876. Middle of the Linked List

Easy 10.4K 307

Companies

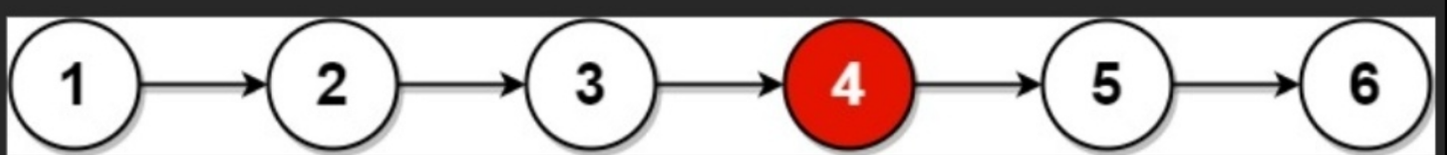
Given the `head` of a singly linked list, return *the middle node of the linked list*.

If there are two middle nodes, return **the second middle** node.

Example 1:


```
Input: head = [1,2,3,4,5]
Output: [3,4,5]
Explanation: The middle node of the list is node 3.
```

Example 2:

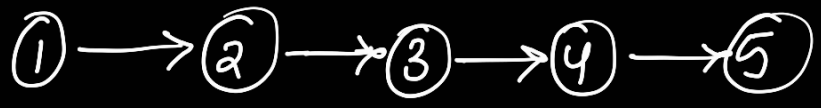



```
Input: head = [1,2,3,4,5,6]
Output: [4,5,6]
Explanation: Since the list has two middle nodes with
values 3 and 4, we return the second one.
```

Constraints:

- The number of nodes in the list is in the range `[1, 100]`.
- `1 <= Node.val <= 100`

Accepted **1.5M** | Submissions **1.9M** | Acceptance Rate **76.5%**



Basic intuition:

If 2 persons start at point A and if P_2 moves with speed 2x and P_1 moves with speed x then by the time P_2 reaches destination P_1 will have reached half of the distance.

→ Take 2 pointers slow and fast and initialize them with head.

→ now make slow pointer make 1 step and fast pointer make 2 steps.

case 1:

If there are odd no. of nodes then by the time fast reaches last node, slow would be exactly on middle of linked list.

case 2:

If there are even no. of nodes, then by the time fast becomes null, the slow pointer would be exactly on 2nd middle node

```
slow = fast = head

while( fast && fast->next)
{
    slow = slow->next
    fast = fast->next->next
}
```

return slow

$T(n) : O(n)$

Brute force:

-> find length of linked list n.
-> now make exactly $n/2$ steps from the head of list and now return the node pointing by the temp pointer.

```
while(temp)
{
    n++
    temp = temp->next
}
x = n/2
temp = head
while(x--)
    temp = temp->next

return temp
```

$T(n) : O(n) + O(n/2)$