Previous Next

24. Swap Nodes in Pairs

Dec. 1, 2019 | 23.6K views

Average Rating: 4.74 (19 votes)

Given a linked list, swap every two adjacent nodes and return its head.

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

Example:

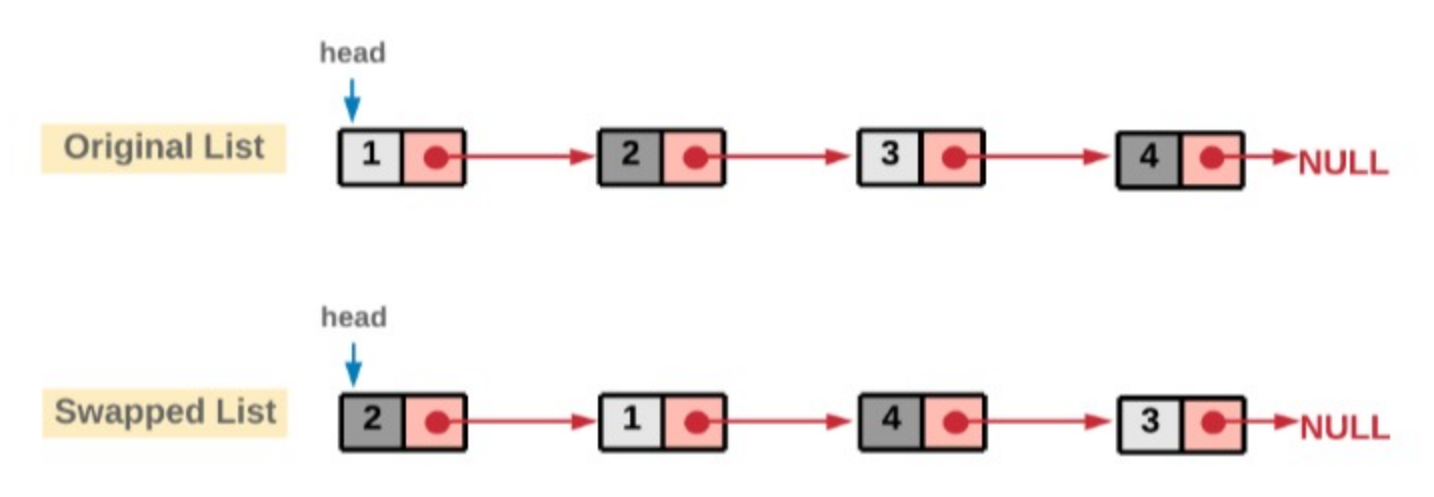
```
Given 1->2->3->4, you should return the list as 2->1->4->3.
```

