Linked Lists: Interview / LeetCode exercise

Created	@August 31, 2025 10:35 AM
Course	Python Data Structures & Algorithms + LEETCODE Exercises

Find Middle Node

https://leetcode.com/problems/middle-of-the-linked-list/

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def middleNode(self, head: Optional[ListNode]) -> Optional[ListNode]:
        slow = head
        fast = head
        while fast is not None and fast.next is not None:
            slow = slow.next
            fast = fast.next.next
        return slow
```

Has Loop

Determine if there is a loop inside the linked list.

If use 2 pointers and there is a loop inside the linked list, eventually both these pointers will be in same position

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def hasLoop(self, head: Optional[ListNode]) -> Optional[bool]:
        slow = head
        fast = head
        while fast is not None and fast.next is not None:
            slow = slow.next
            fast = fast.next.next
            if slow == fast:
                return True
        return False
```

Kth Node from End

Given this linked list: 1 → 2 → 3 → 4 → 5

- if k = 1, return the first node from the end, which is 5
- if k = 2, return 4
- if k = 5, return 1
- if k > 5, return None

Use two pointers, both of them are initialized with head of list

- fast pointer needs to jump k position ahead
  - if fast is None during this move, then the list has length less than k
  - else, the distance between fast and slow is k
- move both pointers forward, until fast reaches the end of the list
  - since k is **ordinary**, then need to check fast is None, not fast is tail

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def kThNodeFromEnd(self, head: Optional[ListNode], k) → Optional[ListNod
e]:
        slow = head
        fast = head
        for _ in range(k):
            if fast is not None:
                fast = fast.next
            else:
                return None
        while fast is not None:
            fast = fast.next
            slow = slow.next
        return slow
```

Remove Duplicates

Given a linked list, remove nodes with duplicated values.

Example: 1 → 2 → 3 → 1 → 4 → 2 → 5

output: 1 → 2 → 3 → 4 → 5

2 approaches:

- Use a set, gives `O(N)` time complexity
- Use nested loop, gives `O(N^2)` time complexity
  - `cur` keeps track the node where its value is being compared
  - `runner` keeps track the previous node of the node being compared with `cur`

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def removeDuplicates(self, head: Optional[ListNode]) -> Optional[ListNode]:
        runner = head
        cur = head
        while cur is not None:
                value = cur.value
                while runner.next is not None:
                        if runner.next.value == value:
                                runner.next = runner.next.next
                        else:
                                runner = runner.next
                cur = cur.next
                runner = cur
```

Binary to Decimal

Given the linked list: 1 → 1 → 1 → 1

output: 15

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def binToDec(self, head: Optional[ListNode]) -> Optional[int]:
        number = 0
        cur = head
        while cur is not None:
                number = number * 2 + cur.value
```

```
            cur = cur.next
        return number
```

Partition List

Given a linked list and number k:

example: 3 → 8 → 5 → 10 → 2 → 1 and k = 5

partition the linked list into 2 list, 1 list where all node has value less than k, 1 list where all node has value equal or greater than k, while maintaining the relative order of node.

example: 3 → 2 → 1 and 8 → 5 → 10

then merge these 2 partition together, the partition less than k will go first, then the partition equal or greater than k

output: 3 → 2 → 1 → 8 → 5 → 10

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def partitionList(self, head: Optional[ListNode], k) → Optional[ListNode]:
        dummy1 = ListNode("dummy1")
        dummy2 = ListNode("dummy2")
        prev1 = dummy1
        prev2 = dummy2
        cur = head
        while cur is not None:
            if cur.value < k:
                prev1.next = cur
                prev1 = prev1.next
            else:
                prev2.next = cur
                prev2 = prev2.next
```

```
            cur = cur.next
        prev2.next = None
        prev1.next = dummy2.next
        return dummy1.next
```

Reverse Between

Given a linked list, index i and j. Reverse the order of the nodes from (inclusive) i to (inclusive) j.

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def reverseBetween(self, head: Optional[ListNode], start_index, end_index)
→ Optional[ListNode]:
        if head is None or head.next is None:
            return None # nothing to be reversed :)
        dummy = Node("dummmy") # in case i = 0
        dummy.next = head
        prev = dummy
        for _ in range(start_index):
            prev = prev.next # prev points to index i - 1
        current = prev.next
        for _ in range(end_index - start_index):
            to_move = current.next # to_move will then be moved to index i
            current.next = to_move.next
            to_move.next = prev.next
            prev.next = to_move
        head = dummy.next
        return head
```

Swap Nodes in Pairs

Given a linked list, for each pair of adjacent nodes, swaps their positions. If there are odd number of nodes, leave the last node.

Example: 1 → 2 → 3 → 4

output: 2 → 1 → 4 → 3

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def swapPairs(self, head: Optional[ListNode]) → Optional[ListNode]:
        if head is None or head.next is None:
            return None
        dummy = Node("dummmy")
        dummy.next = head
        prev = dummy # points to the previous position of the pair to be swapp
ed
        while prev is not None:
                m1 = prev.next
                if m1 is None:
                    break
                m2 = m1.next:
                if m2 is None:
                    break
                m1.next = m2.next
                m2.next = m1
                prev.next = m2
                prev = prev.next.next
            head = dummy.next
```