

Solution Review: Find Middle Node of a Linked List

This review provides a detailed analysis of the different ways to solve the Find the Middle Value in a Linked List challenge.

We'll cover the following



- Solution #1: Brute Force Method
 - Time Complexity
- Solution #2: Two Pointers
 - Time Complexity

Solution #1: Brute Force Method

main.py

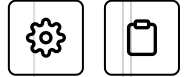
LinkedList.py

Node.py

```
8
9 def find_mid(lst):
10     if lst.is_empty():
11         return None
12
13     node = lst.get_head()
14     mid = 0
15     if lst.length() % 2 == 0:
16         mid = lst.length()//2
17     else:
18         mid = lst.length()//2 + 1
```



```
19
20     for i in range(mid - 1):
21         node = node.next_element
22
23     return node.data
24
25
26 lst = LinkedList()
27 lst.insert_at_head(22)
28 lst.insert_at_head(21)
29 lst.insert_at_head(10)
30 lst.insert_at_head(14)
31 lst.insert_at_head(7)
32
33 lst.print_list()
34 print(find_mid(lst))
35
```



This is the simplest way to go about this problem. We traverse the whole list to find its length. The middle position can be calculated by halving the length.

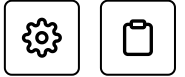
Note: For odd lengths, the middle value would be,

```
mid = length/2 + 1
```

Then, we iterate till the middle index and return the value of that node.

Time Complexity

The algorithm makes a linear traversal over the list. Hence, the time complexity is $O(n)$.



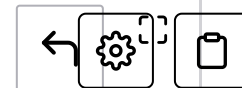
Solution #2: Two Pointers

main.py

LinkedList.py

Node.py

```
1 from LinkedList import LinkedList
2 from Node import Node
3 def find_mid(lst):
4     if lst.is_empty():
5         return -1
6     current_node = lst.get_head()
7     if current_node.next_element == None:
8         #Only 1 element exist in array so return its value.
9         return current_node.data
10
11     mid_node = current_node
12     current_node = current_node.next_element.next_element
13     #Move mid_node (Slower) one step at a time
14     #Move current_node (Faster) two steps at a time
15     #When current_node reaches at end, mid_node will be at the middle
16     while current_node:
17         mid_node = mid_node.next_element
18         current_node = current_node.next_element
19         if current_node:
20             current_node = current_node.next_element
21     if mid_node:
22         return mid_node.data
23     return -1
24
25 lst = LinkedList()
26 lst.insert_at_head(22)
27 lst.insert_at_head(21)
28 lst.insert_at_head(10)
```



This solution is more efficient as compared to the brute force method. We will use two pointers which will work simultaneously.

Think of it this way:

- The **fast** pointer moves two steps at a time till the end of the list
- The **slow** pointer moves one step at a time
- when the **fast** pointer reaches the end, the **slow** pointer will be at the middle

Using this algorithm, we can make the process faster because the calculation of the length and the traversal till the middle are happening side-by-side.

Time Complexity

We are traversing the linked list at twice the speed, so it is certainly faster. However, the bottleneck complexity is still $O(n)$.

The linked lists we have seen so far had unique values, but what if a list contains duplicates? We'll learn more about this in the next lesson.

← Back

Next →



Mark as Completed



Report an
Issue



Ask a Question

(https://discuss.educative.io/tag/solution-review-find-middle-node-of-a-linked-list__introduction-to-linked-lists__data-structures-for-coding-interviews-in-python)