DESCRIPTION

Project 1 : Linked Lists

Due Thursday, January 24th, 2019 at 8:00 pm.
This is not a team project, do not copy someone else's work

Description

For this project, you will be implementing a Linked List class for singly linked lists. As learned in previous classes, singly linked list nodes have only a 'next' node pointer and a value. The Node class has already been implemented for you, in addition to a few functions of the Linked List class. Your assignment is to complete the remaining functions and solve a problem using your class.

Turning It In

Your completed project must be submitted as a folder named "**Project1**" and must include:

- A **LinkedList.py** python3 file.
- A **README.txt** file that includes:
  - Your name
  - Feedback on the project
  - How long it took to complete the project
  - Any external resources (attributions) that you used in its development.
- After placing these to 2 files in to a folder named as Project1, please zip the folder and upload this zipped folder using Mimir. Read their help documents, if needed.

Assignment Specifications

The **Node class** is included in the skeleton file with the following functions and **should not be edited.**

- __init__(self, value, next_node=None)
  - This function initializes a node with a given value. Optional argument is the next_node reference, which will default to None if not passed in.
- __eq__(self, other)
  - This provides comparison ('==') for nodes. If nodes have the same value, they are considered equal.
- __repr__(self), __str__(self)
  - A node is represented in string form as 'value'. Use str(node) to make into a string.

The **Linked List class** is partially completed in the skeleton file. **Function signatures or provided functions may not be edited in any way.**
The following functions are provided and may be used as needed.

- __init__(self)
  - Self.head - the first node in the linked list
  - Self.tail - the last node in the linked list
  - Self.size - the number of nodes in the list
  - A Linked List object has three variables:
- __eq__(self, other)
  - This function is used to compare two linked lists. This will primarily be used for grading purposes - DO NOT CHANGE
- __repr__(self), __str__(self)
  - This function will return the string form of a list of all of the node values, in order from head to tail. Please make sure you fully understand how to traverse Linked Lists, even though this function is provided to you in this project.

You must complete and implement the following functions. Take note of the specified return values and input parameters. **Do not change the function signatures.**

- **length**(self)
  - returns the number of nodes in the list
  - Complexity: Time O(1)
- **is_empty**(self)
  - returns true if the linked list is empty, false if it is not empty
  - Complexity: Time O(1)
- **front_value**(self)
  - returns the value of the front (head) node, None for empty lists
  - Complexity: Time O(1)
- **back_value**(self)
  - returns the value of the back (tail) node, None for empty lists
  - Complexity: Time O(1)
- **count**(self, val)
  - takes a value ‘val’ and counts how many times that value occurs in the linked list, returns the number of occurrences
  - Complexity: Time O(n)
- **find**(self, val)
  - takes a parameter ‘val’ and returns true if the value is found in the list, false if the value is not found
  - Complexity: Time O(n)
- **push_front**(self, val)
  - takes a parameter ‘val’ and inserts a node with value ‘val’ at the front (head) of the linked list
  - Complexity: Time O(1)
- **push_back**(self, val)
  - takes a parameter ‘val’ and inserts a node with value ‘val’ at the back (tail) of the linked list
  - Complexity: Time O(1)

- **pop_front**(self)
  - removes the front (head) node of the linked list and returns the value of the removed node
  - Complexity: Time O(1)
- **pop_back**(self)
  - removes the back (tail) node of the linked list and returns the value of the removed node
  - Complexity: Time O(n)

Application Problem

When you have completed the LinkedList class, you will be using it to complete the **partition(**linked_list, x**)** function. The function signature is provided in the skeleton file, outside of the other class declarations. Your task is to take the LinkedList object that is passed in as a parameter and partition it such that values less than the parameter x occur before values greater than or equal to x. The linked list should ***not*** be sorted, instead the values in each partition should remain in the same relative order as the unpartitioned linked list.

Example
input: 5 -> 4 -> 2 -> 1, x=3
output: 2 -> 1 -> 5 -> 4
2 and 1 are at the beginning of the list because they are both less than x, but 2 comes before 1 because that is the order they occur in the original list.

Additional Requirements:

- You may not access any of the object member variables directly, for the LinkedList or Node classes. You are only allowed to use the class functions. (This means that if you use .head, .tail, .size, .value, or .next_node anywhere in the partition function, you will loose all points relating to this function.)
- You are not allowed to use any other data structures besides LinkedLists to solve the problem. That means no python lists, sets, dictionaries, etc. You are allowed to use primitive types such as ints, strings, etc.
- Time Complexity O(n)

Assignment Notes

- You will be required to add and complete the docstrings for the functions that you complete. See the provided function for the expected format.
  - Pycharm automatically inserts a template when typing """ on the line after the function declaration.
- You are provided skeleton code for the LinkedList class and must complete the included methods. You may not add additional functions in the LinkedList class, and do not alter the function signatures in any way.
- Grading Rubric
  - 60 pts : Mimir visible test cases (see tests for point breakdown)

- 20 pts : Additional test cases
- 20 pts : Manual grading (Complexity)
  - (2) Docstrings
  - (6) length, is_empty, front_value, back_value, count, find
  - (4) push_front, push_back
  - (4) pop_front, pop_back
  - (4) partition