Solution Review: Problem Challenge 2

We'll cover the following ⌄

- Rearrange a LinkedList (medium)
- Solution
  - Code
- Time Complexity
- Space Complexity

Rearrange a LinkedList (medium)

Given the head of a Singly LinkedList, write a method to modify the LinkedList such that the **nodes from the second half of the LinkedList are inserted alternately to the nodes from the first half in reverse order**. So if the LinkedList has nodes 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null, your method should return 1 -> 6 -> 2 -> 5 -> 3 -> 4 -> null.

Your algorithm should not use any extra space and the input LinkedList should be modified in-place.

Example 1:

```
Input: 2 –> 4 –> 6 –> 8 –> 10 –> 12 –> null
Output: 2 –> 12 –> 4 –> 10 –> 6 –> 8 –> null
```

Example 2:

```
Input: 2 –> 4 –> 6 –> 8 –> 10 –> null
Output: 2 –> 10 –> 4 –> 8 –> 6 –> null
```

Solution

This problem shares similarities with Palindrome LinkedList (https://www.educative.io/collection/page/5668639101419520/5671464854355968/6278770187042816/). To rearrange the given LinkedList we will follow the following steps:

1. We can use the **Fast & Slow pointers** method similar to Middle of the LinkedList (https://www.educative.io/collection/page/5668639101419520/5671464854355968/6033606055034880/) to find the middle node of the LinkedList.

2. Once we have the middle of the LinkedList, we will reverse the second half of the LinkedList.

3. Finally, we'll iterate through the first half and the reversed second half to produce a LinkedList in the required order.

Here is what our algorithm will look like:
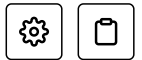
```python
1   from __future__ import print_function
2
3
4   class Node:
5     def __init__(self, value, next=None):
6       self.value = value
7       self.next = next
8
9     def print_list(self):
10      temp = self
11      while temp is not None:
12        print(str(temp.value) + " ", end='')
13        temp = temp.next
14      print()
15
16
17  def reorder(head):
18    if head is None or head.next is None:
19      return
20
21    # find middle of the LinkedList
22    slow, fast = head, head
23    while fast is not None and fast.next is not None:
24      slow = slow.next
25      fast = fast.next.next
26
27    # slow is now pointing to the middle node
28    head_second_half = reverse(slow)  # reverse the second half
```

Time Complexity

The above algorithm will have a time complexity of $O(N)$ where 'N' is the number of nodes in the LinkedList.

Space Complexity

The algorithm runs in constant space $O(1)$.

← Back

Problem Challenge 2

Next →

Problem Challenge 3

✅ Mark as Completed