Intersection Of Two Sorted Linked List

7/3/25

Eg: LL1 = 1 -> 2 -> 3 -> 4 -> 6
LL2 = 2 -> 4 -> 6 -> 8.
IL = 2 -> 4 -> 6

LL1 = 10 -> 20 -> 40 -> 50
LL2 = 15 -> 40
IL = 40

Constraints :-> $1 <= \text{size of LL} <= 10^4$
$1 <= \text{node} \to \text{data} <= 10^4$

Two approaches.

1) Two Pointer approach [Efficient]
- Time complexity = O(N+M)
Space complexity = O(1) for modifying input
= O(min(N,M)) if creating new list

2) Hash Set Approach
Time = O(N+M)
Space = O(N) or O(M)
Extra space for hash set

① Two Pointer Approach

Algorithm: 1) Initialize two pointers, p1 & p2 at the heads of the two linked lists.

2) Compare the values @ p1 & p2
* If p1->val == p2->val, add the node to the result list & move both pointers forward
* If p1->val < p2->val, move p1 forward
* If p1->val > p2->val, move p2 forward

3) Repeat until either list is exhausted

4) Return the new intersection list

Python Code

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val=val
        self.next=next
def intersection_of_sorted_lists(head1,head2):
    dummy=ListNode()
    tail=dummy
    p1,p2=head1,head2

    while p1 and p2:
        if p1.val==p2.val:
            tail.next=ListNode(p1.val)
            tail=tail.next
            p1=p1.next
            p2=p2.next
        elif p1.val<p2.val:
            p1=p1.next
        else:
            p2=p2.next
    return dummy.next
```

C code

```
#include <stdio.h>
#include <stdlib.h>
struct ListNode {
    int val;
    struct ListNode *next; };

struct ListNode* intersectionOfSortedLists(struct ListNode* head1,
                                           struct ListNode* head2) {

    struct ListNode dummy;
    struct ListNode* tail = &dummy;
    dummy.next = NULL;

    while(head1 && head2) {
        if (head1->val == head2->val) {
            struct ListNode* newNode = (struct ListNode*)malloc(sizeof(
                                       struct ListNode));
            newNode->val = head1->val;
            newNode->next = NULL;
            tail->next = newNode;
            tail = newNode;
            head1 = head1->next;
            head2 = head2->next;
        } else if (head1->val < head2->val) {
            head1 = head1->next;
        } else {
            head2 = head2->next;
        }
    }
    return dummy.next;
}
```

C++)

Node * findIntersection (Node * head1 , Node * head2

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def find_intersection (head1, head2):
    head = None
    curr = None
    while head1 and head2:
        if head1.data < head2.data:
            head1 = head1.next
        elif head1.data > head2.data
            head2 = head2.next
    else:
        if head is None:
            head = None (head1.data)
            curr = head
        else:
            curr.next = Node (head1.data)
            curr = curr.next
        head1 = head1.next
        head2 = head2.next
return head
```

Step-by-step Execution

head1.data	head2.data	Action
1	2	move head1 forward
2	2	Add '2' to intersection & move both →
3	4	move head1 →
4	4	Add '4' to intersection, move both →
6	6	Add 6 to intersection move both →
None	8	Stop (one list is empty.