1

— input: tree!

— output: tree!

```
def increment(t):
```

Tree Function Example

Write a function that returns a new tree and increments every value by 1. This is the example from lecture.

Plan:
— “return a new tree” —> call tree()
— “increment by 1” —> root(t)+1
— input: tree!
— output: tree!

```
def increment(t):

    for b in branches(t):
```

iterate through all of the branches (horizontal)

Tree Function Example

Write a function that returns a new tree and increments every value by 1. This is the example from lecture.

Plan:
— “return a new tree” —> call tree()
— “increment by 1” —> root(t)+1
— input: tree!
— output: tree!

```
def increment(t):

    for b in branches(t):
                    increment(b)
```

iterate through all of the branches (horizontal)

do a recursive call to go down into each branch (vertical) **(DON’T OVERTHINK THIS!!)**

Tree Function Example

Write a function that returns a new tree and increments every value by 1. This is the example from lecture.

Plan:
— “return a new tree” —> call tree()
— “increment by 1” —> root(t)+1
— input: tree!
— output: tree!

```
def increment(t):
    new_b = []
    for b in branches(t):
        new_b += [increment(b)]
```

accumulate the result of the recursive call in a variable. Check in Q: Why can we do this? Doesn’t this variable get “erased” with every recursive call?

iterate through all of the branches (horizontal)

do a recursive call to go down into each branch (vertical) **(DON’T OVERTHINK THIS!!)**

Tree Function Example

Write a function that returns a new tree and increments every value by 1. This is the example from lecture.

Plan:
— “return a new tree” —> call tree()
— “increment by 1” —> root(t)+1
— input: tree!
— output: tree!

```
def increment(t):
    new_b = []
    for b in branches(t):
        new_b += [increment(b)]
    return
```

accumulate the result of the recursive call in a variable. Check in Q: Why can we do this? Doesn’t this variable get “erased” with every recursive call?

iterate through all of the branches (horizontal)

do a recursive call to go down into each branch (vertical) **(DON’T OVERTHINK THIS!!)**

Tree Function Example

Write a function that returns a new tree and increments every value by 1. This is the example from lecture.

Plan:
— “return a new tree” —> call tree()
— “increment by 1” —> root(t)+1
— input: tree!
— output: tree!

```
def increment(t):
    new_b = []
    for b in branches(t):
        new_b += [increment(b)]
    return tree(                    )
```

accumulate the result of the recursive call in a variable. Check in Q: Why can we do this? Doesn’t this variable get “erased” with every recursive call?

iterate through all of the branches (horizontal)

do a recursive call to go down into each branch (vertical) **(DON’T OVERTHINK THIS!!)**

create a new tree!
what will be the root?
what are the branches?

Tree Function Example

Write a function that returns a new tree and increments every value by 1. This is the example from lecture.

Plan:
— “return a new tree” —> call tree()
— “increment by 1” —> root(t)+1
— input: tree!
— output: tree!

```
def increment(t):
    new_b = []
    for b in branches(t):
        new_b += [increment(b)]
    return tree( root(t) + 1,          )
```

accumulate the result of the recursive call in a variable. Check in Q: Why can we do this? Doesn’t this variable get “erased” with every recursive call?

iterate through all of the branches (horizontal)

do a recursive call to go down into each branch (vertical) **(DON’T OVERTHINK THIS!!)**

create a new tree!
what will be the root?
what are the branches?

the value of the new root is the value of the current root, incremented by 1

Tree Function Example

Write a function that returns a new tree and increments every value by 1. This is the example from lecture.

Plan:
— “return a new tree” —> call tree()
— “increment by 1” —> root(t)+1
— input: tree!
— output: tree!

```
def increment(t):
    new_b = []
    for b in branches(t):
        new_b += [increment(b)]
    return tree(root(t) + 1, new_b)
```

accumulate the result of the recursive call in a variable. Check in Q: Why can we do this? Doesn’t this variable get “erased” with every recursive call?

iterate through all of the branches (horizontal)

do a recursive call to go down into each branch (vertical) **(DON’T OVERTHINK THIS!!)**

create a new tree!
what will be the root?
what are the branches?

the value of the new root is the value of the current root, incremented by 1

where did we accumulate the incremented branches?