Solution

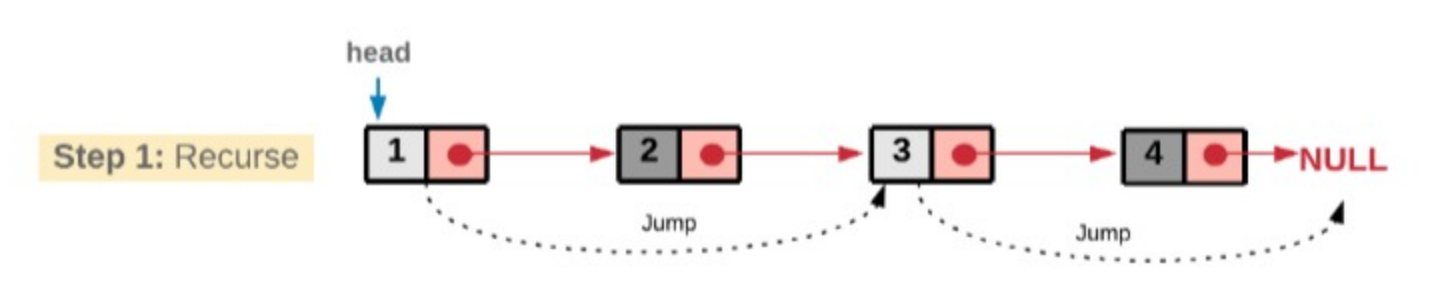
Approach 1: Recursive Approach

Intuition

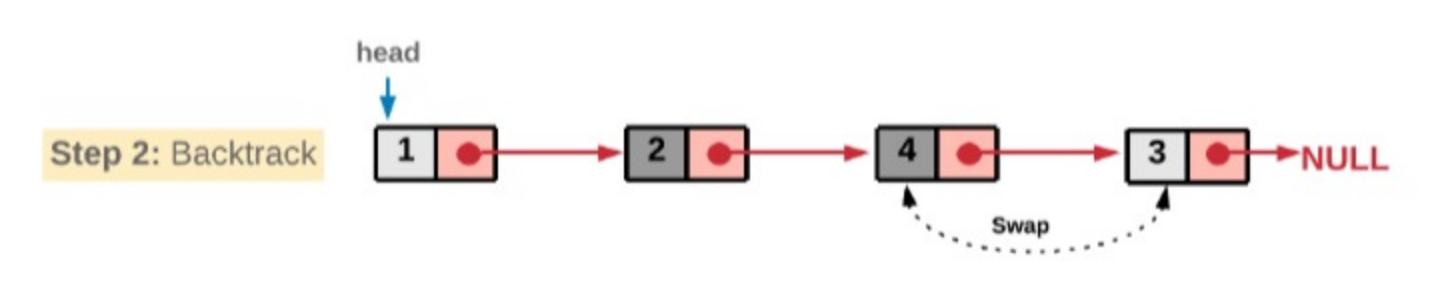
The problem doesn't ask for entire reversal of linked list. It's rather asking us to