Homework 5 | CS 61A Spring 2024

Homework 5: Generators

- hw05.zip

Due by 11:59pm on Thursday, March 7

Instructions

Download hw05.zip. Inside the archive, you will find a file called hw05.py, along with a copy of the `ok` autograder.

Submission: When you are done, submit the assignment by uploading all code files you've edited to Gradescope. You may submit more than once before the deadline; only the final submission will be scored. Check that you have successfully submitted your code on Gradescope. See Lab 0 for more instructions on submitting assignments.

Using Ok: If you have any questions about using Ok, please refer to this guide.

Readings: You might find the following references useful:

- Section 4.2

Grading: Homework is graded based on correctness. Each incorrect problem will decrease the total score by one point. **This homework is out of 2 points.**

Required Questions

Getting Started Videos

These videos may provide some helpful direction for tackling the coding problems on this assignment.

> To see these videos, you should be logged into your berkeley.edu email.

YouTube link

Q1: Infinite Hailstone

Write a generator function that yiels the elements of the hailstone sequence starting at number `n`. After reaching the end of the hailstone sequence, the generator should yield the value 1 indefinitely.

Here's a quick reminder of how the hailstone sequence is defined:

1. Pick a positive integer `n` as the start.
2. If `n` is even, divide it by 2.
3. If `n` is odd, multiply it by 3 and add 1.
4. Continue this process until `n` is 1.

Try to write this generator function recursively. If you're stuck, you can first try writing it iteratively and then seeing how you can turn that implementation into a recursive one.

> **Hint:** Since `hailstone` returns a generator, you can `yield from` a call to `hailstone`!

```
def hailstone(n):
    """Q1: Yields the elements of the hailstone sequence starting at n.
       At the end of the sequence, yield 1 infinitely.

    >>> hail_gen = hailstone(10)
    >>> [next(hail_gen) for _ in range(10)]
    [10, 5, 16, 8, 4, 2, 1, 1, 1, 1]
    >>> next(hail_gen)
    1
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q hailstone
```

Q2: Merge

Write a generator function `merge` that takes in two infinite generators `a` and `b` that are in increasing order without duplicates and returns a generator that has all the elements of both generators, in increasing order, without duplicates.

```
def merge(a, b):
    """Q2:
    >>> def sequence(start, step):
    ...     while True:
    ...         yield start
    ...         start += step
    >>> a = sequence(2, 3) # 2, 5, 8, 11, 14, ...
    >>> b = sequence(3, 2) # 3, 5, 7, 9, 11, 13, 15, ...
    >>> result = merge(a, b) # 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]
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q merge
```

Q3: Yield Paths

Define a generator function `yield_paths` which takes in a tree `t`, a value `value`, and returns a generator object which yields each path from the root of `t` to a node that has label `value`.

Each path should be represented as a list of the labels along that path in the tree. You may yield the paths in any order.

```
def yield_paths(t, value):
    """Q4: 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 ____
    for b in branches(t):
        for ____ in ____:
            yield ____
```

Hint: If you're having trouble getting started, think about how you'd approach this problem if it wasn't a generator function. What would your recursive calls be? With a generator function, what happens if you make a "recursive call" within its body?

Hint: Try coming up with a few simple cases of your own! How should this function work when `t` is a leaf node?

Hint: Remember, it's possible to loop over generator objects because generators are iterators!

Note: Remember that this problem should **yield paths** -- do not return a list of paths!

Use Ok to test your code:

```
python3 ok -q yield_paths
```

Check Your Score Locally

You can locally check your score on each question of this assignment by running

```
python3 ok --score
```

This does NOT submit the assignment! When you are satisfied with your score, submit the assignment to Gradescope to receive credit for it.

Submit

Submit this assignment by uploading any files you've edited **to the appropriate Gradescope assignment.** Lab 00 has detailed instructions.

Exam Practice

Homework assignments will also contain prior exam questions for you to try. These questions have no submission component; feel free to attempt them if you'd like some practice!

1. Summer 2018 Final Q7a,b: Streams and Jennyrators
2. Spring 2019 Final Q1: Iterators are inevitable
3. Spring 2021 MT2 Q8: The Tree of L-I-F-E
4. Summer 2016 Final Q8: Zhen-erators Produce Power
5. Spring 2018 Final Q4a: Apply Yourself