Lab 8: Linked Lists, Trees **lab08.zip (lab08.zip)**

Due by 11:59pm on Tuesday, October 20.

Starter Files

Download lab08.zip (lab08.zip). Inside the archive, you will find starter files for the questions in this lab, along with a copy of the Ok (ok) autograder.

Topics

Consult this section if you need a refresher on the material for this lab. It's okay to skip directly to the questions and refer back here should you get stuck.

Linked Lists

Motivation

Trees

Required Questions

What Would Python Display?

Q1: WWPD: Linked Lists

Read over the `Link` class in `lab08.py`. Make sure you understand the doctests.

> Use Ok to test your knowledge with the following "What Would Python Display?" questions:
>
> ```
> python3 ok -q link -u
> ```
>
> Enter `Function` if you believe the answer is `<function ...>`, `Error` if it errors, and `Nothing` if nothing is displayed.
>
> If you get stuck, try drawing out the box-and-pointer diagram for the linked list on a piece of paper or loading the `Link` class into the interpreter with `python3 -i lab09.py`.

```
>>> from lab08 import *
>>> link = Link(1000)
>>> link.first
------

>>> link.rest is Link.empty
------

>>> link = Link(1000, 2000)
------

>>> link = Link(1000, Link())
------
```

```
>>> from lab08 import *
>>> link = Link(1, Link(2, Link(3)))
>>> link.first
______

>>> link.rest.first
______

>>> link.rest.rest.rest is Link.empty
______

>>> link.first = 9001
>>> link.first
______

>>> link.rest = link.rest.rest
>>> link.rest.first
______

>>> link = Link(1)
>>> link.rest = link
>>> link.rest.rest.rest.rest.first
______

>>> link = Link(2, Link(3, Link(4)))
>>> link2 = Link(1, link)
>>> link2.first
______

>>> link2.rest.first
______
```

```
>>> from lab08 import *
>>> link = Link(5, Link(6, Link(7)))
>>> link                  # Look at the __repr__ method of Link
______

>>> print(link)          # Look at the __str__ method of Link
______
```

Linked Lists

Q2: Convert Link

Write a function `convert_link` that takes in a linked list and returns the sequence as a Python list. You may assume that the input list is shallow; none of the elements is another linked list.

Try to find both an iterative and recursive solution for this problem!

```
def convert_link(link):
    """Takes a linked list and returns a Python list with the same elements.

    >>> link = Link(1, Link(2, Link(3, Link(4))))
    >>> convert_link(link)
    [1, 2, 3, 4]
    >>> convert_link(Link.empty)
    []
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q convert_link
```

Q3: Every Other

Implement `every_other`, which takes a linked list `s`. It mutates `s` such that all of the odd-indexed elements (using 0-based indexing) are removed from the list. For example:

```
>>> s = Link('a', Link('b', Link('c', Link('d'))))
>>> every_other(s)
>>> s.first
'a'
>>> s.rest.first
'c'
>>> s.rest.rest is Link.empty
True
```

If `s` contains fewer than two elements, `s` remains unchanged.

> Do not return anything! `every_other` should mutate the original list.

```
def every_other(s):
    """Mutates a linked list so that all the odd-indiced elements are removed
    (using 0-based indexing).

    >>> s = Link(1, Link(2, Link(3, Link(4))))
    >>> every_other(s)
    >>> s
    Link(1, Link(3))
    >>> odd_length = Link(5, Link(3, Link(1)))
    >>> every_other(odd_length)
    >>> odd_length
    Link(5, Link(1))
    >>> singleton = Link(4)
    >>> every_other(singleton)
    >>> singleton
    Link(4)
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q every_other
```

Trees

Q4: Cumulative Mul

Write a function `cumulative_mul` that mutates the Tree `t` so that each node's label becomes the product of all labels in the subtree rooted at the node.

```
def cumulative_mul(t):
    """Mutates t so that each node's label becomes the product of all labels in
    the corresponding subtree rooted at t.

    >>> t = Tree(1, [Tree(3, [Tree(5)]), Tree(7)])
    >>> cumulative_mul(t)
    >>> t
    Tree(105, [Tree(15, [Tree(5)]), Tree(7)])
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q cumulative_mul
```

Optional Problems

Q5: Cycles

The `Link` class can represent lists with cycles. That is, a list may contain itself as a sublist.

```
>>> s = Link(1, Link(2, Link(3)))
>>> s.rest.rest.rest = s
>>> s.rest.rest.rest.rest.rest.first
3
```

Implement `has_cycle`,that returns whether its argument, a `Link` instance, contains a cycle.

> *Hint*: Iterate through the linked list and try keeping track of which `Link` objects you've already seen.

```
def has_cycle(link):
    """Return whether link contains a cycle.

    >>> s = Link(1, Link(2, Link(3)))
    >>> s.rest.rest.rest = s
    >>> has_cycle(s)
    True
    >>> t = Link(1, Link(2, Link(3)))
    >>> has_cycle(t)
    False
    >>> u = Link(2, Link(2, Link(2)))
    >>> has_cycle(u)
    False
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q has_cycle
```

As an extra challenge, implement `has_cycle_constant` with only constant space (https://web.archive.org/web/20201214030141/http://composingprograms.com/pages/28-efficiency.html#growth-categories). (If you followed the hint above, you will use linear space.) The solution is short (less than 20 lines of code), but requires a clever idea. Try to discover the solution yourself before asking around:

```
def has_cycle_constant(link):
    """Return whether link contains a cycle.

    >>> s = Link(1, Link(2, Link(3)))
    >>> s.rest.rest.rest = s
    >>> has_cycle_constant(s)
    True
    >>> t = Link(1, Link(2, Link(3)))
    >>> has_cycle_constant(t)
    False
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q has_cycle_constant
```

Q6: Reverse Other

Write a function `reverse_other` that mutates the tree such that **labels** on *every other* (odd-depth) level are reversed. For example, `Tree(1,[Tree(2, [Tree(4)]), Tree(3)])` becomes `Tree(1, [Tree(3, [Tree(4)]), Tree(2)])`. Notice that the nodes themselves are *not* reversed; only the labels are.

```
def reverse_other(t):
    """Mutates the tree such that nodes on every other (odd-depth) level
    have the labels of their branches all reversed.

    >>> t = Tree(1, [Tree(2), Tree(3), Tree(4)])
    >>> reverse_other(t)
    >>> t
    Tree(1, [Tree(4), Tree(3), Tree(2)])
    >>> t = Tree(1, [Tree(2, [Tree(3, [Tree(4), Tree(5)]), Tree(6, [Tree(7)])]), Tree(8)])
    >>> reverse_other(t)
    >>> t
    Tree(1, [Tree(8, [Tree(3, [Tree(5), Tree(4)]), Tree(6, [Tree(7)])]), Tree(2)])
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q reverse_other
```

CS 61A

Weekly Schedule (/web/20201214030141/https://cs61a.org/weekly.html)

Office Hours (/web/20201214030141/https://cs61a.org/office-hours.html)

Staff (/web/20201214030141/https://cs61a.org/staff.html)

Resources

Studying Guide (/web/20201214030141/https://cs61a.org/articles/studying.html)

Debugging Guide (/web/20201214030141/https://cs61a.org/articles/debugging.html)

Composition Guide (/web/20201214030141/https://cs61a.org/articles/composition.html)

Policies

Assignments (/web/20201214030141/https://cs61a.org/articles/about.html#as

Exams (/web/20201214030141/https://cs61a.org/articles/about.html#ex

Grading (/web/20201214030141/https://cs61a.org/articles/about.html#gr