[Click on the logo to find **Problem Statement**]

Intuition

This code employs a two-pointer approach to find the intersection node of two linked lists, headA and headB. The key intuition lies in the fact that if two linked lists have an intersection, they will have the same length after the intersection point. By using two pointers, 'one' and 'two', initialized to the heads of the linked lists, we traverse both lists simultaneously. Whenever one of the pointers reaches the end of its list, it wraps around to the head of the other list. This way, both pointers continue to move until they meet at the intersection node or reach the end of both lists. If they meet at an intersection, the loop terminates, and the common node (intersection point) is returned. Otherwise, if they both reach the end without meeting, indicating no intersection, None is returned. This approach optimally utilizes two pointers to achieve linear time complexity and constant space complexity.

Approach

1. Initialize two pointers, 'one' and 'two', to the heads of the two linked lists, headA and headB, respectively.
2. Traverse both linked lists simultaneously using the two pointers until either the intersection node is found or both pointers reach the end of their respective lists.
3. In each iteration, move the pointers to the next nodes of their respective lists.
4. If either pointer reaches the end of its list, wrap it around to the head of the other list.
5. Continue this process until the pointers either meet at the intersection node or both reach the end of their lists.
6. If the pointers meet at an intersection, return the intersection node.
7. If the pointers reach the end without meeting, indicating no intersection, return None.

Complexity

- Time complexity: O(m + n)
- Space complexity: O(1)

Code

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> Optional[ListNode]:
        one = headA
        two = headB
        while one != two:
            if one is None:
                one = headB
            else:
                one = one.next
            if two is None:
                two = headA
            else:
                two = two.next
        return one
```

If you want to see more solutions to coding problems, you can visit: