CS 61A
Spring 2026

Linked Lists

Discussion 8: March 18, 2026

Linked Lists

A linked list is a `Link` object or `Link.empty`.

You can mutate a `Link` object `s` in two ways:

- Change the first element with `s.first = ...`
- Change the rest of the elements with `s.rest = ...`

```
class Link:
    """A linked list is either a Link object or Link.empty

    >>> s = Link(3, Link(4, Link(5)))
    >>> s.rest
    Link(4, Link(5))
    >>> s.rest.rest.rest is Link.empty
    True
    >>> s.rest.first * 2
    8
    >>> print(s)
    (3 4 5)
    """
    empty = ()

    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest

    def __repr__(self):
        if self.rest:
            rest_repr = ', ' + repr(self.rest)
        else:
            rest_repr = ''
        return 'Link(' + repr(self.first) + rest_repr + ')'

    def __str__(self):
        string = '('
        while self.rest is not Link.empty:
            string += str(self.first) + ' '
            self = self.rest
        return string + str(self.first) + ')'
```

Q1: Strange Loop

In lab, there was a `Link` object with a cycle that represented an infinite repeating list of 1's.

```
>>> ones = Link(1)
>>> ones.rest = ones
>>> [ones.first, ones.rest.first, ones.rest.rest.first, ones.rest.rest.rest.first]
[1, 1, 1, 1]
>>> ones.rest is ones
True
```

Implement `strange_loop`, which takes no arguments and returns a `Link` object `s` for which `s.rest.first.rest` is `s`.

Draw a picture of the linked list you want to create, then write code to create it.

```
def strange_loop():
    """Return a Link s for which s.rest.first.rest is s.

    >>> s = strange_loop()
    >>> s.rest.first.rest is s
    True
    """
    s = Link(1, Link(Link(2)))
    s.rest.first.rest = s
    return s
```

Q2: Sum Two Ways

Implement both `sum_rec` and `sum_iter`. Each one takes a linked list of numbers `s` and a non-negative integer `k` and returns the sum of the first `k` elements of `s`. If there are fewer than `k` elements in `s`, all of them are summed. If `k` is 0 or `s` is empty, the sum is 0.

Use recursion to implement `sum_rec`. Don't use recursion to implement `sum_iter`; use a `while` loop instead.

To get started on the recursive implementation, consider the example `a = Link(1, Link(6, Link(8)))`, and the call `sum_rec(a, 2)`. Write down the recursive call to `sum_rec` that would help compute `sum_rec(a, 2)`. Then, write down what that recursive call should return. Discuss how this return value is useful in computing the return value of `sum_rec(a, 2)`.

```
def sum_rec(s, k):
    """Return the sum of the first k elements in s.

    >>> a = Link(1, Link(6, Link(8)))
    >>> sum_rec(a, 2)
    7
    >>> sum_rec(a, 5)
    15
    >>> sum_rec(Link.empty, 1)
    0
    """
    # Use a recursive call to sum_rec; don't call sum_iter
    if k == 0 or s is Link.empty:
        return 0
    return s.first + sum_rec(s.rest, k - 1)

def sum_iter(s, k):
    """Return the sum of the first k elements in s.

    >>> a = Link(1, Link(6, Link(8)))
    >>> sum_iter(a, 2)
    7
    >>> sum_iter(a, 5)
    15
    >>> sum_iter(Link.empty, 1)
    0
    """
    # Don't call sum_rec or sum_iter
    total = 0
    while k > 0 and s is not Link.empty:
        total, s, k = total + s.first, s.rest, k - 1
    return total
```

Q3: Overlap

Implement `overlap`, which takes two linked lists of numbers called `s` and `t` that are sorted in increasing order and have no repeated elements within each list. It returns the count of how many numbers appear in both lists.

