Introduction to DLL

A Doubly Linked List (DLL) contains an extra pointer, typically called previous pointer, together with next pointer and data which are there in singly linked list.

Doubly Linked List

- Using two pointers prev and next, it can traverse in both directions that is forward and backward
- the node structure of DLL is similar to the SLL, expect that SLL is having only one pointer which points to next node BUT DLL is having an other pointer which points to previous node also.
- The Main difference between SLL and DLL is that, because SLL is having only one pointer so it can traverse only in one direction, while DLL is having two pointer so it can traverse in both forward and backward direction.
- The last node address is always null, with that it is going to identify the last node, if you see in DLL the first node is having prev pointer as null value, because it was the first node and same last node next pointer is also having null value, because it is the last node.

Advantages over singly linked list

- A DLL can be traversed in both forward and backward direction.
- The delete operation in DLL is more efficient if pointer to the node to be deleted is given.
- We can quickly insert a new node before a given node.
- In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using previous pointer.

Disadvantages over singly linked list

- Every node of DLL Require extra space for an previous pointer.
- All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers.

```
 class DDLNode {
    constructor(val) {
        this.data = val
        this.next = null
        this.prev = null
    }
}

let head = null

function insert_at_beginning(val) {
    let new_node = new DDLNode(val)

    new_node.next = head

    if (head != null){
        head.prev = new_node
    }

    head = new_node

    return head
}

function insert_at_end(val) {
    let new_node = new DLLNode(val)

    if (head == null) {
        head = new_node
        return head
    } else {
        let cur = head

        while (cur.next != null) {
            cur = cur.next
        }

        cur.next = new_node
        new_node.prev = cur

        return head
    }
```

```
}

function insert_after_val(val, new_data) {
    let new_node = new DLLNode(new_data)

    let cur = head

    while (cur.data != val) {
        cur = cur.next
    }

    new_node.next = cur.next
    new_node.prev = cur

    if (cur.next != null) {
        cur.next.prev = new_node
    }

    cur.next = new_node

    return head
}

function insert_before_val(val, new_data) {
    let new_node = new DLLNode(new_data)

    let cur = head

    while (cur.data != val) {
        cur = cur.next
    }

    new_node.next = cur
    new_node.prev = cur.prev

    if (cur.prev != null) {
        cur.prev.next = new_node
    } else {
        head = new_node
    }

    cur.prev = new_node

    return head
}

function delete_beginning() {
    let temp = head
    head = head.next
    head.prev = null

    return temp
}

function delete_end() {
    if (head.next == null) {
        let temp = head
        head = null
        return temp
    } else {
        let cur = head
        while (cur.next != null) {
            cur = cur.next
        }

        let temp = cur
        cur.prev.next = null
        cur.prev = null

        return temp
    }
}

function delete_after_val(val) {
    let cur = head

    while (cur.data != val) {
        cur = cur.next
    }

    let temp = cur.next

    if (cur.next != null) {
        cur.next = cur.next.next
        cur.next.prev = cur
    } else {
        cur.next = null
    }

    return temp
}

function delete_before_val(val) {
    let cur = head

    while (cur.data != val) {
        cur = cur.next
    }

    let temp = cur.prev

    if (cur.prev != null) {
        cur.prev = cur.prev.prev
        cur.prev.next = cur
    } else {
        cur.prev = null
        head = cur
    }

    return temp
}
```