How to write functions

There is a structure to how most functions involving trees are written.
You'll notice this as you do more problems!

```
def tree_function(tree):
```

How to write functions

There is a structure to how most functions involving trees are written.
You'll notice this as you do more problems!

```
def tree_function(tree):
```

move horizontally through the branches

```
    for b in branches(tree):
```

How to write functions

There is a structure to how most functions involving trees are written.
You'll notice this as you do more problems!

```
def tree_function(tree):
    for b in branches(tree):
        call tree_function
```

move horizontally through the branches

move vertically down to the leaves of the branch

How to write functions

There is a structure to how most functions involving trees are written.
You'll notice this as you do more problems!

```
def tree_function(tree):
    new_variable = ???
    for b in branches(tree):
        call tree_function
```

depending on what your function is supposed to do, new_variable can accumulate branches, numbers, lists…

move horizontally through the branches

move vertically down to the leaves of the branch

How to write functions

There is a structure to how most functions involving trees are written.
You'll notice this as you do more problems!

```
def tree_function(tree):
    new_variable = ???
    for b in branches(tree):
        call tree_function
    combine results!
```

depending on what your function is supposed to do, new_variable can accumulate branches, numbers, lists…

move horizontally through the branches

move vertically down to the leaves of the branch

use tree and new_variable to build a solution to the original problem

How to write functions

There is a structure to how most functions involving trees are written.
You'll notice this as you do more problems!

```
def tree_function(tree):
    new_variable = ???
    for b in branches(tree):
        call tree_function
    combine results!
    return _________________
```

depending on what your function is supposed to do, new_variable can accumulate branches, numbers, lists…

move horizontally through the branches

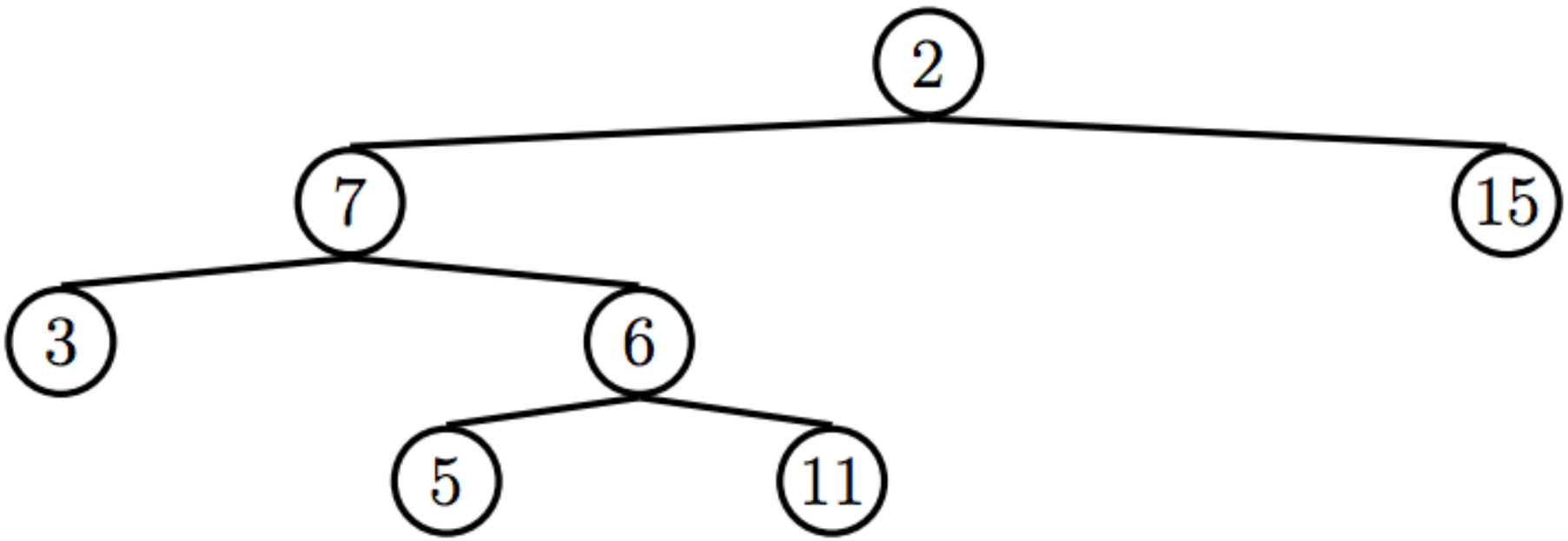
move vertically down to the leaves of the branch

