Objectives

* Double Linked List implementation

Due Date

This assignment is due on Tuesday, February 13, 2024, by 11:00 pm.

**Remarks:**

* It is expected you develop test cases for your code.
* When you are asked to hand in code, you cut-and-paste the definition(s) from your code and paste into this Word document immediately following the activity.
* Include with your code several test case examples and your results.
* This homework is to be individual work.

Value

This assignment is worth 20 points.

Activities

Doubly Linked Lists (DLLs) support O(1) removal from the end of the ADT developed in Lab 4.

1. Copy and save your LinkedList.py to DoublyLinkedList.py.
2. Rename your LinkedList Class to DoublyLinkedList and expand it to implement a Doubly Linked List that will remove a node from the end of the structure remove\_last()in O(1).
3. Use the same TestLinkedList.py file from the lab to test your new implementation.
4. Provide a listing of DoublyLinkedList.py below. If you needed to change your TestLinkedList.py file based on feedback from your Lab 4 submission also include a listing of TestLinkedList.py.

class Node:  
 *""""Class to define a node in a linked list"""* def \_\_init\_\_(self, item, \_next=None, \_prev=None):  
 *""""Constructor of the Node, builds the item (data) and the link to the next node \_next"""* self.item = item  
 self.\_next = \_next  
 self.\_prev = \_prev  
  
 def \_\_repr\_\_(self):  
 *"""Returns the Node data and what it is pointing to, and wthe previous node"""* return f"Node({self.item}, {self.\_next}, {self.\_prev} )"  
  
 def \_\_iter\_\_(self):  
 *""""Allows for the iteration over Nodes"""* yield self.item  
 if self.\_next is not None:  
 yield from self.\_next  
  
  
class DoublyLinkedList:  
 *"""Class defining the Linked List ADT and her methods"""* def \_\_init\_\_(self, items=None):  
 *"""Initialise the LinkedList with a head, tail and length."""* self.\_head = None  
 self.\_tail = None  
 self.\_length = 0  
  
 if items is not None:  
 for item in items:  
 self.addlast(item)  
  
 def addfirst(self, item):  
 *"""Adds a new node at the beginning of the linked list."""* new\_node = Node(item, self.\_head)  
 if self.\_head is not None:  
 self.\_head.\_prev = new\_node  
 self.\_head = new\_node  
 if self.\_tail is None:  
 self.\_tail = self.\_head  
 self.\_length += 1  
  
 def addlast(self, item):  
 *"""Adds a new node at the end of the linked list."""* new\_node = Node(item, None, self.\_tail)  
 if self.\_tail is not None:  
 self.\_tail.\_next = new\_node  
 self.\_tail = new\_node  
 if self.\_head is None:  
 self.\_head = new\_node  
 self.\_length += 1  
  
 def remove\_first(self):  
 *"""Removes the first node from the linked list."""* if self.\_head is None:  
 return None # or raise an exception  
 item = self.\_head.item  
 self.\_head = self.\_head.\_next  
 if self.\_head is not None:  
 self.\_head.\_prev = None  
 else:  
 self.\_tail = None  
 self.\_length -= 1  
 return item  
  
 def remove\_last(self):  
 *"""Removes the last node from the linked list in O(1) time."""* if self.\_tail is None:  
 return None  
 item = self.\_tail.item  
 self.\_tail = self.\_tail.\_prev  
 if self.\_tail is not None:  
 self.\_tail.\_next = None  
 else:  
 self.\_head = None  
 self.\_length -= 1  
 return item  
  
 def \_\_str\_\_(self):  
 *"""Formats the str magic method to return human-readable representation of linked list"""* string = 'Your linked list contains: '  
 currentnode = self.\_head  
 while currentnode is not None:  
 string += str(currentnode.item)  
 currentnode = currentnode.\_next  
 if currentnode is not None:  
 string += " ~and~ "  
 return string  
  
 def \_\_len\_\_(self):  
 *"""Returns length of the linked list"""* return self.\_length  
  
 def \_\_iter\_\_(self):  
 *"""Modifies the iter magic method to allow for iteration on linked list"""* if self.\_head is not None:  
 yield from self.\_head  
  
 def \_\_repr\_\_(self):  
 *"""Returns a more basic representation of the linked list"""* items = []  
 for item in self:  
 items.append(str(item))  
 return f"LinkedList({items})"

import unittest  
from DoublyLinkedList import DoublyLinkedList  
  
class TestDoublyLinkedList(unittest.TestCase):  
  
 def test\_addfirst(self):  
 *"""Test for adding a node to the beginning of a Linked List"""* ll = DoublyLinkedList()  
 ll.addfirst(1)  
 self.assertEqual(repr(ll),"LinkedList(['1'])")  
  
 def test\_addlast(self):  
 *"""Tests for adding a node to the end of a Linked List"""* ll = DoublyLinkedList()  
 ll.addlast(5)  
 self.assertEqual(repr(ll), "LinkedList(['5'])")  
  
 def test\_removefirst(self):  
 *"""Tests for removing the first node of a Linked List"""* ll = DoublyLinkedList()  
 ll.addfirst(1)  
 ll.addfirst(2)  
 removed\_item = ll.remove\_first()  
 self.assertEqual(removed\_item, 2)  
 self.assertEqual(repr(ll), "LinkedList(['1'])")  
 ll.remove\_first()  
 self.assertEqual(repr(ll), "LinkedList([])")  
 self.assertIsNone(ll.remove\_first())  
  
 def test\_removelast(self):  
 *"""Tests removing the last node of a Linked List"""* ll = DoublyLinkedList()  
 ll.addfirst(1)  
 ll.addfirst(2)  
 removed\_item = ll.remove\_last()  
 self.assertEqual(removed\_item, 1)  
 self.assertEqual(repr(ll), "LinkedList(['2'])")  
  
 # Test removing from an empty list  
 ll.remove\_last()  
 self.assertEqual(repr(ll), "LinkedList([])")  
 self.assertIsNone(ll.remove\_last())  
  
 def test\_length(self):  
 *"""Tests for the length of the Linked List"""* ll = DoublyLinkedList()  
 self.assertEqual(len(ll), 0)  
 ll.addfirst(1)  
 self.assertEqual(len(ll), 1)  
 ll.addlast(2)  
 self.assertEqual(len(ll), 2)  
 ll.remove\_first()  
 self.assertEqual(len(ll), 1)  
 ll.remove\_last()  
 self.assertEqual(len(ll), 0)  
  
 def test\_str\_and\_repr\_consistency(self):  
 *"""Test to show consistent behavior of repr and str for the Linked List"""* ll = DoublyLinkedList([1, 2, 3])  
 expected\_repr = "LinkedList(['1', '2', '3'])"  
 self.assertEqual(repr(ll), expected\_repr)  
 expected\_str = 'Your linked list contains: 1 ~and~ 2 ~and~ 3'  
 self.assertEqual(str(ll), expected\_str)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()