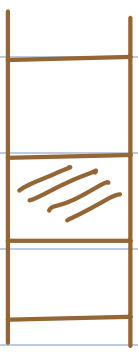
SCALER

LinkedList - 1

TABLE OF CONTENTS

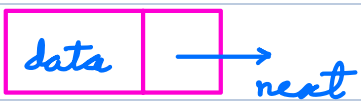
Issues with Array

Continuous memory allocation.

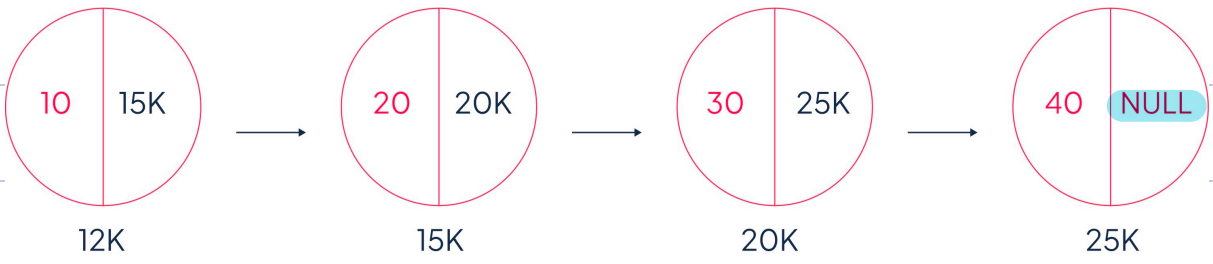


Linked List → Linear Data Structure

Non-continuous memory allocation.



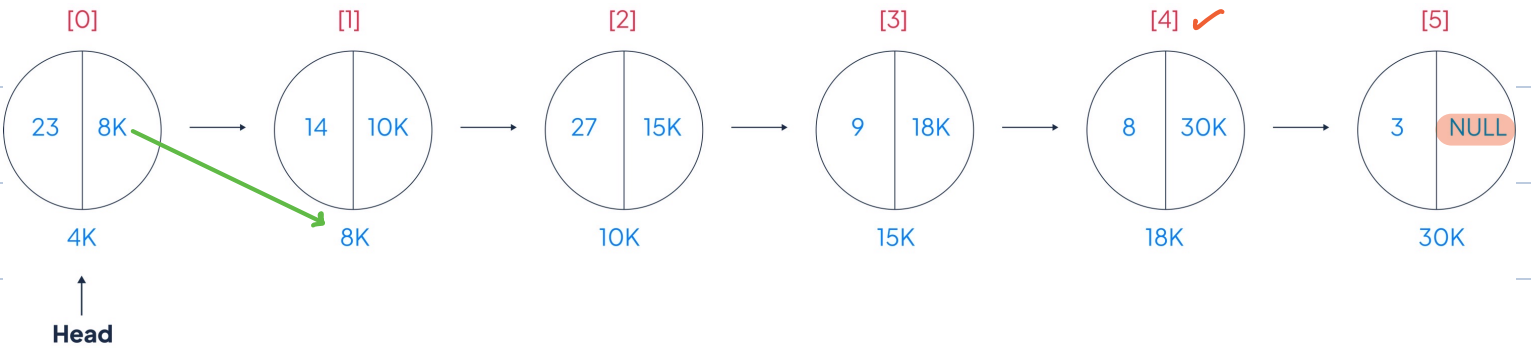
LinkedList



Head

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

How to access the element at kth.idx in a linked-list

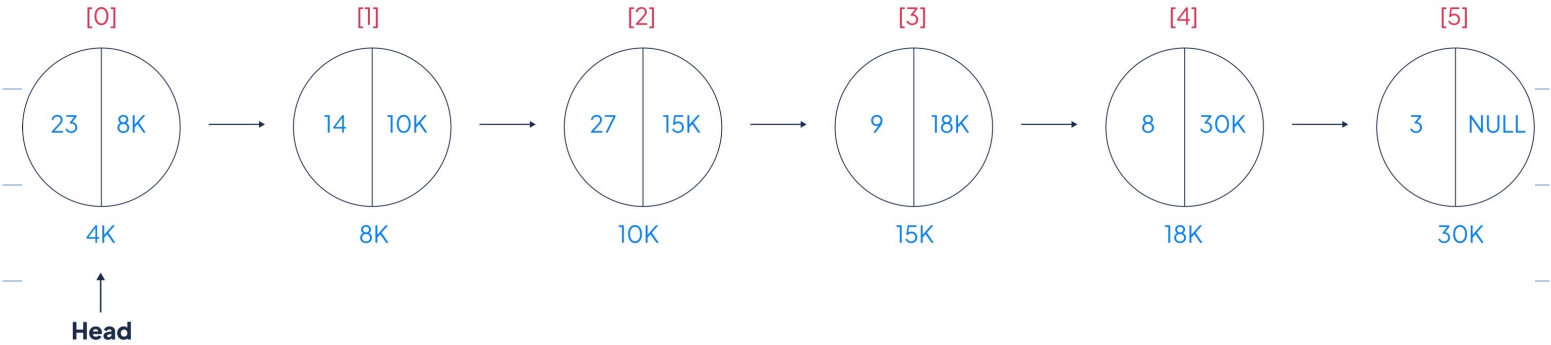


idx = 4 → 8

```
if (Head == null)    return -1
temp = Head
for i → 1 to K {
    temp = temp.next
    if (temp == null)    return -1
}
return temp.val
```

TC = O(K)

