Linked List

Mustafa Muhammad

12th November 2021

List of Questions:

1. General format

1 Add two numbers

Input: l1 = [2,4,3], l2 = [5,6,4]

Output: [7,0,8]

Explanation: 342 + 465 = 807.

```
class Solution:
    def addTwoNumbers(self, l1: Optional[ListNode], l2: Optional[ListNode]) -> Optional
        list1 = l1
        list2 = l2
        ans = ListNode()
        head = ans
        carry = 0
        while(list1 or list2):
            added = 0
            if list1 != None:
                added += list1.val
            else:
                added += 0

            if list2 != None:
                added += list2.val
            else:
                added += 0

            added += carry

            ans.next = ListNode(added % 10)

            ans = ans.next
            carry = int(added/10)

            if list1 != None:
                list1 = list1.next
            if list2 != None:
                list2 = list2.next

        if carry > 0:
            ans.next = ListNode(carry)
            ans = ans.next

        return head.next
```

2 Remove Nth Node from end of list

Input: head = [1,2,3,4,5], n = 2

Output: [1,2,3,5]

Input: head = [1], n = 1

Output: []

```
class Solution:
    def removeNthFromEnd(self, head: Optional[ListNode], n: int) -> Optional[ListNode]:
        right = head

        # so that we initialize 1 step back
        dummy = ListNode(0, head)
        left = dummy

        for i in range(0, n):
            right = right.next

        while right != None:
            right = right.next
            left = left.next

        left.next = left.next.next

        return dummy.next
```

3 Merge K sorted Lists

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Input: lists = [[1,4,5],[1,3,4],[2,6]] Output: [1,1,2,3,4,4,5,6] Explanation: The linked-lists are: [
1-¿4-¿5,

1-¿3-¿4,

2-¿6] merging them into one sorted list: 1-¿1-¿2-¿3-¿4-¿4-¿5-¿6

```
import heapq
class Solution:
    def mergeKLists(self, lists: List[Optional[ListNode]]) -> Optional[ListNode]:
        heap = []
        for i in range(0, len(lists)):
            linked_list = lists[i]
            while linked_list != None:
                heapq.heappush(heap, linked_list.val)
                linked_list = linked_list.next
        ans = ListNode()
        head = ans
        while len(heap) != 0:
            ans.next = ListNode(heapq.heappop(heap))
            ans = ans.next
        return head.next
```

4 Merge Two sorted lists

Input: l1 = [1,2,4], l2 = [1,3,4]

Output: [1,1,2,3,4,4]

```
class Solution:
    def mergeTwoLists(self, l1: Optional[ListNode], l2: Optional[ListNode]) -> Optional
        first = l1
        second = l2

        ans = ListNode()
        head = ans
        while first and second:
            if first.val < second.val:
                ans.next = ListNode(first.val)
                ans = ans.next
                first = first.next

            elif first.val > second.val:
                ans.next = ListNode(second.val)
                ans = ans.next
                second = second.next
            else:
                ans.next = ListNode(second.val)
                ans = ans.next
                ans.next = ListNode(second.val)
                ans = ans.next
                second = second.next
                first = first.next

        if first != None:
            while first != None:
                ans.next = ListNode(first.val)
                ans = ans.next
                first = first.next

        if second != None:
            while second != None:
                ans.next = ListNode(second.val)
                ans = ans.next
                second = second.next

        return head.next
```

5 Reverse Linked List

Input: head = [1,2,3,4,5] Output: [5,4,3,2,1]

```
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        prev = None

        current = head
        while current != None:
            next = current.next
            current.next = prev
            prev = current
            current = next

        return prev
```

6 LinkedList Cycle

Given head, the head of a linked list, determine if the linked list has a cycle in it.

