

# 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

```
1 def differences(it):
2     """
3     Yields the differences between successive terms of iterable IT.
4
5     >>> d = differences(iter([5, 2, -100, 103]))
6     >>> [next(d) for _ in range(3)]
7     [-3, -102, 203]
8     >>> list(differences([1]))
9     []
10    """
11    it = iter(it)
12    first = next(it, None)
13    second = next(it, None)
14    if first == None:
15        return
16    if second == None:
17        return
18    yield second - first
19    for curr in it:
20        yield curr - second
21        second = curr
22
23 def merge(x, y):
24     """
25     >>> def sequence(start, step):
26     ...     while True:
27     ...         yield start
28     ...         start += step
29     >>> x = sequence(2, 3) # 2, 5, 8, 11, 14, ...
30     >>> y = sequence(3, 2) # 3, 5, 7, 9, 11, 13, 15, ...
31     >>> result = merge(x, y) # 2, 3, 5, 7, 8, 9, 11, 13, 14, 15
32     >>> [next(result) for _ in range(10)]
33     [2, 3, 5, 7, 8, 9, 11, 13, 14, 15]
34     """
35    list_x = next(x)
36    list_y = next(y)
37    while True:
38        if list_x < list_y:
39            yield list_x
40            list_x = next(x)
41        if list_y < list_x:
42            yield list_y
43            list_y = next(y)
44        else:
45            yield list_x
46            list_x = next(x)
47            list_y = next(y)
48
49
```

```

50 def perms(seq):
51     """Generates all permutations of the given sequence. Each permutation is a
52     list of the elements in SEQ in a different order. The permutations may be
53     yielded in any order.
54
55     >>> p = perms([100])
56     >>> type(p)
57     <class 'generator'>
58     >>> next(p)
59     [100]
60     >>> try: # Prints "No more permutations!" if calling next would cause an error
61     ...     next(p)
62     ... except StopIteration:
63     ...     print('No more permutations!')
64     No more permutations!
65     >>> sorted(perms([1, 2, 3])) # Returns a sorted list containing elements of the generator
66     [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
67     >>> sorted(perms((10, 20, 30)))
68     [[10, 20, 30], [10, 30, 20], [20, 10, 30], [20, 30, 10], [30, 10, 20], [30, 20, 10]]
69     >>> sorted(perms("ab"))
70     [['a', 'b'], ['b', 'a']]
71     """
72     if len(seq) == 1:
73         yield list(seq)
74     elif len(seq) == 2:
75         yield list(seq)
76         yield list(seq[::-1])
77     else:
78         for i in range(len(seq)):
79             next_seq = seq[:i] + seq[i+1:]
80             all_p = perms(next_seq)
81             for a in all_p:
82                 new_p = [seq[i]] + list(a)
83                 yield new_p
84
85 def yield_paths(t, value):
86     """Yields all possible paths from the root of t to a node with the label
87     value as a list.
88
89     >>> t1 = tree(1, [tree(2, [tree(3), tree(4, [tree(6))], tree(5))], tree(5)])
90     >>> print_tree(t1)
91     1
92     2
93     3
94     4
95     6
96     5
97     5
98     >>> next(yield_paths(t1, 6))
99     [1, 2, 4, 6]
100    >>> path_to_5 = yield_paths(t1, 5)
101    >>> sorted(list(path_to_5))

```

```

102 [[1, 2, 5], [1, 5]]
103
104 >>> t2 = tree(0, [tree(2, [t1])])
105 >>> print_tree(t2)
106 0
107   2
108    1
109     2
110     3
111     4
112     6
113     5
114     5
115 >>> path_to_2 = yield_paths(t2, 2)
116 >>> sorted(list(path_to_2))
117 [[0, 2], [0, 2, 1, 2]]
118 """
119 if label(t) == value:
120     yield [label(t)]
121 for b in branches(t):
122     for path in yield_paths(b, value):
123         yield [label(t)] + path
124
125
126 def remainders_generator(m):
127     """
128     Yields m generators. The ith yielded generator yields natural numbers whose
129     remainder is i when divided by m.
130
131     >>> import types
132     >>> [isinstance(gen, types.GeneratorType) for gen in remainders_generator(5)]
133     [True, True, True, True, True]
134     >>> remainders_four = remainders_generator(4)
135     >>> for i in range(4):
136     ...     print("First 3 natural numbers with remainder {0} when divided by 4:".format(i))
137     ...     gen = next(remainders_four)
138     ...     for _ in range(3):
139     ...         print(next(gen))
140     First 3 natural numbers with remainder 0 when divided by 4:
141     4
142     8
143     12
144     First 3 natural numbers with remainder 1 when divided by 4:
145     1
146     5
147     9
148     First 3 natural numbers with remainder 2 when divided by 4:
149     2
150     6
151     10
152     First 3 natural numbers with remainder 3 when divided by 4:
153     3

```

```

154     7
155     11
156     """
157     """*** YOUR CODE HERE ***"""
158
159
160 # Tree ADT
161
162 def tree(label, branches=[]):
163     """Construct a tree with the given label value and a list of branches."""
164     for branch in branches:
165         assert is_tree(branch), 'branches must be trees'
166     return [label] + list(branches)
167
168
169 def label(tree):
170     """Return the label value of a tree."""
171     return tree[0]
172
173
174 def branches(tree):
175     """Return the list of branches of the given tree."""
176     return tree[1:]
177
178
179 def is_tree(tree):
180     """Returns True if the given tree is a tree, and False otherwise."""
181     if type(tree) != list or len(tree) < 1:
182         return False
183     for branch in branches(tree):
184         if not is_tree(branch):
185             return False
186     return True
187
188
189 def is_leaf(tree):
190     """Returns True if the given tree's list of branches is empty, and False
191     otherwise.
192     """
193     return not branches(tree)
194
195
196 def print_tree(t, indent=0):
197     """Print a representation of this tree in which each node is
198     indented by two spaces times its depth from the root.
199
200     >>> print_tree(tree(1))
201     1
202     >>> print_tree(tree(1, [tree(2)]))
203     1
204         2
205     >>> numbers = tree(1, [tree(2), tree(3, [tree(4), tree(5)]), tree(6, [tree(7)])])

```

```
206 >>> print_tree(numbers)
207 1
208 2
209 3
210 4
211 5
212 6
213 7
214 """
215 print(' ' * indent + str(label(t)))
216 for b in branches(t):
217     print_tree(b, indent + 1)
218
219
220 def copy_tree(t):
221     """Returns a copy of t. Only for testing purposes.
222
223     >>> t = tree(5)
224     >>> copy = copy_tree(t)
225     >>> t = tree(6)
226     >>> print_tree(copy)
227     5
228     """
229     return tree(label(t), [copy_tree(b) for b in branches(t)])
230
231
232 def naturals():
233     """A generator function that yields the infinite sequence of natural
234     numbers, starting at 1.
235
236     >>> m = naturals()
237     >>> type(m)
238     <class 'generator'>
239     >>> [next(m) for _ in range(10)]
240     [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
241     """
242     i = 1
243     while True:
244         yield i
245         i += 1
246
```