Merge Two Sorted List(Day 21)

Prepared By Azan Imtiaz

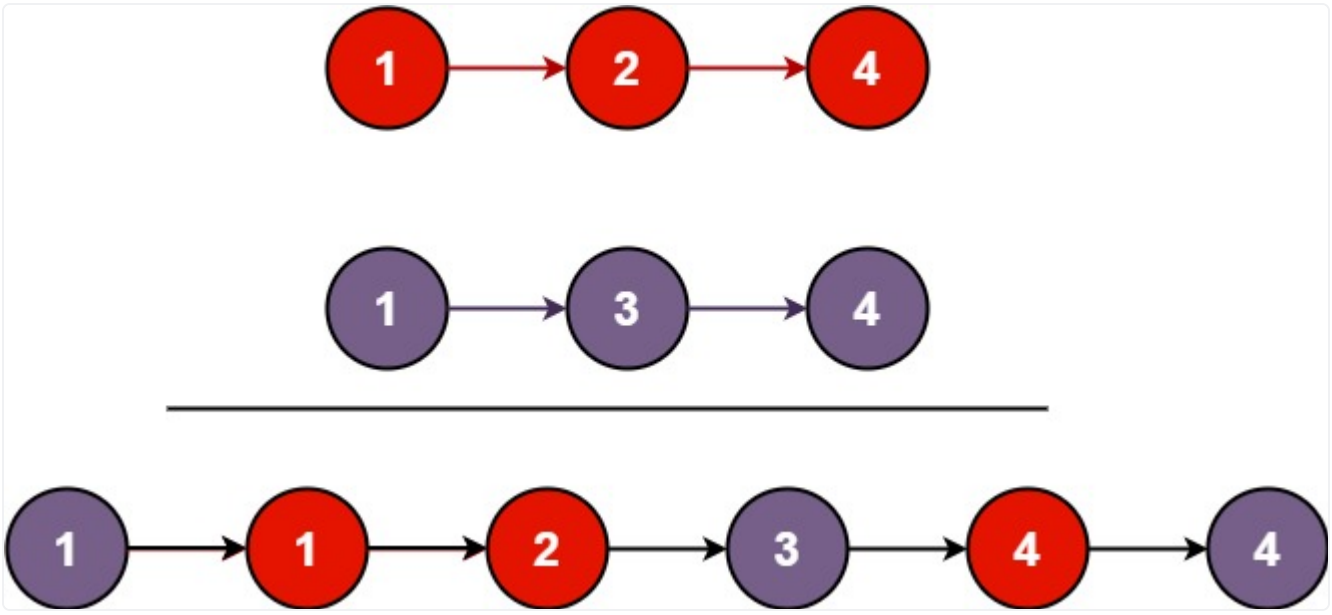
Merge Two Sorted List(Leatcode)

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

Merge the two lists into 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.

Problem

We need to merge two sorted linked lists into a single, sorted linked list.

Intuition

The idea is to compare the nodes of both lists one by one and link them together in a new list based on which node has the smaller value.

Approach

1. Create a dummy node with a value less than any possible list node value.
2. Compare the current nodes of both lists.
3. Link the smaller node to the new list and move to its next node.
4. Repeat until one list is exhausted.
5. Attach the remaining part of the non-exhausted list to the new list.
6. Return the merged list starting from the next node of the dummy.

Code

```
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode newNode = new ListNode(Integer.MIN_VALUE);
        ListNode head = newNode;

        while (list1 != null && list2 != null) {
            if (list1.val <= list2.val) {
                newNode.next = list1;
```

```
                list1 = list1.next;
            } else {
                newNode.next = list2;
                list2 = list2.next;
            }
            newNode = newNode.next;
        }

        if (list1 == null) {
            newNode.next = list2;
        } else {
            newNode.next = list1;
        }

        return head.next;
    }
}
```

Time and Space Complexity

- **Time Complexity:** O(n + m), where n and m are the lengths of the two lists. Each element is looked at once.
- **Space Complexity:** O(1), as we only use a few pointers regardless of the input size.

Dry Run

Test Case 1:

- `list1`: 1 -> 2 -> 4
- `list2`: 1 -> 3 -> 4

Calculation:

- Compare 1 and 1, tie, choose first list's 1.
- Compare 2 and 1, choose second list's 1.
- Compare 2 and 3, choose first list's 2.
- Compare 4 and 3, choose second list's 3.
- Compare 4 and 4, tie, choose first list's 4.
- Second list's 4 is left, attach it to the end.

Result:

- Merged List: 1 -> 1 -> 2 -> 3 -> 4 -> 4

Test Case 2:

- `list1`: empty
- `list2`: 0

Calculation:

- `list1` is empty, so attach `list2` to the dummy node.

Result:

- Merged List: 0

Test Case 3:

- `list1`: 2 -> 4 -> 6
- `list2`: 1 -> 3 -> 5

Calculation:

- Compare 2 and 1, choose second list's 1.
- Compare 2 and 3, choose first list's 2.
- Compare 4 and 3, choose second list's 3.
- Compare 4 and 5, choose first list's 4.
- Compare 6 and 5, choose second list's 5.
- First list's 6 is left, attach it to the end.

Result:

- Merged List: 1 -> 2 -> 3 -> 4 -> 5 -> 6