Tree Class
Tree Review

```
        3
       / \
      1   2
     /   / \
    0   1   1
     \
     0   1
```
Recursive description (wooden trees):  

Relative description (family trees):
Recursive description (wooden trees):  
A tree has a root value and a list of branches

Relative description (family trees):
Tree Review

**Recursive description** (wooden trees):
A tree has a **root** value and a list of **branches**

**Relative description** (family trees):
Recursive description (wooden trees):
A tree has a root value and a list of branches

Relative description (family trees):
Recursive description (wooden trees):
A tree has a root value and a list of branches.
Each branch is a tree.

Relative description (family trees):
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):
**Tree Review**

**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):*

Root value: 3

Branches:
- 1
  - 0
    - Leaf
  - 1
    - Leaf
- 2
  - 1
  - 1
    - 0
    - 1
**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.
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
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
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.
**Tree Review**

**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.
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
Tree Class

A Tree has a root value and a list of branches; each branch is a Tree
A Tree has a root value and a list of branches; each branch is a Tree.

class Tree:
Tree Class

A Tree has a root value and a list of branches; each branch is a Tree

class Tree:
    def __init__(self, root, branches=[]):
Tree Class

A Tree has a root value and a list of branches; each branch is a Tree

class Tree:
    def __init__(self, root, branches=[]):
        self.root = root
A Tree has a root value and a list of branches; each branch is a Tree

class Tree:
    def __init__(self, root, branches=[]):
        self.root = root
        for branch in branches:
            assert isinstance(branch, Tree)
A Tree has a root value and a list of branches; each branch is a Tree

```python
class Tree:
    def __init__(self, root, branches=[]):
        self.root = root
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)
```

Tree Class
Tree Class

A Tree has a root value and a list of branches; each branch is a Tree

class Tree:
    def __init__(self, root, branches=[]):
        self.root = root
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

def tree(root, branches=[]):
    for branch in branches:
        assert is_tree(branch)
    return [root] + list(branches)

def root(tree):
    return tree[0]

def branches(tree):
    return tree[1:]
**Tree Class**

A Tree has a root value and a list of branches; each branch is a Tree

class Tree:
    def __init__(self, root, branches=[]):
        self.root = root
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

def fib_tree(n):
    if n == 0 or n == 1:
        return Tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = left.root + right.root
        return Tree(fib_n, [left, right])
Tree Class

A Tree has a root value and a list of branches; each branch is a Tree.

class Tree:
    def __init__(self, root, branches=[]):
        self.root = root
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

def fib_tree(n):
    if n == 0 or n == 1:
        return Tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = left.root + right.root
        return Tree(fib_n, [left, right])
A Tree has a root value and a list of branches; each branch is a Tree

class Tree:
    def __init__(self, root, branches=[]):
        self.root = root
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

def fib_tree(n):
    if n == 0 or n == 1:
        return Tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = left.root + right.root
        return Tree(fib_n, [left, right])

def tree(root, branches=[]):
    for branch in branches:
        assert is_tree(branch)
    return [root] + list(branches)

def root(tree):
    return tree[0]

def branches(tree):
    return tree[1:]

def fib_tree(n):
    if n == 0 or n == 1:
        return tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = root(left) + root(right)
        return tree(fib_n, [left, right])

(Demo)
Tree Mutation
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

```
        3
       / \  
      1   2  
     / \  / 
    0   1 1  
       /    
      0 1   
```
Example: Pruning Trees

Removing subtrees from a tree is called pruning.

