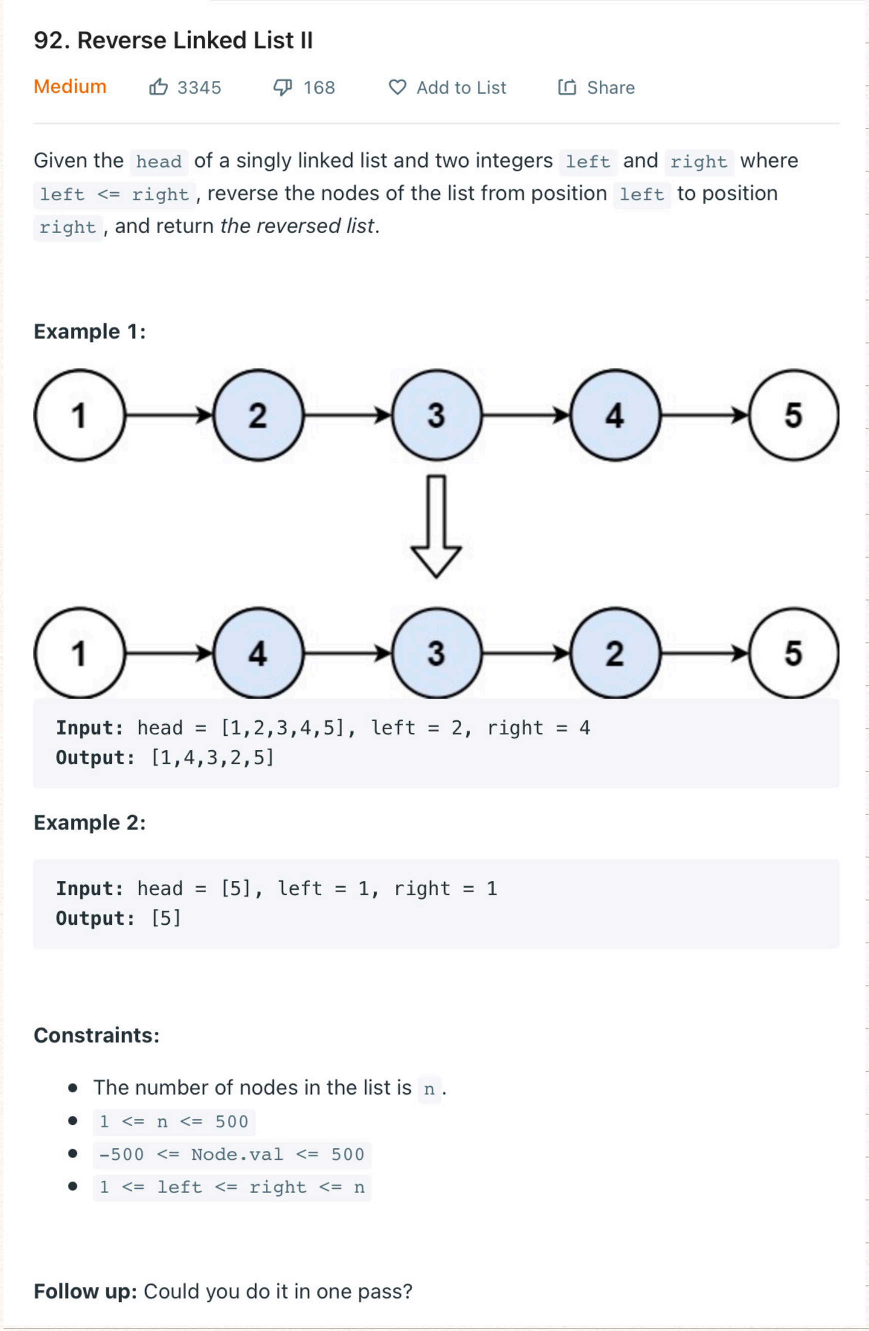
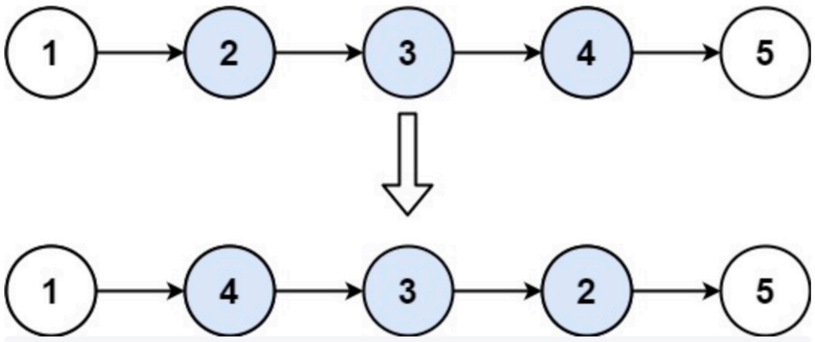


92. Reverse Linked List II

Medium 3345 168 Add to List Share

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?