This can be done in *linear* time in the combined length of `s` and `t` by always advancing forward in the linked list whose first element is smallest until both first elements are equal (add one to the count and advance both) or one list is empty (time to return).

```
def overlap(s, t):
    """For increasing s and t, count the numbers that appear in both.

    >>> a = Link(3, Link(4, Link(6, Link(7, Link(9, Link(10))))))
    >>> b = Link(1, Link(3, Link(5, Link(7, Link(8)))))
    >>> overlap(a, b)  # 3 and 7
    2
    >>> overlap(a.rest, b)  # just 7
    1
    >>> overlap(Link(0, a), Link(0, b))
    3
    """
    if s is Link.empty or t is Link.empty:
        return 0
    if s.first == t.first:
        return 1 + overlap(s.rest, t.rest)
    elif s.first < t.first:
        return overlap(s.rest, t)
    elif s.first > t.first:
        return overlap(s, t.rest)


def overlap_iterative(s, t):
    """For increasing s and t, count the numbers that appear in both.

    >>> a = Link(3, Link(4, Link(6, Link(7, Link(9, Link(10))))))
    >>> b = Link(1, Link(3, Link(5, Link(7, Link(8)))))
    >>> overlap(a, b)  # 3 and 7
    2
    >>> overlap(a.rest, b)  # just 7
    1
    >>> overlap(Link(0, a), Link(0, b))
    3
    """
    res = 0
    while s is not Link.empty and t is not Link.empty:
        if s.first == t.first:
            res += 1
            s = s.rest
            t = t.rest
        elif s.first < t.first:
            s = s.rest
        else:
            t = t.rest
    return res
```

Q4: Iterate in Order

Implement `iterate_in_order`, which takes two linked lists of numbers called `s` and `t` that are sorted in increasing order and have no repeated elements within each list. It returns a generator that iterates over all items in `s` and `t` in non-decreasing order.

```
def iterate_in_order(s, t):
    """For increasing s and t, yields the elements in s and t, in non-decreasing order.

    >>> a = Link(3, Link(4, Link(6, Link(7, Link(9, Link(10))))))
    >>> b = Link(1, Link(3, Link(5, Link(7, Link(8)))))
    >>> t = iterate_in_order(a, b)
    >>> for item in t:
    ...     print(item)
    1
    3
    3
    4
    5
    6
    7
    7
    8
    9
    10
    >>> t = iterate_in_order(Link.empty, b)
    >>> for item in t:
    ...      print(item)
    1
    3
    5
    7
    8
    """
    while s is not Link.empty or t is not Link.empty:
        if s is Link.empty:
            yield t.first
            t = t.rest
        elif t is Link.empty or s.first <= t.first:
            yield s.first
            s = s.rest
        else:
            yield t.first
            t = t.rest
```

Extra Challenging Extra Fun

This last question is similar in complexity to an A+ question on an exam. Feel free to skip it, but it's a fun one, so try it if you have time.

Q5: Decimal Expansion

Definition. The *decimal expansion* of a fraction `n/d` with `n < d` is an infinite sequence of digits starting with the 0 before the decimal point and followed by digits that represent the tenths, hundredths, and thousands place (and so on) of the number `n/d`. E.g., the decimal expansion of 2/3 is a zero followed by an infinite sequence of 6's: 0.6666666….

Implement `divide`, which takes positive integers `n` and `d` with `n < d`. It returns a linked list with a cycle containing the digits of the infinite decimal expansion of `n/d`. The provided `display` function prints the first `k` digits after the decimal point.

For example, 1/22 would be represented as `x` below:

```
>>> 1/22
0.045454545454545456
>>> x = Link(0, Link(0, Link(4, Link(5))))
>>> x.rest.rest.rest.rest = x.rest.rest
>>> display(x, 20)
0.04545454545454545454...
```

```
def display(s, k=10):
    """Print the first k digits of infinite linked list s as a decimal.

    >>> s = Link(0, Link(8, Link(3)))
    >>> s.rest.rest.rest = s.rest.rest
    >>> display(s)
    0.8333333333...
    """
    assert s.first == 0, f'{s.first} is not 0'
    digits = f'{s.first}.'
    s = s.rest
    for _ in range(k):
        assert s.first >= 0 and s.first < 10, f'{s.first} is not a digit'
        digits += str(s.first)
        s = s.rest
    print(digits + '...')
```

```
def divide(n, d):
    """Return a linked list with a cycle containing the digits of n/d.

    >>> display(divide(5, 6))
    0.8333333333...
    >>> display(divide(2, 7))
    0.2857142857...
    >>> display(divide(1, 2500))
    0.0004000000...
    >>> display(divide(3, 11))
    0.2727272727...
    >>> display(divide(3, 99))
    0.0303030303...
    >>> display(divide(2, 31), 50)
    0.06451612903225806451612903225806451612903225806451...
    """
    assert n > 0 and n < d
    result = Link(0)  # The zero before the decimal point
    cache = {}
    tail = result
    while n not in cache:
        q, r = 10 * n // d, 10 * n % d
        tail.rest = Link(q)
        tail = tail.rest
        cache[n] = tail
        n = r
    tail.rest = cache[n]









    return result
```