Prune branches before recursive processing.
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*

Prune branches before recursive processing

```python
def prune(t, n):
    """Prune sub-trees whose root value is n."""
    t.branches = [______________ for b in t.branches if ________________]
    for b in t.branches:
        prune(__________________________, _____________________________)
```
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*

Prune branches before recursive processing

```python
def prune(t, n):
    """Prune sub-trees whose root value is n."""
    t.branches = [b for b in t.branches if b.root != n]
    for b in t.branches:
        prune(b, n)
```

```
0 1 1
  1 1
  2
    3
```

```python
def prune(t, n):
    """Prune sub-trees whose root value is n."""
    t.branches = [b for b in t.branches if b.root != n]
    for b in t.branches:
        prune(b, n)
```
Example: Pruning Trees

Removing subtrees from a tree is called pruning.

Prune branches before recursive processing.

```python
def prune(t, n):
    """Prune sub-trees whose root value is n."""
    t.branches = [b for b in t.branches if b.root != n]
    for b in t.branches:
        prune(b, n)
```

```python
def prune(t, n):
    """Prune sub-trees whose root value is n."""
    t.branches = [b for b in t.branches if b.root != n]
    for b in t.branches:
        prune(b, n)
```
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

```python
def prune(t, n):
    """Prune sub-trees whose root value is n."""
    t.branches = [b for b in t.branches if b.root != n]
    for b in t.branches:
        prune(b, n)
```

(Demo)
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.
Example: Pruning Trees

Removing subtrees from a tree is called **pruning**

Prune branches before recursive processing

Memoization:
Example: Pruning Trees

Removing subtrees from a tree is called pruning.

Prune branches before recursive processing.

Memoization:
- Returned by fib.
Example: Pruning Trees

Removing subtrees from a tree is called \textit{pruning}

Prune branches before recursive processing

\textbf{Memoization:}
- Returned by fib
- Found in cache
Example: Pruning Trees

Removing subtrees from a tree is called **pruning**

Prune branches before recursive processing

**Memoization:**
- Returned by `fib`
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called pruning.

Prune branches before recursive processing.

Memoization:
- Returned by fib
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

Memoization:
- Returned by fib
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

**Memoization:**
- Returned by `fib`
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

**Memoization:**
- Returned by `fib`
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

**Memoization:**

- Returned by `fib`
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

**Memoization:**
- Returned by `fib`
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called **pruning**

Prune branches before recursive processing

**Memoization:**
- Returned by fib
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

**Memoization:**
- Returned by `fib`
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

**Memoization:**
- Returned by fib
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called **pruning**

Prune branches before recursive processing

**Memoization:**
- Returned by `fib`
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

**Memoization:**
- Returned by `fib`
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

**Memoization:**
- Returned by `fib`
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

**Memoization:**
- Returned by `fib`
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called pruning.

Prune branches before recursive processing.

Memoization:
- Returned by fib
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

**Memoization:**
- Returned by fib
- Found in cache
- Skipped
Example: Pruning Trees

Removing subtrees from a tree is called \textit{pruning}.

Prune branches before recursive processing.

\textbf{Memoization:}
- Returned by \texttt{fib}
- Found in cache
- Skipped
Hailstone Trees
Hailstone Trees
Hailstone Trees

Pick a positive integer $n$ as the start
Hailstone Trees

Pick a positive integer \( n \) as the start

If \( n \) is even, divide it by 2
Hailstone Trees

Pick a positive integer n as the start
If n is even, divide it by 2
If n is odd, multiply it by 3 and add 1
Hailstone Trees

Pick a positive integer $n$ as the start
If $n$ is even, divide it by 2
If $n$ is odd, multiply it by 3 and add 1
Continue this process until $n$ is 1
Hailstone Trees

Pick a positive integer $n$ as the start
If $n$ is even, divide it by 2
If $n$ is odd, multiply it by 3 and add 1
Continue this process until $n$ is 1

(Demo)
Hailstone Trees

Pick a positive integer n as the start
If n is even, divide it by 2
If n is odd, multiply it by 3 and add 1
Continue this process until n is 1

(Demo)
Hailstone Trees

Pick a positive integer n as the start
If n is even, divide it by 2
If n is odd, multiply it by 3 and add 1
Continue this process until n is 1

