Linked Lists in DSA: Complete Beginner to Intermediate Guide

Subtitle: Learn how linked lists work, how to manipulate them, and solve common interview problems step by step.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Linked Lists are a fundamental data structure that helps you understand memory, pointers, and dynamic data handling. Unlike arrays, they are not stored in contiguous memory, which makes them powerful for insertions and deletions. This guide will help you build a strong foundation and solve common problems.

Step 1: What is a Linked List?

A Linked List is a **collection of nodes**, where each node contains:

- **Data** (value)
- **Pointer/Reference** to the next node

Structure:

[10] → [20] → [30] → null

Step 2: Types of Linked Lists

1. Singly Linked List

- Each node points to the next node
- Most commonly used

2. Doubly Linked List

- Each node has:
  - Pointer to next
  - Pointer to previous

3. Circular Linked List

- Last node points back to the first node

Step 3: Basic Operations

1. Creating a Node (JavaScript Example)

```
class Node {
  constructor(data) {
    this.data = data;
    this.next = null;
  }
}
```

2. Traversing a Linked List

```
let current = head;

while (current !== null) {
  console.log(current.data);
  current = current.next;
}
```

3. Insertion at Beginning

```
let newNode = new Node(5);
newNode.next = head;
head = newNode;
```

4. Deletion of First Node

```
head = head.next;
```

Step 4: Time Complexity

Operation	Time Complexity
Access	O(n)
Insert (beginning)	O(1)
Delete (beginning)	O(1)
Search	O(n)

Insight:

- Slower access than arrays
- Faster insertion/deletion

Step 5: Important Linked List Patterns

1. Fast & Slow Pointers

Used for:

- Detecting cycles
- Finding middle element

```
let slow = head;

let fast = head;


while (fast && fast.next) {

  slow = slow.next;

  fast = fast.next.next;

}
```

2. Reversing a Linked List

```
let prev = null;

let current = head;
```

```
while (current !== null) {
  let next = current.next;
  current.next = prev;
  prev = current;
  current = next;
}
head = prev;
```

Step 6: Important Problems You Must Practice

1. Reverse a Linked List

(Already covered above)

2. Find Middle Element

Use fast & slow pointer technique.

3. Detect Cycle (Loop)

```
let slow = head;
let fast = head;

while (fast && fast.next) {
  slow = slow.next;
  fast = fast.next.next;

  if (slow === fast) {
    console.log("Cycle detected");
    break;
  }
}
```

4. Merge Two Sorted Lists

```
function merge(l1, l2) {
  let dummy = new Node(0);
  let current = dummy;

  while (l1 && l2) {
    if (l1.data < l2.data) {
      current.next = l1;
      l1 = l1.next;
    } else {
      current.next = l2;
      l2 = l2.next;
    }
    current = current.next;
  }

  current.next = l1 || l2;
  return dummy.next;
}
```

Step 7: Common Mistakes

- Losing reference to the head node
- Not handling **null pointers**
- Infinite loops due to incorrect traversal
- Forgetting to update links properly

Step 8: Practice Plan

Day 1:

- Understand structure + traversal
- 5 basic problems

Day 2–3:

- Insert/Delete operations
- 8–10 problems

Day 4–5:

- Reverse + fast/slow pointers
- 10–12 problems

Day 6–7:

- Mixed problems + revision
- 15 problems

Mini Exercises

1. Count number of nodes in a list
2. Search for a value
3. Reverse a linked list
4. Find the middle node

Key Takeaways

- Linked Lists are **dynamic and flexible** data structures
- Master **pointer manipulation** to solve problems
- Learn patterns like **fast/slow pointers** and **reversal**
- Handle edge cases carefully (null, single node)
- Practice regularly to build confidence

Visit haas.dev for more DSA guides, interview prep resources, and coding practice material.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app