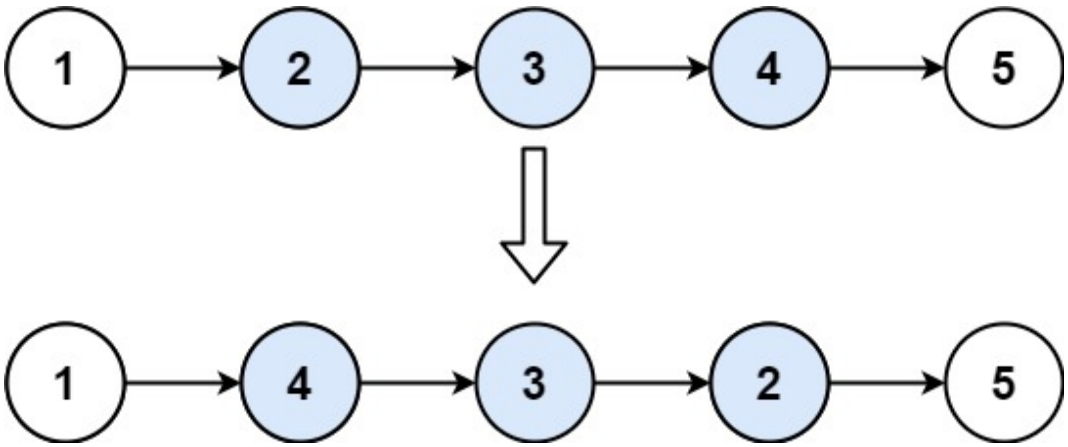
92. Reverse Linked List II

Created	@October 20, 2021 11:34 PM
Difficulty	Medium
LC Url	https://leetcode.com/problems/reverse-linked-list-ii/
Importance	
Tag	Array&Sorting
Video	

Given the `head` of a singly linked list and two integers `left` and `right` where `left <= right`, reverse the nodes of the list from position `left` to position `right`, and return *the reversed list*.

Example 1:




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

Example 2:

```
Input: head = [5], left = 1, right = 1
Output: [5]
```

Constraints:

- The number of nodes in the list is `n`.

- `1 <= n <= 500`
- `500 <= Node.val <= 500`
- `1 <= left <= right <= n`

Follow up:

Could you do it in one pass?

Solution

```
# 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], left: int, right: int) -> Optional[ListNode]:
        def reverseN(head, right):
            if right == 1:
                self.successor = head.next
                return head
            last = reverseN(head.next, right-1)
            head.next.next = head
            head.next = self.successor
            return last

        if left == 1:
            return reverseN(head, right)

        head.next = self.reverseBetween(head.next, left-1, right-1)
        return head
```

```
# 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: ListNode, left: int, right: int) -> ListNode:
        """
        Example:
        [1 -> 2 -> 3 -> 4 -> 5]
        """

        # edge case: left and right point the same
        if left == right:
            return head

        """
        current = 1
```

```
        previous = None
        """
        current = head
        previous = None

        # skip to left - 1 nodes
        for i in range(left - 1):
            previous = current
            current = current.next

        # the node before sublist
        """
        current = 2
        previous = 1
        lastNodeOfFirstPart = 1
        """
        lastNodeOfFirstPart = previous

        # after reversing; last node of sublist will be current
        """
        lastNodeOfSubList = 2
        """
        lastNodeOfSubList = current
        next = None

        # reverse until right
        for i in range(right - left + 1):
            next = current.next
            current.next = previous
            previous = current
            current = next

        """
        After the reverse
        1 -> <- 2 <- 3 <- 4

        current = 5
        """

        # connect the first part
        """
        1 -> 4 -> 3 -> 2
        """
        if lastNodeOfFirstPart:
            lastNodeOfFirstPart.next = previous
        else:
            head = previous

        # connect to the last part
        """
        lastNodeOfSubList = 2
        current = 5

        1 -> 4 -> 3 -> 2 -> 5
        """
        lastNodeOfSubList.next = current

        return head
```

