Solution: Reorder List

Let's solve the Reorder List problem using the In-place Reversal of a Linked List pattern.

We'll cover the following

Statement

Given the head of a singly linked list, reorder the list as if it were folded on itself. For example, if the list is represented as follows:

$L_0 \rightarrow L_1 \rightarrow L_2 \rightarrow ... \rightarrow L_{n-2} \rightarrow L_{n-1} \rightarrow L_n$

This is how you'll reorder it:

$L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \rightarrow ...$

You don't need to modify the values in the list's nodes; only the links between nodes need to be changed.

Constraints:

- The range of number of nodes in the list is $[1, 500]$.
- $-5000 \leq$ `Node.value` ≤ 5000

Solution

So far, you've probably brainstormed some approaches and have an idea of how to solve this problem. Let's explore some of these approaches and figure out which one to follow based on considerations such as time complexity and any implementation constraints.

Naive approach

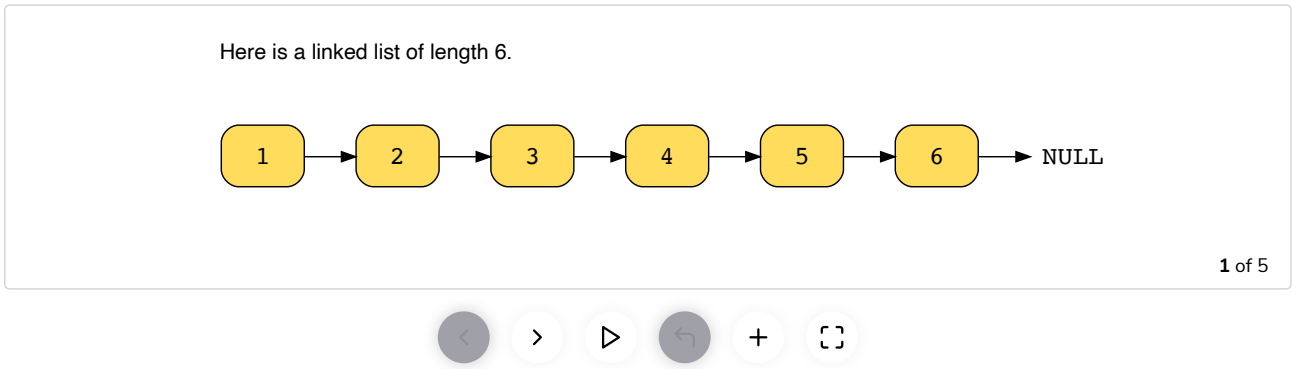
The naive approach for this problem is to traverse the complete linked list twice. First, we need to start from the first node using the current pointer. To find the last node, we traverse the complete linked list and add the last node in front of the current node. After adding the last node, the current node will move to the next node. Each time the last node is added, the current node will move ahead in the list. For this reason we need to find the last node. To do that, we'll need to find the last occurring nodes n times, which will take $O(n)$ time complexity to find the last node each time. We'll end the program when both current and last nodes become equal.

The total time complexity for this solution is $O(n^2)$, which is way too much. However, the space complexity of this naive approach is $O(1)$ as we used the constant extra space.

Optimized approach using in-place reversal of a linked list

We need to perform the in-place reversal for the second half of the linked list, which will allow us to not use any extra memory. To solve the original problem, we can use the modified in-place methodology after the first in-place reversal to merge the two halves of the linked list. We can use two-pointers, $first$ and $second$, to point to the heads of the two halves of the linked list. We'll traverse both halves in lockstep to merge the two linked lists, interleaving the nodes from the second half into the first half.

