Day 23: Reverse a Linked List

Abhinav Yadav

"In a linked list, reversing brings you back to the beginning with a different perspective."
— Anonymous

---

1 Introduction

A **Singly Linked List** is a linear data structure where each element (node) contains two parts:

- The data element.
- A pointer to the next node in the list.

In this problem, we are asked to **reverse the singly linked list**. This means that the head node will be transformed into the last node, and each node will point to the previous one.

We can reverse a singly linked list using the **three-pointer technique**. This method uses three pointers:

- **Previous Pointer:** Points to the previous node.
- **Current Pointer:** Points to the current node.
- **Next Pointer:** Points to the next node.

2 Steps to Reverse a Linked List

The process involves the following steps:

1. Initialize three pointers: `prev` (NULL), `current` (head), and `next` (NULL).
2. Traverse the list, and for each node:
   (a) Set `next` to `current->next`.
   (b) Change `current->next` to `prev`.
   (c) Move `prev` to `current` and `current` to `next`.
3. Repeat this until `current` becomes NULL.
4. The `prev` pointer will be pointing to the new head of the reversed list.

3 Applications of Reversing a Linked List

Reversing a linked list can be useful in several scenarios, including:

- Reversing a list of nodes for printing or traversing.
- Implementing undo operations (reversing actions).
- Reversing a stack of elements, where the linked list can serve as an auxiliary data structure.

4 Code Implementation

```
import java.util.Scanner;

// Define a Node class
class Node {
    int data;
    Node next;

    // Constructor to initialize the node
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

public class ReverseLinkedList {
    // Function to insert a node at the beginning
    public static Node insertAtBeginning(Node head, int value) {
        Node newNode = new Node(value); // Create a new node
        newNode.next = head;            // Link the new node to
           the previous first node
        return newNode;                 // Return the new head (
           new node)
    }

    // Function to reverse the linked list
    public static Node reverseList(Node head) {
        Node prev = null;
        Node current = head;
        Node next = null;

        // Traverse the list and reverse the links
        while (current != null) {
            next = current.next;   // Store the next node
            current.next = prev;   // Reverse the current node's
               pointer
            prev = current;        // Move prev and current one
               step forward
            current = next;
```

```
        }

        // The new head is the previous node at the end of the
           list
        return prev;
    }

    // Function to print the list
    public static void printList(Node head) {
        if (head == null) {
            System.out.println("List is empty.");
            return;
        }

        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        }
        System.out.println("NULL");
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Node head = null; // Initialize an empty list (head is
           null)

        // Insert elements at the beginning
        System.out.print("Enter the value to insert at the
           beginning: ");
        int value = sc.nextInt();
        head = insertAtBeginning(head, value);

        System.out.print("Enter the value to insert at the
           beginning: ");
        value = sc.nextInt();
        head = insertAtBeginning(head, value);

        System.out.print("Enter the value to insert at the
           beginning: ");
        value = sc.nextInt();
        head = insertAtBeginning(head, value);

        // Print the list before reversal
        System.out.println("List before reversal: ");
        printList(head);

        // Reverse the list
        head = reverseList(head);

        // Print the list after reversal
```

```
81          System.out.println("List after reversal: ");
82          printList(head);
83
84          sc.close();
85      }
86  }
```

5 Reversal of Linked List

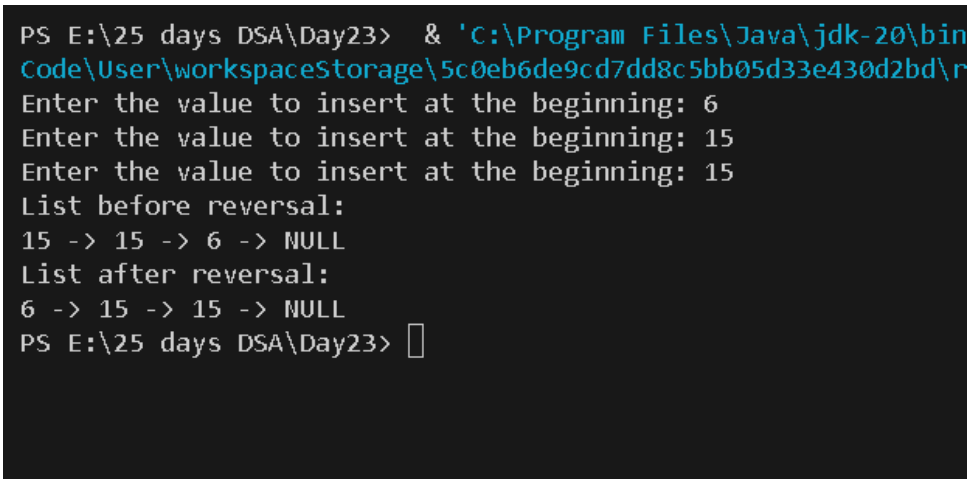

Figure 1: Linked List's Reversal

6 Conclusion

Reversing a singly linked list is a fundamental operation that can be useful in various algorithms, such as undo operations and traversals. The three-pointer technique is an efficient way to reverse a list in place, without requiring additional space, making the operation both time and space-efficient.