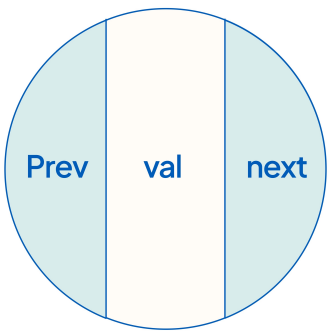
Agenda:

1. **What is doubly linked list?**
2. **LRU Cache**
3. **Check if LL is palindrome**

Doubly LinkedList

```
class Node {
        int val;
        Node next;
        Node prev;
        public Node (int v) {
                this.val = v;
                this.next = null;
                this.prev = null;
        }
}
```



Scenario

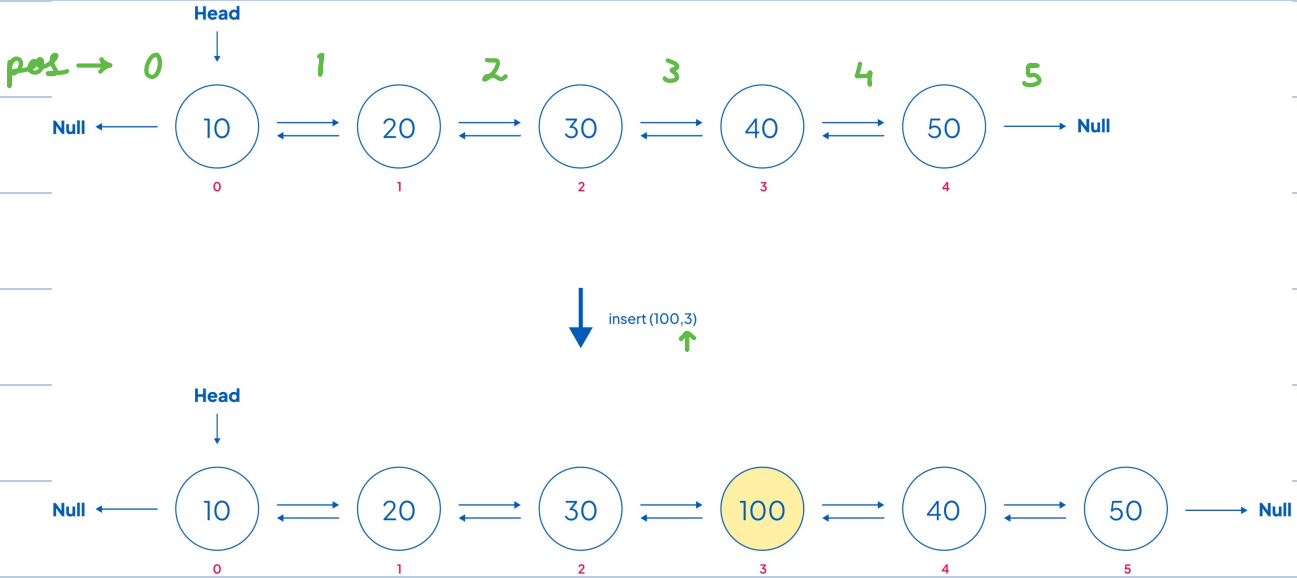
Spotify wants to enhance its user experience by allowing users to navigate through their music playlist seamlessly using "next" and "previous" song functionalities.

Problem

You are tasked to implement this feature using a doubly linked list where each node represents a song in the playlist. The system should support the following operations:

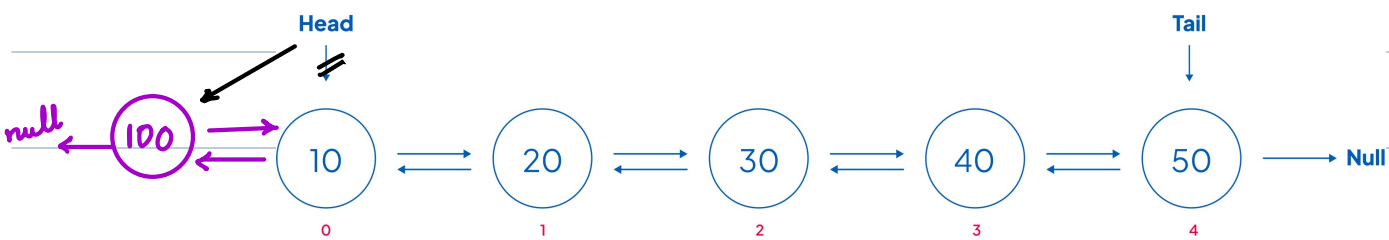
- **Add Song :** Insert a new song into the playlist. If the playlist is currently empty, this song becomes the "Current song".
- **Play Next Song :** Move to the next song in the playlist and display its details.
- **Play Previous Song :** Move to the previous song in the playlist and display its details.
- **Current Song :** Display the details of the current song being played.