The slides below illustrate how we want the algorithm to run:



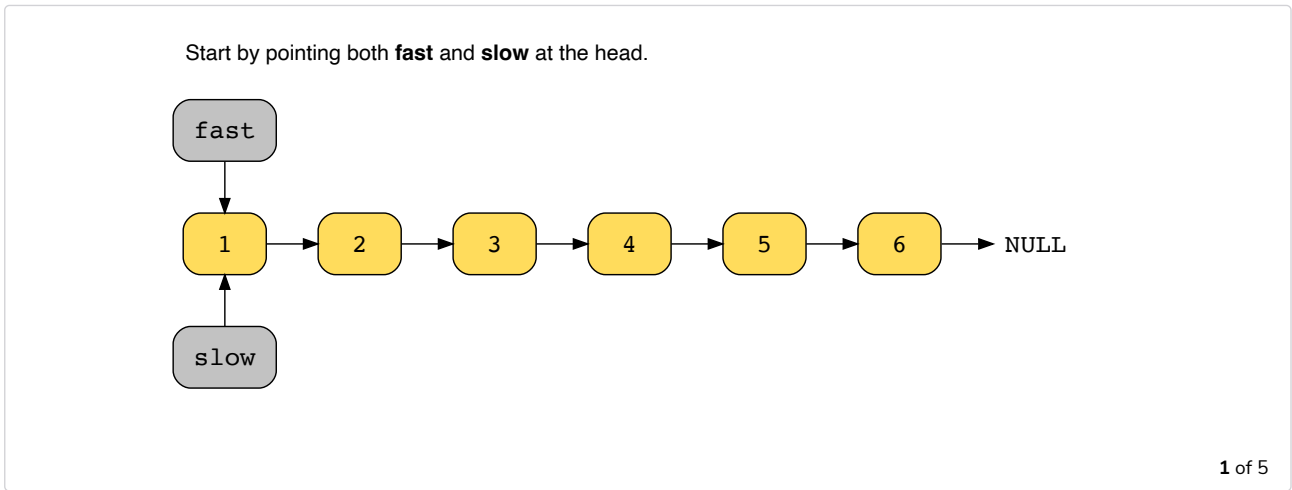
Let's go over the steps in detail:

Find the middle node:

We use two pointers to find the middle node, `slow` and `fast`.

- The `slow` pointer moves one step at a time: `slow = slow.next`.
- The `fast` pointer moves two steps at a time: `fast = fast.next.next`.

This way when the `fast` pointer reaches the end of the `LinkedList`, the `slow` pointer points at the middle node.

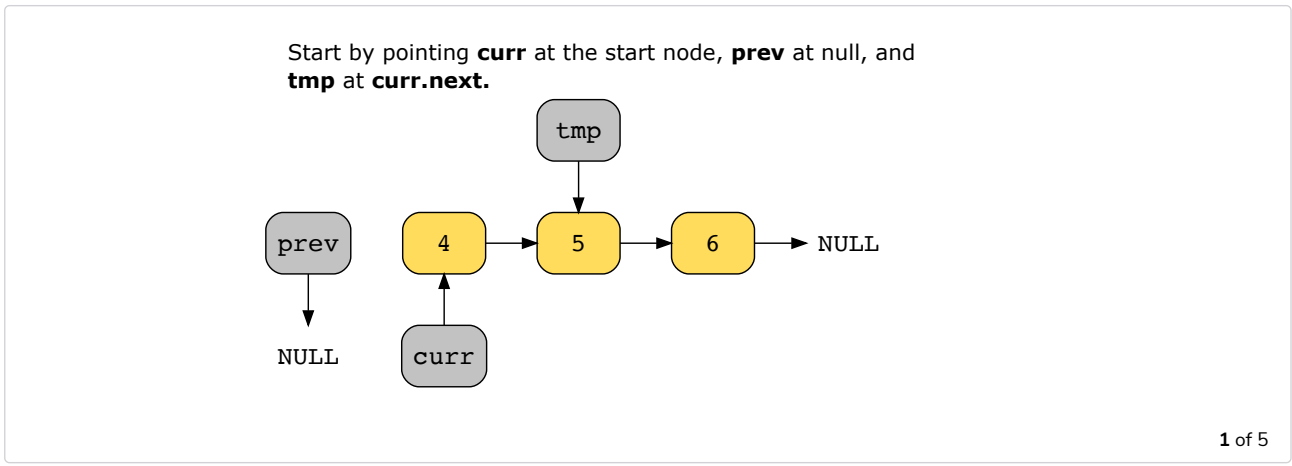


Reverse the second part of the list:

1. We traverse the list from the middle node till the end of the linked list using the `slow` pointer.

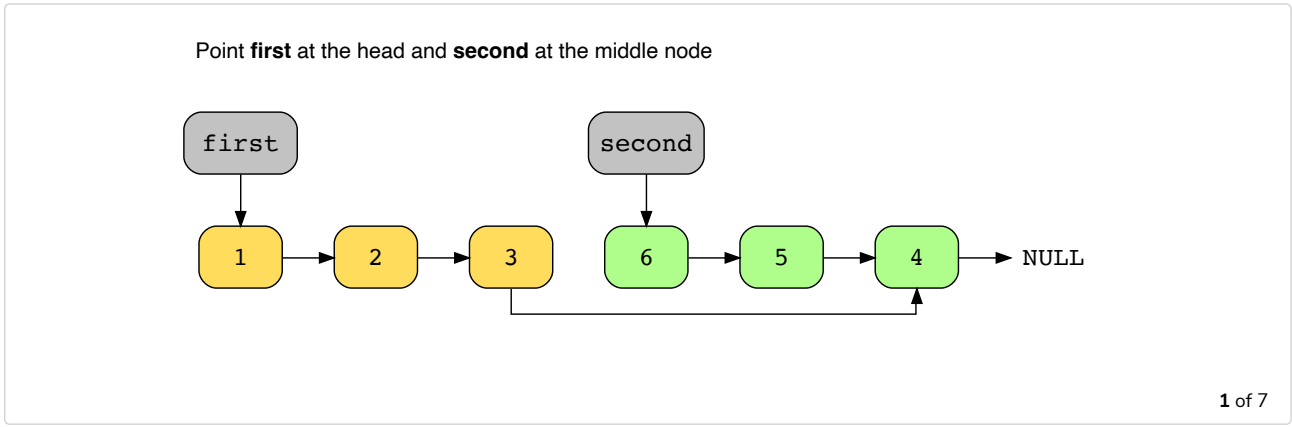
2. We save the neighbors of each node.
3. The `prev` pointer points at the previous node and `tmp` points at `curr.next`.

While moving through the linked list, we point `curr.next` at `prev`. Then shift the current node to the right for the next iteration, `prev = curr` and `curr = tmp`.



Merge lists:

1. We start by taking the first node of the first and second half of the linked list. Let's call them `first` and `second` respectively. We also save their successors.
2. While traversing the list, we set `first.next = second`, and point `second.next` at the successor of the `first` node. The first iteration is completed here.
3. Now we move to the successors of the previously saved nodes for the next iteration.



Let's look at the solution code:

main.java

PrintList.java

LinkedList.java

LinkedListNode.java

```
1  class Reorder {
2      public static LinkedListNode reorderList(LinkedListNode head)
```

```
3     {
4         if(head == null)
5             return head;
6         // find the middle of linked list
7         // in 1->2->3->4->5->6 find 4
8         LinkedListNode slow = head;
9         LinkedListNode fast = head;
10        while(fast!= null && fast.next != null)
11        {
12            slow = slow.next;
13            fast = fast.next.next;
```

```
16        // convert 1->2->3->4->5->6 into 1->2->3 and 6->5->4
17        // reverse the second half in-place
18        LinkedListNode prev = null;
19        LinkedListNode curr = slow;
20        LinkedListNode next = null;
21        while(curr != null)
22        {
23            next = curr.next;
24            curr.next = prev;
25            prev = curr;
26            curr = next;
27        }
28        // merge two sorted linked lists
```

Reorder List

Solution summary

To recap, the solution to this problem can be divided into the following three parts:

1. Find the middle node. If there are two middle nodes, then choose the second node.
2. Reverse the second half of the linked list.
3. Merge both halves of the linked lists alternatively.

Time complexity

The time complexity of this solution is linear, $O(n)$, where n is the number of nodes in a linked list.

Space complexity

The space complexity of the solution is constant, $O(1)$.

Back

Reorder List

Next

Swapping Nodes in a ...

Mark as Completed