Reorder List(Day 23)

Prepared By Azan Imtiaz

Reorder List(Leatcode)

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

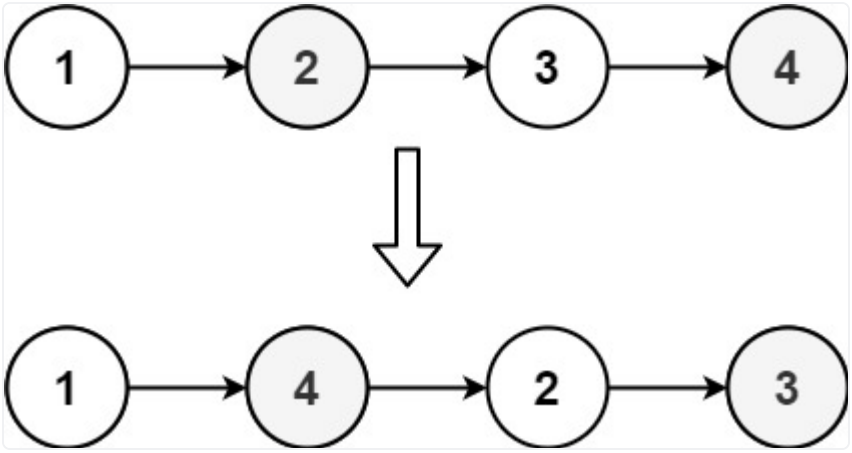
```
L0 → L1 → … → Ln - 1 → Ln
```

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`

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`

Problem Explanation

The task is to reorder a singly linked list in a zig-zag pattern: first node, then last node, followed by second node, then second-to-last node, and so on.

Intuition to Solve the Problem

To achieve this reordering:

1. **Find the Middle**: Split the list into two halves.
2. **Reverse the Second Half**: Make merging easier by reversing the latter half.

3. **Merge the Two Halves**: Combine the halves in an alternating pattern.

Approach

Find the Middle of the List:

- Use slow and fast pointers to find the middle. Slow moves one step, fast moves two steps. When fast reaches the end, slow is at the middle.

Split the List:

- Divide the list into two halves at the middle point.

Reverse the Second Half:

- Flip the order of the second half.

Merge the Two Halves:

- Alternately merge nodes from each half.

Code

```
class Solution {
    public void reorderList(ListNode head) {
        if (head == null || head.next == null) {
            return;
        }

        // Step 1: Find the middle
        ListNode slow = head;
        ListNode fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }

        // Step 2: Split the list
```

```
        ListNode second = slow.next;
        slow.next = null;

        // Step 3: Reverse the second half
        second = reverse(second);

        // Step 4: Merge the halves
        ListNode first = head;
        while (second != null) {
            ListNode tmp1 = first.next;
            ListNode tmp2 = second.next;
            first.next = second;
            second.next = tmp1;
            first = tmp1;
            second = tmp2;
        }
    }

    private ListNode reverse(ListNode head) {
        ListNode prev = null;
        ListNode curr = head;
        while (curr != null) {
            ListNode next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
}
```

Code Explanation

- **Finding the Middle**: Slow and fast pointers locate the middle.
- **Splitting the List**: The list is divided into two at the middle.
- **Reversing the Second Half**: The second half is reversed for merging.
- **Merging**: Alternate merging of the two halves using temporary pointers.

Time Complexity

- **Finding the Middle**: O(n)

  Reversing the Second Half: O(n/2) = O(n)

- **Merging Two Halves**: O(n)

Overall time complexity is O(n), where n is the number of nodes.

Space Complexity

The space complexity is O(1) as it uses constant extra space.

Dry Runs

Test Case 1

Input: `1 -> 2 -> 3 -> 4`

- Middle found at node 2.
- Split into `1 -> 2` and `3 -> 4`.
- Reverse second half to `4 -> 3`.
- Merged list: `1 -> 4 -> 2 -> 3`.

Test Case 2

Input: `1 -> 2 -> 3 -> 4 -> 5`

- Middle found at node 3.
- Split into `1 -> 2 -> 3` and `4 -> 5`.
- Reverse second half to `5 -> 4`.
- Merged list: `1 -> 5 -> 2 -> 4 -> 3`.

Test Case 3

Input: `1 -> 2`

- Middle found at node 1.
- Split into `1` and `2`.
- Reverse second half to `2`.
- Merged list: `1 -> 2`.

Thank for Reading

Subscribe to My Youtube Channel Azan's Coding Corner

Follow Me on Linkedin Azan Imtiaz