21. Merge Two Sorted Lists

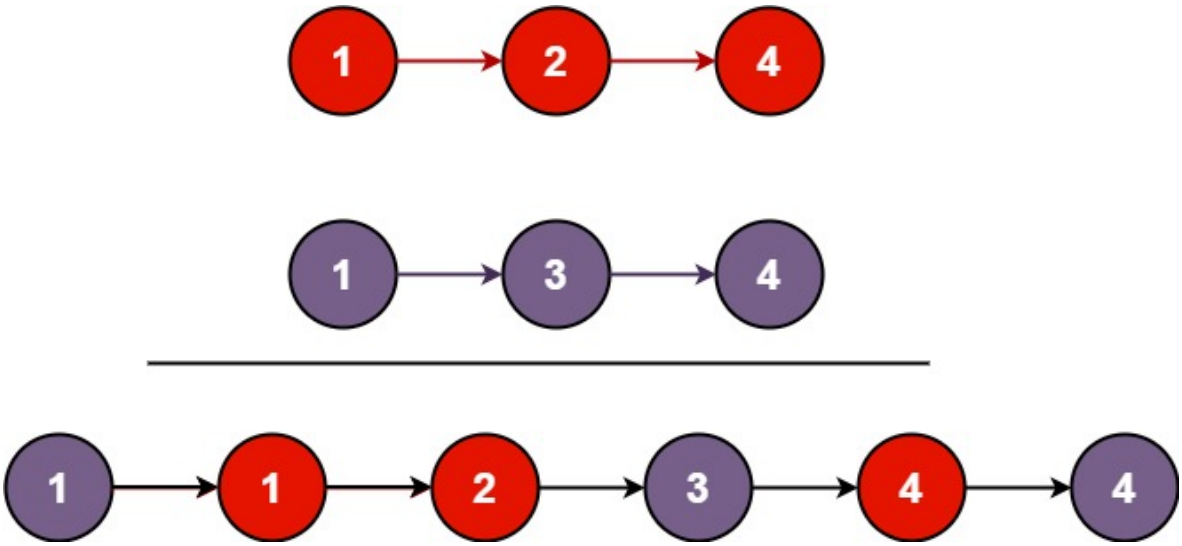
Created	@February 23, 2022 10:21 PM
Difficulty	Easy
LC Url	https://leetcode.com/problems/merge-two-sorted-lists/
Importance	
Tag	Array&Sorting Two pointers
Video	

You are given the heads of two sorted linked lists `list1` and `list2`.

Merge the two lists in a one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*.

Example 1:




```
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]
```

Example 2:

```
Input: list1 = [], list2 = []
Output: []
```

Example 3:

```
Input: list1 = [], list2 = [0]
Output: [0]
```

Constraints:

- The number of nodes in both lists is in the range `[0, 50]`.
- `100 <= Node.val <= 100`
- Both `list1` and `list2` are sorted in **non-decreasing** order.

Solution

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode]) -> Optional[ListNode]:
        dummy = ListNode(-1)
        p = dummy
        p1, p2 = list1, list2

        while p1 and p2:
            if p1.val > p2.val:
                p.next = p2
                p2 = p2.next
            else:
                p.next = p1
                p1 = p1.next
            p = p.next

        if p1:
            p.next = p1

        if p2:
            p.next = p2

        return dummy.next
```

linked list不耗费额外的空间

88. Merge Sorted Array

O(n)的额外的空间

```
def mergeSortedArray(self, A, B):
    i, j = 0, 0
    C = []
    while i < len(A) and j < len(B):
        if A[i] < B[j]:
            C.append(A[i])
            i += 1
        else:
            C.append(B[j])
            j += 1

    while i < len(A):
        C.append(A[i])
        i += 1

    while j < len(B):
        C.append(B[j])
        j += 1

    return C
```

143. Reorder List

Created	@October 17, 2022 3:50 PM
Difficulty	Medium
LC Url	https://leetcode.com/problems/reorder-list/
Importance	
Tag	LinkedList NEET
Video	

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

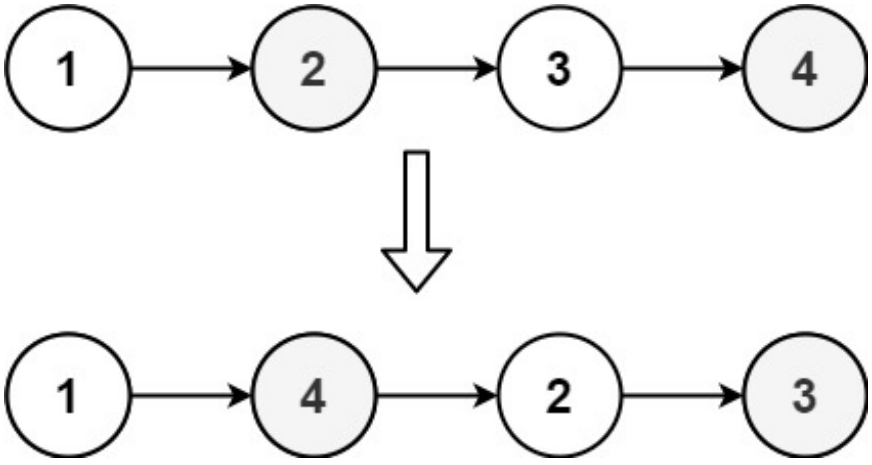
```
L0 → L1 → … → Ln - 1 → Ln
```

Reorder the list to be on the following form:

```
L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …
```

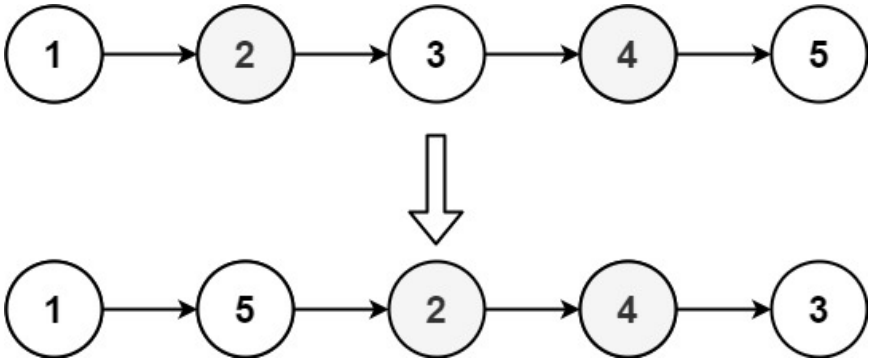
You may not modify the values in the list's nodes. Only nodes themselves may be changed.

Example 1:



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

Example 2:




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

Constraints:

- The number of nodes in the list is in the range `[1, 5 * 10 4 ]`.
- `1 <= Node.val <= 1000`

Solution

```
class Solution:
    def reorderList(self, head: ListNode) -> None:
        # find middle
        slow, fast = head, head.next
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next

        # reverse second half
        second = slow.next
        prev = slow.next = None
        while second:
            tmp = second.next
            second.next = prev
            prev = second
```

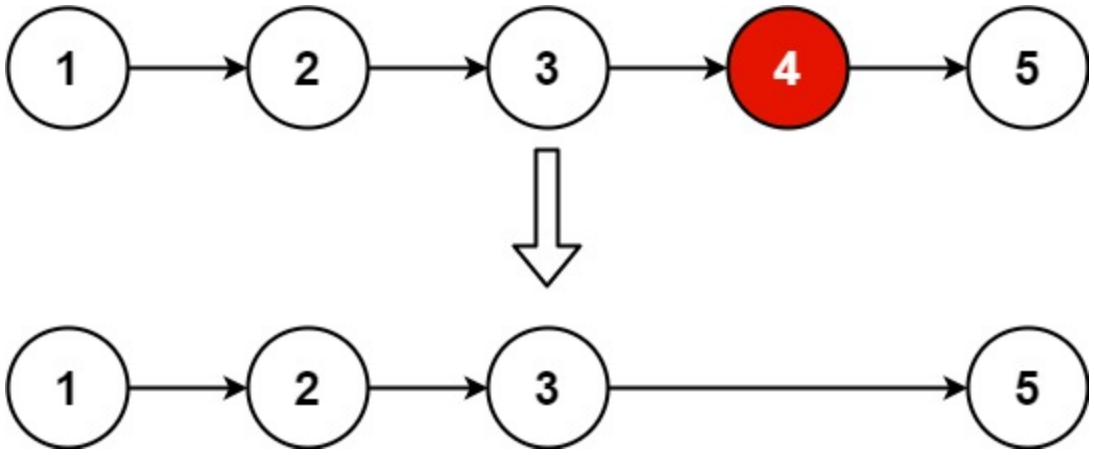
```
            second = tmp

