Day 24 (09/07/2025)

---

1. Reverse a Linked List

A problem that introduces fundamental linked list manipulation techniques and teaches how to reverse the direction of node connections efficiently.

Given the **head of a linked list**, the task is to **reverse this list** and return the reversed head. This is one of the most **fundamental operations** in linked list manipulation and appears frequently in **interviews** and **real-world applications** like implementing undo functionality, reversing data streams, or processing data in reverse order. You can solve this iteratively by changing the links between nodes, or recursively by processing nodes from the end. Understanding pointer manipulation is crucial for this problem.

This teaches **pointer manipulation** and **iterative/recursive thinking** techniques that are essential for **advanced linked list operations and data structure transformations**.

Your task: Reverse the direction of a linked list by changing node connections using efficient pointer manipulation.

Examples

Input:

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

Output:

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

Input:

head: 2 -> 7 -> 10 -> 9 -> 8 -> NULL

Output:

head: 8 -> 9 -> 10 -> 7 -> 2 -> NULL

Input:

head: 2 -> NULL

Output:

2 -> NULL

---

2. Delete N nodes after M nodes of a linked list

A problem that demonstrates pattern-based linked list modification and teaches how to delete nodes following specific intervals and patterns.

Given a **linked list**, delete **n nodes** after skipping **m nodes** of a linked list until the last of the linked list. This operation is commonly used in **data filtering** and **pattern-based processing** applications where you need to **remove elements at regular intervals** while preserving specific sections. The challenge involves understanding how to maintain proper node connections while following a deletion pattern throughout the entire list.

This introduces **pattern-based traversal** and **conditional node deletion** techniques that are crucial for **data stream processing and interval-based operations**.

Your task: Delete nodes following a specific pattern while maintaining proper list connections and handling edge cases.

Examples

Input:

Linked List: 9->1->3->5->9->4->10->1, n = 1, m = 2

Output:

9->1->5->9->10->1

Input:

Linked List: 1->2->3->4->5->6, n = 1, m = 6

Output:

1->2->3->4->5->6

---

3. Delete in a Singly Linked List

A problem that teaches position-based node deletion and demonstrates how to remove specific nodes by their index while maintaining list integrity.

Given a **singly linked list** and an integer **x**, delete the **xth node** (1-based indexing) from the singly linked list. This operation is fundamental in **dynamic data management** and **position-based removal** applications where you need to **delete elements at specific positions** efficiently. The challenge involves understanding how to traverse to the correct position, handle edge cases like deleting the head node, and properly reconnect the remaining nodes.

This teaches **position-based deletion** and **index-based traversal** techniques that are essential for **dynamic list management and efficient element removal**.

Your task: Delete a node at a specific position while properly handling edge cases and maintaining list connections.

Examples

Input:

Linked list: 1 -> 3 -> 4, x = 3

Output:

1 -> 3

Input:

Linked list: 1 -> 5 -> 2 -> 9, x = 2

Output:

1 -> 2 -> 9