**4. (8.0 points) Ratios**

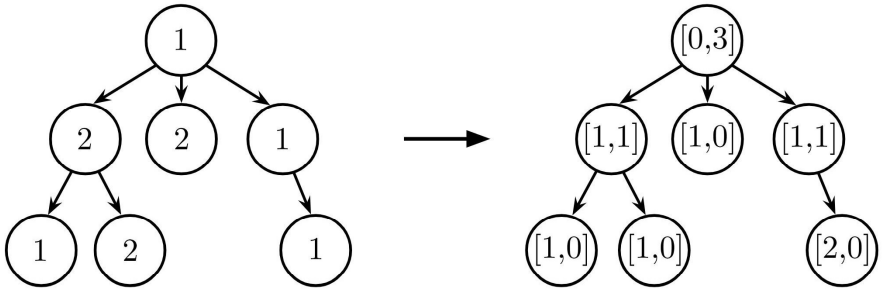
**(a) Definition:** The number of labels **above** tree `t` is the number of labels along the path from this node to the root of `t` **excluding the current node**.

**Definition:** The number of labels **below** tree `t` is the number of labels within a subtree `t` **excluding the current node**.

Implement `ratio`, which takes a Tree `t` and predicate function `f`, and **mutates** `t` into a Ratio Tree. A Ratio Tree replaces the value at each label with a 2-element list. Index 0 holds the number of labels above this node for which `f(label)` returns true. Index 1 holds the number of labels below this node for which `f(label)` returns true.

For example, consider the label 2 in the first branch of the root of the tree shown below. 2 has one label along the path to the root that is odd (1). 2 has one label within its subtrees that are odd (1). As a result, the label is replaced with the list [1, 1]

```
>>> t = Tree(1, [Tree(2, [Tree(1), Tree(2)]), Tree(2), Tree(1, [Tree(1)])])
>>> is_odd = lambda x: x % 2 == 1
>>> ratio(t, is_odd)
```




```
def ratio(t, f):
    """Mutate each label of t to the ratio of the number of labels above
       it to the number of labels below it, represented as a 2-element
       list - provided that f(label) == true.
    >>> t = Tree(15, [Tree(1, [Tree(8)]), Tree(5), (Tree(4, [Tree(6), Tree(7, [Tree(9)])]))])
    >>> pred = lambda x: x % 2 == 1 # odd?
    >>> ratio(t, pred)
    >>> t
    Tree([0, 4], [Tree([1, 0], [Tree([2, 0])]), Tree([1, 0]), Tree([1, 2],
    [Tree([1, 0]), Tree([1, 1], [Tree([2, 0])])])])
    """
    def helper(t, filtered_depth):
        above = _________
                   (a)
        if _________:
              (b)
            _________
               (c)
        below = _________
                   (d)
        for b in t.branches:
            _________
               (e)
            below += _________
                        (f)
        t.label = [above, below]
    return helper(t, 0)
```

i. **(1.0 pt)** Fill in blank (a)

ii. **(1.0 pt)** Fill in blank (b)

- ◯ `t.label`
- ◯ `t.is_leaf()`
- ◯ `f(t.label)`
- ◯ `f(t.branches)`
- ◯ `any([f(b.label) for b in t.branches])`
- ◯ `all([f(b.label) for b in t.branches])`

iii. **(1.0 pt)** Fill in blank (c)

- ◯ `return f(t.label)`
- ◯ `return above`
- ◯ `t.label = [above, 0]`
- ◯ `t.label = [0, 0]`
- ◯ `filtered_depth += 1`
- ◯ `above += 1`

iv. **(1.5 pt)** Fill in blank (d)

- ◯ `len([b for b in t.branches if f(b.label)])`
- ◯ `sum([b for b in t.branches if f(b.label)])`
- ◯ `sum([b.label for b in t.branches])`
- ◯ `sum([helper(b, filtered_depth) for b in t.branches])`
- ◯ `len([helper(b, filtered_depth) for b in t.branches])`
- ◯ `all([helper(b, filtered_depth) for b in t.branches])`

v. **(2.0 pt)** Fill in blank (e). You **cannot** use `and`, `or`, `if`, or `else`.