use tree and new_variable to build a solution to the original problem

figure out what the return type is!!

Find path

Write a function that takes in a tree t and a value x, and returns the path to x as a list of values.

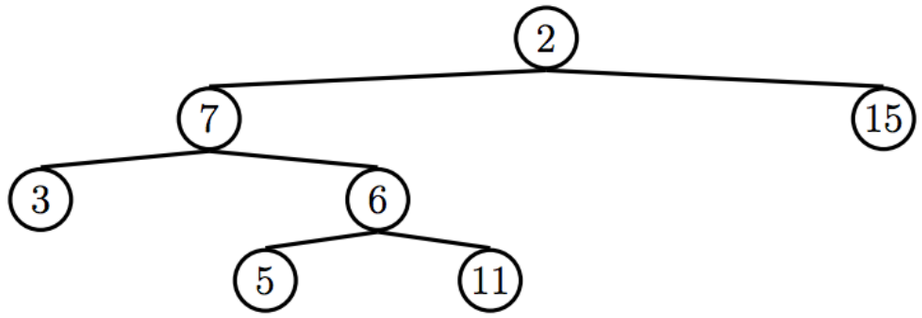



```
>>> find_path(t, 5)
[2, 7, 6, 5]
>>> find_path(t, 10)   # returns None
```

Look at the tree and doctests. Make sure you understand what this function is supposed to do!

Find path

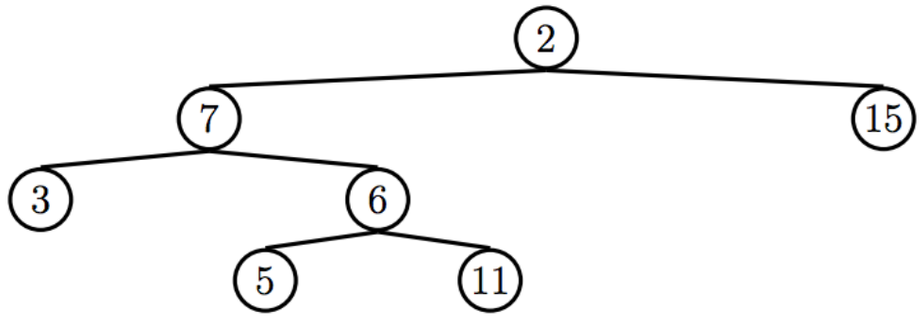
```
def find_path(t, x):
```



Plan:

— input: a tree and a value
— output: a list of values
— compare each value to x
— build a new list
— return None if no path found

Find path



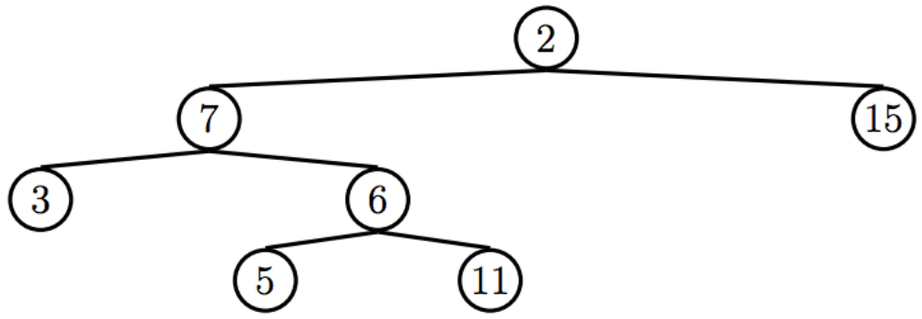
Plan:

