Homework 4

● Graded

Student

Sangwon Ji

Total Points

2 / 2 pts

Autograder Score

2.0 / 2.0

Autograder Results

```
=====================================================================
Assignment: Homework 4
OK, version v1.18.1
=====================================================================

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Scoring tests

---------------------------------------------------------------------
Doctests for differences

>>> from hw04 import *
>>> d = differences(iter([5, 2, -100, 103]))
>>> [next(d) for _ in range(3)]
[-3, -102, 203]
>>> list(differences([1]))
[]
Score: 1.0/1

---------------------------------------------------------------------
Doctests for merge

>>> from hw04 import *
>>> def sequence(start, step):
... while True:
... yield start
... start += step
>>> x = sequence(2, 3) # 2, 5, 8, 11, 14, ...
>>> y = sequence(3, 2) # 3, 5, 7, 9, 11, 13, 15, ...
>>> result = merge(x, y) # 2, 3, 5, 7, 8, 9, 11, 13, 14, 15
>>> [next(result) for _ in range(10)]
[2, 3, 5, 7, 8, 9, 11, 13, 14, 15]
Score: 1.0/1

---------------------------------------------------------------------
Doctests for perms

>>> from hw04 import *
>>> p = perms([100])
```

```
>>> type(p)

>>> next(p)
[100]
>>> try: # Prints "No more permutations!" if calling next would cause an error
... next(p)
... except StopIteration:
... print('No more permutations!')
No more permutations!
>>> sorted(perms([1, 2, 3])) # Returns a sorted list containing elements of the generator
[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
>>> sorted(perms((10, 20, 30)))
[[10, 20, 30], [10, 30, 20], [20, 10, 30], [20, 30, 10], [30, 10, 20], [30, 20, 10]]
>>> sorted(perms("ab"))
[['a', 'b'], ['b', 'a']]
Score: 1.0/1

---------------------------------------------------------------------
Doctests for yield_paths

>>> from hw04 import *
>>> t1 = tree(1, [tree(2, [tree(3), tree(4, [tree(6)]), tree(5)]), tree(5)])
>>> print_tree(t1)
1
2
3
4
6
5
5
>>> next(yield_paths(t1, 6))
[1, 2, 4, 6]
>>> path_to_5 = yield_paths(t1, 5)
>>> sorted(list(path_to_5))
[[1, 2, 5], [1, 5]]
>>> t2 = tree(0, [tree(2, [t1])])
>>> print_tree(t2)
0
2
1
2
3
4
6
5
5
>>> path_to_2 = yield_paths(t2, 2)
>>> sorted(list(path_to_2))
[[0, 2], [0, 2, 1, 2]]
Score: 1.0/1

---------------------------------------------------------------------
Point breakdown
```

```
differences: 1.0/1
merge: 1.0/1
perms: 1.0/1
yield_paths: 1.0/1

Score:
Total: 4.0

Cannot backup when running ok with --local.
---------------------------------------------------------------------
Final Score:2.0
```

Submitted Files

hw04.py

Download

```python
def differences(it):
    """
    Yields the differences between successive terms of iterable IT.

    >>> d = differences(iter([5, 2, -100, 103]))
    >>> [next(d) for _ in range(3)]
    [-3, -102, 203]
    >>> list(differences([1]))
    []
    """
    it = iter(it)
    first = next(it,None)
    second = next(it, None)
    if first == None:
        return
    if second == None:
        return
    yield second - first
    for curr in it:
        yield curr - second
        second = curr

def merge(x, y):
    """
    >>> def sequence(start, step):
    ...     while True:
    ...         yield start
    ...         start += step
    >>> x = sequence(2, 3) # 2, 5, 8, 11, 14, ...
    >>> y = sequence(3, 2) # 3, 5, 7, 9, 11, 13, 15, ...
    >>> result = merge(x, y) # 2, 3, 5, 7, 8, 9, 11, 13, 14, 15
    >>> [next(result) for _ in range(10)]
    [2, 3, 5, 7, 8, 9, 11, 13, 14, 15]
    """
    list_x = next(x)
    list_y = next(y)
    while True:
      if list_x < list_y:
         yield list_x
         list_x = next(x)
      if list_y < list_x:
         yield list_y
         list_y = next(y)
      else:
          yield list_x
          list_x = next(x)
          list_y = next(y)