swap every two adjacent nodes of a linked list starting at the very first node.



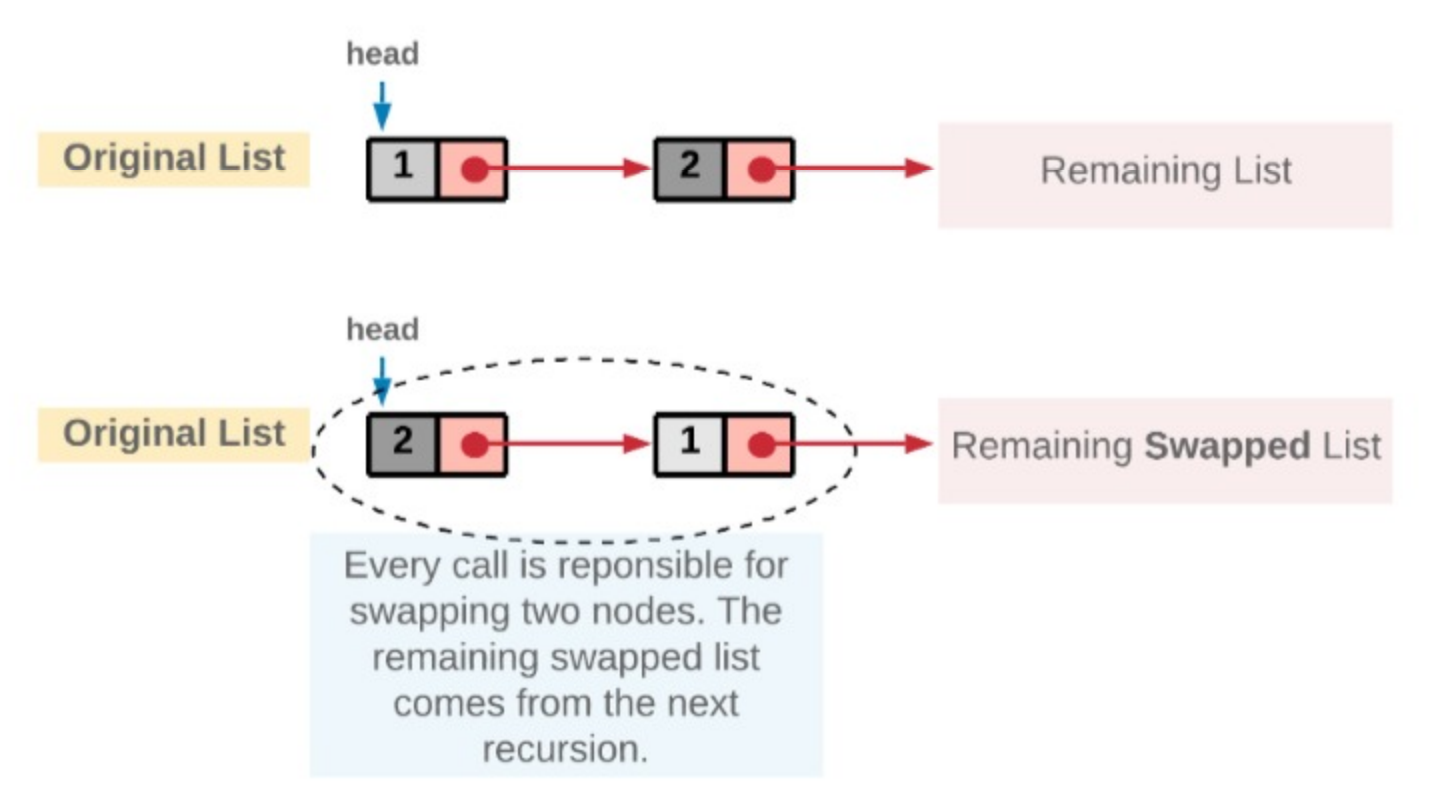
The basic intuition is to reach to the end of the linked list in steps of two using recursion.