— input: a tree and a value
— output: a list of values
— compare each value to x
— build a new list
— return None if no path found

```
def find_path(t, x):
    for b in branches(t):
```

we must check all the branches for a path
so iterate through them using a for loop (horizontal)

Find path



Plan:

— input: a tree and a value
— output: a list of values
— compare each value to x
— build a new list
— return None if no path found

```
def find_path(t, x):
```

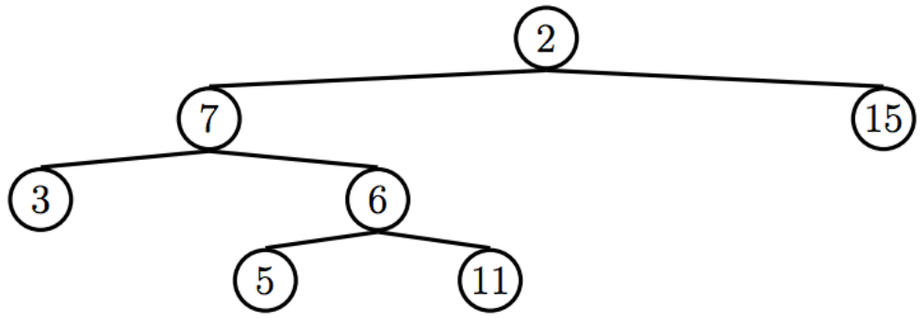
```
    for b in branches(t):
```

we must check all the branches for a path
so iterate through them using a for loop (horizontal)

```
        find_path(b)
```

try to find a path in each branch

Find path



Plan:

— input: a tree and a value
— output: a list of values
— compare each value to x
— build a new list
— return None if no path found

```
def find_path(t, x):
```

```
    for b in branches(t):
```

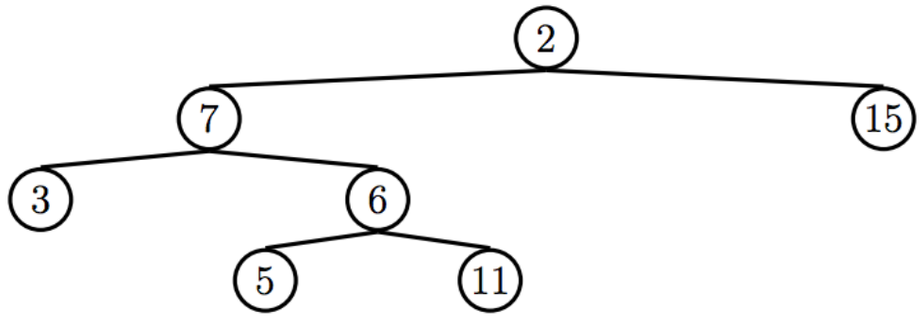
we must check all the branches for a path
so iterate through them using a for loop (horizontal)

```
        new_path = find_path(b)
```

try to find a path in each branch

assign it to a name. Checkpoint: why do we need to do this?

Find path



Plan:

— input: a tree and a value
— output: a list of values
— compare each value to x
— build a new list
— return None if no path found

```
def find_path(t, x):
    for b in branches(t):
        new_path = find_path(b)
        if new_path:
```

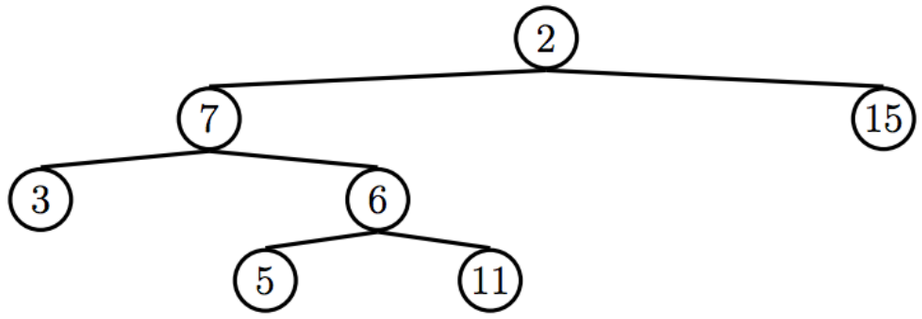
we must check all the branches for a path so iterate through them using a for loop (horizontal)

try to find a path in each branch

assign it to a name. Checkpoint: why do we need to do this?