Search in linked-list



search(3) → true

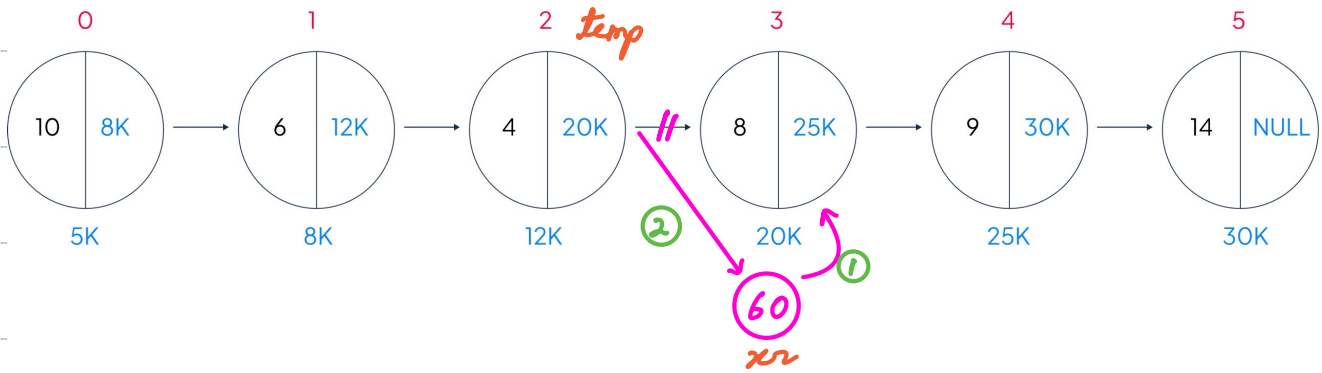
X

```
temp = Head
while (temp != null) {
    if (temp.val == X)   return true
    temp = temp.next
}
return false
```

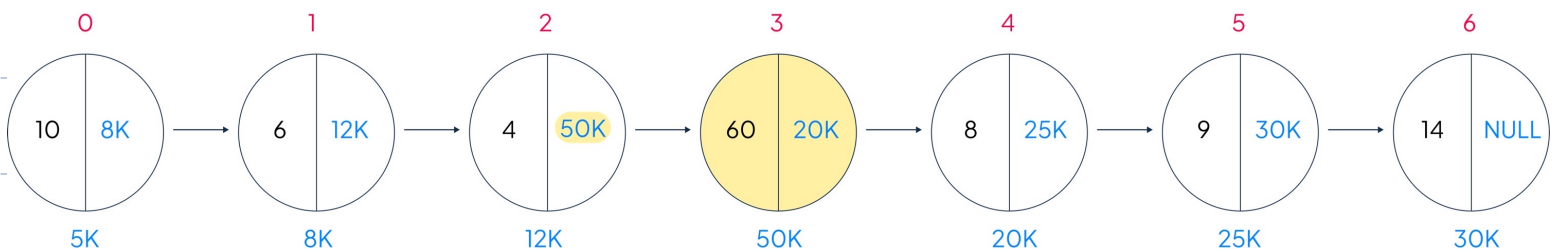
TC = O(N) // Linear Search

Binary Search → Not possible ∵ we cannot jump to middle element.

Insertion in linked-list at Kth index

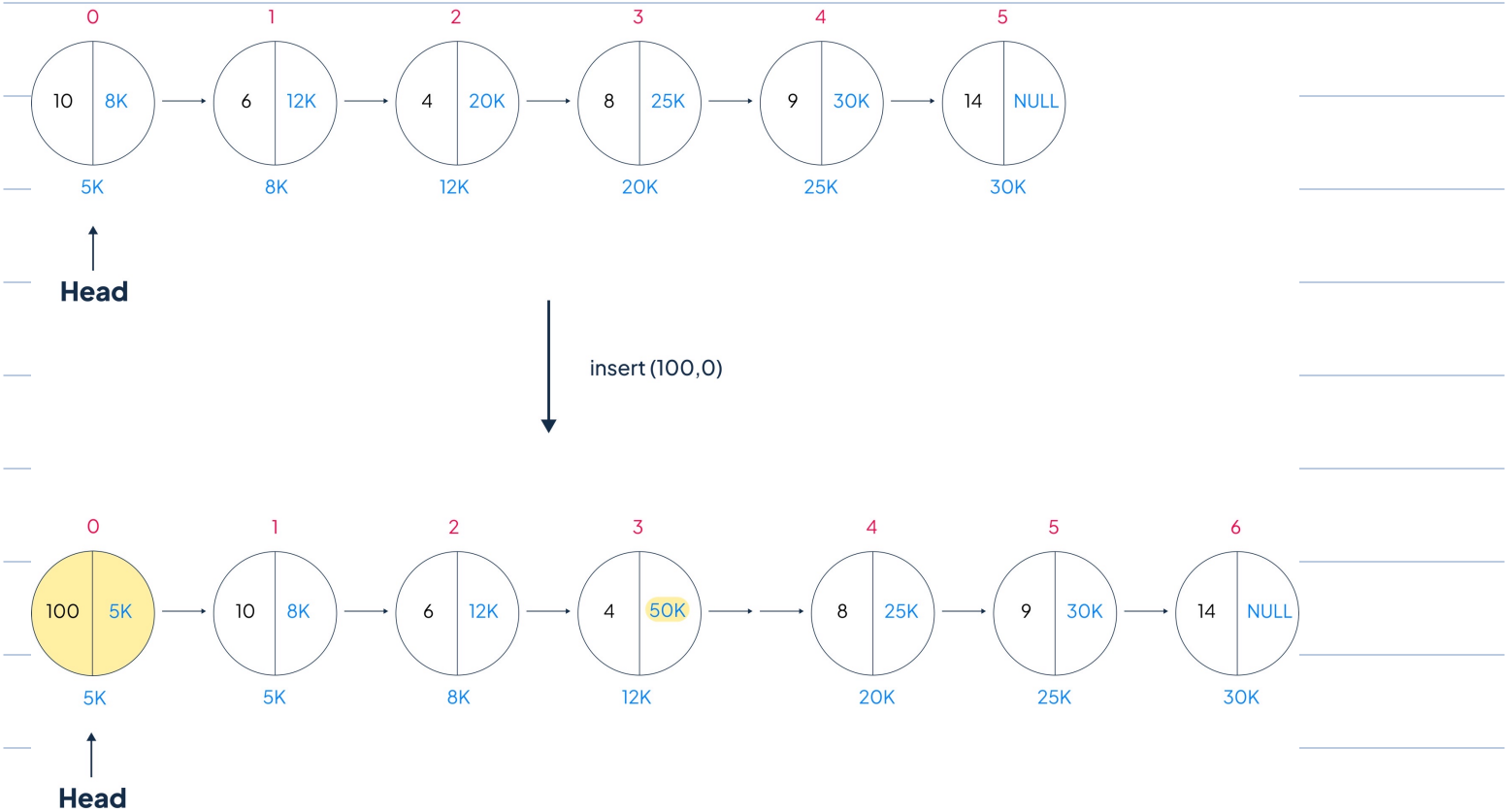


idx value
insert At (3,60)