Insertion in Doubly LinkedList

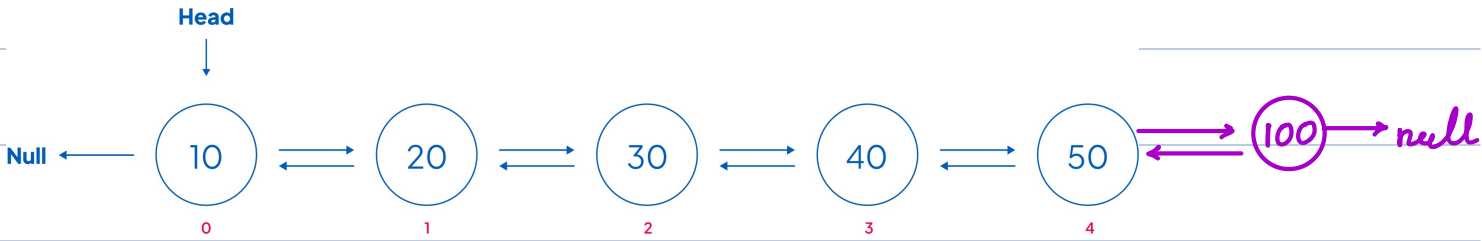


Edge - Case

- **At index 0**



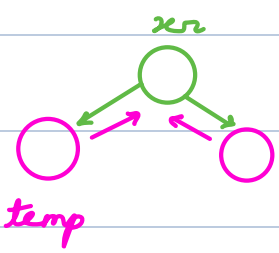
- **At index 5**



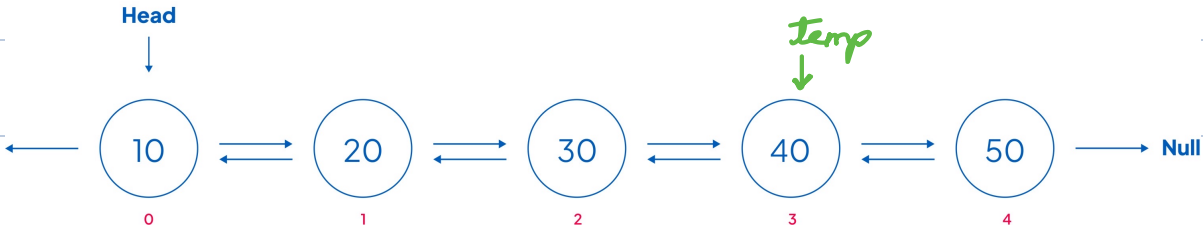

```
// Head , data , pos        0 <= pos <= length of LL
xr = new Node (data)
if ( pos == 0) {
    xr.next = Head
    if ( Head != null )    Head.pre = xr
    return xr    // new Head
}

temp = Head
for i → 1 to (pos-1) {
    temp = temp.next
}

xr.pre = temp
xr.next = temp.next
if ( temp.next != null)    temp.next.pre = xr
temp.next = xr
return Head                TC = O(N)    SC = O(1)
```

