21. Merged Two Sorted Lists

Nov. 30, 2017 | 188.2K views

Merge two sorted linked lists and return it as a new **sorted** list. The new list should be made by splicing together the nodes of the first two lists.

Example:

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

Approach 1: Recursion

Intuition

We can recursively define the result of a `merge` operation on two lists as the following (avoiding the corner case logic surrounding empty lists):

$$\begin{cases} list1[0] + merge(list1[1:], list2) & list1[0] < list2[0] \\ list2[0] + merge(list1, list2[1:]) & otherwise \end{cases}$$

Namely, the smaller of the two lists' heads plus the result of a `merge` on the rest of the elements.

Algorithm

We model the above recurrence directly, first accounting for edge cases. Specifically, if either of `l1` or `l2` is initially `null`, there is no merge to perform, so we simply return the non-`null` list. Otherwise, we determine which of `l1` and `l2` has a smaller head, and recursively set the `next` value for that head to the next merge result. Given that both lists are `null`-terminated, the recursion will eventually terminate.

Java | Python3 — Copy

```python
class Solution:
    def mergeTwoLists(self, l1, l2):
        if l1 is None:
            return l2
        elif l2 is None:
            return l1
        elif l1.val < l2.val:
            l1.next = self.mergeTwoLists(l1.next, l2)
            return l1
        else:
            l2.next = self.mergeTwoLists(l1, l2.next)
            return l2
```

Complexity Analysis

- Time complexity : $O(n+m)$

  Because each recursive call increments the pointer to `l1` or `l2` by one (approaching the dangling `null` at the end of each list), there will be exactly one call to `mergeTwoLists` per element in each list. Therefore, the time complexity is linear in the combined size of the lists.

- Space complexity : $O(n+m)$

  The first call to `mergeTwoLists` does not return until the ends of both `l1` and `l2` have been reached, so $n+m$ stack frames consume $O(n+m)$ space.

Approach 2: Iteration

Intuition

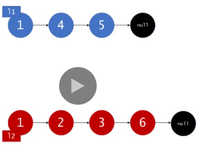
We can achieve the same idea via iteration by assuming that `l1` is entirely less than `l2` and processing the elements one-by-one, inserting elements of `l2` in the necessary places in `l1`.

Algorithm

First, we set up a false "`prehead`" node that allows us to easily return the head of the merged list later. We also maintain a `prev` pointer, which points to the current node for which we are considering adjusting its `next` pointer. Then, we do the following until at least one of `l1` and `l2` points to `null`: if the value at `l1` is less than or equal to the value at `l2`, then we connect `l1` to the previous node and increment `l1`. Otherwise, we do the same, but for `l2`. Then, regardless of which list we connected, we increment `prev` to keep it one step behind one of our list heads.

After the loop terminates, at most one of `l1` and `l2` is non-`null`. Therefore (because the input lists were in sorted order), if either list is non-`null`, it contains only elements greater than all of the previously-merged elements. This means that we can simply connect the non-`null` list to the merged list and return it.

To see this in action on an example, check out the animation below:



1 / 35

Java | Python3 — Copy

```python
class Solution:
    def mergeTwoLists(self, l1, l2):
        # maintain an unchanging reference to node ahead of the return node.
        prehead = ListNode(-1)

        prev = prehead
        while l1 and l2:
            if l1.val <= l2.val:
                prev.next = l1
                l1 = l1.next
            else:
                prev.next = l2
                l2 = l2.next
            prev = prev.next

        # exactly one of l1 and l2 can be non-null at this point, so connect
        # the non-null list to the end of the merged list.
        prev.next = l1 if l1 is not None else l2

        return prehead.next
```

Complexity Analysis

- Time complexity : $O(n+m)$

  Because exactly one of `l1` and `l2` is incremented on each loop iteration, the `while` loop runs for a number of iterations equal to the sum of the lengths of the two lists. All other work is constant, so the overall complexity is linear.

- Space complexity : $O(1)$

  The iterative approach only allocates a few pointers, so it has a constant overall memory footprint.

Rate this article:

Previous Next

Comments: 19

Sort By

Type comment here... (Markdown is supported)

Preview Post

sfdye ★ 860 ⏲ September 1, 2018 9:58 PM

For Approach 2 Python 3 solutions, the second last line can be just `prev.next = l1 or l2`

53 Share Reply

SHOW 1 REPLY

azhukov67 ★ 39 ⏲ February 12, 2019 10:30 PM — Report

The problem requires to return "new list": *"... and return it as a new list. The new list should be ..."*

Both Approaches modify input lists!

39 Share Reply

SHOW 1 REPLY

Hai_dee ★ 1169 ⏲ August 11, 2018 2:45 PM

I'm being somewhat pedantic, however, this comment is somewhat incorrect.

```
# exactly one of l1 and l2 can be non-null at this point, so connect
```

Read More

20 Share Reply

SHOW 2 REPLIES

shaikarshad29 ★ 7 ⏲ March 14, 2019 12:45 PM — Report

isn't the space complexity of approach 2 O(M+N), since we are creating a new linked list of that size?

7 Share Reply

SHOW 4 REPLIES

lienne ★ 10 ⏲ March 23, 2020 5:58 AM

I'm copy-pasting my answer to another question here bc I feel like it's something that could be really useful to people solving this in Java:

Question:
In the iterative solution:

Read More

4 Share Reply

SHOW 2 REPLIES

roireshef ★ 39 ⏲ April 16, 2020 9:46 PM — Report

cleaner:

```
    # exactly one of l1 and l2 can be non-null at this point, so connect
    # the non-null list to the end of the merged list.
    prev.next = l1 or l2
```

Read More

3 Share Reply

iamsdhar ★ 8 ⏲ August 23, 2019 12:00 AM

I don't understand the recursive solution. Why do we use l1.next and l2.next? Does that not modify the existing lists?

3 Share Reply

SHOW 1 REPLY

henghaaya ★ 69 ⏲ April 20, 2019 4:18 AM

for the time complexity of first approach, more accurate, is this should be Max(m, n)???

2 Share Reply

SHOW 2 REPLIES

magedhelmy ★ 3 ⏲ June 1, 2020 6:49 PM

Python 3 solution here:

https://youtu.be/XWnA81_d3O4

Read More

1 Share Reply

al1230 ★ 1 ⏲ March 5, 2020 5:26 AM — Report

What is the point of prev = prev.next? I know the code doesn't work without it, but I can't understand why. Why isn't prev.next enough in the if/else blocks?

1 Share Reply

SHOW 1 REPLY