Write a function that returns true if all of the roots in the tree are unique.

```
>>> t = tree(4, [])
>>> unique_roots(t)
True
>>> t = tree(4, [tree(2, [tree(4)])])
>>> unique_roots(t)
False
```

Approach: We know that we will somehow have to compare all of the nodes to each other. A priori we might do something like:

```
def unique_roots(t):
    past_roots = [root(t)]
    for branch in branches(t):
        do something
        past_roots += [root(branch)] #keep adding roots
```

Running through a quick example of this though will show that this will only compare the roots inside each subtree.
So we need a way to keep adding in roots to past_roots across function calls/across the branches horizontally
⇒nonlocal or parameters

The point of the function is to determine if all of the nodes are unique
⇒ We must keep track of all the roots we have seen so far as we recursive through the tree

Initially it is not clear why we need unique, come back to this variable at the end

```
def unique_roots(t):
    past_roots = [root(t)]
    unique = True
    def helper(t):
        for branch in branches(t):
            nonlocal past_roots, unique
            if root(branch) in past_roots:
                return False
            past_roots += [root(branch)]
            unique = helper(branch)
        return unique
    return helper(t)
```

This function will:
1. Iterate through the branches
2. Add all roots to past_roots
3. Compare each root to the roots in past_roots
4. Return False if there is a repeat

Check Yourself:
1. Why do we check if root is in past_roots first?
2. Why do we declare past_roots and unique as nonlocal?
3. Why do we need to keep track of whether or not we have seen a repeat yet (unique variable)?

```
def unique_roots2(t):
    def helper(t, past_roots, unique=True):
        for branch in branches(t):
            if root(branch) in past_roots:
                return False
            past_roots += [root(branch)]
            unique = helper(branch, past_roots, unique)
        return unique
    return helper(t, [root(t)])
```

This is an alternative approach.
Instead of using nonlocal, here we use function arguments to keep passing past_roots and unique.
The key ideas are the same.

Check Yourself:
1. What is yet another way to keep track of the variables?