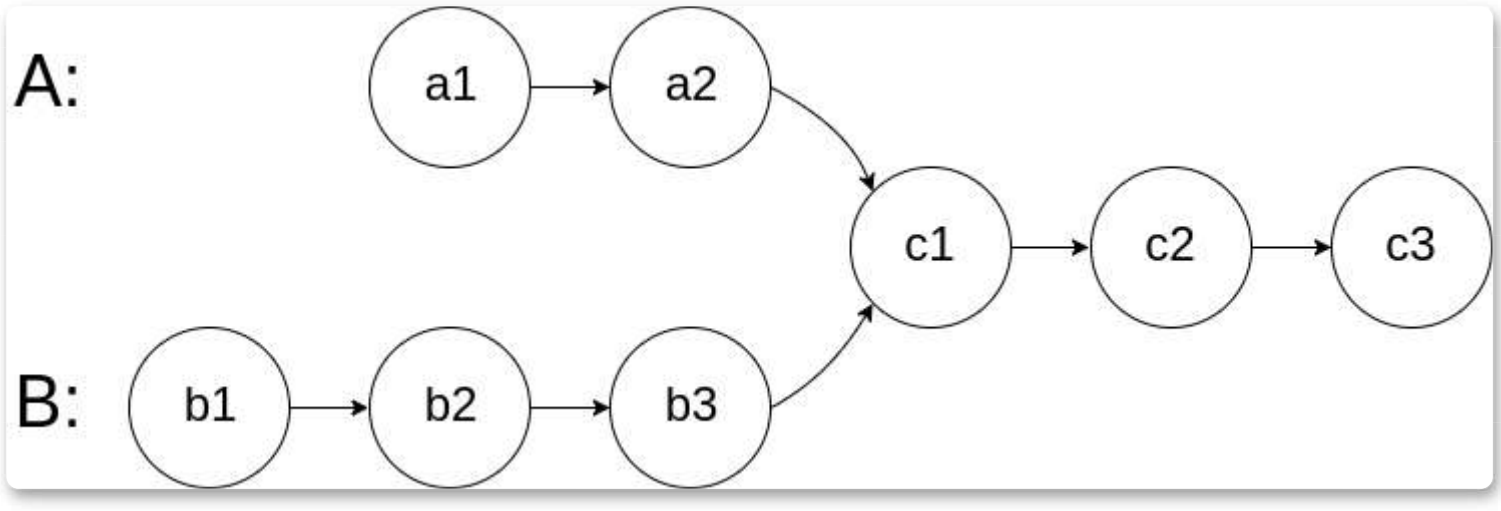
「PS：这道题虽然是 easy 的，但是其实如果不仔细思考，不容易想出最优解，这到题目的最优解其实非常优雅。」

题目介绍[1]