and while back tracking the nodes can be swapped.

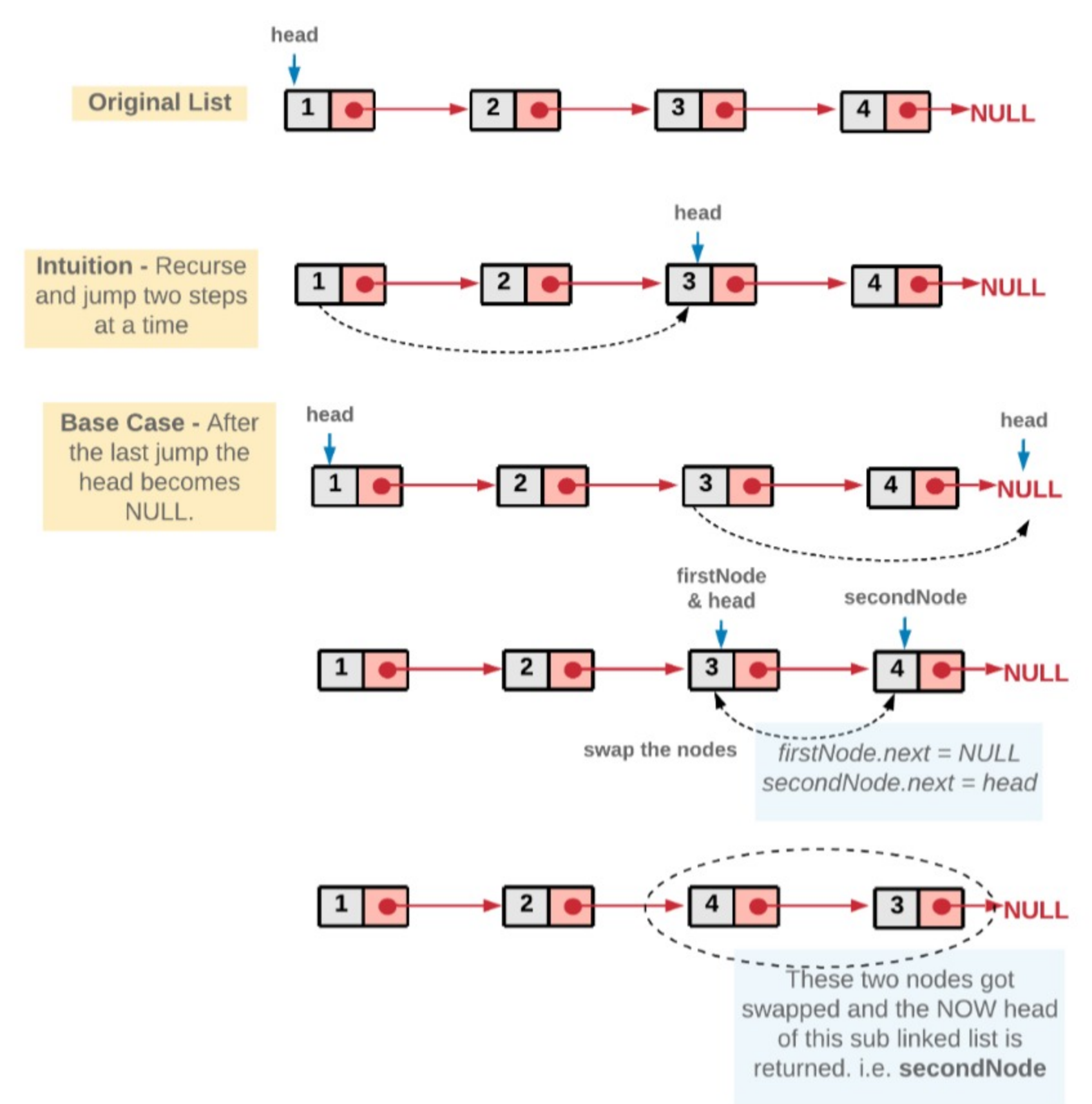


In every function call we take out two nodes which would be swapped and the remaining nodes are passed to the next recursive call. The reason we are adopting a recursive approach here is because a sub-list of the original list would still be a linked list and hence, it would adapt to our recursive strategy. Assuming the recursion would return the swapped `remaining` list of nodes, we just swap the current two nodes and attach the remaining list we get from recursion to these two swapped pairs.



Algorithm

1. Start the recursion with `head` node of the original linked list.
2. Every recursion call is responsible for swapping a pair of nodes. Let's represent the two nodes to be swapped by `firstNode` and `secondNode`.
3. Next recursion is made by calling the function with head of the next pair of nodes. This call would swap the next two nodes and make further recursive calls if there are nodes left in the linked list.
4. Once we get the pointer to the remaining swapped list from the recursion call, we can swap the `firstNode` and `secondNode` i.e. the nodes in the current recursive call and then return the pointer to the `secondNode` since it will be the new head after swapping.



5. Once all the pairs are swapped in the backtracking step, we would eventually be returning the pointer to the head of the now `swapped` list. This head will essentially be the second node in the original linked list.

Java Python Copy

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None


class Solution(object):
    def swapPairs(self, head: ListNode) -> ListNode:
        """
        :type head: ListNode
        :rtype: ListNode
        """

