List Collections (Linked Lists, Stacks and Queues) Problems

225. Implement Stack using Queues (https://leetcode.com/problems/implement-stack-using-queues/)

Easy

Implement the following operations of a stack using queues.

```
push(x) -- Push element x onto stack.
pop() -- Removes the element on top of the stack.
top() -- Get the top element.
empty() -- Return whether the stack is empty.
```

Example:

```
MyStack stack = new MyStack();

stack.push(1);
stack.push(2);
stack.top();   // returns 2
stack.pop();   // returns 2
stack.empty(); // returns false
```

Notes:

- You must use only standard operations of a queue -- which means only push to back, peek/pop from front, size, and is empty operations are valid.
- Depending on your language, queue may not be supported natively. You may simulate a queue by using a list or deque (double-ended queue), as long as you use only standard operations of a queue.
- You may assume that all operations are valid (for example, no pop or top operations will be called on an empty stack).

Performance

- Runtime: 36 ms, faster than 70.60% of Python3 online submissions for Implement Stack using Queues.
- Memory Usage: 13.1 MB, less than 5.45% of Python3 online submissions for Implement Stack using Queues.

Complexity Analysis

- O(1) in time.
- O(1) in space

In [1]:
```
class MyStack:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.stack = []


    def push(self, x: int) -> None:
        """
        Push element x onto stack.
        """
        self.stack.append(x)

    def pop(self) -> int:
        """
        Removes the element on top of the stack and returns that element.
        """
        if len(self.stack) > 0:
            return self.stack.pop()

    def top(self) -> int:
        """
        Get the top element.
        """
        if len(self.stack) > 0:
            return self.stack[-1]

    def empty(self) -> bool:
        """
        Returns whether the stack is empty.
        """
        return len(self.stack) == 0


# Your MyStack object will be instantiated and called as such:
obj = MyStack()
obj.push(1)
param_2 = obj.pop()
param_3 = obj.top()
param_4 = obj.empty()
print('Should print 1, None, True:', param_2, param_3, param_4)
```

```
Should print 1, None, True: 1 None True
```

232. Implement Queue using Stacks (https://leetcode.com/problems/implement-queue-using-stacks/?tab=Description)

Easy

Implement the following operations of a queue using stacks.

```
push(x) -- Push element x to the back of queue.
pop() -- Removes the element from in front of queue.
peek() -- Get the front element.
empty() -- Return whether the queue is empty.
```

Example:

```
MyQueue queue = new MyQueue();

queue.push(1);
queue.push(2);
queue.peek();  // returns 1
queue.pop();   // returns 1
queue.empty(); // returns false
```

Notes:

- You must use only standard operations of a stack -- which means only push to top, peek/pop from top, size, and is empty operations are valid.
- Depending on your language, stack may not be supported natively. You may simulate a stack by using a list or deque (double-ended queue), as long as you use only standard operations of a stack.
- You may assume that all operations are valid (for example, no pop or peek operations will be called on an empty queue).

Performance

- Runtime: 36 ms, faster than 69.07% of Python3 online submissions for Implement Queue using Stacks.
- Memory Usage: 13.2 MB, less than 5.32% of Python3 online submissions for Implement Queue using Stacks.

Complexity Analysis

- O(1)
- O(1)

In [2]:
```
from collections import deque

class MyQueue:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.queue = deque([])

    def push(self, x: int) -> None:
        """
        Push element x to the back of queue.
        """
        self.queue.append(x)

    def pop(self) -> int:
        """
        Removes the element from in front of queue and returns that element.
        """
        if len(self.queue) > 0:
            return self.queue.popleft()

    def peek(self) -> int:
        """
        Get the front element.
        """
        if len(self.queue) > 0:
            return self.queue[0]

    def empty(self) -> bool:
        """
        Returns whether the queue is empty.
        """
        return len(self.queue) == 0


# Your MyQueue object will be instantiated and called as such:
obj = MyQueue()
obj.push(1)
param_2 = obj.pop()
param_3 = obj.peek()
param_4 = obj.empty()
print('Should print 1, None, True:', param_2, param_3, param_4)
```

```
Should print 1, None, True: 1 None True
```

206. Reverse Linked List (https://leetcode.com/articles/reverse-linked-list/)

Reverse a singly linked list.

Example:

```
Input: 1=>2=>3=>4=>5=>NULL
Output: 5=>4=>3=>2=>1=>NULL
```

Follow up:

A linked list can be reversed either iteratively or recursively. Could you implement both?

Performance

- Runtime: 56 ms, faster than 83.14% of Python3 online submissions for Path Sum II.
- Memory Usage: 14.7 MB, less than 55.81% of Python3 online submissions for Path Sum II.

