Solution: Swapping Nodes in a Linked List

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

We'll cover the following

Statement

Given the linked list and an integer, `k`, return the head of the linked list after swapping the values of the k^{th} node from the beginning and the k^{th} node from the end of the linked list.

> **Note:** We'll number the nodes of the linked list starting from 1 to n.

Constraints:

- The linked list will have `n` number of nodes.
- $1 \leq$ `k` $\leq$ `n` ≤ 500
- $-5000 \leq$ `Node.value` ≤ 5000

Pattern: In-place Reversal of a Linked List

We need to find the k^{th} node from the start of the linked list and the k^{th} node from the end of the linked list. We find the two nodes in the linked list using the in-place reversal method. We use two pointers to traverse the linked list to find the k^{th} node from the start and the k^{th} node from the end of the linked list.

Once we've found these nodes, we swap their values without changing their positions.

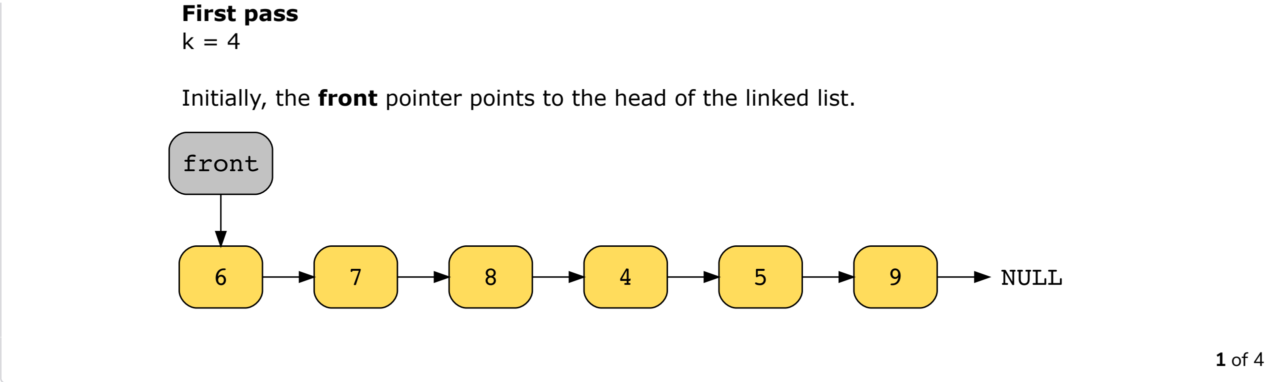
Solution

Let's look at the different approaches to solving the problem.

The three-pass approach:

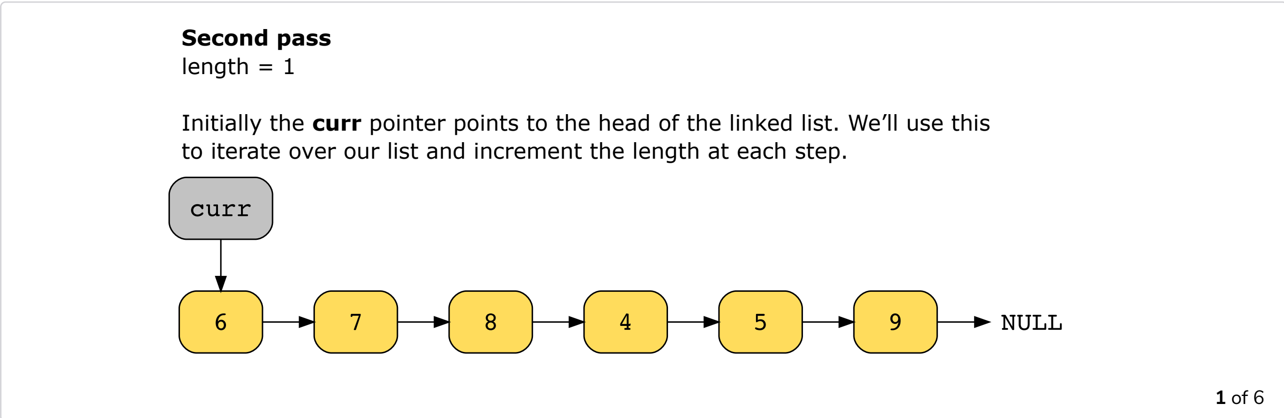
Let's use two pointers, `front` and `end`, to help traverse a linked list and find the k^{th} node at the start and end of the linked list:

- **First pass:** To find the k^{th} node at the start of the linked list, we traverse the linked list from the head to the k^{th} node using the `front` pointer.

