Understanding Linked List

Overview

- What is a Linked List?
- Key Features of Linked Lists
- Types of Linked Lists
- Structure of a Linked List
- Common Operations
- When to Use Linked Lists?

DEFINITION: LINKED LIST

- A linked list is a dynamic data structure that stores elements (called nodes) in a sequential order using pointers.

- Unlike arrays, linked lists allocate memory dynamically, meaning they do not require a predefined size and can grow or shrink during runtime.

Key Features of Linked List

Dynamic Size

- Can grow or shrink during runtime, unlike arrays with fixed size.

Efficient Insertions/Deletions

- Inserting or deleting a node requires only pointer adjustments, which avoids the overhead of shifting elements as in arrays.

Sequential Access

- Linked lists do not allow direct access to elements by index, as each element must be accessed sequentially starting from the head.

Types of Linked List

Singly Linked List

- Each node points to the next node.
- Traversal is one-directional.
- Example:

Doubly Linked List

- Each node points to both its previous and next nodes.
- Allows bidirectional traversal.
- Example:

Circular Linked List

- The last node links back to the first node.
- Can be singly or doubly circular.
- Example:

Structure of a Linked List

Node

- A building block of a linked list. Contains two parts
  - Data: Holds the value of the element.
  - Next: A pointer/reference to the next node in the sequence.

Head

- Points to the first node in the linked list.

Tail (Optional)

- Points to the last node in some implementations
- It is more common in doubly and circular linked lists

Common Operations

Traversal

- Visiting each node in sequence starting from the head.

Insertion

- Adding a new node either at the beginning, end, or specific position by adjusting pointers

Deletion

- Removing a node by updating the pointer of the previous node

Search

- Iterating through the list to find a node with specific data

- Traversal/Search: O(n)
- Insertion/Deletion: O(1) at the head; O(n) for other positions

When to Use Linked Lists?

Use when you

- Need dynamic memory allocation.
- Perform frequent insertions/deletions.
- Implement other data structures like stacks, queues, or adjacency lists for graphs.

Avoid when

- Random access is a priority.
- Memory usage is critical, as linked lists use more memory due to pointer storage.