CWB-Module-3

Circular Linked List and Doubly Linked List

Recap of Singly Linked List

Definition: Linear collection of nodes (data + pointer to next node).

| Head |--->| **A** |--->| **B** |--->| **C** |---> NULL

Limitations:

- Can only traverse forward.
- No direct access to previous node.
- Requires O(n) time for tail operations.

Circular Linked List (CLL)

Definition:

- Last node points back to the head (instead of `NULL`).
- Types: Singly CLL and Doubly CLL.

Head → [A] → [B] → [C] → [D] → Head

Use Cases:

- Round-robin scheduling.
- Music playlists (loop mode).

1. Insertion at Head

Goal: Add a new node as the first node while maintaining circularity.

Steps:

1. **Create New Node**: `[New|•]`
2. **Link New Node**:
   - New node's `next` points to current head: `[New|•] → [Head]`
3. **Update Tail's Next**:
   - Traverse to the last node (tail) where `tail.next == head`.
   - Set `tail.next = New` (now: `[Tail] → [New]`).
4. **Update Head**: `Head = New`

2. Insertion at End

Goal: Add a new node as the last node while maintaining circularity.

Steps:

1. **Create New Node**: `[New|•]`
2. **Link New Node**:
   - New node's `next` points to head: `[New] → [Head]`
3. **Update Tail's Next**:
   - Find the current tail (`tail.next == head`).
   - Set `tail.next = New` (now: `[Old Tail] → [New] → [Head]`).

1. Delete Head Node

Steps:

1. Find the **tail** node (where `tail.next == head`).
2. Update `tail.next` to point to `head.next`.
3. Move `head` to `head.next`.
4. Free the old head.

2. Delete Tail Node

Steps:

1. Traverse to the **node before tail** (`prev`).
2. Set `prev.next = head` .
3. Free the old tail.

Introduction to Doubly Linked List

What is a Doubly Linked List?

- A linked list where each node contains:
  - Data
  - Next pointer → Points to the next node
  - Prev pointer → Points to the previous node
- Head → Points to the first node (`prev = NULL`)
- Tail → Points to the last node (`next = NULL`)

NULL ⇄ [A] ⇄ [B] ⇄ [C] ⇄ NULL

Advantages of DLL

1. Bidirectional Traversal
    - Can move forward and backward.
2. Efficient Deletions
    - O(1) deletion at head/tail (if tail pointer is maintained).
3. Easier Reverse Traversal
    - No need for recursion/stack (unlike Singly Linked List).