if you found a path (find_path returned something that was not None) then add the current node to the list and return it

Find path



Plan:

— input: a tree and a value
— output: a list of values
— compare each value to x
— build a new list
— return None if no path found

```
def find_path(t, x):
```

```
    for b in branches(t):
```

we must check all the branches for a path so iterate through them using a for loop (horizontal)

```
        new_path = find_path(b)
```

assign it to a name. Checkpoint: why do we need to do this?

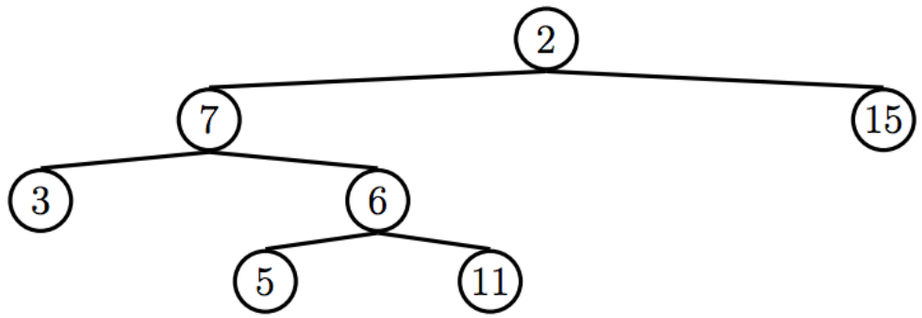
try to find a path in each branch

```
        if new_path:
```

if you found a path (find_path returned something that was not None) then add the current node to the list and return it

```
            return [root(t)] + new_path
```

Find path



Plan:

— input: a tree and a value
— output: a list of values
— compare each value to x
— build a new list
— return None if no path found

```
def find_path(t, x):
    if root(t) == x:
        for b in branches(t):
            new_path = find_path(b)
            if new_path:
                return [root(t)] + new_path
```

we're missing out base case! how do we know for sure, that a node is included in the path?

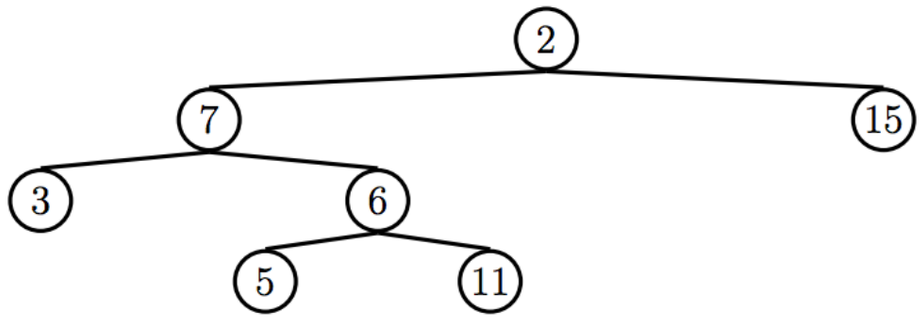
we must check all the branches for a path so iterate through them using a for loop (horizontal)

assign it to a name. Checkpoint: why do we need to do this?

try to find a path in each branch

if you found a path (find_path returned something that was not None) then add the current node to the list and return it

Find path



Plan:

— input: a tree and a value
— output: a list of values
— compare each value to x
— build a new list
— return None if no path found

```
def find_path(t, x):
    if root(t) == x:
        return [root(t)]
    for b in branches(t):
        new_path = find_path(b)
        if new_path:
            return [root(t)] + new_path
```

we're missing out base case! how do we know for sure, that a node is included in the path?

we must check all the branches for a path so iterate through them using a for loop (horizontal)

assign it to a name. Checkpoint: why do we need to do this?

try to find a path in each branch

if you found a path (find_path returned something that was not None) then add the current node to the list and return it