DS 5010 Homework 3

Instructions

- Submit your solutions on Canvas by the deadline displayed online.
- Your submission must include a single Python module (file with extension ".py") that includes all of the code necessary to answer the problems. All of your code should run without error.
- Problem numbers must be clearly marked with code comments. Problems must appear in order, but later problems may use functions defined in earlier problems.
- Functions must be documented with a docstring describing at least (1) the function's purpose, (2) any and all parameters, (3) the return value. Code should be commented such that its function is clear.
- All solutions to the given problems must be your own work. If you use third-party code for ancillary tasks, you **must** cite them. (You may use code from class notes without citation.)
- You may use functions from built-in modules (e.g., `math`). You may **NOT** use external modules (e.g., `numpy`, `pandas`, etc.).

In this assignment, you will implement a double-ended queue (**deque**) using a doubly-linked list. You may use the code from "hw3-skeleton.py" on Piazza as a starting point.

The "hw3-skeleton.py" module defines a `Node` class and a `Deque` class. The `Node` class is complete and ready to use. Several methods for `Deque` are already implemented for you, but you will need to implement the remaining methods. (The pre-implemented methods may not work properly until you provide working implementations for the missing methods.)

Note that the `__iter__()` and `__reversed__()` methods are implemented assuming a double-linked list, but that the linked list is ***not** circular*. That is, the "head" and "tail" nodes are not directly linked with each other. That is, `head.getprev()` and `tail.getnext()` both return `None`.

Please review the course slides for the definition of a double-ended queue (deque) data structure. A working example of a singly-linked list can also be found in the course notes.

For a deque, don't forget to consider both the "head" and "tail" when updating the object! Also note that for a length-one deque, the head and the tail will be the same node.

Problem 1 Define the method `Deque.push(self, data)` satisfying the following criteria:

- Adds a new item (`data`) to the front ("head") of the deque.
- Appropriately updates the "head" and "tail" nodes as necessary.

Examples:

```
In : x = Deque()

In : x.push("1!")

In : x.push("2!")

In : x.push("3!")

In : print(x)
3! -> 2! -> 1!
```

Problem 2 Define the method `Deque.pop(self)` satisfying the following criteria:

- Removes and returns the item at the front ("head") of the deque.
- Appropriately updates the "head" and "tail" nodes as necessary.
- If the deque is empty, return `None`, with no changes to the object

Examples:

```
In : x = Deque()

In : x.push("1!")

In : x.push("2!")

In : x.push("3!")

In : pop(x)
Out: '3!'

In : print(x)
2! -> 1!
```

Problem 3 Define the method `Deque.push_back(self, data)` satisfying the following criteria:

- Adds a new item (`data`) to the back ("tail") of the deque.
- Appropriately updates the "head" and "tail" nodes as necessary.

Examples:

```
In : y = Deque()

In : y.push_back(1.11)

In : y.push_back(2.22)

In : y.push_back(3.33)

In : print(y)
1.11 -> 2.22 -> 3.33
```

Problem 4 Define the method `Deque.pop_back(self)` satisfying the following criteria:

- Removes and returns the item at the back ("tail") of the deque.
- Appropriately updates the "head" and "tail" nodes as necessary.
- If the deque is empty, return `None`, with no changes to the object

Examples:

```
In : y = Deque()

In : y.push_back(1.11)

In : y.push_back(2.22)

In : y.push_back(3.33)

In : pop_back(y)
Out: 3.33
```

```
In : print(y)
1.11 -> 2.22
```

Problem 5 Define the method `Deque.find(self, value)` satisfying the following criteria:

- Find and return the index (as an offset) of the given value in the deque
- If the value does not exist in the deque, return `None`

Examples:

```
In : y = Deque()

In : y.push_back(1.11)

In : y.push_back(2.22)

In : y.push_back(3.33)

In : print(y)
1.11 -> 2.22 -> 3.33

In : y.find(1.11)
Out: 0

In : y.find(2.22)
Out: 1

In : y.find("a") # returns None (not printed)
```