Faculty of Mathematical Economics

Data Structures and Algorithms

Instructor: **Nguyen Thanh Tuan**
DSEB Class of 2021 - 2024

Homework Assignment Week 7

Topic: Linked List
Date Created: March 3, 2023

Side Notes

The name `Linked List` might confuse some of you about the intuitiveness of this abstract data type. In fact, we do not declare any "list" that you have already known to construct a linked list.

By definition, a linked list, in its simplest form, is a "collection" of nodes that collectively form a linear sequence. Each node stores a reference to an object that is an element of the sequence, as well as a reference to the next or last node of the list. Calling it a list, but we literally hold in hand only the head and size of it. Then the nodes will automatically reference to their neighbors. Problems in this assignment will require you to build methods to traverse or to get access to elements in the middle of the list. As the definition above, you *must* implement these methods in a natural, naive way, i.e not using the `list` data type or Python built-in functions.

Why using linked lists instead of lists? People didn't invent linked lists just to make life more complicated. The referencing mechanism of linked lists has paved the way for building more complex abstract data types such as tree, graph... and helped to optimize time complexity and space complexity when working with large data infrastructures.

Problem 1: Singly Linked List

a. Implement a `Node` class and a `SinglyLinkedList` class storing multiple nodes. Each node maintains a reference to its element and one reference to the next node in the list. The linked list class should follow the below structure:

```python
class SinglyLinkedList:
    """Singly Linked List implementation."""

    def __init__(self):
        """Initialize a singly linked list object.
        Attributes: head, size"""
        pass

    def __len__(self):
        """Return the current number of nodes in the list"""
        pass

    def is_empty(self):
        """Return True if the list is empty"""
        pass

    def __getitem__(self, k):
        """Return content in the k-th node of the list"""
        pass

    def insert(self, k):
        """Insert a new node to position k of the list.
        If k = 0 or list is empty, insert a new head"""
        pass

    def __delitem__(self, k):
        """Delete node at position k of the list.
        Return the deleted node.
        """
        pass

    def delete_by_value(self, val):
        """Delete all nodes that store the input value.
        Return all deleted nodes."""
        pass

    def search(self, val):
        """Return  the positions and contents of all nodes that store
        the input value. Print a message if the value is not found"""
        pass

    def update(self, k, val):
        """Update content in the k-th node to new input value.
        Print out the old and new updated values of the node"""
        pass

    def __repr__(self):
        """Return string representation of the list"""
        pass
```

Note:

- In `search` and `delete_by_value` methods, you are allowed to store outputs in built-in Python list.
- Raise Empty exception if the list is empty for non-inserting methods.

b. Check your implementation by performing these tasks:

- Create a `Node` object with attributes: name, score, class and next. Then create a `SinglyLinkedList` object to insert these students with their information into the list:

```
('Hai', 13.5, 'BFI'), ('Nam', 12, 'Actuary'), ('Vanh', 15, 'DSEB'),
('Ly', 10, 'TKT'), ('Chiu', 13, 'DSEB'), ('Bach', 16, 'DSEB'),
('Chau', 11, 'BFI'),('Huy', 11, 'Actuary')
```

- Insert Hoang who is in TKT class and has score 16 to position 3.
- Delete all students whose class is BFI.
- Search for a student whose name is Vanh.
- No surprised, Bach gave right answer for a hard question so Mr. Tuan decide to add 1 point to his current score. Update his score.

Problem 2: Doubly Linked List

a. Implement a `DoublyLinkedList` class storing multiple nodes. Each node maintains a reference to its element and reference to its **last and next** nodes in the list. The class should follow the below structure:

```
class DoublyLinkedList:
    """Doubly Linked List implementation."""

    def __init__(self):
        """Initialize a doubly linked list object.
        Attributes: head, size"""
        pass

    def __len__(self):
        """Return the current number of nodes in the list"""
        pass

    def is_empty(self):
        """Return True if the list is empty"""
        pass

    def __getitem__(self, k):
        """Return content in the k-th node of the list"""
        pass

    def insert(self, k):
        """Insert a new node to position k of the list.
        If k = 0 or list is empty, insert a new head"""
        pass

    def __delitem__(self, k):
        """Delete node at position k of the list.
        Return the deleted node.
        """
        pass

    def reverse(self):
        """Reverse the list."""
        pass
```

```python

    def sort_by_value(self):
        """Sort the list by values of nodes in descending order."""
        pass

    def __repr__(self):
        """Return string representation of the list"""
        pass
```

Note:

- Raise Empty exception if the list is empty for non-inserting methods.
- In `reverse` and `sort_by_value` methods, you are NOT allowed to use a second sequence (list, tuple, linked list,...) to solve the problem, meaning that space complexity of these methods must be O(1).

b. Check your implementation by performing these tasks:

- Create a `DoublyLinkedList` object and insert these stock codes into the list:

```python
('VNM', 100.6), ('HPG', 46.05), ('GAS', 94), ('MSN', 86.8),
('FPT', 75.7), ('VIC', 104.7), ('VCB', 94.3),('MWG', 128.2),
('PNJ', 83.2), ('DHG', 98.6)
```

- Delete all nodes whose stock prices are smaller than 80.
- Insert VJC with price 101.2 into position 3 of the list.
- Print out the current list and its reversed order.
- Print out the list sorted by stock price in from highest to lowest.

Problem 3: Minimum and Maximum Element of A Linked Stack

Re-implement a `LinkedStack` class which is a stack using singly linked list for storage in textbook *Data Structures and Algorithms in Python (2013, Wiley) page 262* with common methods `push`, `pop`, `top`, `is_empty`. Then, implement two methods `get_min`, `get_max` to return the elements with minimum and maximum value in the stack. Each method must have O(1) time complexity and O(1) space complexity.

Optional Problem: Palindrome Linked List

Implement a function to check if a singly linked list is palindrome. Your solution must maintain space complexity O(1).

Note: Optional problems are like challenges for who is interested in the topic and desires to advance their coding skills and algorithm. These problems are not counted in your homework marks but there is a probability that it will be in your final exam (This is fair because solutions to these problems will be given in tutor class).

Guidelines for submission

- Your submission must be under the `.ipynb` format.
- Your submission will be graded and it is likely that homework grade will contribute as a component in your GPA.
- If your submission is later than the due date without special consideration approval, you will receive a penalty on your mark.