Complexity Analysis

Iterative

```
O(n) in time
O(1) in space
```

Recursive

```
O(n) in time
O(n) in space
```

In [3]:
```
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

class LinkedList:
    def __init__(self, value, head = None):
        if head: self.head = head
        else:    self.head = ListNode(value)

    def append(self, new_tail):
        n0 = self.head
        n1 = self.head.next
        while n1:
            n0 = n0.next
            n1 = n1.next
        n0.next = ListNode(new_tail)

    def printList(self):
        node = self.head
        string = ''
        while node:
            string = string + str(node.val) + ' => '
            node = node.next
        print(string)

    def reverseListIterative(self):
        n1 = None
        n2 = self.head
        while (n2 != None):
            n3 = n2.next
            n2.next = n1
            n1 = n2
            n2 = n3
        self.head = n1

    def reverseListRecursive(self, node):
        if node == None or node.next == None:
            self.head = node
            return
        self.reverseListRecursive(node.next)
        node.next.next = node
        node.next = None

my_list = LinkedList(1)
my_list.append(2)
my_list.append(3)
my_list.append(4)
my_list.append(5)
my_list.printList()

my_list.reverseListIterative()
my_list.printList()

my_list.reverseListRecursive(my_list.head)
my_list.printList()
```

```
1 => 2 => 3 => 4 => 5 =>
5 => 4 => 3 => 2 => 1 =>
1 => 2 => 3 => 4 => 5 =>
```

83. Remove Duplicates from Sorted List (https://leetcode.com/problems/remove-duplicates-from-sorted-

list/)

Easy

Given a sorted linked list, delete all duplicates such that each element appears only once.

Example 1:

```
Input: 1=>1=>2
Output: 1=>2
```

Example 2:

```
Input: 1=>1=>2=>3=>3
Output: 1=>2=>3
```

Performance

- Runtime: 52 ms, faster than 61.73% of Python3 online submissions for Remove Duplicates from Sorted List.
- Memory Usage: 13 MB, less than 5.26% of Python3 online submissions for Remove Duplicates from Sorted List.

Complexity Analysis

```
O(n) in time
O(1) in space
```

In [4]:

```
# Definition for singly-linked list.
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

class LinkedList:
    def __init__(self, value, head = None):
        if head: self.head = head
        else:    self.head = ListNode(value)

    def append(self, new_tail):
        n0 = self.head
        n1 = self.head.next
        while n1:
            n0 = n0.next
            n1 = n1.next
        n0.next = ListNode(new_tail)

    def printList(self):
        node = self.head
        string = ''
        while node:
            string = string + str(node.val) + ' => '
            node = node.next
        print(string)

class Solution:
    def deleteDuplicates(self, head: ListNode) -> ListNode:
        if head:
            n1, n2 = head, head.next
            while n2:
                if n2.val == n1.val:
                    n2 = n2.next
                    n1.next = n2
                else:
                    n1, n2 = n2, n2.next
        return head

my_sol = Solution()

#[1=>1=>2]
my_list = LinkedList(1)
my_list.append(1)
my_list.append(2)
my_list.printList()
my_sol.deleteDuplicates(my_list.head)
my_list.printList()

#[1=>1=>2=>3=>3]
my_list = LinkedList(1)
my_list.append(1)
my_list.append(2)
my_list.append(3)
my_list.append(3)
my_list.printList()
my_sol.deleteDuplicates(my_list.head)
my_list.printList()
```

```
1 => 1 => 2 =>
1 => 2 =>
1 => 1 => 2 => 3 => 3 =>
1 => 2 => 3 =>
```

328. Odd Even Linked List (https://leetcode.com/problems/odd-even-linked-list/)

Medium

Given a singly linked list, group all odd nodes together followed by the even nodes. **Please note here we are talking about the node number and not the value in the nodes.**

You should try to do it in place. The program should run in O(1) space complexity and O(nodes) time complexity.

Example 1:

```
Input:  1=>2=>3=>4=>5=>NULL
Output: 1=>3=>5=>2=>4=>NULL
```

Example 2:

```
Input:  2=>1=>3=>5=>6=>4=>7=>NULL
Output: 2=>3=>6=>7=>1=>5=>4=>NULL
```

Note:

The relative order inside both the even and odd groups should remain as it was in the input. The first node is considered odd, the second node even and so on ...

Performance

- Runtime: 52 ms, faster than 55.48% of Python3 online submissions for Odd Even Linked List.
- Memory Usage: 15.1 MB, less than 6.02% of Python3 online submissions for Odd Even Linked List.

