Module 05 Homework - Recursion

Part 1 - Recursion on a Linked List

Linked lists can be thought of as recursive structures. Every linked list is either:

- the empty list, or
- a node that links to a linked list

This makes them a great playground for implementing various types of recursive functions. We provide starter code that demonstrates a few methods of recursion in a LinkedList, and you will implement a few more.

Starter Code

The `LinkedList` class is already fully implemented. It is essentially a wrapper - the `LLNode` class does all the interesting recursion.

In `LLNode`, we have implemented:

- `__init__`
- `__len__()`
- `__repr__()`
- `get_tail()`

`__len__()`, `__repr__()`, and `get_tail()` show various examples of recursive `LLNode` functions. Study them to understand how they work, then implement the following `LLNode` methods. Note that we have listed the parameters you should use - this is a great hint into how you can efficiently structure your code. Do not change these.

- `add_last(self, item)`
  - recursively call on linked node until you hit the tail, at which point you should add a new node
  - we are not keeping track of a tail node, so this has $\mathcal{O}(n)$ running time
- `sum_all(self)`
  - recursively add all items in linked list and return the sum
  - $\mathcal{O}(n)$
- `contains(self, item)`
  - recursively call on linked node until you either:
    - find the item and return True, or
    - hit the end of the list without finding the item and return False
  - $\mathcal{O}(n)$ running time
- `remove_all(self, item)`
  - recursively removes all nodes holding item in linked list
  - $\mathcal{O}(n)$ running time
- `reverse(self)`
  - recursively reverses the linked list contained in `self.link` unless it is the tail node
  - after reversing the linked list in `self.link`, puts itself at the end of the reversed list
  - this method should return the new head of the list (i.e., the *old tail*)
  - $\mathcal{O}(n)$ running time

Part 2 - Rolling Dice to a Total

Given an `n`-sided die, we want to determine how many different ways one could repeatedly sum die rolls and get to a given total.

For instance, given a 4-sided die, we could obtain a total of 3 in four different ways:

- 1 + 1 + 1 (rolling a 1 three times)
- 1 + 2 (rolling a 1 and then rolling a 2)
- 2 + 1 (rolling a 2 and then rolling a 1)
- 3 (rolling a 3)

We will write two different functions to solve this problem. Each functions takes `n` (the number of sides on the die) and `total` (the total to which we'll sum rolls) as input. Each function must use its corresponding dynamic programming methodology in order to solve the problem.

- `ways_to_sum_memo(n, total)`
  - must use memoization
- `ways_to_sum_tab(n, total)`
  - must use tabulation

```
>>> ways_to_sum_memo(n=4, total=3)
4
>>> ways_to_sum_tab(n=4, total=3)
4
```

Imports

No imports allowed on this assignment, with the following exceptions:

- Any modules you have written yourself
- `typing` - this is not required, but some students have requested it
- For testing only (do not use these for functionality in any other classes/algorithms):
  - `unittest`
  - `random`

Submission Requirements

Students must submit **individually** by the due date.

At a minimum, you must submit the following files:

- `linked_list.py`
- `ways_to_sum.py`

Additionally, you must include any other files necessary for your code to run, such as modules containing data structures you wrote yourself.

Note that we are not requiring tests here. We have instead provided tests for you. We'll begin requiring tests again for the next homework, but we are prioritizing giving you multiple chances to practice recursion this week, since it's an area where we typically need a lot of practice. See here for some more practice problems.

Grading

This assignment is 100% manually graded. Your code will be graded on structure, efficiency, readability, and correctness. Autograder results are not fully indicative of your submission's grade. Egregiously poor structural descions, non-recursive solutions, or brute forcing unittests with will not receive any credit.