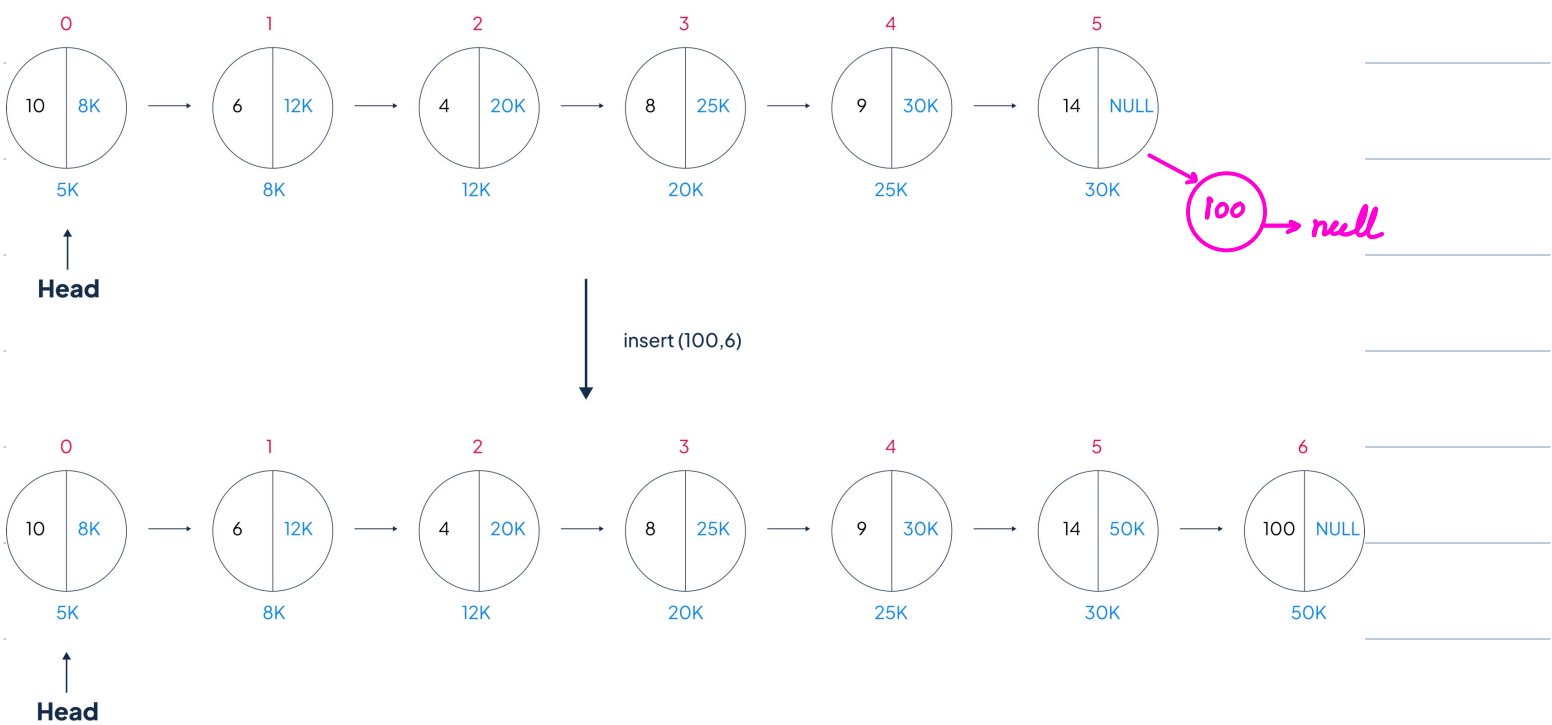


Edge-cases

If K = 0



If K = N



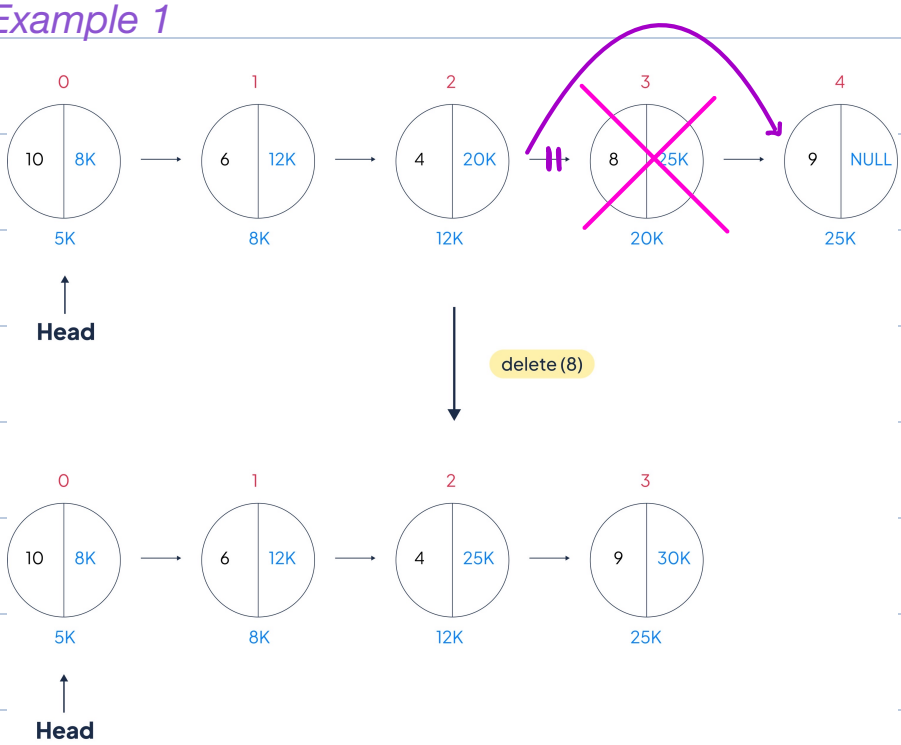
```
// insert (X, K)
Node xr = new Node (X)
if (K == 0) {
    xr.next = Head
    Head = xr
    return Head
}
temp = Head
for i → 1 to (K-1) {
    temp = temp.next
}
xr.next = temp.next
temp.next = xr
return Head
```