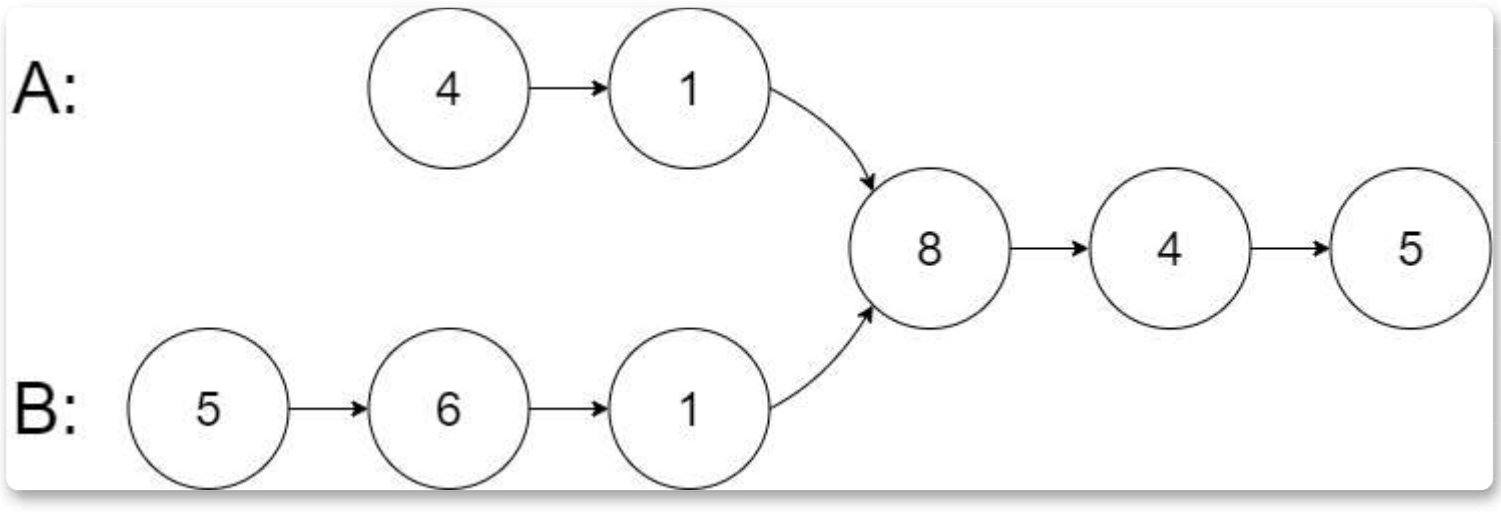
给你两个单链表的头节点 headA 和 headB ，请你找出并返回两个单链表相交的起始节点。如果两个链表不存在相交节点，返回 null 。 图示两个链表在节点 c1 开始相交：



题目数据

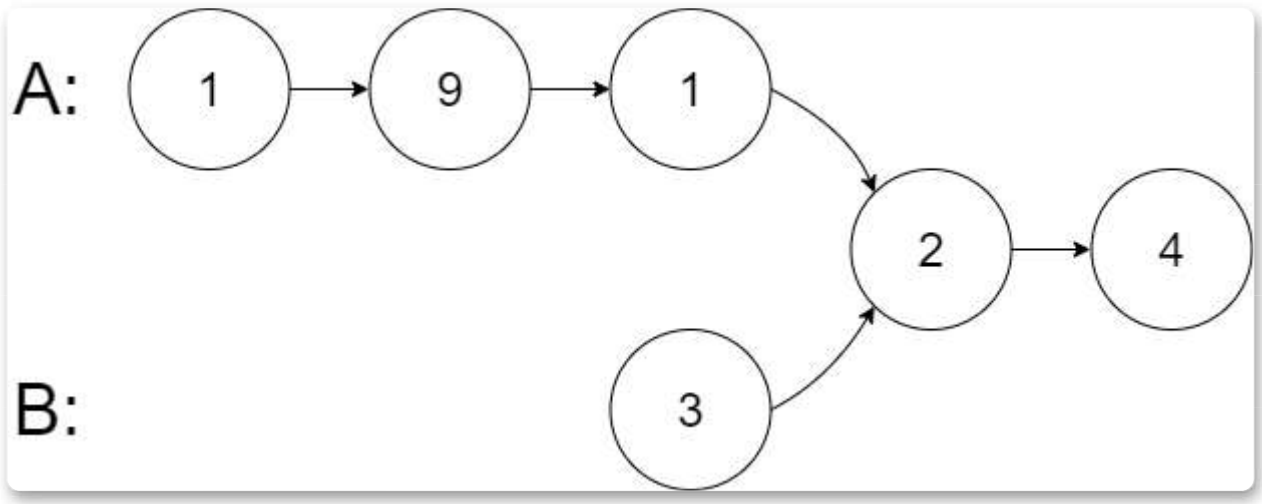
保证整个链式结构中不存在环。 注意，函数返回结果后，链表必须 保持其原始结构 。

示例 1



> 「输入」：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5], skipA = 2, skipB = 3
>
> 「输出」：Intersected at '8'
>
> 「解释」：相交节点的值为 8 （注意，如果两个链表相交则不能为 0）。 从各自的表头开始算起，链表 A 为 [4,1,8,4,5]，链表 B 为 [5,6,1,8,4,5]。 在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3 个节点。