Complexity Analysis

```
O(n) in time.
O(1) in space.
```

In [5]:

```
# Definition for singly-linked list.
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

class LinkedList:
    def __init__(self, value, head = None):
        if head: self.head = head
        else:    self.head = ListNode(value)

    def append(self, new_tail):
        n0 = self.head
        n1 = self.head.next
        while n1:
            n0 = n0.next
            n1 = n1.next
        n0.next = ListNode(new_tail)

    def printList(self):
        node = self.head
        string = ''
        while node:
            string = string + str(node.val) + ' => '
            node = node.next
        print(string)

class Solution:
    def oddEvenList(self, head: ListNode) -> ListNode:
        if head and head.next:
            odd_tail = head
            even_head = head.next
            even_tail = head.next
            n1 = head.next.next
            while n1:
                n2 = n1.next

                odd_tail.next = n1
                n1.next = even_head
                odd_tail = n1

                even_tail.next = n2
                even_tail = n2

                if n2: n1 = n2.next
                else:  break
        return head

my_sol = Solution()

#[1=>2=>3=>4=>5=>]
my_list = LinkedList(1)
my_list.append(2)
my_list.append(3)
my_list.append(4)
my_list.append(5)
print('Input:')
my_list.printList()
my_sol.oddEvenList(my_list.head)
print('Should print:')
print('1 => 3 => 5 => 2 => 4 => ')
my_list.printList()
print()
```

```
#2=>1=>3=>5=>6=>4=>7=>
my_list = LinkedList(2)
my_list.append(1)
my_list.append(3)
my_list.append(5)
my_list.append(6)
my_list.append(4)
my_list.append(7)
print('Input:')
my_list.printList()
my_sol.oddEvenList(my_list.head)
print('Should print:')
print('2 => 3 => 6 => 7 => 1 => 5 => 4 => ')
my_list.printList()
```

```
Input:
1 => 2 => 3 => 4 => 5 =>
Should print:
1 => 3 => 5 => 2 => 4 =>
1 => 3 => 5 => 2 => 4 =>

Input:
2 => 1 => 3 => 5 => 6 => 4 => 7 =>
Should print:
2 => 3 => 6 => 7 => 1 => 5 => 4 =>
2 => 3 => 6 => 7 => 1 => 5 => 4 =>
```

82. Remove Duplicates from Sorted List II (https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/)

Medium

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.

Example 1:

```
Input: 1=>2=>3=>3=>4=>4=>5
Output: 1=>2=>5
```

Example 2:

```
Input: 1=>1=>1=>2=>3
Output: 2=>3
```

Performance

- Runtime: 48 ms, faster than 81.13% of Python3 online submissions for Remove Duplicates from Sorted List II.
- Memory Usage: 13.2 MB, less than 5.75% of Python3 online submissions for Remove Duplicates from Sorted List II.

Complexity Analysis

- O(n) time
- O(1) space

In [6]:

```
# Definition for singly-linked list.
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

def print_list(node):
    string = ''
    while node:
        string = string + str(node.val) + ' => '
        node = node.next
    print(string)

class Solution:
    def deleteDuplicates(self, head: ListNode) -> ListNode:
        head_set = False
        new_head = None
        tail = None
        if head and head.next:
            # Compare the first 2 elements of the linked list
            n1, n2 = head, head.next
            if n1.val != n2.val:
                head_set = True
                new_head = head
                tail = new_head
            # Compare 3 consecutive elements of the linked list
            while n2.next:
                n3 = n2.next
                if n1.val != n2.val and n2.val != n3.val:
                    if not(head_set):
                        head_set = True
                        new_head = n2
                        tail = new_head
                    else:
                        tail.next = n2
                        tail = tail.next
                n1, n2 = n2, n2.next
            # Compare the last 2 elements of the linked list
            if n1.val != n2.val:
                if not(head_set):
                    new_head = n2
                    tail = new_head
                else:
                    tail.next = n2
                    tail = tail.next
            if tail:
                tail.next = None
            return new_head
        else:
            return head

my_sol = Solution()

#[1=>2=>3=>3]
head = ListNode(1)
head.next = ListNode(2)
head.next.next = ListNode(3)
head.next.next.next = ListNode(3)
print_list(head)
print_list(my_sol.deleteDuplicates(head))
```

```
1 => 2 => 3 => 3 =>
1 => 2 =>
```

86. Partition List (https://leetcode.com/problems/partition-list/)

Medium

Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

Example:

```
Input: head = 1=>4=>3=>2=>5=>2, x = 3
Output: 1=>2=>2=>4=>3=>5
```

Performance

- Runtime: 40 ms, faster than 97.40% of Python3 online submissions for Partition List.
- Memory Usage: 13.2 MB, less than 6.12% of Python3 online submissions for Partition List.