TC = O(K)

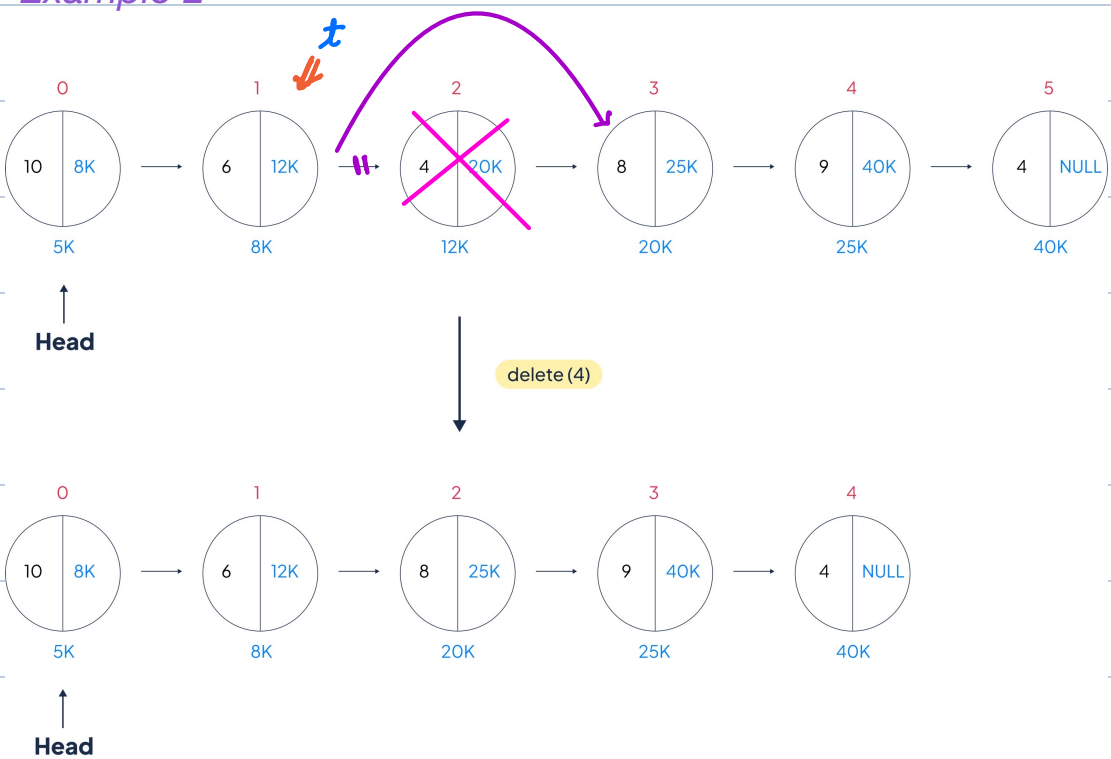
Deletion in linked-list

Delete the first occurrence of value X in the given lined-list. If element is not present, leave as is.

Example 1



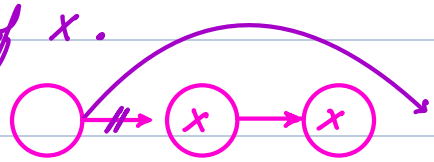
Example 2



```
if ( Head == null)  return null
if ( Head.val == x ) { Head = Head.next
                       return Head }

temp = Head
while ( temp.next != null ) {
    if ( temp.next.val == x ) {
        temp.next = temp.next.next
        break
    }
    temp = temp.next
}

return Head
```

TC = O(N)

H.W → Delete all occurrences of x.



Scenario

Comment

OnePlus has a lineup of N mobile phones ready in their manufacturing line. It has detected a defect in one of their phone models during production. They have decided to recall all phones of the defective model from their manufacturing line. Your task is to help OnePlus remove all defective phones from their production lineup efficiently

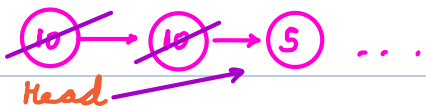
Problem

You are given a linked list A of N nodes where each node represents a specific model type of a OnePlus mobile phone in the manufacturing line. Each node contains an integer representing the model number of the phone. You will also be given an integer B which represents the model number of the defective phone that needs to be removed.

Your goal is to remove all nodes (phones) from the linked list that have the model number B and return the modified linked list representing the updated manufacturing line.

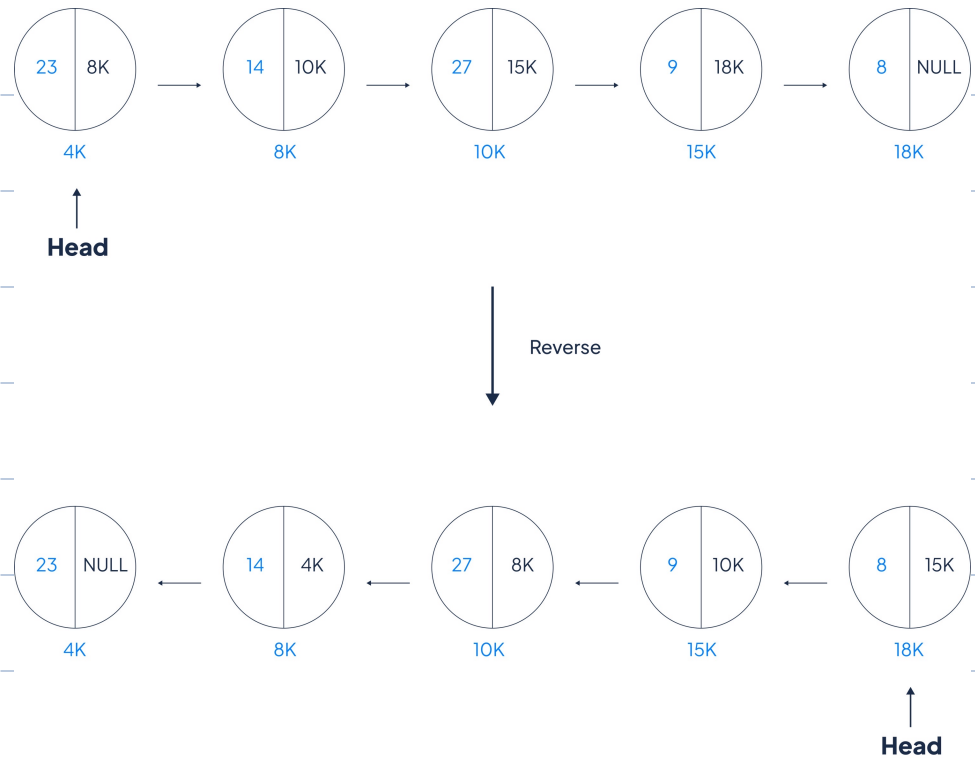
Delete all occurrences of B from list A.