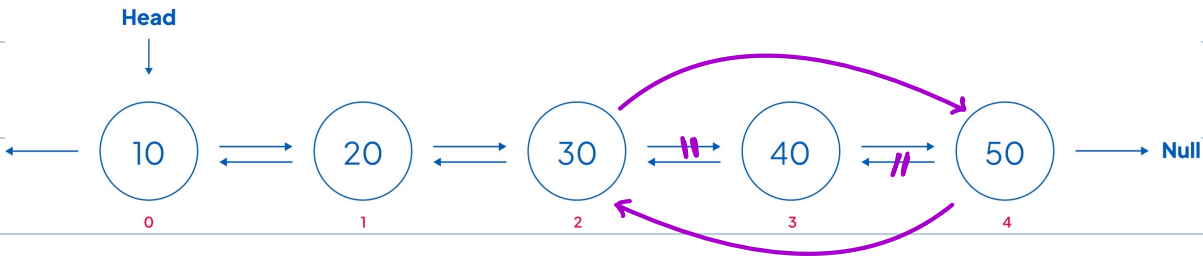


Delete a node from D.L.L



Val = 40

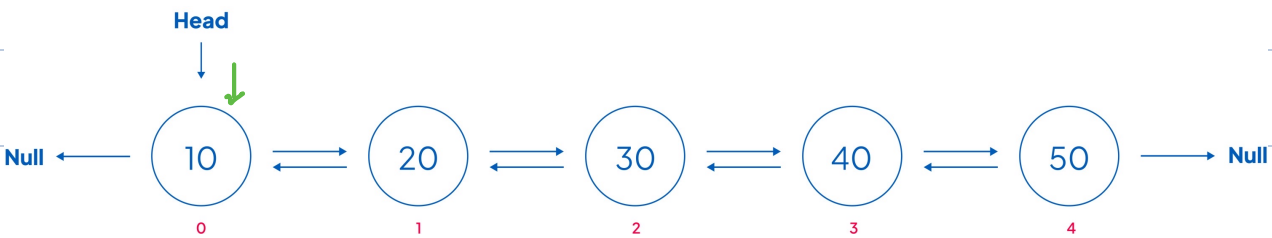
Delete the node with val = 40 →

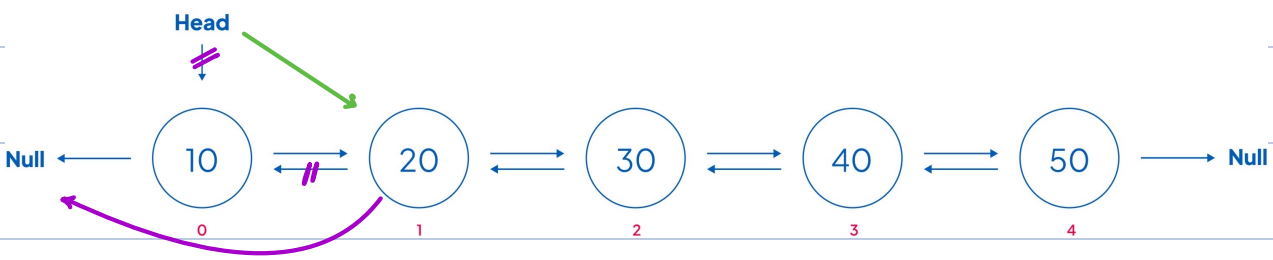


Edge - Case

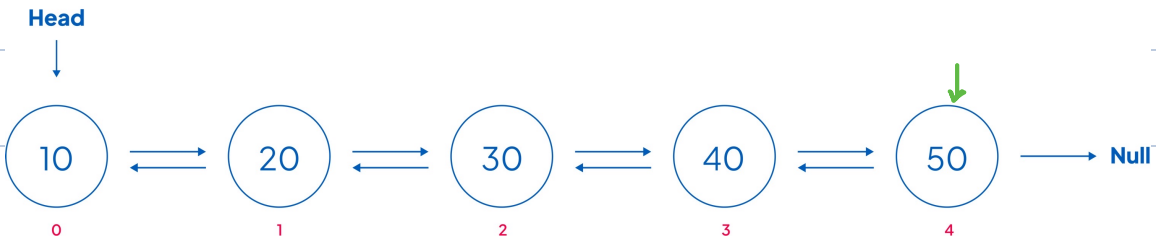
- **idx → 0**

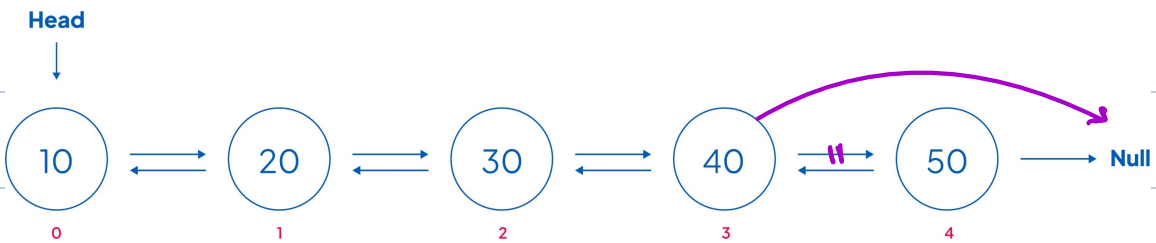
val → 10



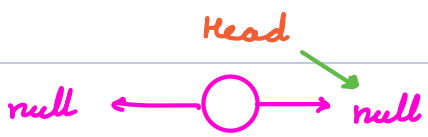


- **last - node** **val → 50**






```
temp = Head
while ( temp != null) {
    if ( temp.val == X)   break        // X: data to delete
    temp = temp.next
}
```




```
if ( temp == null ) return Head
if ( temp.pre == null )      Head = Head.next
if ( temp.next != null )     temp.next.pre = temp.pre
if ( temp.pre != null)       temp.pre.next = temp.next
return Head
```

