Solution: Reverse Nodes in Even Length Groups

Let's solve the Reverse Nodes in Even Length Groups problem using the In-place Reversal of a Linked List pattern.

We'll cover the following

Statement

You're given a linked list. Your task is to reverse all of the nodes that are present in the groups with an even number of nodes in them. The nodes in the linked list are sequentially assigned to non-empty groups whose lengths form the sequence of the natural numbers $(1, 2, 3, 4...)$. The length of a group is the number of nodes assigned to it. In other words:

- The 1^{st} node is assigned to the first group.
- The 2^{nd} and 3^{rd} nodes are assigned to the second group.
- The 4^{th}, 5^{th}, and 6^{th} nodes are assigned to the third group and so on.

You have to return the head of the modified linked list.

> **Note:** The length of the last group may be less than or equal to 1 + the length of the second to the last group.

Constraints:

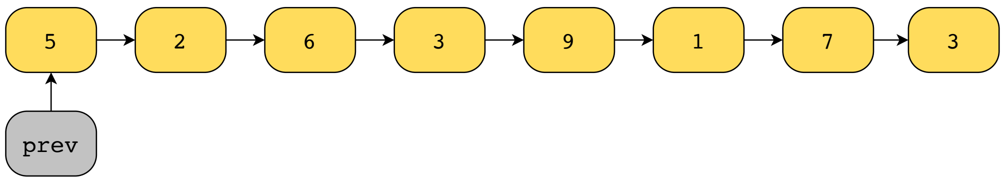
- $1 \leq$ Number of nodes ≤ 500
- $0 \leq$ `LinkedListNode.data` $\leq 10^3$

Solution

We need to reverse each group that has an even number of nodes in it. We can think of each of these groups of nodes as separate linked lists. So, for each of these groups applying an in-place linked list reversal solves the original problem. We need to invoke the in-place reversal of linked list code for every group with an even number of nodes present. However, we'll need to modify the terminating condition of the loop that is used in the in-place reversal of a linked list pattern. We constantly need to check if the current group even requires any sort of reversal or not and if it is the last group do we even need to modify it or not.

The slides below illustrate how we would like the algorithm to run:

We set a **prev** pointer pointing at the last node of the previous group. We set a variable l which indicates the length of the current group. l is initially set to 2 because the very first node can't be reversed as it's contained in the first group of length 1 which is odd. Lastly, we keep a track of the successfully traversed nodes in the current group by the variable n, which is initially set to 0 as no nodes have been currently traversed.



l = 2

n = 0

1 of 24

We can see the code of this solution below:

Java

main.java

PrintList.java

LinkedList.java

LinkedListNode.java

```java
class ReverseNodes {
    public static LinkedListNode reverseEvenLengthGroups(LinkedListNode head)
    {
        LinkedListNode prev = head; // Node immediately before the current gr
        LinkedListNode node, reverse, nodeNext, curr, prevNext = null;
        int l = 2; // The head doesn't need to be reversed since it's a group
        int n = 0;
        while(prev.next!= null)
        {
```

```java
            for (int i = 0; i < l; i ++)
            {
                if(node.next == null)
                    break;
                n += 1;
                node=node.next;
            }
            if(n % 2 != 0) // odd length
               prev = node;
            else
            {
                reverse = node.next;
                curr = prev.next;
                for(int j=0; j < n;j++){
                    nodeNext = curr.next;
                    curr.next = reverse;
```

Reverse Nodes in Even Length Groups

Time complexity

The time complexity will be $O(n)$, since we traverse the n nodes of the linked list.

Space complexity

The space complexity will be $O(1)$, since we use a constant number of additional variables to maintain the proper connections between the nodes during reversal.

Back
Reverse Nodes In Eve...

Next
Swap Nodes in Pairs

✓ Mark as Completed