        # merge two halfs
        first, second = head, prev
        while second:
            tmp1, tmp2 = first.next, second.next
            first.next = second
            second.next = tmp1
            first, second = tmp1, tmp2
```

19. Remove Nth Node From End of List

Created	@July 15, 2020 7:25 PM
Difficulty	Medium
LC Url	https://leetcode.com/problems/remove-nth-node-from-end-of-list/vv
Importance	****
Tag	LinkedList NEET Two pointers
Video	https://www.youtube.com/watch?v=XVuQxVej6y8

Given the `head` of a linked list, remove the `n th` node from the end of the list and return its head.

Example 1:




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

Example 2:

```
Input: head = [1], n = 1
Output: []
```

Example 3:

```
Input: head = [1,2], n = 1
Output: [1]
```

Constraints:

- The number of nodes in the list is `sz`.
- `1 <= sz <= 30`
- `0 <= Node.val <= 100`
- `1 <= n <= sz`

Follow up: Could you do this in one pass?

Solution

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
        dummy = ListNode(0)
        dummy.next = head
        slow = fast = dummy

        for _ in range(n):
            fast = fast.next

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

        slow.next = slow.next.next

        return dummy.next
```

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        ListNode dummy = new ListNode(0);
        dummy.next = head;
        ListNode fast = dummy;
        ListNode slow = dummy;
        for (int i = 0; i < n; i++) {
            fast = fast.next;
        }
        while (fast.next != null) {
            fast = fast.next;
            slow = slow.next;
        }
        slow.next = slow.next.next;
        return dummy.next;
    }
}
```

138. Copy List with Random Pointer

Created	@October 30, 2022 9:02 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/copy-list-with-random-pointer/
Importance	
Tag	LinkedList NEET
Video	

A linked list of length `n` is given such that each node contains an additional random pointer, which could point to any node in the list, or `null`.

Construct a **deep copy** of the list. The deep copy should consist of exactly `n` **brand new** nodes, where each new node has its value set to the value of its corresponding original node. Both the `next` and `random` pointer of the new nodes should point to new nodes in the copied list such that the pointers in the original list and copied list represent the same list state. **None of the pointers in the new list should point to nodes in the original list**.

For example, if there are two nodes `X` and `Y` in the original list, where `X.random --> Y`, then for the corresponding two nodes `x` and `y` in the copied list, `x.random --> y`.

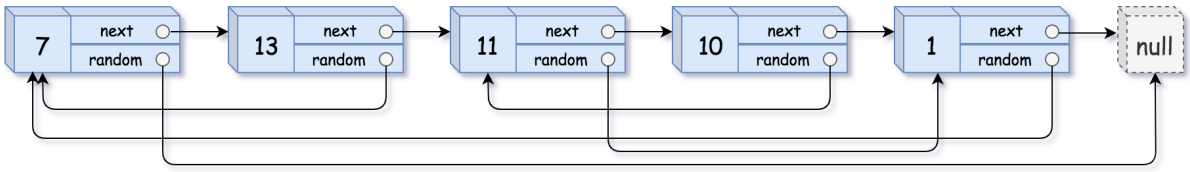
Return *the head of the copied linked list*.

The linked list is represented in the input/output as a list of `n` nodes. Each node is represented as a pair of `[val, random_index]` where:

- `val`: an integer representing `Node.val`
- `random_index`: the index of the node (range from `0` to `n-1`) that the `random` pointer points to, or `null` if it does not point to any node.

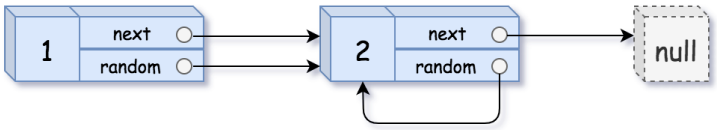
Your code will **only** be given the `head` of the original linked list.

Example 1:



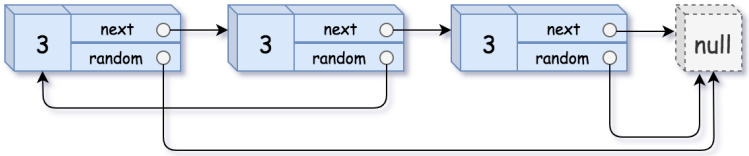

```
Input: head = [[7,null],[13,0],[11,4],[10,2],[1,0]]
Output: [[7,null],[13,0],[11,4],[10,2],[1,0]]
```

Example 2:




```
Input: head = [[1,1],[2,1]]
Output: [[1,1],[2,1]]
```

Example 3:




