To find the middle node of a doubly linked list using a "link hopping" method, without relying on the explicit size of the list, you can use the following approach. This technique involves using two pointers: one that moves one step at a time (slow pointer) and another that moves two steps at a time (fast pointer). This allows the slow pointer to reach the middle node when the fast pointer reaches the end of the list.

Here are the steps:

1. **Initialize Pointers:**
   * Start by initializing two pointers: a **slow pointer** and a **fast pointer**. Set both pointers to the header sentinel node.
2. **Advance the Fast Pointer:**
   * Move the fast pointer two steps at a time. This means the fast pointer will move from one node to the next, and then from that node to the next again, skipping over nodes.
3. **Advance the Slow Pointer:**
   * Move the slow pointer one step at a time. The slow pointer will move from one node to the next after each step.
4. **Termination Condition:**
   * Continue this process until the fast pointer reaches the trailer sentinel node, which indicates the end of the list. If the fast pointer is about to move two steps and the next node is the trailer sentinel, stop the iteration.
5. **Middle Node:**
   * When the fast pointer reaches the trailer sentinel node (or just before it), the slow pointer will be positioned at the middle node of the list. If the list contains an even number of nodes, the slow pointer will be at the node just slightly left of the exact middle.

**Example:**

For a list with nodes 1, 2, 3, 4, 5, 6:

* Initially, both pointers start at the header sentinel.
* In the first iteration:
  + The fast pointer moves to node 3.
  + The slow pointer moves to node 1.
* In the second iteration:
  + The fast pointer moves to node 5.
  + The slow pointer moves to node 2.
* In the third iteration:
  + The fast pointer moves to the trailer sentinel.
  + The slow pointer moves to node 3.
* The slow pointer now points to node 3, which is considered the middle for a list with an even number of nodes.

This method efficiently finds the middle node in linear time O(n)O(n)O(n) and does not require knowing the size of the list explicitly.