```
#########
# Trees #
#########

def tree(label, branches=[]):
    """Construct a tree with the given label value and a list of branches."""
    for branch in branches:
        assert is_tree(branch), 'branches must be trees'
    return [label] + list(branches)

def label(tree):
    """Return the label value of a tree."""
    return tree[0]

def branches(tree):
    """Return the list of branches of the given tree."""
    return tree[1:]

def is_tree(tree):
    """Returns True if the given tree is a tree, and False otherwise."""
    if type(tree) != list or len(tree) < 1:
        return False
    for branch in branches(tree):
        if not is_tree(branch):
            return False
    return True

def is_leaf(tree):
    """Returns True if the given tree's list of branches is empty, and False
    otherwise.
    """
    return not branches(tree)

def print_tree(t, indent=0):
    """Print a representation of this tree in which each node is
    indented by two spaces times its depth from the root.

    >>> print_tree(tree(1))
    1
    >>> print_tree(tree(1, [tree(2)]))
    1
      2
    >>> numbers = tree(1, [tree(2), tree(3, [tree(4), tree(5)]), tree(6, [tree(7)])])
    >>> print_tree(numbers)
    1
      2
      3
        4
        5
      6
        7
    """
    print('  ' * indent + str(label(t)))
    for b in branches(t):
        print_tree(b, indent + 1)

def copy_tree(t):
    """Returns a copy of t. Only for testing purposes.

    >>> t = tree(5)
    >>> copy = copy_tree(t)
    >>> t = tree(6)
    >>> print_tree(copy)
    5
    """
    return tree(label(t), [copy_tree(b) for b in branches(t)])

######################
# Required questions #
######################

def replace_leaf(t, old, new):
    """Returns a new tree where every leaf value equal to old has
    been replaced with new.

    >>> yggdrasil = tree('odin',
    ...                  [tree('balder',
    ...                        [tree('thor'),
    ...                         tree('loki')]),
    ...                   tree('frigg',
    ...                        [tree('thor')]),
    ...                   tree('thor',
    ...                        [tree('sif'),
    ...                         tree('thor')]),
    ...                   tree('thor')])
    >>> laerad = copy_tree(yggdrasil) # copy yggdrasil for testing purposes
    >>> print_tree(replace_leaf(yggdrasil, 'thor', 'freya'))
    odin
      balder
        freya
        loki
      frigg
        freya
      thor
        sif
        freya
      freya
    >>> laerad == yggdrasil # Make sure original tree is unmodified
    True
    """
    if is_leaf(t) and label(t) == old:
        return tree(new)
    else:
        bs = [replace_leaf(b, old, new) for b in branches(t)]
        return tree(label(t), bs)

def prune_leaves(t, vals):
    """Return a modified copy of t with all leaves that have a label
    that appears in vals removed.  Return None if the entire tree is
    pruned away.

    >>> t = tree(2)
    >>> print(prune_leaves(t, (1, 2)))
    None
    >>> numbers = tree(1, [tree(2), tree(3, [tree(4), tree(5)]), tree(6, [tree(7)])])
    >>> print_tree(numbers)
    1
      2
      3
        4
        5
      6
        7
    >>> print_tree(prune_leaves(numbers, (3, 4, 6, 7)))
    1
      2
      3
        5
      6
    """
    if is_leaf(t) and (label(t) in vals):
      return None
    new_branches = []
    for b in branches(t):
      new_branch = prune_leaves(b, vals)
      if new_branch:
        new_branches += [new_branch]
    return tree(label(t), new_branches)

# Mobiles

def mobile(left, right):
    """Construct a mobile from a left side and a right side."""
    assert is_side(left), "left must be a side"
    assert is_side(right), "right must be a side"
    return ['mobile', left, right]

def is_mobile(m):
    """Return whether m is a mobile."""
    return type(m) == list and len(m) == 3 and m[0] == 'mobile'

def left(m):
    """Select the left side of a mobile."""
    assert is_mobile(m), "must call left on a mobile"
    return m[1]

def right(m):
    """Select the right side of a mobile."""
    assert is_mobile(m), "must call right on a mobile"
    return m[2]

def side(length, mobile_or_weight):
    """Construct a side: a length of rod with a mobile or weight at the end."""