示例 2

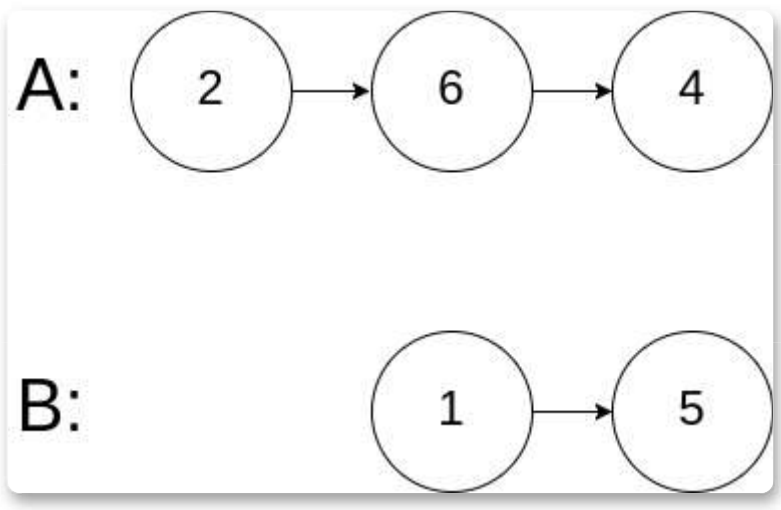


「输入」：intersectVal = 2, listA = [1,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1

「输出」：Intersected at '2'

「解释」：相交节点的值为 2 （注意，如果两个链表相交则不能为 0）。 从各自的表头开始算起，链表 A 为 [1,9,1,2,4]，链表 B 为 [3,2,4]。 在 A 中，相交节点前有 3 个节点；在 B 中，相交节点前有 1 个节点。

示例 3



「输入」：intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2

「输出」：null

「解释」：从各自的表头开始算起，链表 A 为 [2,6,4]，链表 B 为 [1,5]。 由于这两个链表不相交，所以 intersectVal 必须为 0，而 skipA 和 skipB 可以是任意值。 这两个链表不相交，因此返回 null

题目解答

方法一：哈希集合

思路和算法

最容易想到的方法就是用 hash 表来解决。

1. 先创建一个 hash 集合，然后遍历链表 A，每遍历一个节点 nodeA 的时候就存到 hash 集合里，遍历完链表 A 之后再遍历链表 B。
2. 在遍历链表 B 的过程中遇到的每一个链表 B 节点 nodeB 都去查一次 hash 集合，判断是否已经存在 hash 集合里，如果存在那么就说明我们找到这两个链表相交的节点了。
3. 如果所有的链表 B 节点 nodeB 都不在 hash 集合里，就说明这两个链表不相交。

代码实现

```
public class Solution {
    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        Set<ListNode> visited = new HashSet<ListNode>();
        ListNode temp = headA;
        while (temp != null) {
            visited.add(temp);
            temp = temp.next;
        }
        temp = headB;
        while (temp != null) {
            if (visited.contains(temp)) {
                return temp;
            }
            temp = temp.next;
        }
        return null;
    }
}
```

复杂度分析

n 为链表 A 的长度，m 为链表 B 的长度

- 时间复杂度：O(n+m)
- 空间复杂度：O(n)

