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. 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 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.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```java

class ListNode {
  int value = 0;
  ListNode next;

  ListNode(int value) {
    this.value = value;
  }
}

class RearrangeList {

  public static void reorder(ListNode head) {
    if (head == null || head.next == null)
      return;

    // find the middle of the LinkedList
    ListNode slow = head, fast = head;
    while (fast != null && fast.next != null) {
      slow = slow.next;
      fast = fast.next.next;
    }

    // slow is now pointing to the middle node
    ListNode headSecondHalf = reverse(slow); // reverse the second half
    ListNode headFirstHalf = head;
    // rearrange to produce the LinkedList in the required order
```

Run Save Reset

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

✓ Completed

Report an Issue Ask a Question