Middle Node in Linked List

Give notes for the below DSA problem: 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.

Here are the detailed notes for the problem of finding the middle node of a singly linked list:

Problem Understanding

- **Input**: A singly linked list represented by its head node.
- **Output**: The middle node of the linked list.
  - If the list has an odd number of nodes, return the single middle node.
  - If the list has an even number of nodes, return the second of the two middle nodes.

Key Points

1. **Middle Node Identification**:
   - If the number of nodes is odd: Only one middle node exists.
   - If the number of nodes is even: The second middle node is considered.
2. **Approaches**:
   - **Two-Pass Approach**:
     - In the first pass, determine the total length of the linked list.
     - In the second pass, traverse to the middle node by iterating `length / 2` steps.
   - **Two-Pointer (Optimal) Approach**:
     - Use two pointers: a slow pointer and a fast pointer.
     - Move the fast pointer two steps and the slow pointer one step at a time.

    - When the fast pointer reaches the end, the slow pointer will be at the middle node.

3. **Complexity**:
    - **Time Complexity**:
        - Two-Pass: $O(n)$ for both passes combined.
        - Two-Pointer: $O(n)$ for one pass.
    - **Space Complexity**:
        - Both approaches are $O(1)$, as they use constant extra space.

Algorithm for the Two-Pointer Approach

1. Initialize two pointers: `slow` and `fast`, both pointing to the head of the linked list.
2. Traverse the list:
    - Move `fast` two steps forward.
    - Move `slow` one step forward.
3. Stop when `fast` reaches the end (`fast == null` or `fast.next == null`).
4. The `slow` pointer now points to the middle node.

Edge Cases

- Single-node linked list: The middle node is the only node.
- Two-node linked list: Return the second node as per the problem description.
- Empty linked list: Ensure the input is valid; if not, handle gracefully (e.g., return `null`).

Pseudo Code

```
text
```

```
function findMiddleNode(head):
    if head is null:
        return null

    slow = head
    fast = head

    while fast is not null and fast.next is not null:
        slow = slow.next
        fast = fast.next.next

    return slow
```

Advantages of the Two-Pointer Approach

- Efficient in terms of runtime since it only requires a single traversal of the linked list.
- Minimal memory usage as no additional data structures are required.

Would you like me to include code for this problem in Java, or provide examples for further clarification?