LE-4
Doubly Linked List and Tagged Queues

Introduction:

This exercise is a hands-on component for the concepts acquired in the lecture on Double Linked List and Queue ADT. This introduces you to the Abstraction concept and how a data structure and its operations can be created according to the requirements.

Doubly Linked List

You have already implemented Linked List in LE-2, you will now be implementing a Doubly Linked List. You have been provided with a Node class similar to LE-2 but with an extra attribute called *prev* that points to the Node before a given node in the linked list. The DoublyLinkedList class also contains an extra attribute called tail which keeps a track of the tail node of the list. Modify the *push()* and *pop()* methods from LE-2 and add the appropriate definition for *insert,* and *delete* methods to respectively insert and delete elements from the linked list (inclusive of the head and tail).

The *insert* method takes the following two inputs: **insert(data, position)**

- **data** - the value to be inserted.
- **position** - the position at which to insert the element, should take values from 1 till some large value x, where 1 will represent insertion at the head of the list and x will represent an arbitrary position in the list if x > 1. If the value of x is greater than the size of your linked list then it should default to the value of the last position for insertion (analogous to append).

Example:

```
LinkedList A: 1 <-> 2 <-> 3 <-> 4 <-> 5

Input:
A.insert(6, 2)

Output:
A: 1 <-> 6 <-> 2 <-> 3 <-> 4 <-> 5
```

The *delete* method takes one input: **delete(position)**

- **position** - the position from which to remove the value (range similar to the description noted above for *insert*).

Example:

```
LinkedList A: 5 <-> 4 <-> 3 <-> 2 <-> 1

Input:
A.delete(4)

Output:
A: 5 <-> 4 <-> 3 <-> 1
```

For all these methods if the value of position is given as less than 1 then your method should return the exact string 'Invalid Position'.

Queue With Tags

Modifying your Queue implementation from LE-3, create a Queue ADT (adds elements at the back and removes elements from the front by FIFO), but with a special property. *Every element that is added should also have a tag attribute attached to it which will act as a 'grouping' criterion while enqueueing elements.* The idea is that the queue structure is still maintained but with a sub-queueing capability which queues elements also by the tags associated with them. The enqueued element should be added as a tuple which contains both the data and the tag in the form of (data, tag) within the larger global queue.

The *enqueue* method should take two values: **enqueue(data, tag)**

- **data** - the value to be enqueued: int.
- **tag** - the grouping criteria: str.

Example: ***queue structure: left - front, right - rear***

```
A: (10, 'B') <- (2, 'A') <- (13, 'C')

Input:
Queue A.enqueue(6, 'A')

Output:
A: (10, 'B') <- (2, 'A') <- (6, 'A') <- (13, 'C')
```

```
Input:
A.enqueue(3, 'C')

Output:
A: (10, 'B') <- (2, 'A') <- (6, 'A') <- (13, 'C') <- (3, 'C')
```

Rubric:

Your code will be tested with provided test cases.

Location of the code:

The code would be provided at
https://colab.research.google.com/drive/1ltawhk0WGE4dTvtNYhUixZ7e8XTqGsXZ?usp=sharing

What to do when done:

Once you have completed the exercise, you should upload it to the codePost. Please ensure the following while submitting:

- Once satisfied with your code, you should download the file as a python script (.py file), by going to **File** > **Download** > **Download .py**
- The name of the file should be LE4.py
- Upload the python script file to codePost under the LE-4 assignment.
- You can run the test cases on your script up to a limit of 50 times.
- Once satisfied with the test runs, complete your submission.