        # If the list has no node or has only one node left.
        if not head or not head.next:
            return head

        # Nodes to be swapped
        first_node = head
        second_node = head.next

        # Swapping
        first_node.next  = self.swapPairs(second_node.next)
        second_node.next = first_node

        # Now the head is the second node
```

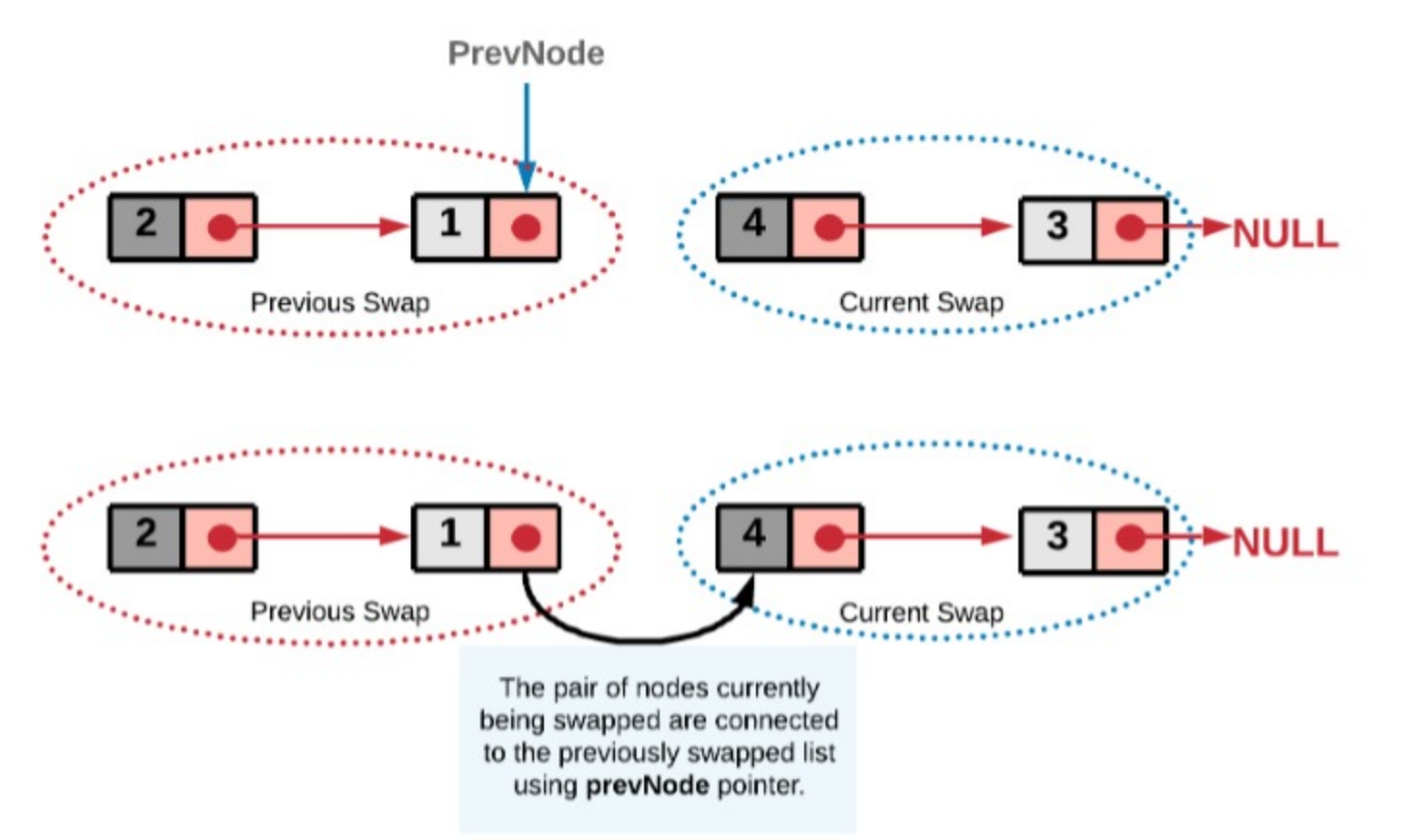
Complexity Analysis

- Time Complexity: $O(N)$ where N is the size of the linked list.
- Space Complexity: $O(N)$ stack space utilized for recursion.

Approach 2: Iterative Approach

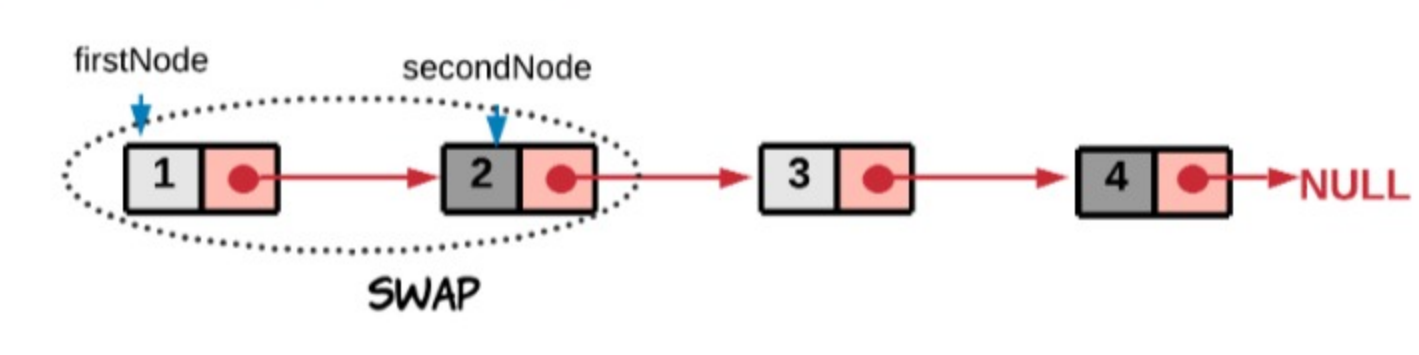
Intuition

The concept here is similar to the recursive approach. We break the linked list into pairs by jumping in steps of two. The only difference is, unlike recursion, we swap the nodes on the go. After swapping a pair of nodes, say `A` and `B`, we need to link the node `B` to the node that was right before `A`. To establish this linkage we save the previous node of node `A` in `prevNode`.



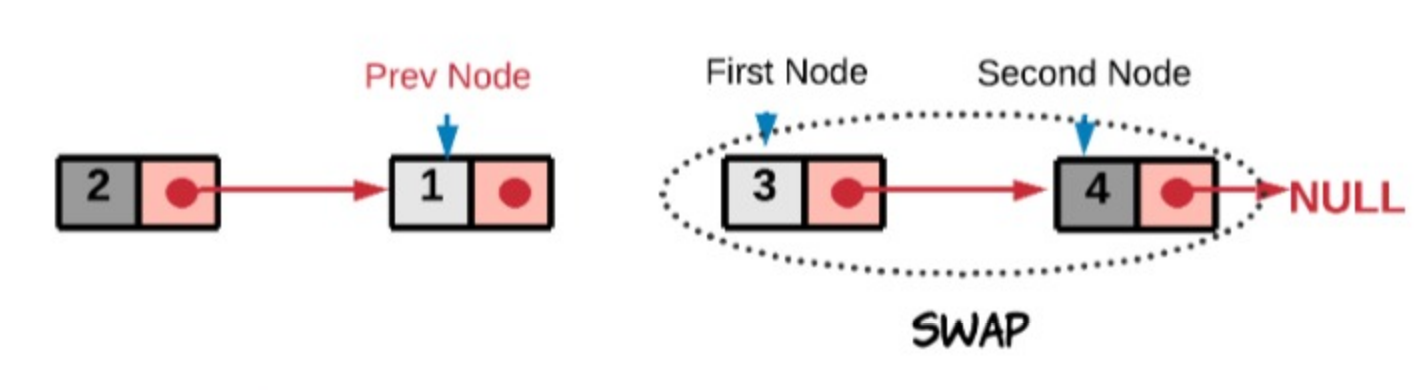
Algorithm

1. We iterate the linked list with jumps in steps of two.
2. Swap the pair of nodes as we go, before we jump to the next pair. Let's represent the two nodes to be swapped by `firstNode` and `secondNode`.