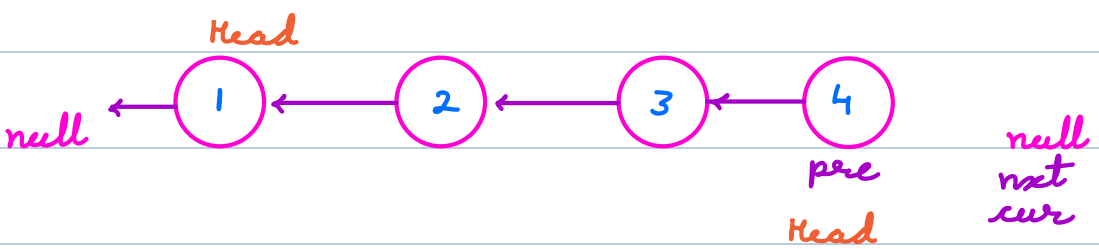
```
if ( Head == null) return null
while ( Head != null && Head.val == B )
        Head = Head.next          // B = 10
temp = Head
while ( temp.next != null ) {
    if ( temp.next.val == B )
        temp.next = temp.next.next
    else
        temp = temp.next
}
return Head
```



TC = O(N)

Reverse a linked-list



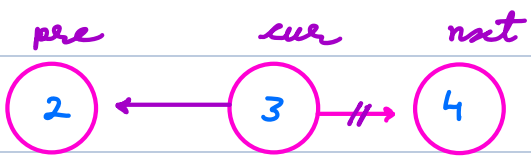


nxt = cur.next

cur.next = pre ✓

pre = cur

cur = nxt



```
cur = Head
pre = null
while (cur != null) {
    nxt = cur.next
    cur.next = pre
    pre = cur
    cur = nxt
}
return pre  // head
```

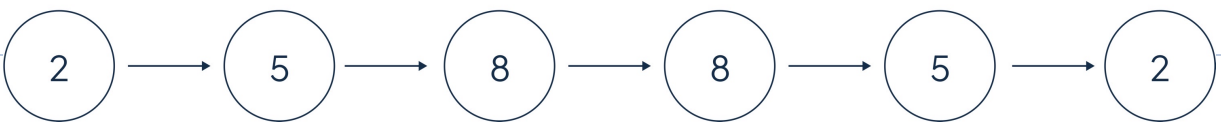
TC = O(N) SC = O(1)

- Check if a linked-list is a palindrome or not

"racecar" → = ←

X

✓



✓

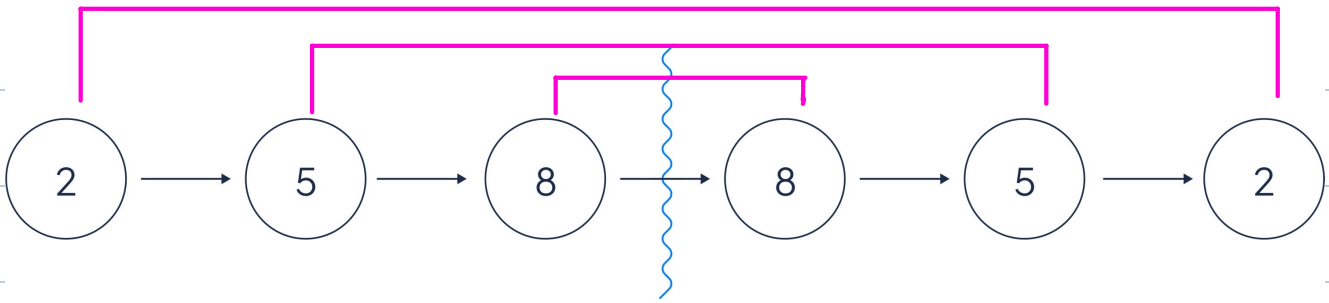
✓

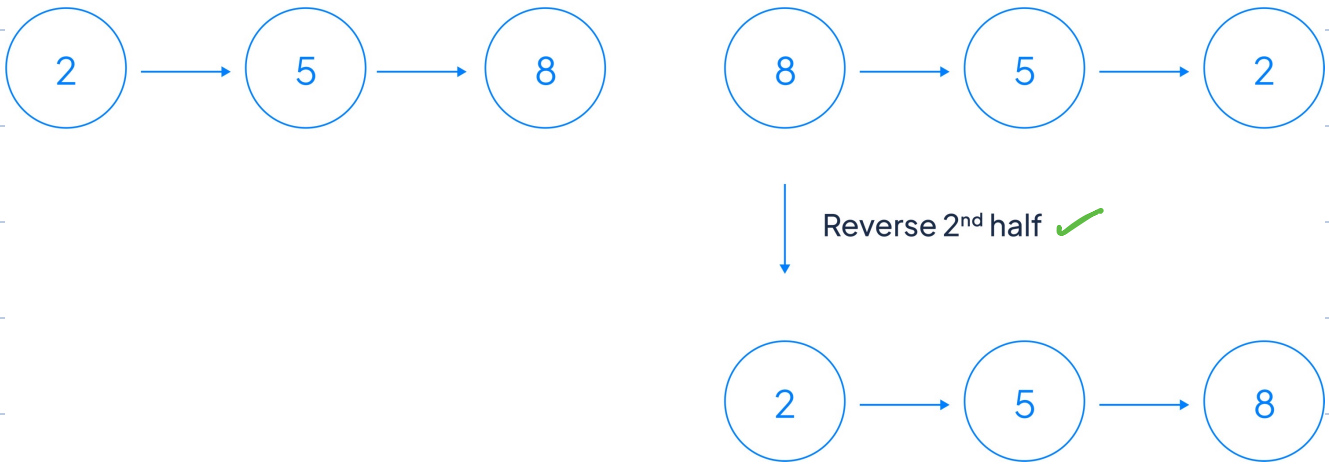
Sol 1 → 1) create copy of list

2) Reverse the copy

3) check if reverse of copy & original list is same.

