Linked List

A linked list is a linear data structure that consists of a sequence of elements, where each element is a separate object known as a node. Each node contains two parts:

1. Data: Stores the actual value or data.

2. Next: A reference (or pointer) to the next node in the sequence.

Linked lists can be categorized into different types based on their structure:

- **Types of Linked Lists**

1. Singly Linked List:

- Each node has a single link to the next node.

- The last node's next reference points to `null`, indicating the end of the list.

2. Doubly Linked List:

- Each node has two links: one to the next node and one to the previous node.

- Allows traversal in both forward and backward directions.

- The first node's previous reference and the last node's next reference are `null`.

3. Circular Linked List:

- Similar to a singly linked list, but the last node points back to the first node, forming a circle.

- Can also be doubly circular, where the first node's previous reference points to the last node.

4. Circular Doubly Linked List:

- Combines properties of both doubly linked lists and circular linked lists.

- Each node points to both the next and previous nodes, and the last node points back to the first node and vice versa.

- **Characteristics of Linked Lists**

- Dynamic Size: The size of the linked list can grow or shrink as nodes are added or removed.

- Ease of Insertion/Deletion: Adding or removing nodes is more efficient compared to arrays since there is no need to shift elements.

- Sequential Access: Nodes are accessed sequentially starting from the head, making random access inefficient compared to arrays.

- **Basic Operations**

1. Insertion: Adding a node at the beginning, end, or a specified position.

2. Deletion: Removing a node from the beginning, end, or a specified position.

3. Traversal: Visiting each node in the list to process the data.

4. Searching: Finding a node containing a specific value.

- **Advantages**

- Dynamic Size: Can grow or shrink as needed without requiring contiguous memory.

- Ease of Modification: Inserting or deleting nodes does not require shifting elements.

- **Disadvantages**

- Memory Usage: Each node requires extra memory for storing the reference to the next (and possibly previous) node.

- Access Time: Accessing elements requires traversal from the head, resulting in O(n) time complexity for search operations.

- **Use Cases**

Linked lists are often used in scenarios where frequent insertion and deletion of elements are required, such as:

- Implementing data structures like stacks, queues, and graphs.

- Managing dynamic memory allocation.

- Handling runtime growth of data, such as in dynamic arrays or lists in programming languages.