Intersection of two linked lists (LeetCode 160 - Easy)

Problem Description

Write a program to find the node at which the intersection of two singly linked lists begins.

Notes:

- If the two linked lists have no intersection at all, return null.
- The linked lists must retain their original structure after the function returns.
- You may assume there are no cycles anywhere in the entire linked structure.
- Your code should preferably run in O(n) time and use only O(1) memory.

```
/**
 * LeetCode 160
 * Find the node
 * at which the intersection of two singly linked lists begins.
 * If the two linked lists have no intersection at all,
  * return null.
 * The linked lists must retain their original structure
 * after the function returns.
 * You may assume there are no cycles
 * anywhere in the entire linked structure
 * @param headA
 * @param headB
 * @return The node
 * at which the intersection of two singly linked lists begins.
 */
```

Solution 1 – HashSet

```
/*
 * HashSet
 * time complexity: O(n)
 * space complexity: O(n)
 */
public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
	HashSet<ListNode> set = new HashSet<ListNode>();
	while (headA != null) {
		set.add(headA);
		headA = headA.next;
	}
	while (headB != null) {
		if (set.contains(headB)) {
			return headB;
		}
		headB = headB.next;
	}
	return null;
}
```

Solution 2 – Two Pointers

```
/*
 * Two Pointers
 * time complexity: O(n)
 * space complexity: O(1)
 * a1 -> a2 ->
 *                          c1 -> c2 -> c3
 * b1 -> b2 -> b3 ->
 * We want a and b to meet at c1
 * How? -> Two iterations.
 */
public ListNode getIntersectionNode2(ListNode headA, ListNode headB) {
	ListNode a = headA;
	ListNode b = headB;
	while(a != b) {
		a = a == null ? headB : a.next;
		// a1->a2->c1->c2->c3->b1->b2->b3->c1
		b = b == null ? headA : b.next;
		// b1->b2->b3->c1->c2->c3->a1->a2->c1
	}
	return a == b ? a : null;
}
```