TC = O(N) SC = O(N) → copy





1) Find middle element & divide the list. → Find size & travel half. ✓

2) Reverse second half. ✓ TC = O(N)

3) compare first half with reversed second half. ✓

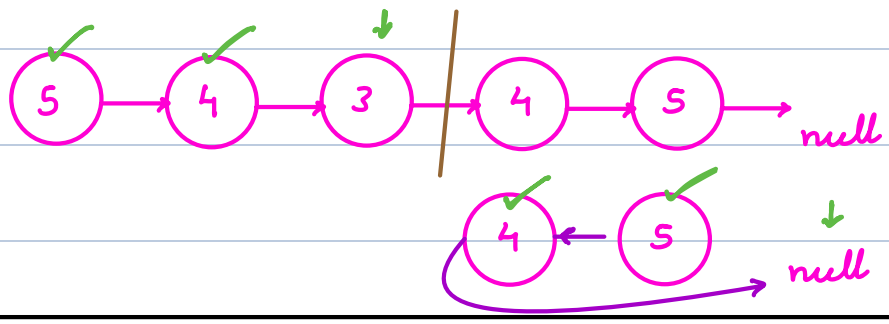
TC = O(N) SC = O(1)

If first half and reversed second half are equal

→ return true;

otherwise

→ return false;

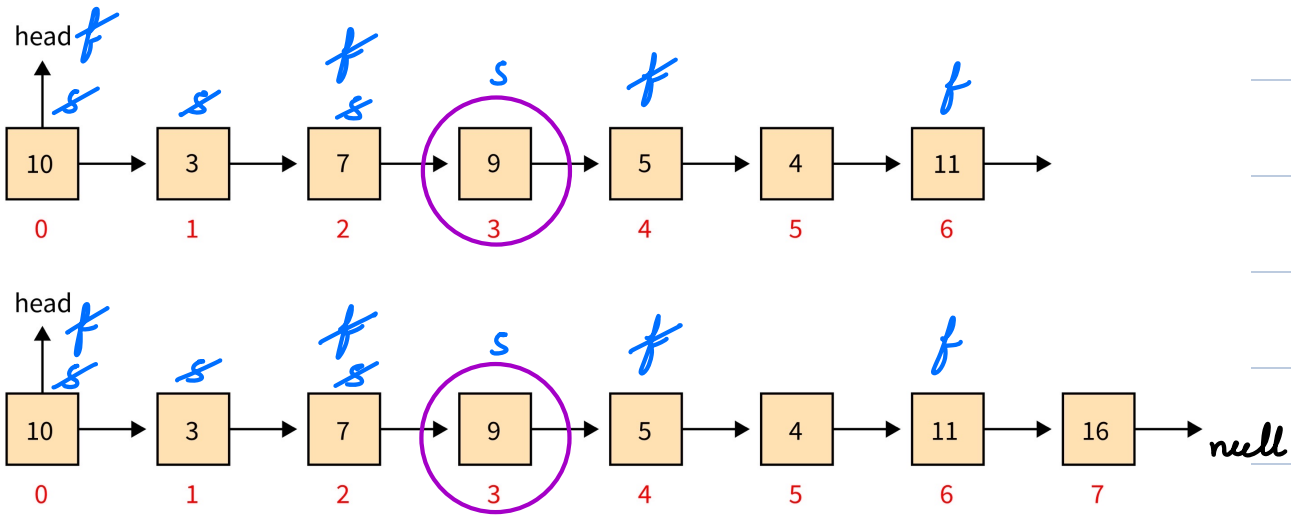


SCALER

LinkedList - 2

TABLE OF CONTENTS

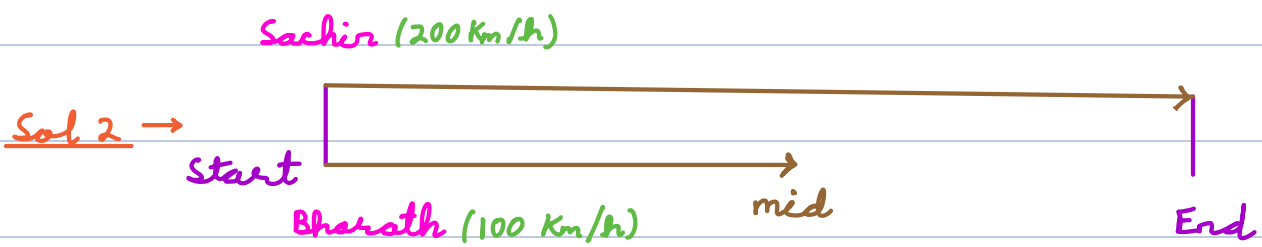
Middle of a Linked-list



Sol 1 → 1) Find length of linked list (N). → O(N)

2) Travel half length (N/2). → O(N/2)

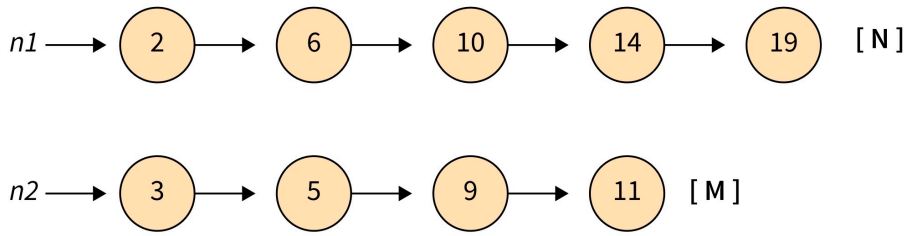
TC = O(N) SC = O(1)




```
if (Head == null) return null
s = f = Head
while (f.next != null && f.next.next != null) {
    s = s.next
    f = f.next.next
} return s
```

TC = O(N/2) = O(N) SC = O(1)

