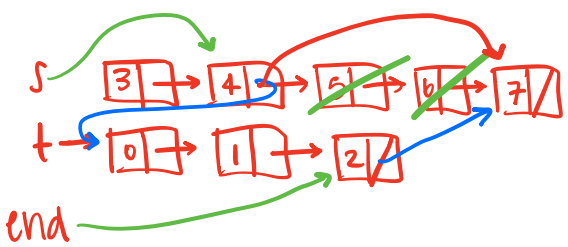
CS61A Exam Prep 7

Fall 2018 Midterm 2 Q6



6. (4 points) Dr. Frankenlink

Implement `replace`, which takes two non-empty linked lists `s` and `t`, as well as **positive** integers `i` and `j` with `i < j`. It mutates `s` by removing elements with indices from `i` to `j` (removing element i but not removing element `j`) and replacing them with `t`. Afterward, `s` contains all of the objects in `t`, so a change to `t` would be reflected in `s` as well. `t` may change as a result of calling `replace`. Assume `s` has at least `j` elements.

```
def replace(s, t, i, j):
    """Replace the slice of s from i to j with t.
                                0  1  2  3  4
    >>> s, t = Link(3, Link(4, Link(5, Link(6, Link(7)))), Link(0, Link(1, Link(2)))
    >>> replace(s, t, 2, 4)
    >>> print(s)
    <3, 4, 0, 1, 2, 7>
    >>> t.rest.first = 8
    >>> print(s)
    <3, 4, 0, 8, 2, 7>
    """
    assert s is not Link.empty and t is not Link.empty and i > 0 and i < j

    if i > 1:
        replace(s.rest, t, i-1, j-1)
    else:
        for k in range(j - i):
            s.rest = s.rest.rest
        end = t
        while end.rest is not Link.empty:
            end = end.rest
        s.rest, end.rest = t, s.rest
```

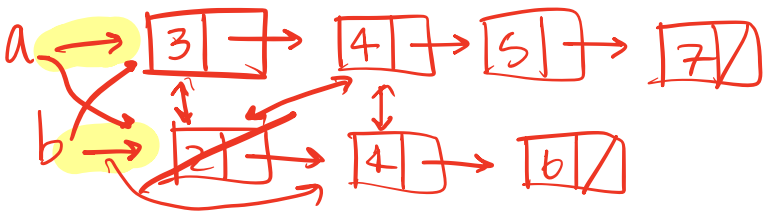
Fall 2017 Midterm 2 Q4a

(a) **(4 pt)** Implement `both`, which takes two *sorted* linked lists composed of `Link` objects and returns whether some value is in both of them. The `Link` class is defined on the midterm 2 study guide.
Important: You may **not** use `len`, `in`, `for`, `list`, slicing, element selection, addition, or list comprehensions.

```
def both(a, b):
    """Return whether there is any value that appears in both a and b, two sorted Link instances.

    >>> both(Link(1, Link(3, Link(5, Link(7)))), Link(2, Link(4, Link(6))))
    False
    >>> both(Link(1, Link(3, Link(5, Link(7)))), Link(2, Link(7, Link(9))))  # both have 7
    True
    >>> both(Link(1, Link(4, Link(5, Link(7)))), Link(2, Link(4, Link(5))))  # both have 4 and 5
    True
    """
    if a is Link.empty or b is Link.empty:
        return False
    if a.first > b.first:
        a, b = b, a
    return a.first == b.first or both(a.rest, b)
```

base case



Fall 2016 Midterm 2 Q5

5. (6 points) Trick or Tree

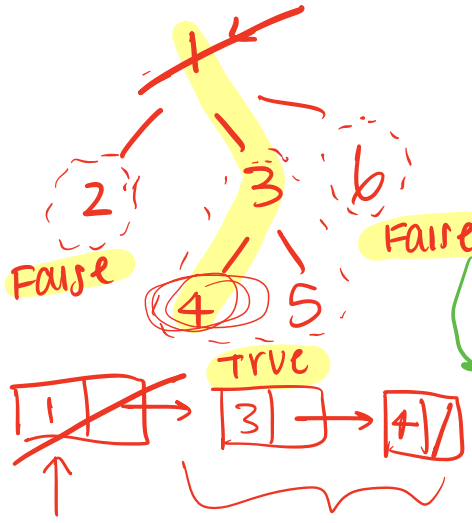
Implement `path`, which takes a linked list `s` and a `Tree` instance `t`. It returns whether `s` is a path from the root of `t` to some leaf. The `Tree` and `Link` classes are on page 2 of the midterm 2 study guide.