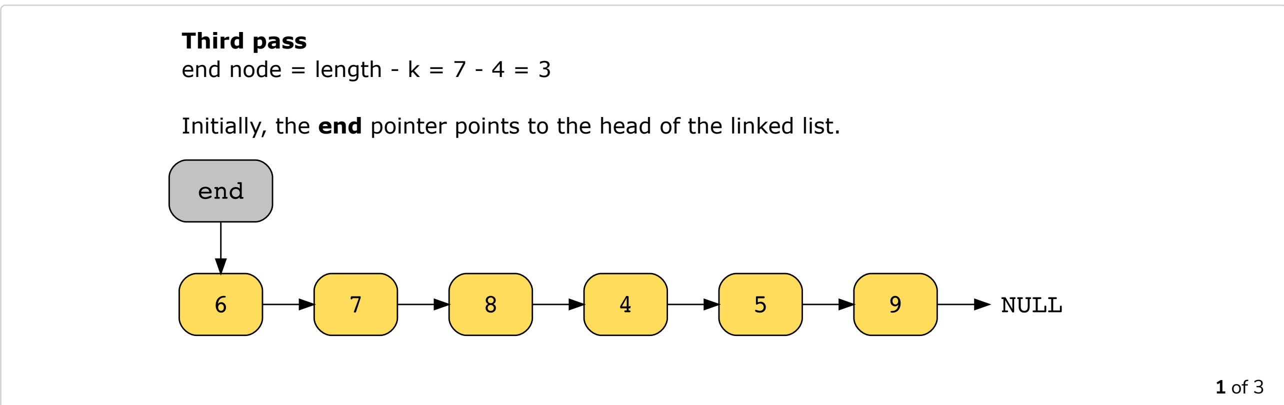


To find the k^{th} node at the end of the linked list, we need to traverse the linked list two times.

- **Second pass:** We must find the length of the linked list to find the exact position of the end node. We traverse the linked list from the head to the last node to find the length of the linked list.



- **Third pass:** The k^{th} node from the end will be located at the $(length - k)^{th}$ position from the start. Therefore, we traverse the linked list again from the head to the $(length - k)^{th}$ node to find the k^{th} last node.



After finding the `front` and `end` nodes, we can swap the values of the nodes.

The two-pass approach:

We can optimize the approach above by finding the `front` node (the k^{th} node from the start) and the length of the linked list in a single pass, and then finding the `end` node (the k^{th} node from the end) in another pass.

Now, let’s try to solve the problem in a single pass.