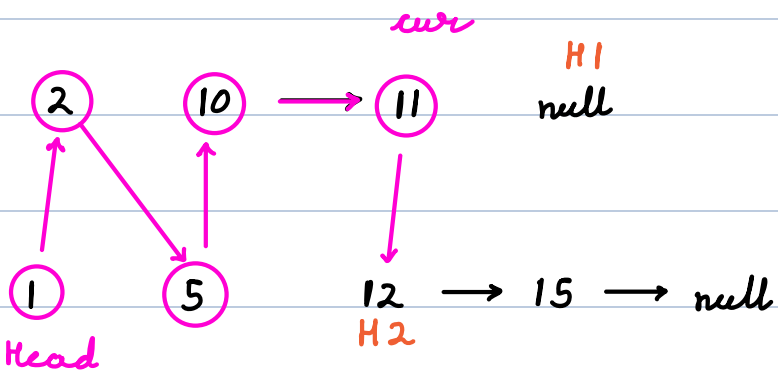
Merge Two Sorted Linked-list



Head 2 → 3 → 5 → 6 → 9 → 10 → 11 → 14 → 19 → null

H1 2 → 10 → 11 → null

H2 1 → 5 → 12 → 15 → null



if (H1 == null) return H2

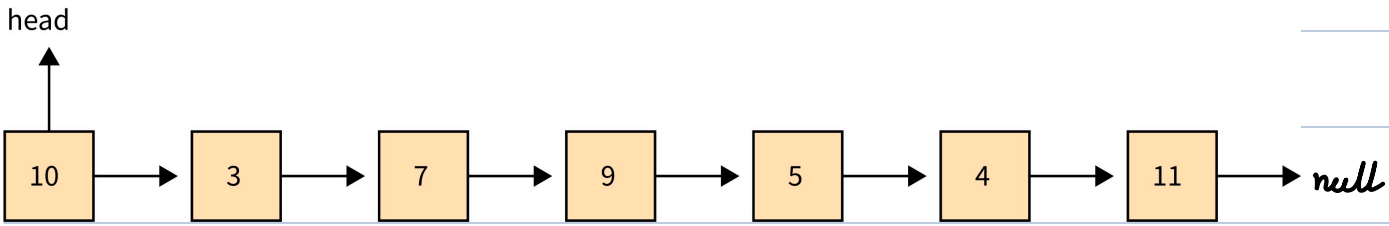
if (H2 == null) return H1

Head = null

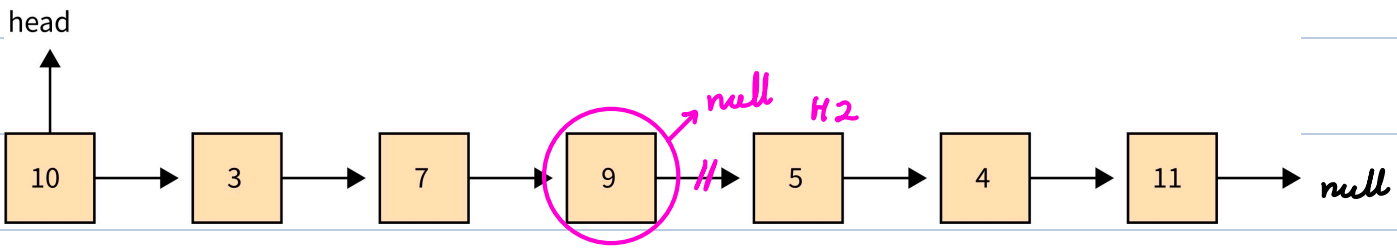
```
if (H1.data <= H2.data) {
    Head = H1
    H1 = H1.next
} else {
    Head = H2
    H2 = H2.next
}

cur = Head
while (H1 != null && H2 != null) {
    if (H1.data <= H2.data) {
        cur.next = H1
        H1 = H1.next
    } else {
        cur.next = H2
        H2 = H2.next
    } cur = cur.next
}

if (H1 == null)   cur.next = H2
if (H2 == null)   cur.next = H1
return Head
```

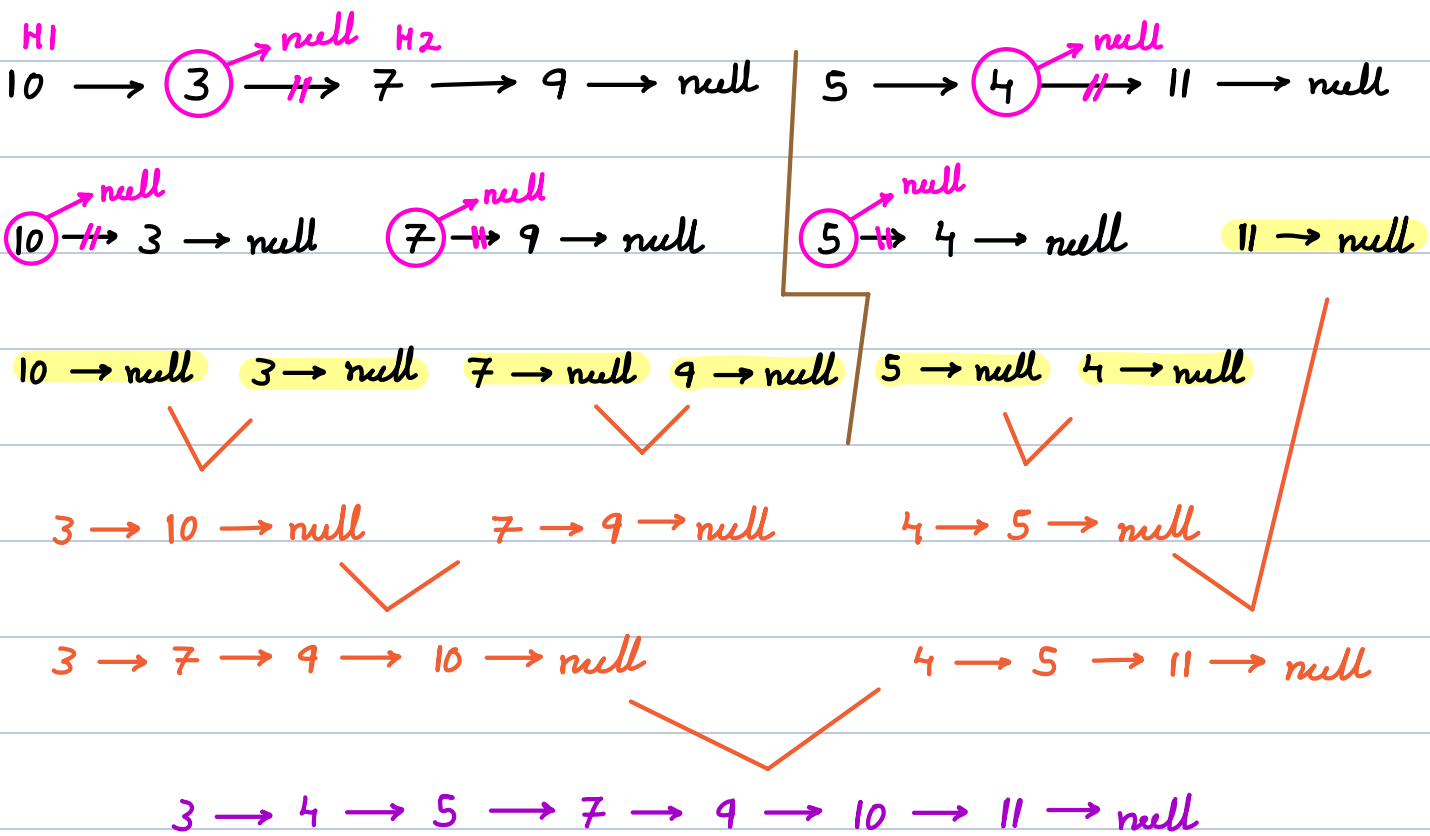
TC = O(N+M) SC = O(1)

