Assignment 2
Due Monday, 9/17

September 13, 2018
CS: DS&A
Proof School

Instructions

Write the functions below in a single .py document. Import and use `linked_list.py` from class; your first line should be something like `import linked_list as LL`.

Use the exact function names given in the problems below. You don't have to use the input variable names I specify, but your function has to preserve the order I gave. For instance, in problem 1, you can start your function definition with `def get_Nth(linked_list, num)`. But keep the name `get_Nth`, and make sure the linked list is the first argument.

Also, your code should handle special cases, like a list with one node, or with zero nodes. But you may assume that your input lists have no loops (i.e. there are no repeated Node objects within a list.)

Start early, and have fun!

0. Warm-up: Write a function called `length(my_list)` that returns the number of nodes in the linked list.

1. Write a function called `get_Nth(my_list,N)` that takes a linked list and returns the data in the Nth node (starting at N=0). If N is greater than the length of the list, print a warning, and/or raise an exception.

For example, given the list

```
"a" --> "c" --> "e",
```

`get_Nth(my_list,0)` returns `a` and `get_Nth(my_list,3)` gives an error.

2. Write a function called `rotate(my_list, k)` that alters the linked list `my_list` by "rotating" it k steps to the right. For instance, given the list

```
1 --> 2 --> 3 --> 4 --> 5,
```

`rotate(my_list, 2)` produces

```
4 --> 5 --> 1 --> 2 --> 3...
```

...and `rotate(my_list, 9)` produces

`2 --> 3 --> 4 --> 5 --> 1`.

K can be any integer ≥ 0. Make sure to correctly reset the head of the list!

3. Write a function called `remove_consecutive_duplicates(my_list)` that alters the linked list `my_list` by, umm, removing consecutive duplicates. For instance, given

`1 --> 1 --> 1 --> 5 --> 5 --> 6 --> 2 --> 2 --> 2 --> 1 --> 1`,

this function should change the list to

`1 --> 5 --> 6 --> 2 --> 1`.

4. *(A classic problem. Please do not look up a solution online!)* Given a linked list, write a function that reverses it. In other words, given a list

`1 --> 2 --> 3 --> 4 --> 5`,

your code should alter this list and return

`1 <-- 2 <-- 3 <-- 4 <-- 5`

where the head of the linked list now points to the `5` node.

Your code shouldn't create a brand-new list; it should solve the problem "in place", which means that it should modify the given list. It should take O(n) time and O(1) space. (The latter means that you have only a bounded amount of memory to use for scratchwork, which must suffice no matter how long the input list is. You cannot, for instance, copy down all of the data from the list into scratchwork memory.)

This is trickier than it seems. Think through it carefully before starting to code. How many pointer variables will you need?

Two non-coding problems

These aren't due next Monday; they'll probably be due next Wednesday. I include them now so you can start thinking about them!

5. Design an algorithm that takes two linked lists and determines whether they intersect; i.e. determines whether they share a Node object in common. (Not whether they have data in common, but whether they have an actual Node object in common.)

Your algorithm should:

- Not alter the input lists.
- Use $O(1)$ space.
- Take $O(n)$ time.

6. We've seen in class that the stack data type has two operations, `pop()` and `push()`. When the stack is implemented by an array or a linked list, these both take O(1) time.

Design a data structure that behaves like a stack, with `pop()` and `push()`, but that also has a third command `min()`, which tells you the *minimum value contained in the stack*. (You may assume that all your values are real numbers.) For example, suppose you ran:

```
> push(5)
> push(7)
> push(10)
> push(1)
> push(33)
```

At this point, `min()` should return 1. But after running `pop()` twice, `min()` should return 5.

Oh, I forgot to tell you the thing that makes this problem challenging! The `min()` function must run *in O(1) time.*