Exam Prep 4: Trees

Students from past semesters wanted more content and structured time to prepare for exams. Exam Prep sections are a way to solidify your understanding of the week's materials. The problems are typically designed to be a bridge between discussion/lab/homework difficulty and exam difficulty.

Reminder: There is nothing to turn in and there is no credit given for attending Exam Prep Sections.

We try to make these problems **exam level** , so you are not expected to be able to solve them coming straight from lecture without additional practice. To get the most out of Exam Prep, we recommend you **try these problems first on your own** before coming to the Exam Prep section, where we will explain how to solve these problems while giving tips and advice for the exam. Do not worry if you struggle with these problems, **it is okay to struggle while learning**.

You can work with anyone you want, including sharing solutions. We just ask you don't spoil the problems for anyone else in the class. Thanks!

You may only put code where there are underscores for the codewriting questions.

You can test your functions on their doctests by clicking the red test tube in the top right corner after clicking Run in 61A Code. Passing the doctests is not necessarily enough to get the problem fully correct. You must fully solve the stated problem.

Q1: Perfectly Balanced and Pruned

Difficulty: ☆

Part A: Implement `sum_tree`, which takes adds together all the labels in a tree.

Part B: Implement `balanced`, a function which takes in a tree and returns whether each of the branches have the same total sum, and each branch is balanced.

Part C: Implement `prune_tree`, a function which takes in a tree `t` as well as a function `predicate` that returns True or False for each label of each tree node. `prune_tree` returns a new tree where any node for which `predicate` of the label of that node returns True, then all the branches for that tree are pruned (not included in the new tree).

IMPORTANT: You may use as many lines as you want for these two parts.

Challenge: Solve both of these parts with just 1 line of code each.

```
14
15
16  def balanced(t):
17      """
18      Checks if each branch has same sum of all elements,
19      and each branch is balanced.
20
21      >>> t = tree(1, [tree(3), tree(1, [tree(2)]), tree(1, [tree(1), tree(1)])])
22      >>> balanced(t)
23      True
24      >>> t = tree(t, [t, tree(1)])
25      >>> balanced(t)
26      False
27      """
28      "*** YOUR CODE HERE ***"
29      if not branches(t):
30          return True
31      else:
```

```
            for c in branches(t):
                if sum_tree(branches(t)[0]) != sum_tree(c) or not balanced(c):
                    return False
        return True


def prune_tree(t, predicate):
    """
    Returns a new tree where any branch that has the predicate of the label
    of the branch returns True has its branches pruned.

    >>> prune_tree(tree(1, [tree(2)]), lambda x: x == 1) # prune at root
    [1]
    >>> prune_tree(tree(1, [tree(2)]), lambda x: x == 2) # prune at leaf
    [1, [2]]
    >>> prune_tree(test_tree, lambda x: x >= 3) # prune at 3, 4, and 5
    [1, [2, [4], [5]], [3]]
    >>> sum_tree(prune_tree(test_tree, lambda x: x > 10)) # prune nothing, add 1 to 9
    45
    >>> prune_tree(test_tree, lambda x: x > 10) == test_tree # prune nothing
    True
    """
    "*** YOUR CODE HERE ***"
    if predicate(label(t)) or is_leaf(t):
        return tree(label(t))
    return tree(label(t), [prune_tree(c, predicate) for c in branches(t)])


test_tree = tree(
    1, [tree(2, [tree(4, [tree(8), tree(9)]), tree(5)]), tree(3, [tree(6), tree(7)])]
)
draw(test_tree)

```

Q2: Closest - Spring 2015 Midterm 2 Q3(c)

IMPORTANT: For this problem, you will be given time during the Exam Prep section to solve on your own before we go over it.

Difficulty: ☆☆

Implement `closest`, which takes a tree of numbers `t` and returns the smallest absolute difference anywhere in the tree between an entry and the sum of the entries of its branches. The sum only compares the any node value to the sum of its branches, or nodes one level below that node.

The built-in `min` function takes a sequence and returns its minimum value.

Reminder: A branch of a branch of a tree t is not considered to be a branch of t.

```
def closest(t):
    """Return the smallest difference between an entry and the sum of the
    root entries of its branches .
    >>> t = tree(8 , [tree(4), tree(3)])
    >>> closest(t) # |8 - (4 + 3)| = 1
    1
    >>> closest(tree(5, [t])) # Same minimum as t
    1
    >>> closest(tree(10, [tree(2), t])) # |10 - (2 + 8)| = 0
    0
    >>> closest(tree(3)) # |3 - 0| = 3
    3
    >>> closest(tree(8, [tree(3, [tree(1, [tree(5)])])])) # 3 - 1 = 2
    2
    >>> sum([])
    0
    """
    diff = abs(label(t) - sum([label(c) for c in branches(t)]))
    return min([diff] + [closest(c) for c in branches(t)])

```

Q3: Recursion on Tree ADT - Summer 2014 Midterm 1 Q7

Difficulty: ⭐⭐

Define a function `dejavu`, which takes in a tree of numbers `t` and a number `n`. It returns `True` if there is a path from the root to a leaf such that the sum of the numbers along that path is `n` and `False` otherwise.