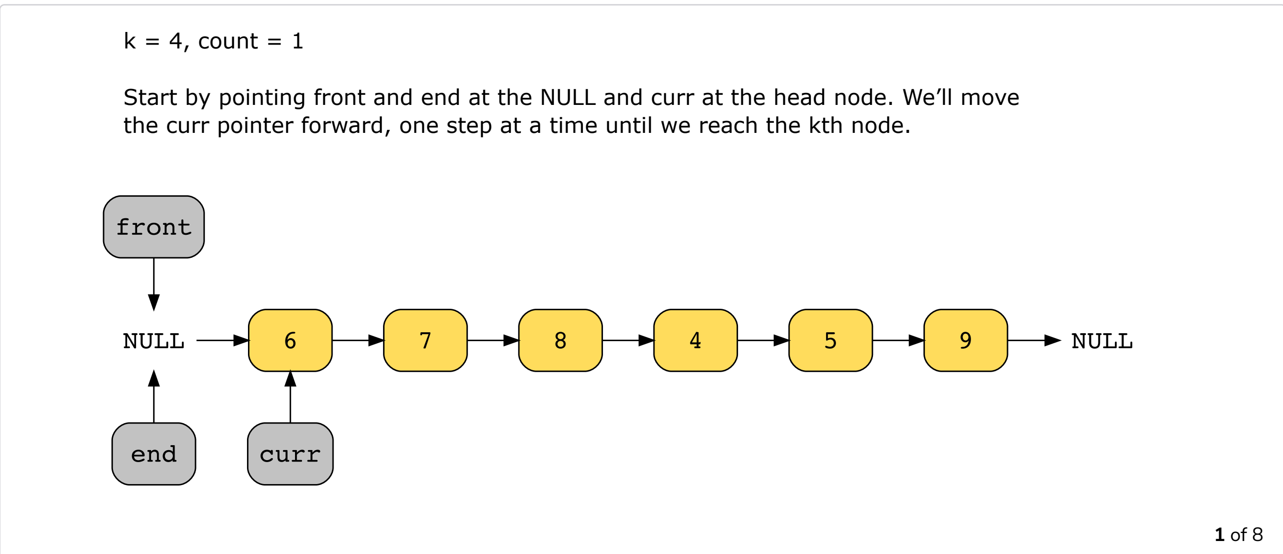
The one-pass approach:

In the two-pass approach, before finding the `end` node (the k^{th} node from the end), we first need to find the length of the linked list by traversing the complete list. We can optimize this by finding the `end` node without calculating the length of the linked list. We traverse the linked list using two pointers, `end` and `curr`, by keeping the `end` pointer `k` positions behind the `curr` pointer. When the `curr` pointer reaches the last node, the `end` pointer is at the k^{th} last node.

Let’s look at the algorithm of the approach discussed above:

- Initialize the `count` variable with 0.
- Set the `front` and `end` pointers to NULL and the `curr` pointer to the head node.
- Traverse the linked list using the `curr` pointer and increment `count` on every step.
- When `count` becomes equal to `k`, it means that we’ve reached k^{th} node from the start. At this point, we perform the following two steps:
    - We set the `front` pointer to `curr` node.
    - We set the `end` pointer to the head node. After doing this, the `end` node is exactly k nodes behind the `curr` node.
- We continue traversing the linked list, but along with the `curr` pointer, we move the `end` pointer too.
- When `curr` reaches the end of the linked list, the `end` pointer will be pointing to the k^{th} node from the end of the linked list.
- We swap the values of the `front` and `end` nodes and return the head of the linked list.

Let’s look at the following slides to get a better understanding of the steps:



Let’s implement the algorithm as discussed above:

Java

main.java

LinkedList.java

PrintList.java

LinkedListNode.java

```java
class SwapNodes {
    public static void swap(LinkedListNode node1, LinkedListNode node2) {
        int temp = node1.data;
        node1.data = node2.data;
        node2.data = temp;
    }
    public static LinkedListNode swapNodes(LinkedListNode head, int k) {
        if (head == null) {
            return head;
        }
        int count = 0;

        // front and end pointers will be used to track the kth node from
        // the start and end of the linked list, respectively
        LinkedListNode front = null;
        LinkedListNode end = null;
        LinkedListNode curr = head;

        while (curr != null) {
```

```java
            // kth node from the end of the linked list
            if (end != null) {
                end = end.next;
            }
            // if the count has become equal to k, it means the curr is
```

Swapping Nodes in a Linked List

Solution summary

Let's summarize the steps we performed to solve the problem:

- Initialize a `count` variable with 0, the `front` and `end` pointers with NULL, and point the `curr` pointer to the head of the linked list.
- Iterate the linked list using `curr`, and increment the `count` variable at each step.
- When `count` becomes equal to k, set `front` equal to the `curr` pointer and move the `end` pointer to the head.
- Continue moving the `end` and `curr` pointers forward until `curr` reaches the last node.
- Swap the values of these `front` and `end` nodes.

Time complexity

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

Space complexity

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

← Back

Next →

Mark as Completed