143. Reorder List

April 13, 2020 | 13.2K views

Given a singly linked list L: $L_0 \rightarrow L_1 \rightarrow \ldots \rightarrow L_{n-1} \rightarrow L_n$,
reorder it to: $L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \rightarrow \ldots$

You may **not** modify the values in the list's nodes, only nodes itself may be changed.

Example 1:

```
Given 1->2->3->4, reorder it to 1->4->2->3.
```

Example 2:

```
Given 1->2->3->4->5, reorder it to 1->5->2->4->3.
```

Solution

Solution Bricks

This problem is a combination of these three easy problems:

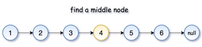
- Middle of the Linked List.
- Reverse Linked List.
- Merge Two Sorted Lists.

Approach 1: Reverse the Second Part of the List and Merge Two Sorted Lists

Overview

- Find a middle node of the linked list. If there are two middle nodes, return the second middle node. Example: for the list `1->2->3->4->5->6`, the middle element is `4`.
- Once a middle node has been found, reverse the second part of the list. Example: convert `1->2->3->4->5->6` into `1->2->3->4` and `6->5->4`.
- Now merge the two sorted lists. Example: merge `1->2->3->4` and `6->5->4` into `1->6->2->5->3->4`.

find a middle node



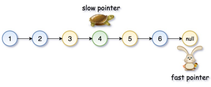
reverse the second part of the list

merge two sorted lists

Now let's check each algorithm part in more detail.

Find a Middle Node

Let's use two pointers, `slow` and `fast`. While the slow pointer moves one step forward `slow = slow.next`, the fast pointer moves two steps forward `fast = fast.next.next`, i.e. `fast` traverses twice as fast as `slow`. When the fast pointer reaches the end of the list, the slow pointer should be in the middle.



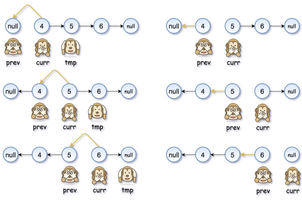
Java | Python3 — Copy

```
# find the middle of linked list [Problem 876]
# in 1->2->3->4->5->6 find 4
slow = fast = head
while fast and fast.next:
    slow = slow.next
    fast = fast.next.next
```

Reverse the Second Part of the List

Let's traverse the list starting from the middle node `slow` and its virtual predecessor `None`. For each current node, save its neighbours: the previous node `prev` and the next node `tmp = curr.next`.

While you're moving along the list, change the node's next pointer to point to the previous node: `curr.next = prev`, and shift the current node to the right for the next iteration: `prev = curr`, `curr = tmp`.



Java | Python3 — Copy

```
# reverse the second part of the list [Problem 206]
# convert 1->2->3->4->5->6 into 1->2->3->4 and 6->5->4
# reverse the second half in-place
prev, curr = None, slow
while curr:
    tmp = curr.next

    curr.next = prev
    prev = curr
    curr = tmp
```

There is a more elegant way to do it in Python:

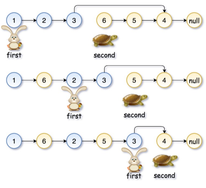
Python3 — Copy

```
# reverse the second part of the list [Problem 206]
# convert 1->2->3->4->5->6 into 1->2->3->4 and 6->5->4
# reverse the second half in-place
prev, curr = None, slow
while curr:
    curr.next, prev, curr = prev, curr, curr.next
```

Merge Two Sorted Lists

This algorithm is similar to the one for list reversal.

Let's pick the first node of each list - first and second, and save their successors. While you're traversing the list, set the first node's next pointer to point to the second node, and the second node's next pointer to point to the successor of the first node. For this iteration the job is done, and for the next iteration move to the previously saved nodes' successors.



Java | Python3 — Copy

```
# merge two sorted linked lists [Problem 21]
# merge 1->2->3->4 and 6->5->4 into 1->6->2->5->3->4
first, second = head, prev
while second.next:
    tmp = first.next
    first.next = second
    first = tmp

    tmp = second.next
    second.next = first
    second = tmp
```

Once again, there is a way to make things simple in Python

Python3 — Copy

```
# merge two sorted linked lists [Problem 21]
# merge 1->2->3->4 and 6->5->4 into 1->6->2->5->3->4
first, second = head, prev
while second.next:
    first.next, first = second, first.next
    second.next, second = first, second.next
```

Implementation

Now it's time to put all the pieces together.

Java | Python3 — Copy

```
class Solution:
    def reorderList(self, head: ListNode) -> None:
        if not head:
            return

        # find the middle of linked list [Problem 876]
        # in 1->2->3->4->5->6 find 4
        slow = fast = head
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next

        # reverse the second part of the list [Problem 206]
        # convert 1->2->3->4->5->6 into 1->2->3->4 and 6->5->4
        # reverse the second half in-place
        prev, curr = None, slow
        while curr:
            curr.next, prev, curr = prev, curr, curr.next

        # merge two sorted linked lists [Problem 21]
        # merge 1->2->3->4 and 6->5->4 into 1->6->2->5->3->4
        first, second = head, prev
        while second.next:
            first.next, first = second, first.next
            second.next, second = first, second.next
```

Complexity Analysis

- Time complexity: $\mathcal{O}(N)$. There are three steps here. To identify the middle node takes $\mathcal{O}(N)$ time. To reverse the second part of the list, one needs $N/2$ operations. The final step, to merge two lists, requires $N/2$ operations as well. In total, that results in $\mathcal{O}(N)$ time complexity.
- Space complexity: $\mathcal{O}(1)$, since we do not allocate any additional data structures.

Comments: 14 Sort By

Type comment here... (Markdown is supported)

Preview Post

mohitgupta1194 ★ 37 May 7, 2020 10:56 PM
Diagrams are cute.
33 Share Reply

strann28 ★ 18 May 3, 2020 12:06 PM
This should be reclassified as hard.
16 Share Reply
SHOW 3 REPLIES

lunarprom ★ 10 May 31, 2020 11:24 AM
"Approach 1: blablabla."
Sounds like there is an actual approach 2.