LINKED LISTS

Definition

A Linked List is a linear data structure in which elements are stored in separate objects called nodes. Unlike arrays, which store elements in contiguous memory locations, linked lists store nodes in different memory locations that are connected using pointers (references).

Each node contains:

1. Data – The value stored in the node

2. Pointer (Reference) – The address of the next node

The first node is called the Head, and the last node points to None.

Types of Linked Lists

1. Singly Linked List

Each node points only to the next node.

2. Doubly Linked List

Each node points to both the next node and the previous node.

3. Circular Linked List

The last node points back to the head node instead of pointing to None.

Key Operations

Insert – Add a new node to the list

Delete – Remove a node from the list

Traverse – Move through all nodes in sequence

Search – Find a specific element

Implementation in Python (Singly Linked List)

```
# Node class
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
```

```
# Linked List class
class LinkedList:
    def __init__(self):
        self.head = None

    # Insert at end
    def append(self, data):
        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            return

        current = self.head
        while current.next:
            current = current.next

        current.next = new_node

    # Display list
    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")



# Creating and testing the linked list
```

```
ll = LinkedList()
ll.append("A")
ll.append("B")
ll.append("C")

ll.display()
```

Output:

```
A -> B -> C -> None
```

Time Complexity

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

Linked lists require traversal to access elements, which makes random access slower compared to arrays.

Applications of Linked Lists

1. Music Streaming Applications

Applications such as Spotify use linked lists to manage playlists.

Each song can be treated as a node, with references to the next and previous songs (doubly linked list).

Reason:

It allows users to move forward and backward efficiently between songs.

2. Web Browsers

Web browsers such as Google Chrome use doubly linked lists to manage browsing history.

When a user:

Clicks Back, the browser moves to the previous node

Clicks Forward, the browser moves to the next node

Reason:

Two-way traversal is required for back and forward navigation.

3. Operating Systems

Operating systems use linked lists to manage running processes and memory allocation.

Reason:

Processes are frequently added and removed, and linked lists allow dynamic memory management without resizing.

4. Image Viewers and Media Applications

Applications that allow users to move to the next or previous image often use doubly linked lists.

Reason:

Efficient navigation in both directions is required.

How Linked Lists Work Within Systems

Within computer systems:

Memory is allocated dynamically.

Each node stores a reference (memory address) to the next node.

The system follows these references to traverse the list.

Data does not need to be stored in contiguous memory.

This makes linked lists suitable for systems where:

The size of data changes frequently.

Insertions and deletions occur often.

Memory flexibility is required.