```
Input: head = [[3,null],[3,0],[3,null]]
Output: [[3,null],[3,0],[3,null]]
```

Constraints:

- `0 <= n <= 1000`
- `10 4 <= Node.val <= 10 4`

- `Node.random` is `null` or is pointing to some node in the linked list.

Solution

```
"""
# Definition for a Node.
class Node:
    def __init__(self, x: int, next: 'Node' = None, random: 'Node' = None):
        self.val = int(x)
        self.next = next
        self.random = random
"""

class Solution:
    def copyRandomList(self, head: 'Optional[Node]') -> 'Optional[Node]':
        oldToCopy = {None: None}

        cur = head
        while cur:
            old_copy = Node(cur.val)
            oldToCopy[cur] = old_copy
            cur = cur.next

        cur = head
        while cur:
            new_copy = oldToCopy[cur]
            new_copy.next = oldToCopy[cur.next]
            new_copy.random = oldToCopy[cur.random]
            cur = cur.next

        return oldToCopy[head]
```

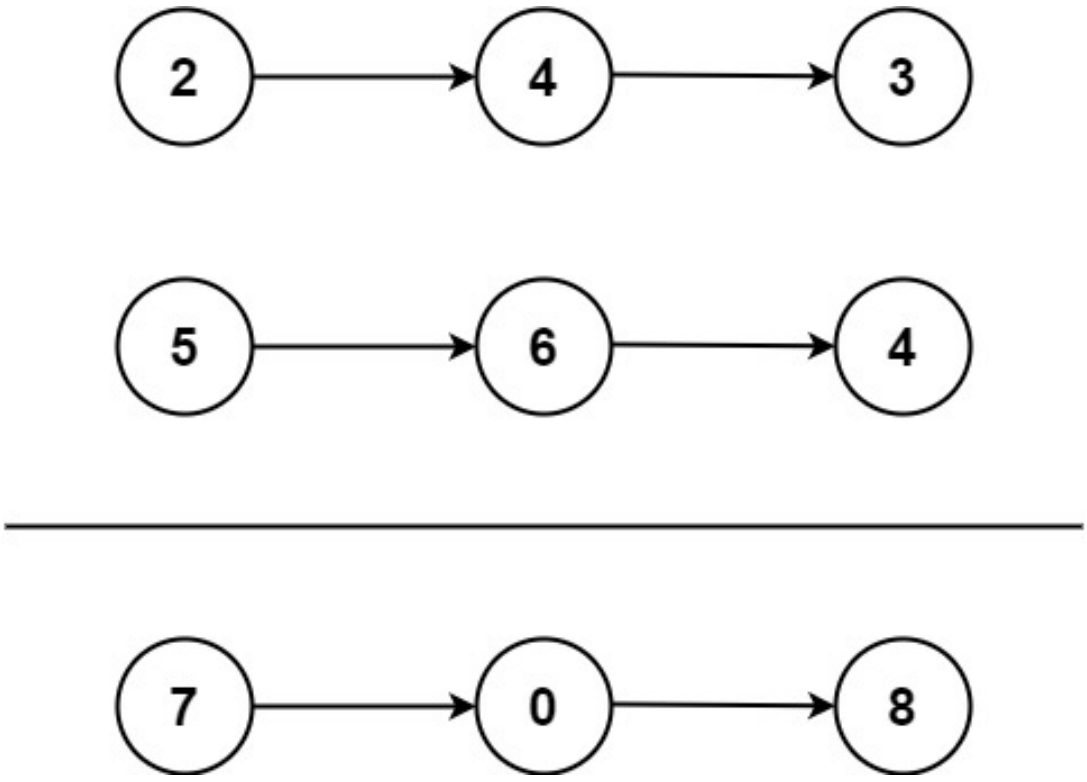
2. Add Two Numbers

Created	@July 10, 2020 6:17 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/add-two-numbers/
Importance	
Tag	LinkedList NEET
Video	

You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example 1:



```
Input: l1 = [2,4,3], l2 = [5,6,4]
Output: [7,0,8]
Explanation: 342 + 465 = 807.
```

Example 2:

```
Input: l1 = [0], l2 = [0]
Output: [0]
```

Example 3:

```
Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]
Output: [8,9,9,9,0,0,0,1]
```

Constraints:

- The number of nodes in each linked list is in the range `[1, 100]`.
- `0 <= Node.val <= 9`
- It is guaranteed that the list represents a number that does not have leading zeros.

Solution

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def addTwoNumbers(self, l1: Optional[ListNode], l2: Optional[ListNode]) -> Optional[ListNode]:
        # 新建dummy node
        dummy = ListNode()
        cur = dummy

        # 标识进位的数值，初始化为0，表示没有进位
        carry = 0
        # 只要l1或者l2还没有遍历结束 或者有进位（carry不为0）导致长度增加
        # 则继续循环
        while l1 or l2 or carry:
            # 如果l1不为空，则获取l1的值，否则给定为0
            v1 = l1.val if l1 else 0
            # 同理
            v2 = l2.val if l2 else 0
```

```
            # 计算两数之和，注意包括carry（上一次的进位）
            isum = v1 + v2 + carry
            # 求进位
            carry = isum // 10
            # 求余数
            isum = isum % 10
            # 当前指针指向新建的Node，里面包括了当前的余数
            cur.next = ListNode(isum)

            # 更新node信息：移动到下一位
            cur = cur.next
            l1 = l1.next if l1 else None
            l2 = l2.next if l2 else None

        # 返回dummy node的next，即目标的链表的第一个node
        return dummy.next
```

Ref: https://github.com/neetcode-gh/leetcode/blob/main/python/2-Add-Two-Numbers.py

141. Linked List Cycle

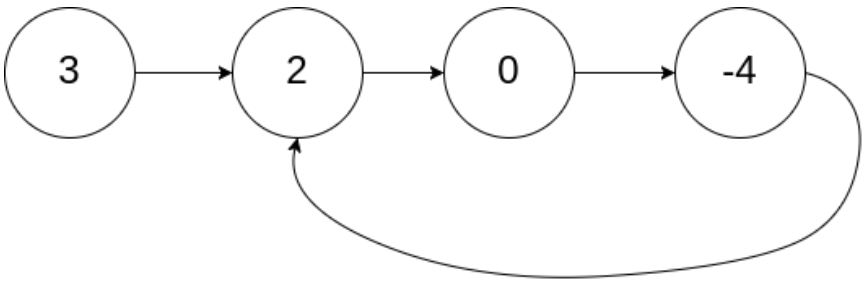
Created	@October 30, 2022 3:11 PM
Difficulty	Easy
LC Url	https://leetcode.com/problems/linked-list-cycle/
Importance	
Tag	LinkedList NEET Two pointers
Video	

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

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter**.

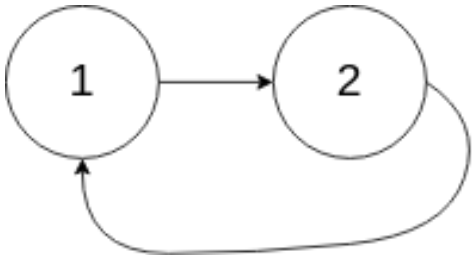
Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.

Example 1:



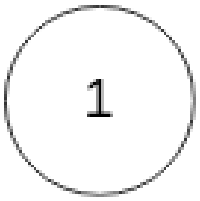

```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 1st node
 (0-indexed).
```

Example 2:




```
Input: head = [1,2], pos = 0
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.
```

Example 3:




```
Input: head = [1], pos = -1
Output: false
Explanation: There is no cycle in the linked list.
```

Constraints:

- The number of the nodes in the list is in the range `[0, 10 4 ]`.
- `10 5  <= Node.val <= 10 5`
- `pos` is `1` or a **valid index** in the linked-list.

Follow up: Can you solve it using `O(1)` (i.e. constant) memory?

Solution

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

```
class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:
        # 初始化快慢双指针
        slow, fast = head, head

        # 当快指针走到末尾的时候停止
        while fast and fast.next:
            # update 快慢指针
            slow = slow.next
            fast = fast.next.next
            # 如果相遇，则返回True
            if slow == fast:
                return True

        # 已经推出了循环，则返回False，没有环
        return False
```

Ref:

- https://labuladong.github.io/algo/2/19/18/
- https://github.com/neetcode-gh/leetcode/blob/main/python/141-Linked-List-Cycle.py

287. Find the Duplicate Number

Created	@October 30, 2022 4:17 PM
Difficulty	Medium
LC Url	https://leetcode.com/problems/find-the-duplicate-number/
Importance	
Tag	LinkedList NEET
Video	

Given an array of integers `nums` containing `n + 1` integers where each integer is in the range `[1, n]` inclusive.