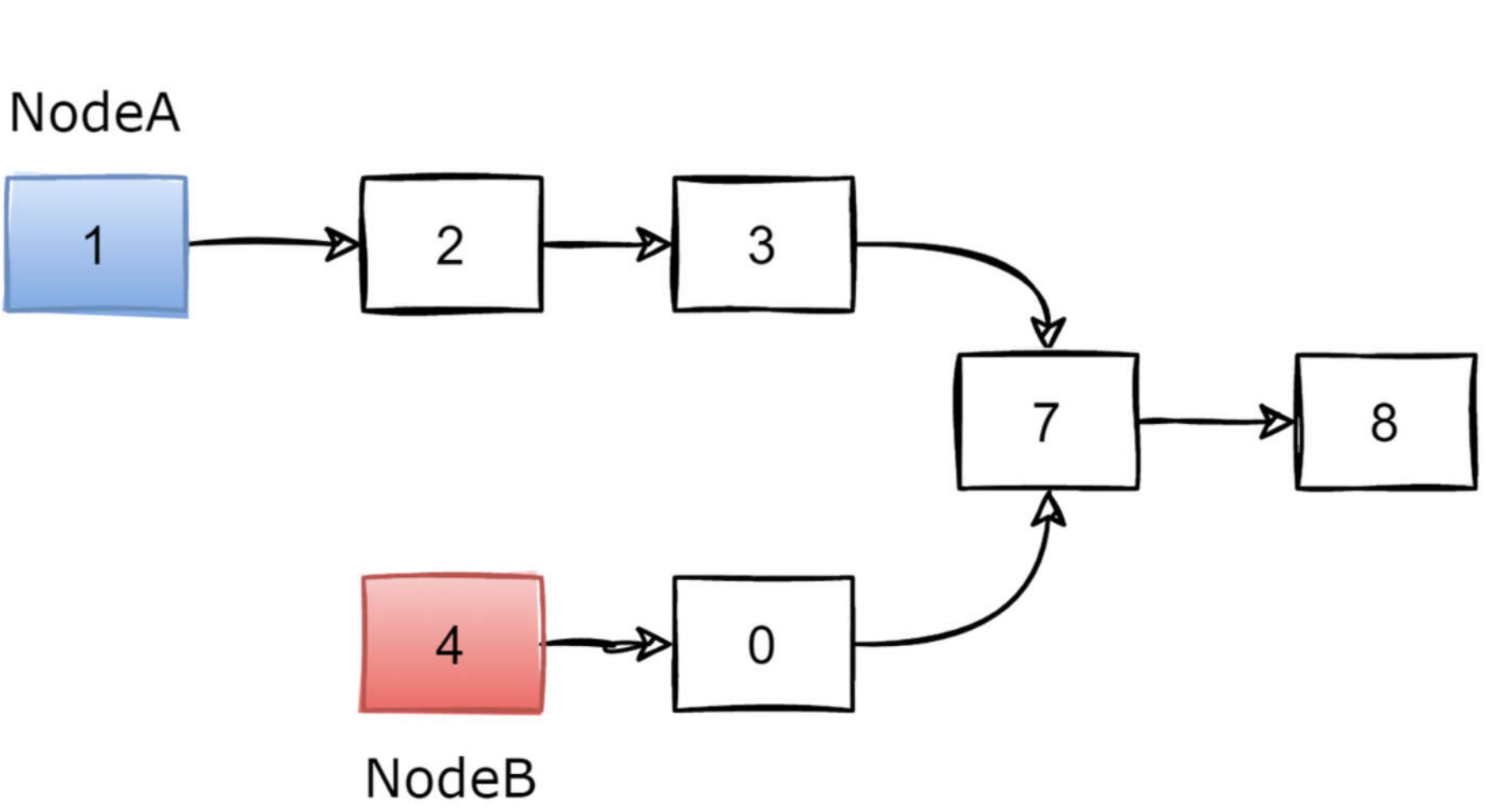
方法二：双指针

思路和算法

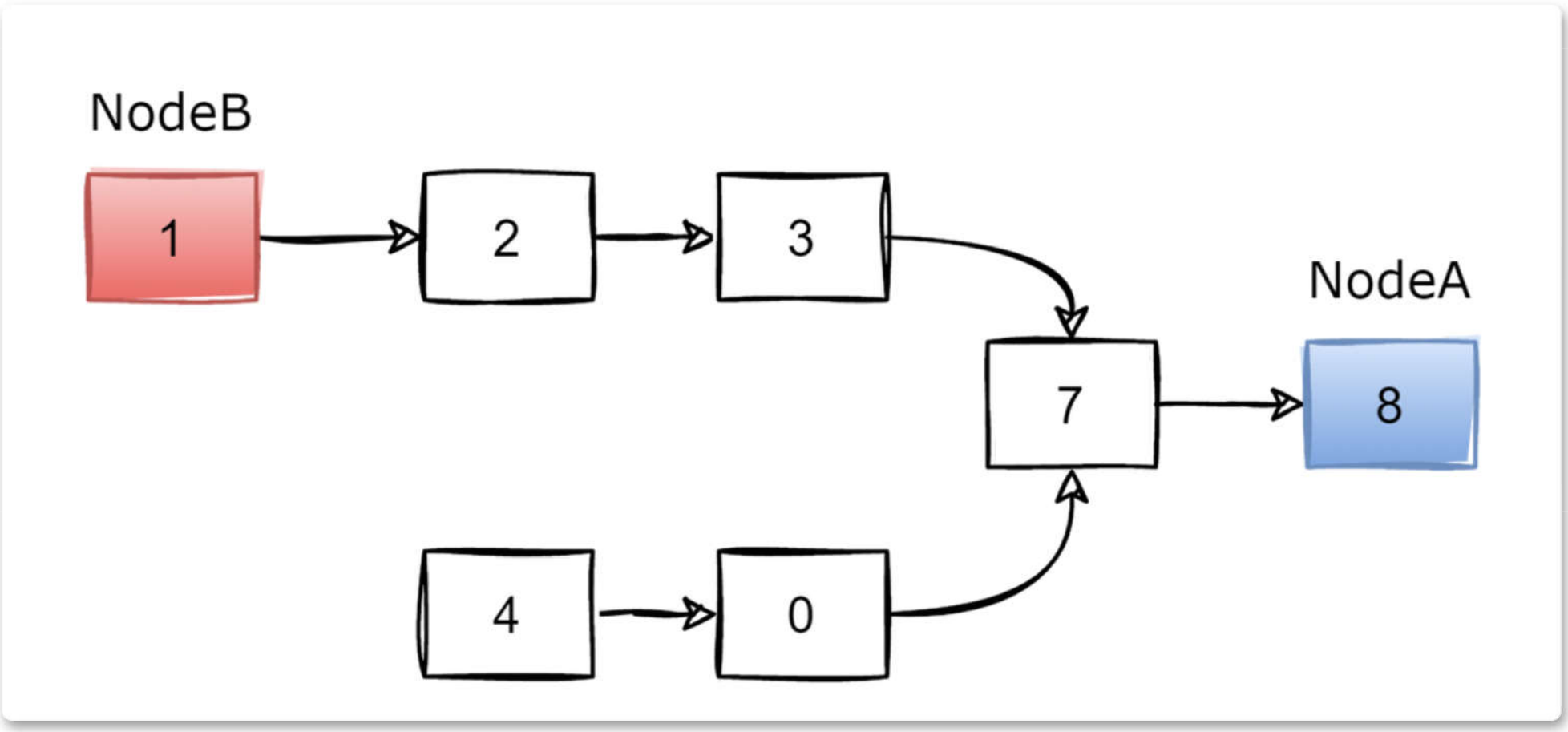
双指针法才是最终的解法，主要思路是可以把两个链表进行合并视为一个链表（在实现上不需要正在合并），这样两个指针 nodeA 和 nodeB 在同时遍历时走过的节点数永远是相等的。如果这两个链表是相交的，那么指针 nodeA 和 nodeB 必定会同时到达相交处。