TC = O(N) // searching SC = O(1)

L.R.U Cache (Google)

↓

Least Recently Used

Q → Given a running stream of integers & a fixed memory M (SC = O(M)). Maintain most recent M items i.e. if the memory is full, delete least recent element.

i/p → 7 3 9 2 6 10 14 2 10 15 8 14

M = 5

~~7~~ ~~3~~ ~~9~~ ~~2~~ ~~6~~

~~10~~ ~~14~~ 2 10 15

8 14

new i/p → x

search(x)

✓ present

1) delete(x)

2) insert(x)

x not present

size

full

1) deleteLRU()

2) insert(x)

not full

insert(x)

searching(x) → ✗ HashSet / ✓ HashMap <data, node>

deleteLRU() → Array (static / Dynamic) ✗

(deleteHead()) LinkedList (Single / ✓ Doubly) ✗

delete (x) → data x → xn = findNode(x) ✓

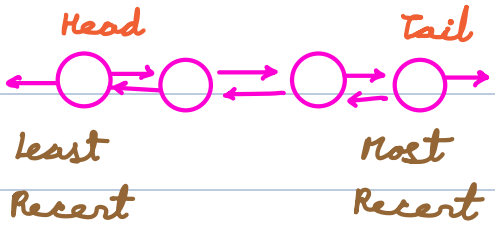
deleteNode(xn)

insert (x) → ✓

1) search (x) → hm.contains (x)

2) deleteLRU() → x = deleteHead()

hm.remove (x)



3) delete (x) → xn = hm.get (x)

deleteNode (xn)

4) insert (x) → xn = new Node (x)

insertTail (xn)

hm.put (x, xn)

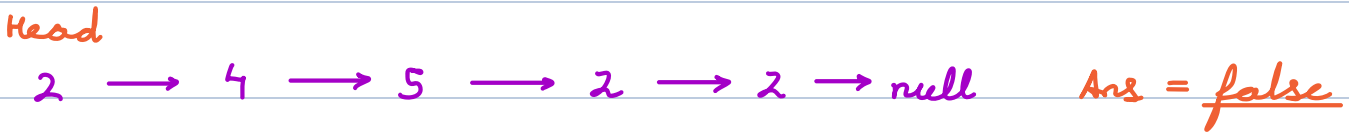
TC = O(1) SC = O(M)

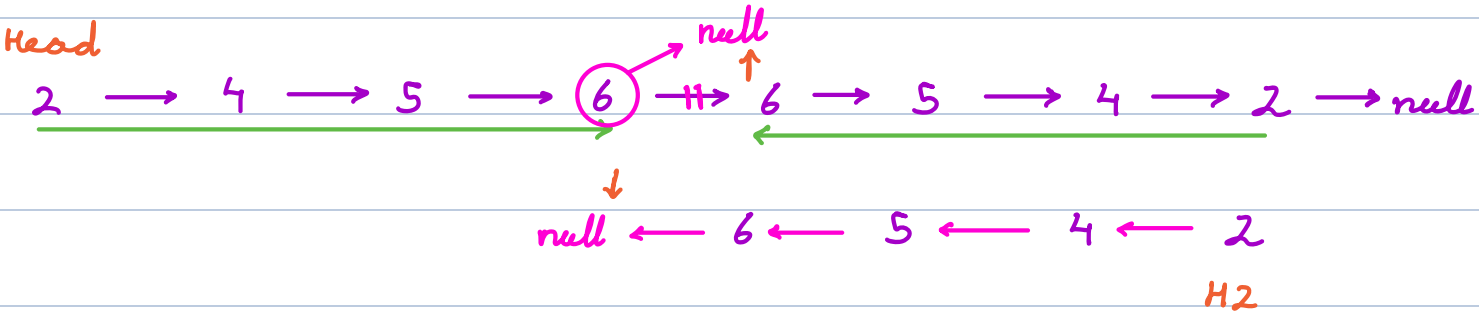
Q → check if the given linked list is palindrome.

SC = O(1)

→ = ←

x = rev(x)





Sol → 1) Find middle element.

2) Reverse second half of LL.

3) Compare first & second half.

TC = O(N) SC = O(1)