CSC148 Summer 2018: Lab 4

Introduction

The goals of this lab are:

- To get you familiar with how LinkedListNodes work
- To get you familiar with how LinkedLists work
- To give you practice traversing LinkedLists
- To give you practice updating LinkedLists

Don't hesitate to make use of other resources for this lab, including the course notes, your TAs, instructor, or other students.

General Lab Notes

1. Make sure you have lab_pyta.txt downloaded and placed in the directory (or directories) where you'll be working.
2. To use PythonTA, include the following code (if you already have a main block, just add the body to the end of it):

```
if __name__ == '__main__':
    import python_ta
    python_ta.check_all(config="lab_pyta.txt")
```

Your lab_pyta.txt should be in the same folder as the .py files you're running. PythonTA will raise errors regarding style, specifying the lines you need to fix. You should get familiar with what the errors mean, and how to fix them: this will be important for your exercises and assignments.

Getting Started

This lab will have you using LinkedLists and LinkedListNodes which only have very basic functionality. Download linkedlist_midterm.py or write your own LinkedList and LinkedListNode classes. The contents of linkedlist_midterm.py is also the code that will be provided to you on the midterm (i.e. you only have `__str__`, `__init__`, and the `prepend` method).

Finding the second last node in a LinkedList (find_second_last)

Write a method in the LinkedList class called find_second_last. This should return the node that points to the back of the LinkedList. If there is no such node (i.e. there are less than 2 items in the LinkedList), return None.

Get all values in a LinkedList (get_values)

Write a method in the LinkedList class called get_values which returns a list of all of the values (from front to back) within the LinkedList.

Comparing 2 LinkedLists (___eq___)

Write the ___eq___ method for the LinkedList class. Two LinkedLists are equal if they have the same size, and all of the elements in the LinkedLists have the same values in the same order.

Try 2 approaches to this: 1 that uses get_values(), and another that traverses both LinkedLists at the same time using the pattern:

```
cur_self = self.front
cur_other = other.front
while cur_self != None:
    # Put something here
    cur_self = cur_self.next_
    cur_other = cur_other.next_
```

Find a LinkedListNode with a given value (find_value)

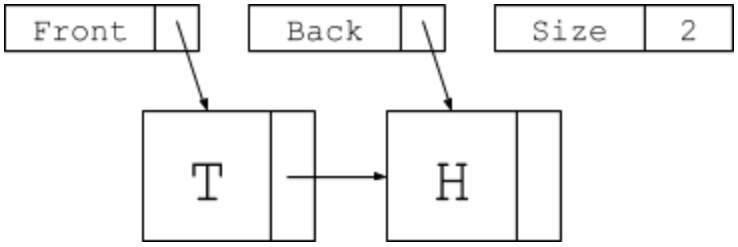
Write a method in the LinkedList class called find_value which returns the LinkedListNode in the LinkedList with a given value.

If no such node is found, return None.

Add a value after a given value (add_after)

Write a method in the LinkedList class called add_after which takes 2 parameters: A value to add, and the value of the node that we want to add our new value after. Assume we only want to add after the first occurrence of that value.

Before writing the method, consider the case of a LinkedList called `lnk` that looks like this:



Draw what the LinkedList will look like after we call `lnk.add_after("E", "H")`.

What next_ pointers do we have to change? Do we have to change back? Size? Front?

Add a value after all occurrences of a given value (add_after_every)

Write a method in the LinkedList class called add_after_every which works like add_after, but instead of adding after the first occurrence, add the value after every occurrence.

Suppose we have a LinkedList called `lnk` with the items (from front to back): "B", "A", "N", "N". Draw how the LinkedList should look after calling `lnk.add_after_every("A", "N")`.