IMPORTANT: For this problem, the starter code template is just a suggestion. You are welcome to add/delete/modify the starter code template, or even write your own solution that doesn't use the starter code at all.

```
1   def dejavu(t, n):
2       """
3       >>> my_tree = tree(2, [tree(3, [tree(5), tree(7)]), tree(4)])
4       >>> dejavu(my_tree, 12) # 2 -> 3 -> 7
5       True
6       >>> dejavu(my_tree, 5) # Sums of partial paths like 2 -> 3 don't count
7       False
8       """
9       if is_leaf(t):
10          return label(t) == n
11      for c in branches(t):
12          if dejavu(c, n - label(t)):
13              return True
14      return False
15
```

Q4: Forest Path - Fall 2015 Final Q3 (a)(b)(d)

Difficulty: ☆☆☆

Definition: A *path* through a `tree` is a list of adjacent node values that starts with the root value and ends with a leaf value. For example, the paths of `tree(1, [tree(2), tree(3, [tree(4), tree(5)])])` are

```
[1, 2]
[1, 3, 4]
[1, 3, 5]
```

Part A: Implement `bigpath`, which takes a tree `t` and an integer `n`. It returns the number of paths in `t` whose sum is at least `n`. Assume that all node values of `t` are integers.

Part B: Implement `allpath` which takes a tree `t`, a one-argument predicate `f`, a two-argument reducing function `g`, and a starting value `s`. It returns the number of paths `p` in `t` for which `f(reduce(g, p, s))` returns a truthy value. The `reduce` function is in the code. Pay close attention to the order of arguments to the `f` function in `reduce`. You do not need to call it, though.

Part C: Re-implement `bigpath` (Part A) using `allpath` (Part B). Assume `allpath` is implemented correctly.

```
4
5       >>> reduce(mul, [2, 4, 8], 1)
6       64
7       >>> reduce(pow, [2, 3, 1], 2)
8       64
9       """
10      for x in s:
11          initial = f(initial, x)
12      return initial
13
```

```
14
15  # The one function defined below is used in the questions below
16  # to convert truthy and falsy values into the numbers 1 and 0, respectively.
17  def one(b):
18      if b:
19          return 1
20      else:
21          return 0
22
23
24  def bigpath(t, n):
25      """Return the number of paths in t that have a sum larger or equal to n.
26
27      >>> t = tree(1, [tree(2), tree(3, [tree(4), tree(5)])])
28      >>> bigpath(t, 3)
29      3
30      >>> bigpath(t, 6)
31      2
32      >>> bigpath(t, 9)
33      1
34      """
35      if is_leaf(t):
36          return one(label(t) >= n)
37      return sum([bigpath(c, n - label(t)) for c in branches(t)])
38
39
40  def allpath(t, f, g, s):
41      """Return the number of paths p in t for which f(reduce(g, p, s)) is truthy.
42
43      >>> t = tree(1, [tree(2), tree(3, [tree(4), tree(5)])])
44      >>> even = lambda x: x % 2 == 0
45      >>> allpath(t, even, max, 0) # Path maxes are 2, 4, and 5
46      2
47      >>> allpath(t, even, pow, 2) # E.g., pow(pow(2, 1), 2) is even
48      3
49      >>> allpath(t, even, pow, 1) # Raising 1 to any power is odd
50      0
51      """
```

```
    if is_leaf(t):
        return one(f(reduce(g, t, s)))
    return sum([allpath(c, f, g, g(s,label(t))) for c in branches(t)])


from operator import add, mul


def bigpath_allpath(t, n):
    """Return the number of paths in t that have a sum larger or equal to n.

    >>> t = tree(1, [tree(2), tree(3, [tree(4), tree(5)])])
    >>> bigpath_allpath(t, 3)
    3
    >>> bigpath_allpath(t, 6)
    2
    >>> bigpath_allpath(t, 9)
```

Q5: Extra Practice

Difficulty: >☆☆☆

Fall 2020 Exam Prep on Trees
(https://drive.google.com/file/d/1yzQF16ZOp_iUzOTY6rK2eXPdwCmcQnAj/view?usp=sharing)