    assert is_mobile(mobile_or_weight) or is_weight(mobile_or_weight)
    return ['side', length, mobile_or_weight]

def is_side(s):
    """Return whether s is a side."""
    return type(s) == list and len(s) == 3 and s[0] == 'side'

def length(s):
    """Select the length of a side."""
    assert is_side(s), "must call length on a side"
    return s[1]

def end(s):
    """Select the mobile or weight hanging at the end of a side."""
    assert is_side(s), "must call end on a side"
    return s[2]

def weight(size):
    """Construct a weight of some size."""
    assert size > 0
    return ['weight', size]

def size(w):
    """Select the size of a weight."""
    assert is_weight(w), 'must call size on a weight'
    return w[1]

def is_weight(w):
    """Whether w is a weight."""
    return type(w) == list and len(w) == 2 and w[0] == 'weight'
```

```python
def examples():
    t = mobile(side(1, weight(2)),
               side(2, weight(1)))
    u = mobile(side(5, weight(1)),
               side(1, mobile(side(2, weight(3)),
                              side(3, weight(2)))))
    v = mobile(side(4, t), side(2, u))
    return (t, u, v)

def total_weight(m):
    """Return the total weight of m, a weight or mobile.

    >>> t, u, v = examples()
    >>> total_weight(t)
    3
    >>> total_weight(u)
    6
    >>> total_weight(v)
    9
    """
    if is_weight(m):
        return size(m)
    else:
        assert is_mobile(m), "must get total weight of a mobile or a weight"
        return total_weight(end(left(m))) + total_weight(end(right(m)))

def balanced(m):
    """Return whether m is balanced.

    >>> t, u, v = examples()
    >>> balanced(t)
    True
    >>> balanced(v)
    True
    >>> w = mobile(side(3, t), side(2, u))
    >>> balanced(w)
    False
    >>> balanced(mobile(side(1, v), side(1, w)))
    False
    >>> balanced(mobile(side(1, w), side(1, v)))
    False
    """
    if is_weight(m):
        return True
    else:
        left_end, right_end = end(left(m)), end(right(m))
        torque_left = length(left(m)) * total_weight(left_end)
        torque_right = length(right(m)) * total_weight(right_end)
        return balanced(left_end) and balanced(right_end) and torque_left == torque_right

def totals_tree(m):
    """Return a tree representing the mobile with its total weight at the root.

    >>> t, u, v = examples()
    >>> print_tree(totals_tree(t))
    3
      2
      1
    >>> print_tree(totals_tree(u))
    6
      1
      5
        3
        2
    >>> print_tree(totals_tree(v))
    9
      3
        2
        1
      6
        1
        5
          3
          2
    """
    if is_weight(m):
        return tree(size(m))
    else:
        branches = [totals_tree(end(f(m))) for f in [left, right]]
        return tree(sum([label(b) for b in branches]), branches)

###################
# Extra Questions #
###################

def zero(f):
    return lambda x: x

def successor(n):
    return lambda f: lambda x: f(n(f)(x))

def one(f):
    """Church numeral 1: same as successor(zero)"""
    return lambda x: f(x)

def two(f):
    """Church numeral 2: same as successor(successor(zero))"""
    return lambda x: f(f(x))

three = successor(two)

def church_to_int(n):
    """Convert the Church numeral n to a Python integer.

    >>> church_to_int(zero)
    0
    >>> church_to_int(one)
    1
    >>> church_to_int(two)
    2
    >>> church_to_int(three)
    3
    """
    return n(lambda x: x + 1)(0)

def add_church(m, n):
    """Return the Church numeral for m + n, for Church numerals m and n.

    >>> church_to_int(add_church(two, three))
    5
    """
    return lambda f: lambda x: m(f)(n(f)(x))

def mul_church(m, n):
    """Return the Church numeral for m * n, for Church numerals m and n.

    >>> four = successor(three)
    >>> church_to_int(mul_church(two, three))
    6
    >>> church_to_int(mul_church(three, four))
    12
    """
    return lambda f: m(n(f))

def pow_church(m, n):
    """Return the Church numeral m ** n, for Church numerals m and n.

    >>> church_to_int(pow_church(two, three))
    8
    >>> church_to_int(pow_church(three, two))
    9
    """
    return n(m)
```