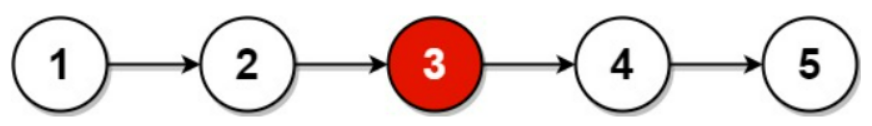
<https://leetcode.com/problems/middle-of-the-linked-list/>

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:**

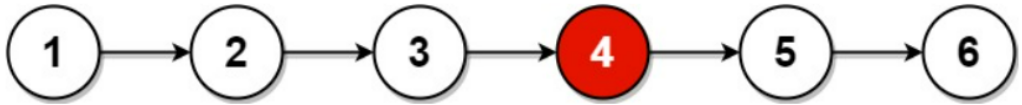


Input: head = [1,2,3,4,5]

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

**Attempt 1: 2023-02-10**

**Solution 1: Count the length and store the node by the way (10 min)**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode middleNode(ListNode head) {

ListNode[] list = new ListNode[100];

ListNode dummy = new ListNode();

dummy.next = head;

ListNode iter = dummy;

int count = 0;

while(iter.next != null) {

iter = iter.next;

list[count++] = iter;

}

return list[count / 2];

}

}

=========================================================================

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode middleNode(ListNode head) {

ListNode[] list = new ListNode[100];

int count = 0;

while(head != null) {

list[count++] = head;

head = head.next;

}

return list[count / 2];

}

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to**

<https://leetcode.com/problems/middle-of-the-linked-list/solutions/154715/middle-of-the-linked-list/>

#### **Approach 1: Output to Array**

**Intuition and Algorithm**

Put every node into an array A in order. Then the middle node is just A[A.length // 2], since we can retrieve each node by index.

We can initialize the array to be of length 100, as we're told in the problem description that the input contains between 1 and 100 nodes.

class Solution {

public ListNode middleNode(ListNode head) {

ListNode[] A = new ListNode[100];

int t = 0;

while (head != null) {

A[t++] = head;

head = head.next;

}

return A[t / 2];

}

}

**Complexity Analysis**

* Time Complexity: O(N), where N is the number of nodes in the given list.
* Space Complexity: O(N), the space used by A.

**Solution 2: Fast and Slow pointer (10 min)**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode middleNode(ListNode head) {

ListNode slow = head;

ListNode fast = head;

while(fast != null && fast.next != null) {

fast = fast.next.next;

slow = slow.next;

}

return slow;

}

}

Time Complexity: O(n)

Space Complexity: O(1)

**Refer to**

<https://leetcode.com/problems/middle-of-the-linked-list/solutions/154715/middle-of-the-linked-list/>

#### **Approach 2: Fast and Slow Pointer**

**Intuition and Algorithm**

When traversing the list with a pointer slow, make another pointer fast that traverses twice as fast. When fast reaches the end of the list, slow must be in the middle.

class Solution {

public ListNode middleNode(ListNode head) {

ListNode slow = head, fast = head;

while (fast != null && fast.next != null) {

slow = slow.next;

fast = fast.next.next;

}

return slow;

}

}

**Complexity Analysis**

* Time Complexity: O(N), where N is the number of nodes in the given list.
* Space Complexity: O(1), the space used by slow and fast