Restrictions:

- You may not call the built-in `len` function on a linked list or invoke its `__len__` method.
- You may not apply element selection (e.g., `s[2]`) on a linked list or invoke its `__getitem__` method.

```
def path(s, t):
    """Return whether Link S is a path from the root to a leaf in Tree T.

    >>> t = Tree(1, [Tree(2), Tree(3, [Tree(4), Tree(5)]), Tree(6)])
    >>> a = Link(1, Link(3, Link(4)))  # A full path
    >>> path(a, t)
    True
    >>> b = Link(1, Link(3))           # A partial path
    >>> path(b, t)
    False
    >>> c = Link(1, Link(2, Link(7)))  # A path and an extra value
    >>> path(c, t)
    False
    >>> d = Link(3, Link(4))           # A path of a branch
    >>> path(d, t)
    False
    """
    if t.label != s.first or s is Link.empty:
        return False
    if t.is_leaf() and s.rest is Link.empty
        and t.label == s.first:
        return True
    return any([path(s.rest, b) for b in t.branches])
```

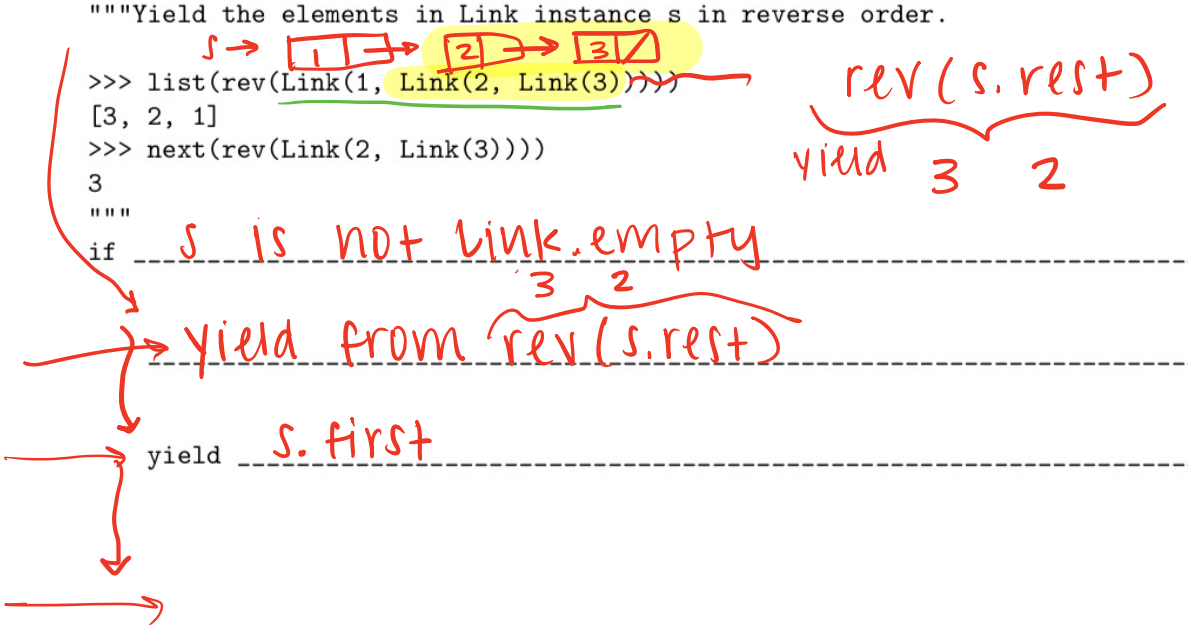


any →
all

Fall 2018 Final Q6

(a) (4 pt) Implement `rev`, a generator function that takes a `Link` instance and yields the elements of that linked list in reverse order. The `Link` class appears on Page 2 of the Midterm 2 Study Guide.

```
def rev(s):
    """Yield the elements in Link instance s in reverse order.
    >>> list(rev(Link(1, Link(2, Link(3)))))
    [3, 2, 1]
    >>> next(rev(Link(2, Link(3))))
    3
    """
    if ____s is not Link.empty____:
        ____yield from rev(s.rest)____
    yield ____s.first____
```



melaniecooray@berkeley.edu