```

```
def perms(seq):
    """Generates all permutations of the given sequence. Each permutation is a
    list of the elements in SEQ in a different order. The permutations may be
    yielded in any order.

    >>> p = perms([100])
    >>> type(p)
    <class 'generator'>
    >>> next(p)
    [100]
    >>> try: # Prints "No more permutations!" if calling next would cause an error
    ...     next(p)
    ... except StopIteration:
    ...     print('No more permutations!')
    No more permutations!
    >>> sorted(perms([1, 2, 3])) # Returns a sorted list containing elements of the generator
    [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
    >>> sorted(perms((10, 20, 30)))
    [[10, 20, 30], [10, 30, 20], [20, 10, 30], [20, 30, 10], [30, 10, 20], [30, 20, 10]]
    >>> sorted(perms("ab"))
    [['a', 'b'], ['b', 'a']]
    """
    if len(seq) == 1:
        yield list(seq)
    elif len(seq) == 2:
        yield list(seq)
        yield list(seq[::-1])
    else:
        for i in range(len(seq)):
            next_seq = seq[:i] + seq[i+1:]
            all_p = perms(next_seq)
            for a in all_p:
                new_p = [seq[i]] + list(a)
                yield new_p

def yield_paths(t, value):
    """Yields all possible paths from the root of t to a node with the label
    value as a list.

    >>> t1 = tree(1, [tree(2, [tree(3), tree(4, [tree(6)]), tree(5)]), tree(5)])
    >>> print_tree(t1)
    1
      2
        3
        4
          6
        5
      5
    >>> next(yield_paths(t1, 6))
    [1, 2, 4, 6]
    >>> path_to_5 = yield_paths(t1, 5)
    >>> sorted(list(path_to_5))
    [[1, 2, 5], [1, 5]]

    >>> t2 = tree(0, [tree(2, [t1])])
    >>> print_tree(t2)
    0
      2
        1
          2
            3
            4
              6
            5
          5
    >>> path_to_2 = yield_paths(t2, 2)
    >>> sorted(list(path_to_2))
    [[0, 2], [0, 2, 1, 2]]
    """
    if label(t) == value:
        yield [label(t)]
    for b in branches(t):
        for path in yield_paths(b, value):
            yield [label(t)] + path


def remainders_generator(m):
    """
    Yields m generators. The ith yielded generator yields natural numbers whose
    remainder is i when divided by m.

    >>> import types
    >>> [isinstance(gen, types.GeneratorType) for gen in remainders_generator(5)]
    [True, True, True, True, True]
    >>> remainders_four = remainders_generator(4)
    >>> for i in range(4):
    ...     print("First 3 natural numbers with remainder {0} when divided by 4:".format(i))
    ...     gen = next(remainders_four)
    ...     for _ in range(3):
    ...         print(next(gen))
    First 3 natural numbers with remainder 0 when divided by 4:
    4
    8
    12
    First 3 natural numbers with remainder 1 when divided by 4:
    1
    5
    9
    First 3 natural numbers with remainder 2 when divided by 4:
    2
    6
    10
    First 3 natural numbers with remainder 3 when divided by 4:
    3
```

```python
    7
    11
    """
    "*** YOUR CODE HERE ***"


# Tree ADT

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)])


def naturals():
    """A generator function that yields the infinite sequence of natural
    numbers, starting at 1.

    >>> m = naturals()
    >>> type(m)
    <class 'generator'>
    >>> [next(m) for _ in range(10)]
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    """
    i = 1
    while True:
        yield i
        i += 1

```