Documentation: Finding the Intersection Node of Two Singly Linked Lists

Problem Overview

The task is to identify the intersection point of two singly linked lists. Given the heads of two linked lists, the goal is to return the node at which the two lists intersect. If they do not intersect, the function should return None.

Key Concepts

1. Singly Linked List:

- A singly linked list is a sequence of nodes where each node contains a value and a reference to the next node in the sequence. The last node points to None, indicating the end of the list.

2. Intersection of Linked Lists:

- Two linked lists are said to intersect if they share a common node. This common node and all subsequent nodes are identical for both lists, meaning they share the same memory address.

3. No Intersection:

- If the two lists do not intersect, they have entirely separate sequences of nodes, and there is no shared memory address between any nodes of the two lists.

Approach: Two-Pointer Technique

Conceptual Steps:

1. **Initialization:**
    - Begin by setting up two pointers, pA and pB. Initially, pA points to the head of the first list (headA), and pB points to the head of the second list (headB).

2. **Traversal:**
    - Traverse through both linked lists using the pointers pA and pB. Move each pointer forward to the next node in its respective list.
    - If either pointer reaches the end of its list (None), reset it to the head of the other list. For example, if pA reaches the end of headA, reset it to the head of headB.

3. **Intersection Detection:**
    - Continue moving the pointers forward. If the two lists intersect, the pointers pA and pB will eventually point to the same node after traversing both lists.
    - If they do not intersect, both pointers will eventually reach the end of the lists (None), and the function will return None.

Why This Works:

- **Equalizing Path Lengths:**

  By resetting each pointer to the head of the other list after reaching the end of its own list, you ensure that both pointers traverse an equal total number of nodes. This equalizes the difference in lengths between the two lists, allowing the pointers to meet at the intersection point if one exists.

- **Guaranteed Termination:**

  The loop will terminate when either both pointers point to the same node (indicating an intersection) or when both pointers are None (indicating no intersection). This ensures that the function always produces a result.

Example 1:

- **Input:** listA = [4,1,8,4,5], listB = [5,6,1,8,4,5]
- **Intersection:** Node with value 8
- **Explanation:** The lists intersect at the node with value 8. The output is the node at which the two lists intersect.

Example 2:

- **Input:** listA = [1,9,1,2,4], listB = [3,2,4]
- **Intersection:** Node with value 2
- **Explanation:** The lists intersect at the node with value 2. The output is the node at which the two lists intersect.

Example 3:

- **Input:** listA = [2,6,4], listB = [1,5]
- **No Intersection:** The lists do not intersect.
- **Explanation:** Since the two lists do not intersect, the output is None.

Constraints

- The number of nodes in listA and listB is bounded by 1 <= m, n <= 30,000.
- Node values are within the range 1 <= Node.val <= 100,000.
- intersectVal is the value of the intersected node or 0 if there is no intersection.
- If the lists intersect, intersectVal == listA[skipA] == listB[skipB].

Performance Considerations

- **Time Complexity: O(m + n)**

  Each pointer traverses both lists exactly once. This gives a time complexity linear to the combined length of the two lists.

- **Space Complexity: O(1)**

  The algorithm only uses two additional pointers, regardless of the size of the input lists. Thus, the space complexity is constant.

Follow-Up Considerations

- **Cycle Detection:** This problem assumes no cycles in the linked lists. If cycles are introduced, a more complex algorithm would be required to handle such cases.

- **Node Reordering:** The solution preserves the original structure of the lists, meaning it doesn't modify or reorder the nodes.

This documentation explains the problem, the approach to solve it, and the rationale behind the chosen algorithm without showing the actual code.