

원형 덱 디자인

- `MyCircularDeque(k)`: Constructor, set the size of the deque to be k.
- `insertFront()`: Adds an item at the front of Deque. Return true if the operation is successful.
- `insertLast()`: Adds an item at the rear of Deque. Return true if the operation is successful.
- `deleteFront()`: Deletes an item from the front of Deque. Return true if the operation is successful.
- `deleteLast()`: Deletes an item from the rear of Deque. Return true if the operation is successful.
- `getFront()`: Gets the front item from the Deque. If the deque is empty, return -1.
- `getRear()`: Gets the last item from Deque. If the deque is empty, return -1.
- `isEmpty()`: Checks whether Deque is empty or not.
- `isFull()`: Checks whether Deque is full or not.

1.

class MyCircularDeque:

```
def __init__(self, k: int):
    """
    Initialize your data structure here. Set the size of the deque to be k.
    """
    self.head, self.tail = ListNode(None), ListNode(None)
    self.k, self.len = k, 0
    self.head.right, self.tail.left = self.tail, self.head

def _add(self, node: ListNode, new: ListNode) -> None:
    n = node.right
    node.right = new
    new.left, new.right = node, n
    n.left = new

def _del(self, node: ListNode) -> None:
    n = node.right.right
    node.right = n
    n.left = node

def insertFront(self, value: int) -> bool:
    """
    Adds an item at the front of Deque. Return true if the operation is successful.
    """
    if self.len == self.k:
        return False
    self.len += 1
    self._add(self.head, ListNode(value))
    return True

def insertLast(self, value: int) -> bool:
```

```
"""
```

```
Adds an item at the rear of Deque. Return true if the operation is successful.
```

```
"""
```

```
if self.len == self.k:  
    return False  
self.len += 1  
self._add(self.tail.left, ListNode(value))  
return True
```

```
def deleteFront(self) -> bool:
```

```
"""
```

```
Deletes an item from the front of Deque. Return true if the operation is successful.
```

```
"""
```

```
if self.len == 0:  
    return False  
self.len -= 1  
self._del(self.head)  
return True
```

```
def deleteLast(self) -> bool:
```

```
"""
```

```
Deletes an item from the rear of Deque. Return true if the operation is successful.
```

```
"""
```

```
if self.len == 0:  
    return False  
self.len -= 1  
self._del(self.tail.left.left)  
return True
```

```
def getFront(self) -> int:
```

```
"""
```

```
Get the front item from the deque.
```

```
"""
```

```
return self.head.right.val if self.len else -1
```

```
def getRear(self) -> int:
```

```
"""
```

```
Get the last item from the deque.
```

```
"""
```

```
return self.tail.left.val if self.len else -1
```

```
def isEmpty(self) -> bool:
```

```
"""
```

```
Checks whether the circular deque is empty or not.
```

```
"""
```

```
return self.len == 0
```

```
def isFull(self) -> bool:
```

```
    """
```

```
    Checks whether the circular deque is full or not.
```

```
    """
```

```
    return self.len == self.k
```

```
# Your MyCircularDeque object will be instantiated and called as such:
```

```
# obj = MyCircularDeque(k)
```

```
# param_1 = obj.insertFront(value)
```

```
# param_2 = obj.insertLast(value)
```

```
# param_3 = obj.deleteFront()
```

```
# param_4 = obj.deleteLast()
```

```
# param_5 = obj.getFront()
```

```
# param_6 = obj.getRear()
```

```
# param_7 = obj.isEmpty()
```

```
# param_8 = obj.isFull()
```

```
class MyCircularDeque:
```

```
    def __init__(self, k: int):
```

```
        """
```

```
        Initialize your data structure here. Set the size of the deque to be k.
```

```
        """
```

```
        self.q = collections.deque()
```

```
        self.max = k
```

```
    def insertFront(self, value: int) -> bool:
```

```
        """
```

```
        Adds an item at the front of Deque. Return true if the operation is successful.
```

```
        """
```

```
        if len(self.q) < self.max:
```

```
            self.q.appendleft(value)
```

```
            return True
```

```
        return False
```

```
    def insertLast(self, value: int) -> bool:
```

```
        """
```

```
        Adds an item at the rear of Deque. Return true if the operation is successful.
```

```
        """
```

```
        if len(self.q) < self.max:
```

```
            self.q.append(value)
```

```
            return True
```

```
return False
```

```
def deleteFront(self) -> bool:
```

```
    """
```

```
    Deletes an item from the front of Deque. Return true if the operation is successful.
```

```
    """
```

```
    if len(self.q):
```

```
        self.q.popleft()
```

```
        return True
```

```
    return False
```

```
def deleteLast(self) -> bool:
```

```
    """
```

```
    Deletes an item from the rear of Deque. Return true if the operation is successful.
```

```
    """
```

```
    if len(self.q):
```

```
        self.q.pop()
```

```
        return True
```

```
    return False
```

```
def getFront(self) -> int:
```

```
    """
```

```
    Get the front item from the deque.
```

```
    """
```

```
    if len(self.q):
```

```
        return self.q[0]
```

```
    return -1
```

```
def getRear(self) -> int:
```

```
    """
```

```
    Get the last item from the deque.
```

```
    """
```

```
    print(self.q)
```

```
    if len(self.q):
```

```
        return self.q[-1]
```

```
    return -1
```

```
def isEmpty(self) -> bool:
```

```
    """
```

```
    Checks whether the circular deque is empty or not.
```

```
    """
```

```
    return len(self.q) == 0
```

```
def isFull(self) -> bool:
    """
    Checks whether the circular deque is full or not.
    """
    return len(self.q) >= self.max
```

```
# Your MyCircularDeque object will be instantiated and called as such:
# obj = MyCircularDeque(k)
# param_1 = obj.insertFront(value)
# param_2 = obj.insertLast(value)
# param_3 = obj.deleteFront()
# param_4 = obj.deleteLast()
# param_5 = obj.getFront()
# param_6 = obj.getRear()
# param_7 = obj.isEmpty()
# param_8 = obj.isFull()
```