Solution Review: Reversing First k Elements of Queue

In this lesson, we will do a solution review for the 'Reversing First k Elements of Queue' challenge.

We'll cover the following

Solution: Using a Queue#

main.py

DoublyLinkedList.py

Stack.py

Queue.py

```
from Queue import MyQueue
from Stack import MyStack

# 1.Push first k elements in queue in a stack.
# 2.Pop Stack elements and enqueue them at the end of queue
# 3.Dequeue queue elements till "k" and append them at the end of que


def reverseK(queue, k):
    # Handling invalid input
    if queue.is_empty() is True or k > queue.size() or k < 0:
```

```
12          return None
13
14      stack = MyStack()
15      for i in range(k):
16          stack.push(queue.dequeue())
17      while stack.is_empty() is False:
18          queue.enqueue(stack.pop())
19      size = queue.size()
20      for i in range(size - k):
21          queue.enqueue(queue.dequeue())
22
23      return queue
24
25  if __name__ == "__main__" :
26      # testing our logic
27      queue = MyQueue()
28
```

1. Check for invalid input, i.e., if the queue is empty, if `k` is greater than the queue, and if `k` is negative on **line 10**. If the input is valid, start by creating a Stack. The available stack functions are:
    - `MyStack()` : This constructor is called upon the creation of the *MyStack* object.
    - `push(int)` : Push elements to the stack.
    - `pop()` : Remove the top element from the stack.
    - `is_empty()` : Returns true if the stack is empty and false otherwise.
    - `top()` : Returns the top element (that has been added at the beginning) without removing it from the stack.
2. Our function `reverseK(queue, k)` takes `queue` as an input parameter. `k` represents the number of elements we want to reverse. The available queue functions are:

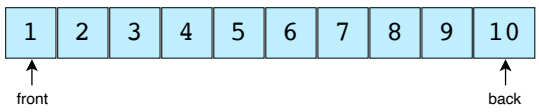
- `MyQueue(size)` : This constructor is called upon the ... *MyQueue* object.
- `enqueue(int)` : Enqueue an element in the back
- `dequeue()` : Dequeue an element from the front
- `is_empty()` : Returns true in case of an empty queue and false otherwise
- `size()` : Returns the size of queue

3. Now, moving on to the actual logic, **dequeue** the first `k` elements from the **front** of the queue and **push** them in the stack we created earlier using `stack.push(queue.dequeue())` in line 16.

4. Once all the `k` values have been pushed to the stack, start **popping** them and **enqueueing** them to the **back** of the queue sequentially. We will do this using `queue.enqueue(stack.pop())` in line 18. At the end of this step, we will be left with an empty stack and the `k` reversed elements will be appended to the **back** of the queue.

5. Now we need to move these reversed elements to the front of the queue. To do this, we used `queue.enqueue(queue.dequeue())` in line 21. Each element is first **dequeued** from the *back*.

The solution is illustrated by the following animation:



1 of 7

Time Complexity#

The time complexity of this function is $O(n)$ where `n` is the size of the queue as the entire queue is iterated over. `k` elements are iterated over in the first two loops and $size - k$ are iterated over in the last loop which sums up to n iterations.