⇓ reverse (head)

head
↓
1 → 2 → 3 → 4 → 5 → 6 → NULL

⇓ last = reverse (head . next)

head
↓
1 → reverse(2 → 3 → 4 → 5 → 6 → NULL)

⇓

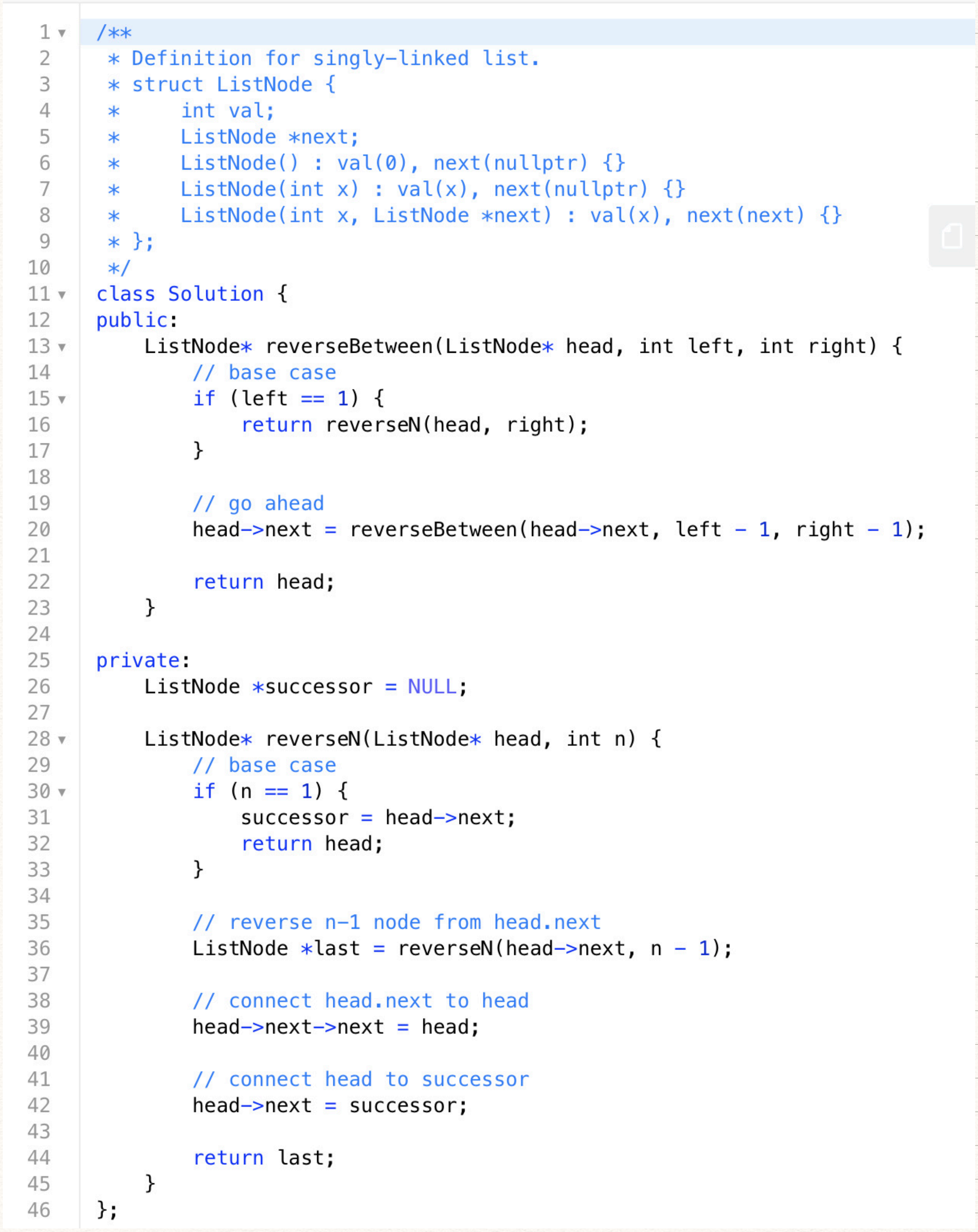
head   last
↓   ↓
1 → 2 ← 3 ← 4 ← 5 ← 6
       ↓
     NULL   ⇓ head . next . next = head

head   last
↓   ↓
1 ⇄ 2 ← 3 ← 4 ← 5 ← 6

⇓ head . next = NULL

head   last
↓   ↓
NULL ← 1 ← 2 ← 3 ← 4 ← 5 ← 6

⇓ return last



```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int left, int right) {
        // base case
        if (left == 1) {
            return reverseN(head, right);
        }

        // go ahead
        head->next = reverseBetween(head->next, left - 1, right - 1);

        return head;
    }

private:
    ListNode *successor = NULL;

    ListNode* reverseN(ListNode* head, int n) {
        // base case
        if (n == 1) {
            successor = head->next;
            return head;
        }

        // reverse n-1 node from head.next
        ListNode *last = reverseN(head->next, n - 1);

        // connect head.next to head
        head->next->next = head;

        // connect head to successor
        head->next = successor;

        return last;
    }
};
```