Find Merge Point (Intersection) of Two Sorted Linked Lists

Mr. Vikas Kumar

Assistant Professor
Industry Embedded Program

Lesson Plan

Subject/Course	**Competitive Coding**
Lesson Title	**Find Merge Point of Two Sorted Linked Lists**

Lesson Objectives
Find Merge Point (Intersection) of Two Sorted Linked Lists
Detect whether two singly linked lists intersect (share nodes).
Achieve $O(m + n)$ time and $O(1)$ extra space.
When lists intersect they share the same tail from the merge point onward.

Problem Statement:

Write a program to find the Merge point of two sorted linked lists.

Concept

A linked list is a linear data structure composed of a sequence of nodes, where each node contains data and a pointer (or reference) to the next node in the sequence.

Common operations:

- Insertion: Adding a new node to the linked list
- Deletion: Removing a node from the linked list.
- Traversal
- Searching

Algorithm/Logic

1. Step by Step Pseudocode:

Compute lengths lenA and lenB of both lists.

Advance the longer list by abs(lenA - lenB) steps.

Walk both lists step-by-step comparing node references.

First node where nodeA == nodeB is the intersection. If none — return null.

Visualization

Two lists intersecting at node X:

List A: A1 -> A2 -> A3 \

                    -> X -> X.next -> ...

List B: B1 -> B2 /

Walk alignment (after advancing longer list or via pointer switching):

After advancing longer:

A3 -> X -> ...

B2 -> X -> ...

Code Implementation

```java
1  public class Practical8_MergePoint {
2
3      static class ListNode {
4          int data;
5          ListNode next;
6          ListNode(int val) { data = val; }
7      }
8
9      static ListNode getMergeNode(ListNode headA, ListNode headB) {
10         ListNode a = headA, b = headB;
11         while (a != b) {
12             a = (a == null) ? headB : a.next;
13             b = (b == null) ? headA : b.next;
14         }
15         return a;
16     }
```

Code Implementation

```java
17 ▾   public static void main(String[] args) {
18         ListNode common = new ListNode(30);
19         ListNode headA = new ListNode(10);
20         headA.next = new ListNode(15);
21         headA.next.next = common;
22
23         ListNode headB = new ListNode(3);
24         headB.next = common;
25
26         ListNode mergePoint = getMergeNode(headA, headB);
27         System.out.println("Merge Point: " + (mergePoint != null ? mergePoint.data : "None"));
28     }
29 }
```

Output :

```
Merge Point: 30
```

Time & Space Complexity

The time complexity of the getIntersectionNode function is O(M + N), where M is the length of the first linked list (headA) and N is the length of the second linked list (headB).
The space complexity of the getIntersectionNode function is O(1).

Summary

- This algorithm cleverly handles lists of different lengths by "re-routing" the pointers to the head of the other list once they reach the end of their own list. This ensures that both pointers will eventually traverse the same total distance before meeting at the intersection or at NULL.

Practice Exercises

Problem Statement 1

```
Input: head = [1,2]
Output: [2,1]
```

Example 3:

```
Input: head = []
Output: []
```

Constraints:

- The number of nodes in the list is the range [0, 5000].

- -5000 <= Node.val <= 5000

Follow up: A linked list can be reversed either iteratively or recursively. Could you implement both?

Problem Statement Path:

Competitive Coding

Linked List

Reverse Linked List

Link :https://leetcode.com/problems/reverse-linked-list/description/
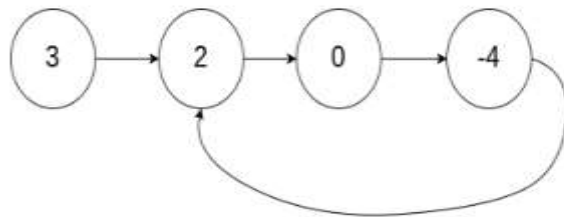
Practice Exercises

Problem Statement 2

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that** `pos` **is not passed as a parameter**

Return `true` *if there is a cycle in the linked list.* Otherwise, return `false`.

Example 1:


```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail
connects to the 1st node (0-indexed).
```

Problem Statement Path:

Competitive Coding

Linked List

Linked List Cycle

Link:

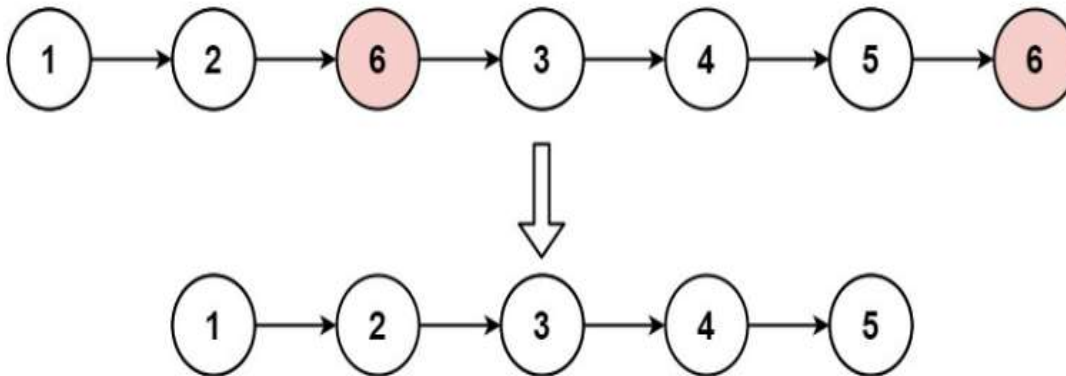https://leetcode.com/problems/linked-list-cycle/description/

Practice Exercises

Problem Statement 3

Given the `head` of a linked list and an integer `val`, remove all the nodes of the linked list that has `Node.val == val`, and return *the new head*.

Example 1:

Input: head = [1,2,6,3,4,5,6], val = 6
Output: [1,2,3,4,5]

Problem Statement Path:

Competitive Coding

Linked List

Remove Linked List Elements

Link:

https://leetcode.com/problems/remove-linked-list-elements/description/

Thanks