3 types:

1. mutate a tree → return original tree
2. create new tree → copy values from existing and return tree(label, branches)
   - *list comprehension
   - create entire new tree
3. return value
   - base case: return value
   - return recursive call

for b in branches(t)

1.3 **Tutorial:** Write a function that takes in a tree and squares every value. It should return a new tree. You can assume that every item is a number.

```
def square_tree(t):
    """Return a tree with the square of every element in t
    >>> numbers = tree(1,
    ...                [tree(2,
    ...                      [tree(3),
    ...                       tree(4)]),
    ...                 tree(5,
    ...                      [tree(6,
    ...                            [tree(7)]),
    ...                       tree(8)])])
    >>> print_tree(square_tree(numbers))
    1
      4
        9
        16
      25
        36
          49
        64
    """
```

creating new tree

```
numbers = [square_tree(b) for b in branches]
return tree(label(t)**2, numbers)
```

---

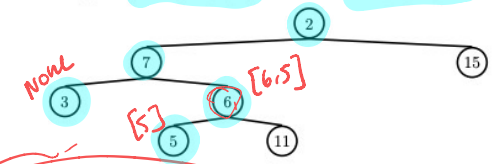
mutating old tree

```
label(t) = label(t) ** 2
for b in branches(t):
    square_tree(b)
return t
```

1.4 **Tutorial:** Write a function that takes in a tree and a value `x` and returns a list containing the nodes along the path required to get from the root of the tree to a node containing `x`.

If `x` is not present in the tree, return `None`. Assume that the entries of the tree are unique.

For the following tree, `find_path(t, 5)` should return `[2, 7, 6, 5]`




```
def find_path(tree, x):
    """
    >>> t = tree(2, [tree(7, [tree(3), tree(6, [tree(5), tree(11)])] ), tree(15)])
    >>> find_path(t, 5)
    [2, 7, 6, 5]
    >>> find_path(t, 10)  # returns None
    """
    if _____:
        return _____
    _____:
        path = _____
        if _____:
            return _____
```