Complexity Analysis

```
O(n) in time.
O(1) in space.
```

In [7]:
```
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

def print_list(node):
    string = ''
    while node:
        string = string + str(node.val) + ' => '
        node = node.next
    print(string)

class Solution:
    def partition(self, head: ListNode, x: int) -> ListNode:
        if head:
            less_head = None
            less_tail = None
            geq_head = None
            geq_tail = None
            n1 = head
            while n1:
                if n1.val < x:
                    if not(less_head):
                        less_head = n1
                    else:
                        less_tail.next = n1
                    less_tail = n1
                else:
                    if not(geq_head):
                        geq_head = n1
                    else:
                        geq_tail.next = n1
                    geq_tail = n1
                n1 = n1.next
            if geq_head:
                head = geq_head
                geq_tail.next = None
            if less_head:
                less_tail.next = geq_head
                head = less_head
        return head

my_sol = Solution()

#[1=>4=>3=>2=>5=>2]
head = ListNode(1)
head.next = ListNode(4)
head.next.next = ListNode(3)
head.next.next.next = ListNode(2)
head.next.next.next.next = ListNode(5)
head.next.next.next.next.next = ListNode(2)
print_list(head)
print('Should print:')
print('1 => 2 => 2 => 4 => 3 => 5 =>')
print_list(my_sol.partition(head, 3))
```

```
1 => 4 => 3 => 2 => 5 => 2 =>
Should print:
1 => 2 => 2 => 4 => 3 => 5 =>
1 => 2 => 2 => 4 => 3 => 5 =>
```

61. Rotate List (https://leetcode.com/problems/rotate-list/)

Medium

Given a linked list, rotate the list to the right by k places, where k is non-negative.

Example 1:

```
Input: 1=>2=>3=>4=>5=>NULL, k = 2
Output: 4=>5=>1=>2=>3=>NULL
```

Explanation:

```
rotate 1 steps to the right: 5=>1=>2=>3=>4=>NULL
rotate 2 steps to the right: 4=>5=>1=>2=>3=>NULL
```

Example 2:

```
Input: 0=>1=>2=>NULL, k = 4
Output: 2=>0=>1=>NULL
```

Explanation:

```
rotate 1 steps to the right: 2=>0=>1=>NULL
rotate 2 steps to the right: 1=>2=>0=>NULL
rotate 3 steps to the right: 0=>1=>2=>NULL
rotate 4 steps to the right: 2=>0=>1=>NULL
```

Solution

- Iterate over the list keeping track of
  - two consecutive nodes (the previous and current nodes)
  - the index to find the length
- Point the tail to the head
- Count (length - k) indices to find the new head and tail
- Point the tail to None

Performance

- Runtime: 44 ms, faster than 85.37% of Python3 online submissions for Rotate List.
- Memory Usage: 13.2 MB, less than 5.77% of Python3 online submissions for Rotate List.

Complexity Analysis

```
O(n+k) in time.
O(1) in space.
```

In [8]:
```
# Definition for singly-linked list.
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

def print_list(node):
    string = ''
    while node:
        string = string + str(node.val) + ' => '
        node = node.next
    print(string)

class Solution:
    def rotateRight(self, head: ListNode, k: int) -> ListNode:
        if head:
            n0 = head
            length = 1
            idx = 0
            find_new_head = False
            while n0.next:
                n1 = n0.next
                idx += 1
                if not n1.next:
                    length = idx + 1
                    n1.next = head
                    idx = -1
                    find_new_head = True
                if find_new_head and idx == (length - k)%length:
                    head = n1
                    n0.next = None
                    break
                n0 = n1
        return head


my_sol = Solution()

#[1=>4=>3=>2=>5=>2]
head = ListNode(1)
head.next = ListNode(2)
head.next.next = ListNode(3)
head.next.next.next = ListNode(4)
head.next.next.next.next = ListNode(5)
print_list(head)
print('Should print:')
print('1 => 2 => 3 => 4 => 5 =>')
print_list(my_sol.rotateRight(head, 3))
```

```
1 => 2 => 3 => 4 => 5 =>
Should print:
1 => 2 => 3 => 4 => 5 =>
3 => 4 => 5 => 1 => 2 =>
```

Systems Design Mock (7:30pm 4 Apr 2019 2019)

How would you design a parking lot management system?

Requirements:

- Direct cars to spaces that are free
- 10K spaces

- Space sizes: S, M, L, XL (different charges for different spaces)
- Loyalty discounts for frequent customers

In [ ]:
```
class ParkingLot:
    def __init__(self, number_of_spaces, car_to_space_dict):

    def allocate_space(self, car):
        space_stack[car.size].pop(car)

    def free_space(space, ):
        space_stack[space.size].push(s_id)

    class space:
        def __init__(self, s_id, size, isfree):

    class car:
        def__init__(self, c_id, size):
```