Day 25 (10/07/2025)

---

1. Palindrome Linked List

A problem that introduces advanced linked list analysis techniques and teaches how to check symmetry in sequential data structures efficiently.

Given a **head singly linked list** of positive integers, the task is to check if the given linked list is **palindrome or not**. A palindrome reads the same forwards and backwards. This problem appears frequently in **interviews** and **real-world applications** like data validation, symmetry detection in sequences, or implementing verification algorithms. You can solve this by converting to an array first, but try to think of **more efficient approaches** using techniques like reversing half the list or using two pointers with recursion.

This teaches **symmetry detection** and **two-pointer validation** techniques that are essential for **pattern recognition and efficient data structure analysis**.

Your task: Check if a linked list is a palindrome using efficient techniques without extra space or with minimal space complexity.

Examples

Input:

head: 1 -> 2 -> 1 -> 1 -> 2 -> 1

Output:

true

Input:

head: 1 -> 2 -> 3 -> 4

Output:

false

---

2. Intersection Sorted Linked Lists

A problem that demonstrates merge-like operations on sorted data structures and teaches how to find common elements efficiently between two sorted lists.

Given that **two linked lists are sorted** in increasing order, create a **new linked list** representing the **intersection** of the two linked lists. The new linked list should be made **without changing the original lists**. The elements of the linked list are not necessarily distinct. This operation is fundamental in **data analysis** and **set operations** where you need to **find common elements between sorted datasets** efficiently without modifying the original data structures.

This introduces **merge-based intersection** and **sorted data processing** techniques that are crucial for **database operations and efficient set computations**.

Your task: Find intersection of two sorted linked lists by creating a new list while preserving original lists.

Examples

Input:

LinkedList1 = 1->2->3->4->6, LinkedList2 = 2->4->6->8

Output:

2->4->6

Input:

LinkedList1 = 10->20->40->50, LinkedList2 = 15->40

Output:

40

---

3. Add Number Linked Lists

A problem that teaches arithmetic operations on linked list representations and demonstrates how to perform addition on numbers stored as linked lists.

Given the **head of two singly linked lists** num1 and num2 representing two **non-negative integers**, the task is to return the head of the linked list representing the **sum of these two numbers**. For example, num1 represented by the linked list: 1 -> 9 -> 0, similarly num2 represented by the linked list: 2 -> 5. Sum of these two numbers is represented by 2 -> 1 -> 5. There can be **leading zeros** in the input lists, but there should not be any leading zeros in the output list. This problem simulates **big number arithmetic** commonly used in cryptography and mathematical computations.

This teaches **digit-by-digit arithmetic** and **carry propagation** techniques that are essential for **big number operations and mathematical computations in linked structures**.

Your task: Add two numbers represented as linked lists while handling carry propagation and leading zero elimination.

Examples

Input:

num1 = 4 -> 5, num2 = 3 -> 4 -> 5

Output:

3 -> 9 -> 0

Input:

num1 = 0 -> 0 -> 6 -> 3, num2 = 0 -> 7

Output:

7 -> 0