一图胜千言

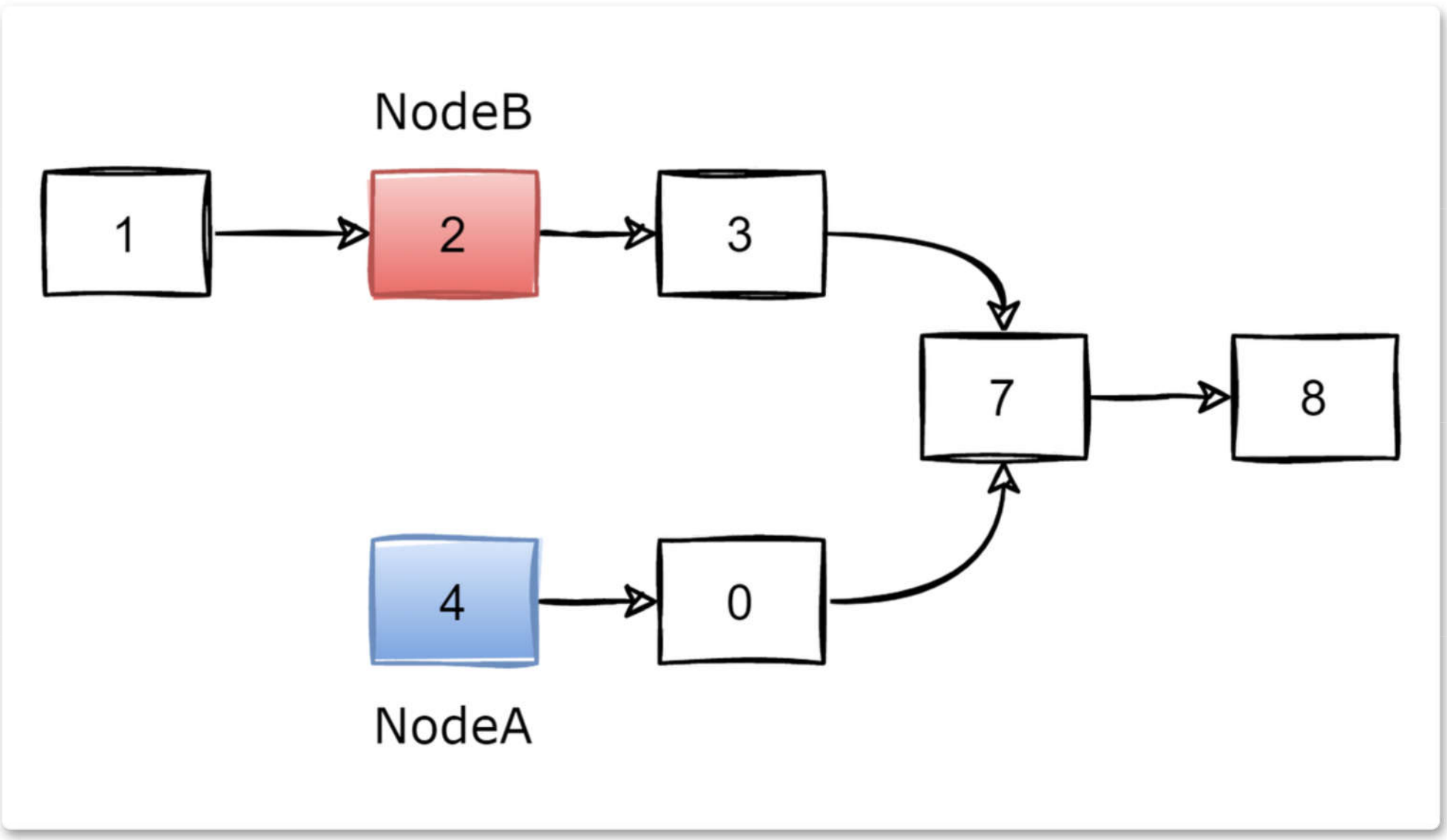
「初始状态」



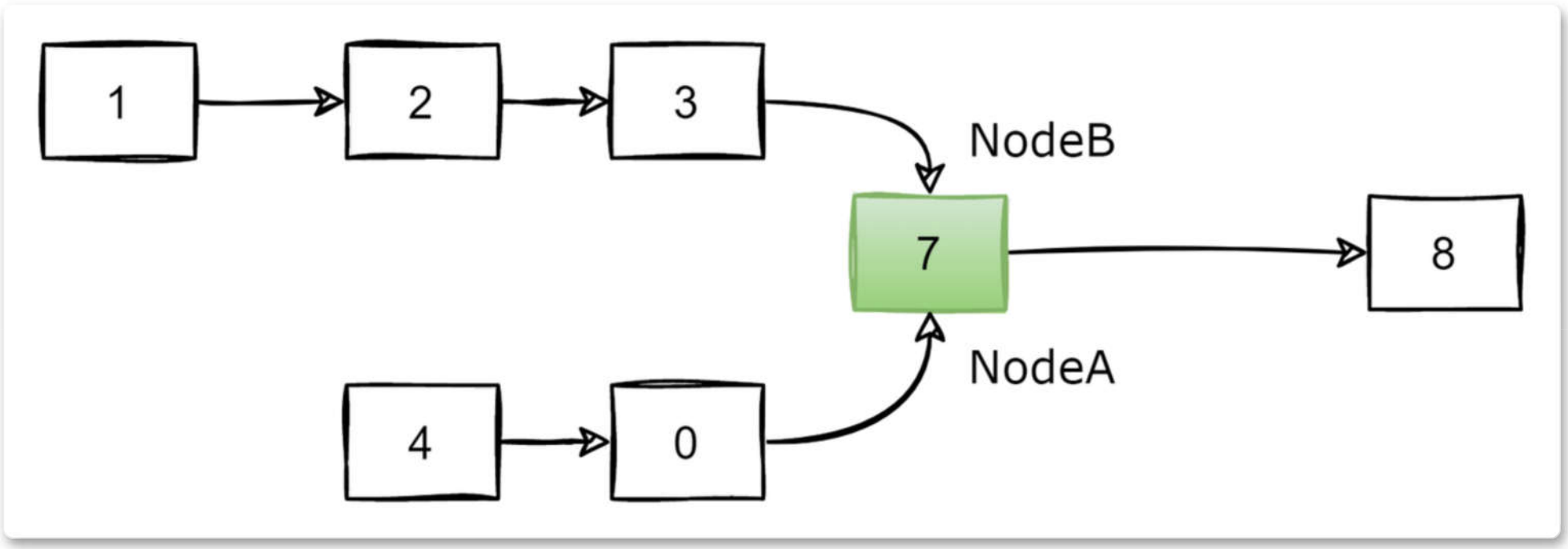
「迭代 4 个节点后的状态」 这时已经遍历完了链表 B，我们让 NodeB 重新指向链表 A 的头节点。



「迭代 5 个节点后的状态」 这时已经遍历完了链表 A，我们让 NodeA 重新指向链表 B 的头节点。此时此刻 NodeA 和 NodeB 到达末尾时的长度一样，如果链表相交，NodeA 和 NodeB 会同时到达相交节点。如果链表不相交，NodeA 和 NodeB 会同时到达末尾。



「迭代 7 个节点后的状态」 这时 NodeA 和 NodeB 相遇了，说明我们找到了相交点



参考代码

```
public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
    if (headA == null || headB == null) return null;
    ListNode nodeA = headA, nodeB = headB;

    // 如果相交时，nodeA 和 nodeB指向了同一个节点。
    // 如果不相交，nodeA 和 nodeB会同时到达末尾，最终都为null。
    while (nodeA != nodeB) {
        nodeA = nodeA == null ? headB : nodeA.next;
        nodeB = nodeB == null ? headA : nodeB.next;
    }
    return nodeA;
}
```

复杂度分析

n 为链表 A 的长度，m 为链表 B 的长度

- 时间复杂度：O(n+m)
- 空间复杂度：O(1)

面试 tips

别看这道题是个 esay 难度的题目，如果在面试过程中只写出了第一种答案，那面试官肯定会继续追问：“还有没有更好的方法呢？”。
如果能写出第二种答案，那可是很大的加分项。

参考资料

[1] **原题链接:**
https://leetcode-cn.com/problems/intersection-of-two-linked-lists/