Naive approach is to use a hashmap with 0(n) worst case space.

```
class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:
        slow = head
        fast = head

        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next

            if fast == slow:
                return True

        return False
```

7 LinkedList Cycle II

```
class Solution:
    def detectCycle(self, head: ListNode) -> ListNode:
        slow = head
        fast = head
        cycle = False
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next
            if fast == slow:
                cycle = True
                break
        if cycle:
            slow = head
            while slow != fast:
                fast = fast.next
                slow = slow.next
            return slow
        return None
```

8 Remove duplicates from sorted list

```
class Solution:
    def deleteDuplicates(self, head: Optional[ListNode]) -> Optional[ListNode]:
        node = head
        ans = node

        if not head or not head.next: return head

        while node.next != None:
            if node.val == node.next.val:
                node.next = node.next.next
            else:
                node = node.next

        return ans
```

9 Partition List

all less than x on one side and all greater than equal to x on the other.

Input: head = [1,4,3,2,5,2], x = 3 Output: [1,2,2,4,3,5]

```
class Solution:
    def partition(self, head: Optional[ListNode], x: int) -> Optional[ListNode]:
        left_part = ListNode()
        ans_head = left_part
        right_part = ListNode()
        right_head = right_part
        node = head
        while node != None:
            if node.val < x:
                left_part.next = ListNode(node.val)
                left_part = left_part.next
            else:
                right_part.next = ListNode(node.val)
                right_part = right_part.next

            node = node.next
        left_part.next = right_head.next
        return ans_head.next
```

10 Rotate List

Given the head of a linked list, rotate the list to the right by k places.

Input: head = [1,2,3,4,5], k = 2 Output: [4,5,1,2,3]

```
class Solution:
    def rotateRight(self, head: Optional[ListNode], k: int) -> Optional[ListNode]:
        if head == None:
            return None
        tail = head
        length = 1
        while tail.next != None:
            tail = tail.next
            length += 1
        k = k%length
        if k == 0:
            return head
        position = length-1-k
        j = 0
        front = head
        while j != position:
            front = front.next
            j+=1
        tail_start = front.next
        front.next = None
        tail.next = head
        return tail_start
```

11 Remove LinkedList Elements

Remove all duplicate elements val from a linkedlist.

```
class Solution:
    def removeElements(self, head: Optional[ListNode], val: int) -> Optional[ListNode]:
        if head == None:
            return None
        while(head != None and head.val == val):
            head = head.next
        current = head
        current_head = current
        while current != None and current.next != None:
            if current.next.val == val:
                current.next = current.next.next
            else:
                current = current.next

        return current_head
```

12 Delete Node

Bouncer question because we're asked to delete the node that we're currently at.

```
class Solution:
    def deleteNode(self, node):
        """
        :type node: ListNode
        :rtype: void Do not return anything, modify node in-place instead.
        """
        node.val = node.next.val
        node.next = node.next.next
```

13 Middle of Linked List

```
class Solution:
    def middleNode(self, head: Optional[ListNode]) -> Optional[ListNode]:
        fast = head
        slow = head

        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next

        return slow
```

14 Odd Even LinkedList

```
class Solution:
    def oddEvenList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if head == None:
            return None

        even = head.next
        even_head = even
        odd = head
        odd_head = odd

        while even and even.next:
            odd.next = even.next
            odd = odd.next
            even.next = odd.next
            even = even.next

        odd.next = even_head
        return odd_head
```

15 Reorder List

You are given the head of a singly linked-list. The list can be represented as: L0 - L1 - ... - Ln - 1 - Ln

Input: head = [1,2,3,4,5]

Output: [1,5,2,4,3]

```
import copy
class Solution:
    def reorderList(self, head: ListNode) -> None:
        if not head or not head.next:
```

```
    return

def reverse(head):
    prev = None
    current = copy.copy(head)
    while current != None:
        next = current.next
        current.next = prev
        prev = current
        current = next
    return prev

slow, fast = head, head
while fast and fast.next:
    slow = slow.next
    fast = fast.next.next

head1 = reverse(slow.next)
slow.next = None
p = head
q = head1
while q:
    temp1 = p.next
    temp2 = q.next
    p.next = q
    q.next = temp1
    p = temp1
    q = temp2
```