CMPUT 175 - Lab 7: Linked Lists

Goal: Learn about singly and doubly-linked lists, and learn to test your methods using assert statements rather than comparing to sample output.

Ensure that you write proper docstrings and follow the Software Quality Requirements.

Exercise 1:

In this task, you will complete the *insert(pos, item)* method for the singly-linked list. Recall that insert adds a new node (containing the item as its data) at the given position in the list. For example, if pos is 0, then the new node goes at the beginning (head) of the list. *pos* must be an integer, and for this exercise, cannot be negative.

Download and save **SLinkedList.py** from eClass. There are 2 classes in the file: the *SLinkedListNode* class and the *SLinkedList* class. The *SLinkedListNode* class is already completed for you based on the implementation in the lecture. The *SLinkedList* class is partially completed – you must complete the implementation of the insert method and use the code in the main function to test it.

Hints:

a. Think of the different cases you may encounter. What if you try to insert it into an empty list? What if you try to insert at the head of a list containing one element or many elements? What if you try to insert in the middle of a list containing many elements - is that different from inserting at the end?
b. Draw out every unique case to see how to update the links without unintentionally losing elements to the garbage collector.
c. Always leave your list in a consistent state.

Sample Output

```
Original List: 6 elements
A->4->2->77->6->Z->
After inserting the word “start” at position 0: 7 elements
start->A->4->2->77->6->Z->
After inserting the word “end” at position 7: 8 elements
start->A->4->2->77->6->Z->end->
After inserting “middle” at position 4: 9 elements
start->A->4->2->middle->77->6->Z->end->
```

Exercise 2:

In this task, you will implement the following doubly-linked list ADT:

- ***search(item)*** – returns True if the item is an element in the list; False otherwise. The item can be any object. *Already completed for you.*
- ***index(item)*** – returns the index of the item in the list (assuming that the head node is at index 0) or -1 if the item is not in the list. *Already completed for you.*
- ***add(item)*** – adds a new node (containing the item as its data) to the head of the list.
- ***remove(item)*** – removes the first element in the list that is equal to the item. If the item is not in the list, the list is not changed, and an exception should NOT be raised. *This is a bit different from the implementation in the lecture.*
- ***append(item)*** – adds a new node (containing the item as its data) to the tail of the list.
- ***insert(pos, item)*** – adds a new node (containing the item as its data) at the given position in the list. For example, if *pos* is 0, then the new node goes at the beginning (head) of the list. *pos* must be an integer, and for this exercise, cannot be negative. *You can assume that if the pos is an integer and positive, it will always be less than or equal to self.__size.*
- ***pop1()*** – removes and returns the last item in the list.
- ***pop(pos)*** – removes and returns the item in the given position. An exception should be raised if the position is outside of the list. *pos* must be an integer, and for this exercise, cannot be negative. *Note: if pos is None, you can assume that the item to be removed is the last item.*
- ***searchLarger(item)*** – returns the position of the first element that is larger than the item, or -1 if there is no larger item.
- ***getSize()*** – returns the number of elements in the list.
- ***getItem(pos)*** – returns the item at the given position. An exception should be raised if the position is outside of the list. *pos* must be an integer, and it can be positive OR Negative.
- ***__str__*** – returns a string representation of the elements in the doubly-linked list, with one space in between each element.

Download and save **DLinkedList.py** from eClass. This file contains the completed *DLinkedListNode* class, as implemented in the lectures. It also contains the unfinished *DLinkedList* class. Your task will be to complete and test the ADT methods described above. Things to keep in mind:

a. All attributes in *DLinkedList* should be PRIVATE.
b. Tests for each method have been provided for you in DLinkedList.py. Run the tests when you complete EACH method.
c. Draw out the linked list structure to help visualize which references need to be updated and in which order (to prevent the garbage collector from removing unintended objects). Be sure to consider all relevant cases. For example, when removing an item, what happens when that item is in the head node, the tail node, or a middle node? When adding an item, is adding to an empty list the same as adding to a list with a single element already in it? Is it the same as adding to a list with many elements already in it?
d. Make sure that the linked list always maintains a consistent state.

Deliverables

You will produce and submit **two** Python (.py) files for this lab on eClass by the submission deadline:

- **SLinkedList.py**: Python solution to Exercise 1 of this lab
- **DLinkedList.py**: Python solution to Exercise 2 of this lab