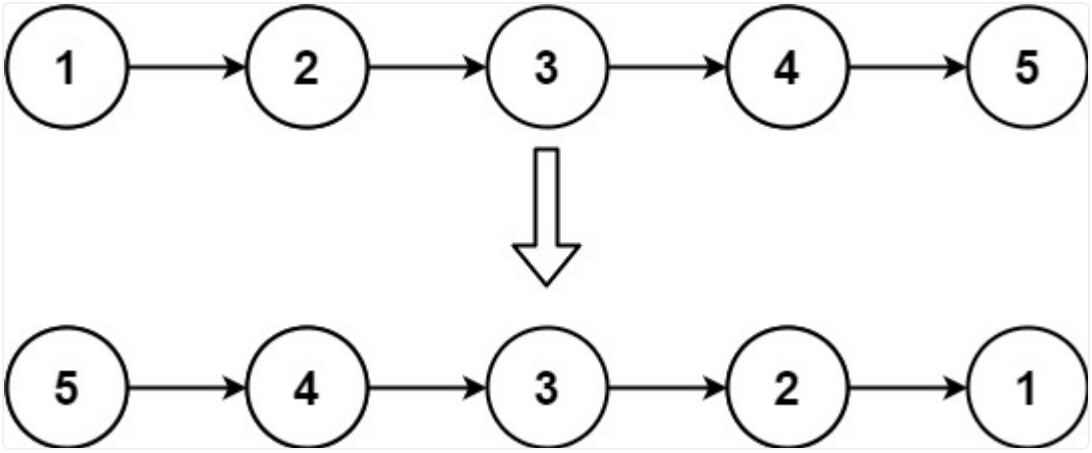
Reverse Linked List(Day 19)

Prepared By Azan Imtiaz

Reverse Linked List (Leatcode)

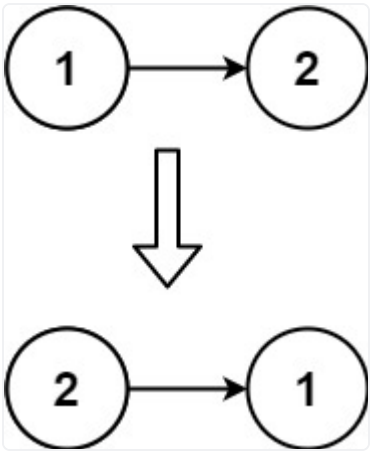
Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:




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

Example 2:




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

Example 3:

```
Input: head = []
Output: []
```

Constraints:

- The number of nodes in the list is the range `[0, 5000]`.
- `-5000 <= Node.val <= 5000`

Problem

We need to reverse a singly linked list. A singly linked list is a data structure where each element, called a node, holds a value and a link to the next node in the sequence. The last node points to `null`, indicating the end of the list.

Intuition

The idea is to change the direction of the links between nodes. We want to make each node point to its previous node instead of its next one.

Approach

1. Initialize two pointers: `prev` as `null` and `cur` as the head of the list.
2. Iterate through the list:
   - Temporarily store the next node.
   - Reverse the current node's link to point to `prev`.
   - Move `prev` to the current node.
   - Advance to the next node.
3. When we reach the end of the list, `prev` will be pointing to the new head of the reversed list.

Solution

```
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode prev = null;
        ListNode cur = head;

        while (cur != null) {
            ListNode nextNode = cur.next; // Store the next node
            cur.next = prev; // Reverse the link
            prev = cur; // Move `prev` to the current node
            cur = nextNode; // Move to the next node in the original list
        }

        // `prev` will be the new head of the reversed list
        return prev;
    }
}
```

Description of Solution

The solution reverses the linked list by reassigning the `next` pointer of each node to its previous node. This is done iteratively until all nodes have been visited and their links reversed. The `prev` pointer ends up at the last node, which becomes the new head of the reversed list.

Dry Run

Let's consider three test cases:

Test Case 1: An empty list

- Input: `head = null`
- Output: `null`
- No nodes to reverse, so the returned list is also `null`.

Test Case 2: A list with one node

- Input: `head = [1]`
- Output: `[1]`

- Only one node is present, so reversing it doesn't change the list.

Test Case 3: A list with multiple nodes

- Input: `head = [1 -> 2 -> 3]`
- Output: `[3 -> 2 -> 1]`

Step-by-step:

1. `prev = null`, `cur = 1`
   - `nextNode = 2`
   - `cur.next = null` (reverse link)
   - `prev = 1`
   - `cur = 2`
2. `prev = 1`, `cur = 2`
   - `nextNode = 3`
   - `cur.next = 1` (reverse link)
   - `prev = 2`
   - `cur = 3`
3. `prev = 2`, `cur = 3`
   - `nextNode = null`
   - `cur.next = 2` (reverse link)
   - `prev = 3`
   - `cur = null` (end of list)

The new head is `prev`, which is `3`. The reversed list is `[3 -> 2 -> 1]`.

Thanks For Reading