There is only **one repeated number** in `nums`, return *this repeated number*.

You must solve the problem **without** modifying the array `nums` and uses only constant extra space.

Example 1:

```
Input: nums = [1,3,4,2,2]
Output: 2
```

Example 2:

```
Input: nums = [3,1,3,4,2]
Output: 3
```

Constraints:

- `1 <= n <= 10 5`
- `nums.length == n + 1`
- `1 <= nums[i] <= n`

- All the integers in `nums` appear only **once** except for **precisely one integer** which appears **two or more** times.

Follow up:

- How can we prove that at least one duplicate number must exist in `nums`?
- Can you solve the problem in linear runtime complexity?

Solution

```
class Solution:
    def findDuplicate(self, nums: List[int]) -> int:
        slow, fast = 0, 0
        while True:
            slow = nums[slow]
            fast = nums[nums[fast]]
            if slow == fast:
                break

        slow2 = 0
        while True:
            slow = nums[slow]
            slow2 = nums[slow2]
            if slow == slow2:
                return slow
```

Code: https://github.com/neetcode-gh/leetcode/blob/main/python/287-Find-The-Duplicate-Number.py

解释推荐：https://leetcode.cn/problems/find-the-duplicate-number/solution/kuai-man-zhi-zhen-tu-jie-by-lin-lin-lu-o8vd/

146. LRU Cache

Created	@April 4, 2022 12:07 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/lru-cache/
Importance	
Tag	Hashmap
Video	https://maxming0.github.io/2020/04/26/LRU-Cache/

Design a data structure that follows the constraints of a **Least Recently Used (LRU) cache**.

Implement the `LRUCache` class:

- `LRUCache(int capacity)` Initialize the LRU cache with **positive** size `capacity`.
- `int get(int key)` Return the value of the `key` if the key exists, otherwise return `1`.
- `void put(int key, int value)` Update the value of the `key` if the `key` exists. Otherwise, add the `key-value` pair to the cache. If the number of keys exceeds the `capacity` from this operation, **evict** the least recently used key.

The functions `get` and `put` must each run in `O(1)` average time complexity.

Example 1:

```
Input
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
Output
[null, null, null, 1, null, -1, null, -1, 3, 4]

Explanation
LRUCache lRUCache = new LRUCache(2);
lRUCache.put(1, 1); // cache is {1=1}
lRUCache.put(2, 2); // cache is {1=1, 2=2}
lRUCache.get(1);    // return 1
lRUCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}
lRUCache.get(2);    // returns -1 (not found)
lRUCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4, 3=3}
```

```
lRUCache.get(1);    // return -1 (not found)
lRUCache.get(3);    // return 3
lRUCache.get(4);    // return 4
```

Constraints:

- `1 <= capacity <= 3000`
- `0 <= key <= 10 4`
- `0 <= value <= 10 5`
- At most 2 `* 10 5` calls will be made to `get` and `put`.

Solution

```
class Node:
    def __init__(self, key, val):
        self.key, self.val = key, val
        self.prev = self.next = None

class LRUCache:

    def __init__(self, capacity: int):
        self.cap = capacity
        self.cache = {} # map key to node

        self.left, self.right = Node(0, 0), Node(0, 0)
        self.left.next, self.right.prev = self.right, self.left

    def remove(self, node):
        prev, nxt = node.prev, node.next
        prev.next, nxt.prev = nxt, prev

    def insert(self, node):
        prev, nxt = self.right.prev, self.right
        prev.next = nxt.prev = node
        node.next, node.prev = nxt, prev

    def get(self, key: int) -> int:
        if key in self.cache:
            self.remove(self.cache[key])
            self.insert(self.cache[key])
            return self.cache[key].val
        return -1

    def put(self, key: int, value: int) -> None:
        if key in self.cache:
```

```
            self.remove(self.cache[key])
        self.cache[key] = Node(key, value)
        self.insert(self.cache[key])

        if len(self.cache) > self.cap:
            # remove from the list and delete the LRU from hashmap
            lru = self.left.next
            self.remove(lru)
            del self.cache[lru.key]

# Your LRUCache object will be instantiated and called as such:
# obj = LRUCache(capacity)
# param_1 = obj.get(key)
# obj.put(key,value)
```