(Demo)
Hailstone Trees

Pick a positive integer $n$ as the start

If $n$ is even, divide it by 2

If $n$ is odd, multiply it by 3 and add 1

Continue this process until $n$ is 1

(Demo)
Hailstone Trees

Pick a positive integer \( n \) as the start
If \( n \) is even, divide it by 2
If \( n \) is odd, multiply it by 3 and add 1
Continue this process until \( n \) is 1

(Demo)

\begin{align*}
1 & \\
2 & \\
4 & \\
8 & \\
\end{align*}
Hailstone Trees

Pick a positive integer n as the start
If n is even, divide it by 2
If n is odd, multiply it by 3 and add 1
Continue this process until n is 1

(Demo)
1
2
4
8
16
Hailstone Trees

Pick a positive integer $n$ as the start

If $n$ is even, divide it by 2

If $n$ is odd, multiply it by 3 and add 1

Continue this process until $n$ is 1

(Demo)

1
2
4
8
16
32
Hailstone Trees

Pick a positive integer n as the start
If n is even, divide it by 2
If n is odd, multiply it by 3 and add 1
Continue this process until n is 1

(Demo)

1
2
4
8
16
32
64
Hailstone Trees

Pick a positive integer n as the start
If n is even, divide it by 2
If n is odd, multiply it by 3 and add 1
Continue this process until n is 1

(Demo)
Hailstone Trees

Pick a positive integer n as the start

If n is even, divide it by 2

If n is odd, multiply it by 3 and add 1

Continue this process until n is 1

(Demo)
Hailstone Trees

Pick a positive integer \( n \) as the start
If \( n \) is even, divide it by 2
If \( n \) is odd, multiply it by 3 and add 1
Continue this process until \( n \) is 1

(Demo)
Hailstone Trees

Pick a positive integer n as the start

If n is even, divide it by 2

If n is odd, multiply it by 3 and add 1

Continue this process until n is 1

(Demo)
Hailstone Trees

Pick a positive integer $n$ as the start

If $n$ is even, divide it by 2

If $n$ is odd, multiply it by 3 and add 1

Continue this process until $n$ is 1

(Demo)
Hailstone Trees

Pick a positive integer $n$ as the start

If $n$ is even, divide it by 2

If $n$ is odd, multiply it by 3 and add 1

Continue this process until $n$ is 1

(Demo)
Hailstone Trees

Pick a positive integer $n$ as the start
If $n$ is even, divide it by 2
If $n$ is odd, multiply it by 3 and add 1
Continue this process until $n$ is 1

(Demo)

All possible $n$ that start a length-8 hailstone sequence

10
1
2
4
8
16
32
64
128 21 20 3
Hailstone Trees

Pick a positive integer \( n \) as the start

If \( n \) is even, divide it by 2

If \( n \) is odd, multiply it by 3 and add 1

Continue this process until \( n \) is 1

(Demo)

```python
def hailstone_tree(k, n=1):
    """Return a Tree in which the paths from the leaves to the root are all possible hailstone sequences of length \( k \) ending in \( n \).""
    
    All possible \( n \) that start a length-8 hailstone sequence
```

```plaintext
1
 |   |   |   |   |   |   |   |   
2
 |   |   |   |   |   |   |   
4
 |   |   |   |   |   |   
8
 |   |   |   |   |   
16
 |   |   |   |   
32
 |   |   |   
64
 |   |   
128
 |
21
 |
20
 |
3
```
Hailstone Trees

Pick a positive integer \( n \) as the start

If \( n \) is even, divide it by 2

If \( n \) is odd, multiply it by 3 and add 1

Continue this process until \( n \) is 1

(Demo)

def hailstone_tree(k, n=1):
    """Return a Tree in which the paths from the
    leaves to the root are all possible hailstone
    sequences of length \( k \) ending in \( n \)."""

    All possible \( n \) that start a
    length-8 hailstone sequence

(Demo)