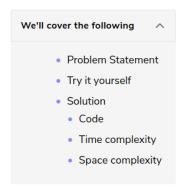




Middle of the LinkedList (easy)



Problem Statement

Given the head of a Singly LinkedList, write a method to return the middle node of the LinkedList.

If the total number of nodes in the LinkedList is even, return the second middle node.

Example 1:

```
Input: 1 -> 2 -> 3 -> 4 -> 5 -> null
Output: 3
```

Example 2:

Example 3:

Try it yourself

Try solving this question here:

```
Python3
🐓 Java
                       Js JS
                                   ⊙ C++
       self.next = next
    def find middle of linked list(head):
     return head
     head = Node(1)
      head.next = Node(2)
      head.next.next = Node(3)
      head.next.next.next = Node(4)
      head.next.next.next = Node(5)
      print("Middle Node: " + str(find middle of linked list(head).value))
      head.next.next.next.next = Node(6)
      print("Middle Node: " + str(find_middle_of_linked_list(head).value))
      head.next.next.next.next.next = Node(7)
      print("Middle Node: " + str(find_middle_of_linked_list(head).value))
```

```
26
27
28 main()
29
Run Save Reset (3
```

Solution

One brute force strategy could be to first count the number of nodes in the LinkedList and then find the middle node in the second iteration. Can we do this in one iteration?

We can use the **Fast & Slow pointers** method such that the fast pointer is always twice the nodes ahead of the slow pointer. This way, when the fast pointer reaches the end of the LinkedList, the slow pointer will be pointing at the middle node.

Code

Here is what our algorithm will look like:

```
🚣 Java
           Python3
                        ⊙ C++
                                   Js JS
    def find_middle_of_linked_list(head):
      slow = head
      fast = head
      while (fast is not None and fast.next is not None):
        slow = slow.next
      return slow
     head = Node(1)
     head.next = Node(2)
      head.next.next = Node(3)
      head.next.next.next = Node(4)
      head.next.next.next.next = Node(5)
      print("Middle Node: " + str(find_middle_of_linked_list(head).value))
      head.next.next.next.next = Node(6)
      print("Middle Node: " + str(find_middle_of_linked_list(head).value))
      head.next.next.next.next.next = Node(7)
      print("Middle Node: " + str(find_middle_of_linked_list(head).value))
    main()
Run
                                                                                               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).

