Solution: Middle of the Linked List

Let's solve the Middle of the Linked List problem using the Fast and Slow Pointers pattern.

We'll cover the following

Statement

Given the `head` of a singly linked list, return the middle node of the linked list. If the number of nodes in the linked list is even, there will be two middle nodes, so return the second one.

Constraints:

- `head` $\neq$ NULL

Solution

So far, you have probably brainstormed some approaches and have an idea of how to solve this problem. Let's explore some of these approaches and figure out which one to follow based on considerations such as time complexity and any implementation constraints.

Naive approach

In the naive approach, we use an external array to store the elements of the linked list, and then we return the element present at the index $\lfloor \frac{array.length}{2} \rfloor$ as the middle node of the linked list. The time and space complexity of this approach is $O(n)$, where n is the number of nodes in the linked list. Let's see if we can solve this problem with better time and space complexity.

Optimized approach using fast and slow pointers

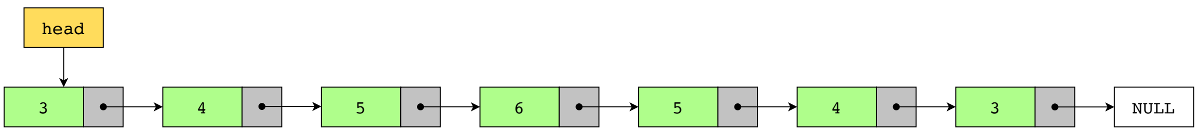
We can use the fast and slow pointers to solve this problem with constant space complexity. The slow pointer traverses the linked list one step at a time, while the fast pointer takes two steps at a time. This makes the fast pointer reach the end of the linked list in $\frac{n}{2}$ iterations, and the slow pointer, by this time, reaches the middle of the linked list.

The following steps are applied when identifying the middle of the linked list:

1. Create two pointers, `slow` and `fast`, initially at the `head` of the linked list, that is, pointing to the first node.
2. Traverse the linked list using both pointers, where the `slow` pointer will move one step forward, and the `fast` pointer will move two steps forward.

3. When the `fast` pointer reaches the last element of the linked list, or becomes equal to NULL, the `slow` pointer, at that time, will point to the middle node. Return the node that the `slow` pointer points to.

The slides below help to understand the solution in a better way.



Given the above linked list, we will find its middle element.

1 of 6

Let's look at the code for this solution below:

Java

main.java

LinkedListNode.java

PrintList.java

LinkedList.java

```
class MiddleNode {

    // Function to find the middle node of the linked list
    public static LinkedListNode middleNode(LinkedListNode head) {

        // Create two pointers, slow and fast ,initially pointing to the heac
        LinkedListNode slow = head;
        LinkedListNode fast = head;

        // Traverse the linked list until fast reaches at the last node or NU
        while (fast != null && fast.next != null) {

            // Move the slow pointer one step ahead
            slow = slow.next;

            // Move the fast pointer two steps ahead
            fast = fast.next.next;
        }
```

```
        return slow;
    }

    // Driver code
    public static void main( String args[] ) {

        int[][] input = {{1, 2, 3, 4, 5}, {1, 2, 3, 4, 5, 6}, {3, 2, 1}, {10]
```

Solution summary

To recap, the solution to this problem can be divided into the following steps:

1. Create two pointers, `slow` and `fast`, initially at the `head` of the linked list.
2. While traversing the linked list, move the `slow` pointer one step forward and the `fast` pointer two steps forward.
3. When the `fast` pointer reaches the last node or NULL, the `slow` pointer will point to the middle node of the linked list. Return the node that the `slow` pointer points to.

Time complexity

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

Space complexity

The space complexity of this solution is constant, that is, $O(1)$.

Back
Middle of the Linked L...

Next
Circular Array Loop

✓ Mark as Completed