Linked List – Traversal of Linked List

1. Introduction

Traversal of a Linked List refers to the process of **visiting each node of the linked list exactly once** in a sequential manner.

Since linked lists do not support random access, traversal is the **only way** to access all elements.

Traversal always starts from the **head node** and continues until the end of the list.

---

2. What is Linked List Traversal?

Linked list traversal means:

- Starting from the **head**
- Moving to the **next node** using the link
- Repeating the process until **NULL** is reached

Each node is accessed in order.

---

3. Why Traversal is Required?

Traversal is required to:

- Print all elements
- Search for an element
- Count nodes
- Perform updates
- Apply algorithms on linked lists

Without traversal, data inside a linked list cannot be accessed.

---

4. Basic Idea of Traversal

The idea is simple:

- Use a temporary pointer (usually called current)
- Initialize it with head
- Move it node by node using current = current.next

---

5. Traversal Logic (Plain English)

1. Check if the linked list is empty
2. If empty, stop traversal
3. Set a pointer to the head node
4. While the pointer is not NULL:
    - Access the data of the current node
    - Move the pointer to the next node
5. Stop when NULL is reached

---

6. Step-by-Step Example

Linked List:

```
Head → 10 → 20 → 30 → 40 → NULL
```

Traversal steps:

```
Visit 10
Visit 20
Visit 30
Visit 40
Stop (NULL)
```

---

7. Visualization of Traversal

```
current
  ↓
[10] → [20] → [30] → [40] → NULL
```

The pointer moves forward **one node at a time**.

---

8. Traversal in Singly Linked List

- Only forward traversal is possible
- Nodes are visited from head to last node
- Reverse traversal is **not possible** without extra memory

---

9. Traversal in Doubly Linked List

- Traversal can be done:
  - Forward (head to tail)
  - Backward (tail to head)
- Each node has both next and previous links

---

10. Time and Space Complexity

Aspect	Complexity
Time Complexity	O(n)
Space Complexity	O(1)

Where n is the number of nodes.

---

11. Edge Cases in Traversal

- Empty list → no traversal
- Single node → visit once

- Large list → sequential traversal only

---

12. Advantages of Linked List Traversal

- Simple to implement
- Works for any list size
- No additional memory required
- Efficient for sequential access

---

13. Limitations of Traversal

- Slow compared to array access
- Cannot jump to a specific position
- Always requires visiting previous nodes

---

14. Real-World Applications

- Printing elements of a list
- Searching for data
- Updating node values
- Counting elements
- Applying algorithms like reversal or deletion

---

15. Summary

- Traversal means visiting each node once
- Starts from head and ends at NULL
- Uses a temporary pointer
- Time complexity is O(n)
- Essential for all linked list operations