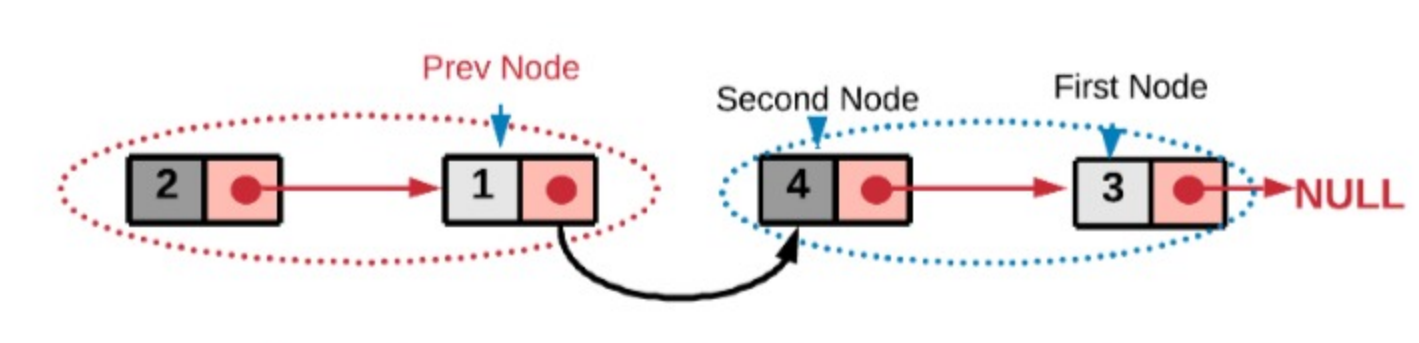
3. Swap the two nodes. The swap step is

```
firstNode.next = secondNode.next
secondNode.next = firstNode
```



4. We also need to assign the `prevNode`'s next to the head of the swapped pair. This step would ensure the currently *swapped* pair is linked correctly to the end of the previously *swapped* list.

```
prevNode.next = secondNode
```



This is an iterative step, so the nodes are swapped on the go and attached to the previously swapped list. And in the end we get the final swapped list.

Java Python Copy

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def swapPairs(self, head: ListNode) -> ListNode:
        """
        :type head: ListNode
        :rtype: ListNode
        """
        # Dummy node acts as the prevNode for the head node
        # of the list and hence stores pointer to the head node.
        dummy = ListNode(-1)
        dummy.next = head

        prev_node = dummy

        while head and head.next:

            # Nodes to be swapped
            first_node = head;
            second_node = head.next;

            # Swapping
            prev_node.next = second_node
```

Complexity Analysis

- Time Complexity : $O(N)$ where N is the size of the linked list.
- Space Complexity : $O(1)$.

Rate this article:

Previous Next