Merge Sort a Linked-list



Head 3 → 4 → 5 → 7 → 9 → 10 → 11 → null





Idea

1. Find the middle node
2. Make recursive calls to sort 1st half and 2nd half
3. Finally, merge two sorted linked-list

```
Node sort (Head) {
    if (Head == null || Head.next == null)
        return Head
    mid = findMiddle (Head)  → TC = O(N)
    H2 = mid.next
    mid.next = null
    H1 = sort (Head)
    H2 = sort (H2)
    return mergeSortedLists (H1, H2) → TC = O(N)
}
```

TC = O(N log (N)) SC = O(log (N))

Scenerio

You are using **Google Maps** to help you find your way around a new place. But, you get lost and end up walking in a circle. **Google Maps** has a way to keep track of where you've been with the help of special **sensors**.

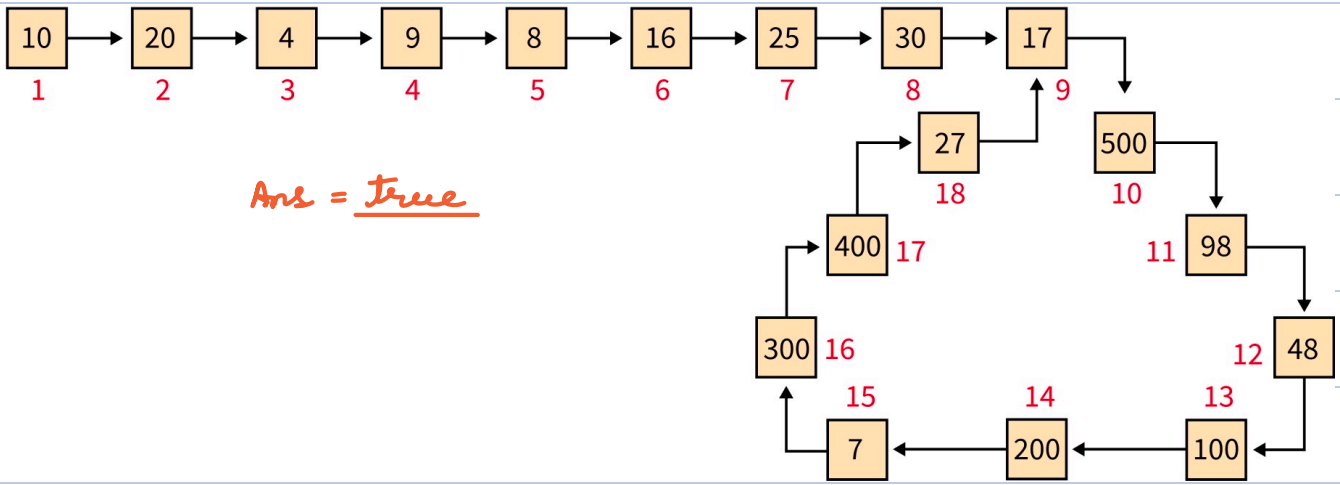
These sensors notice that you're **walking in a loop**, and now, **Google** wants to create a new feature to help you figure out exactly where you started going in circles.

Problem

You have a **linked list** that shows each **step** of your **journey**, like a chain of events. Some of these steps have accidentally led you back to a place you've already been, making you **walk in a loop**. The goal is to find out the exact point where you first started walking in this loop.

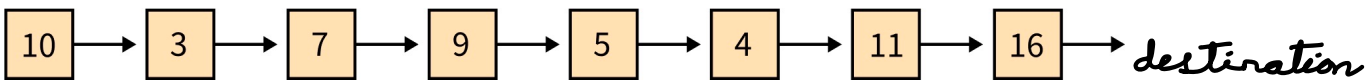
Check if there is a loop

1



Ans = true

2



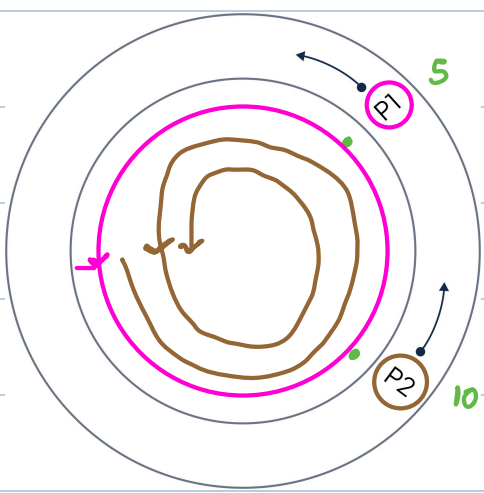
Ans = false

Sol 1 → Use hashset to check already visited node. ✓

SC = O(1)

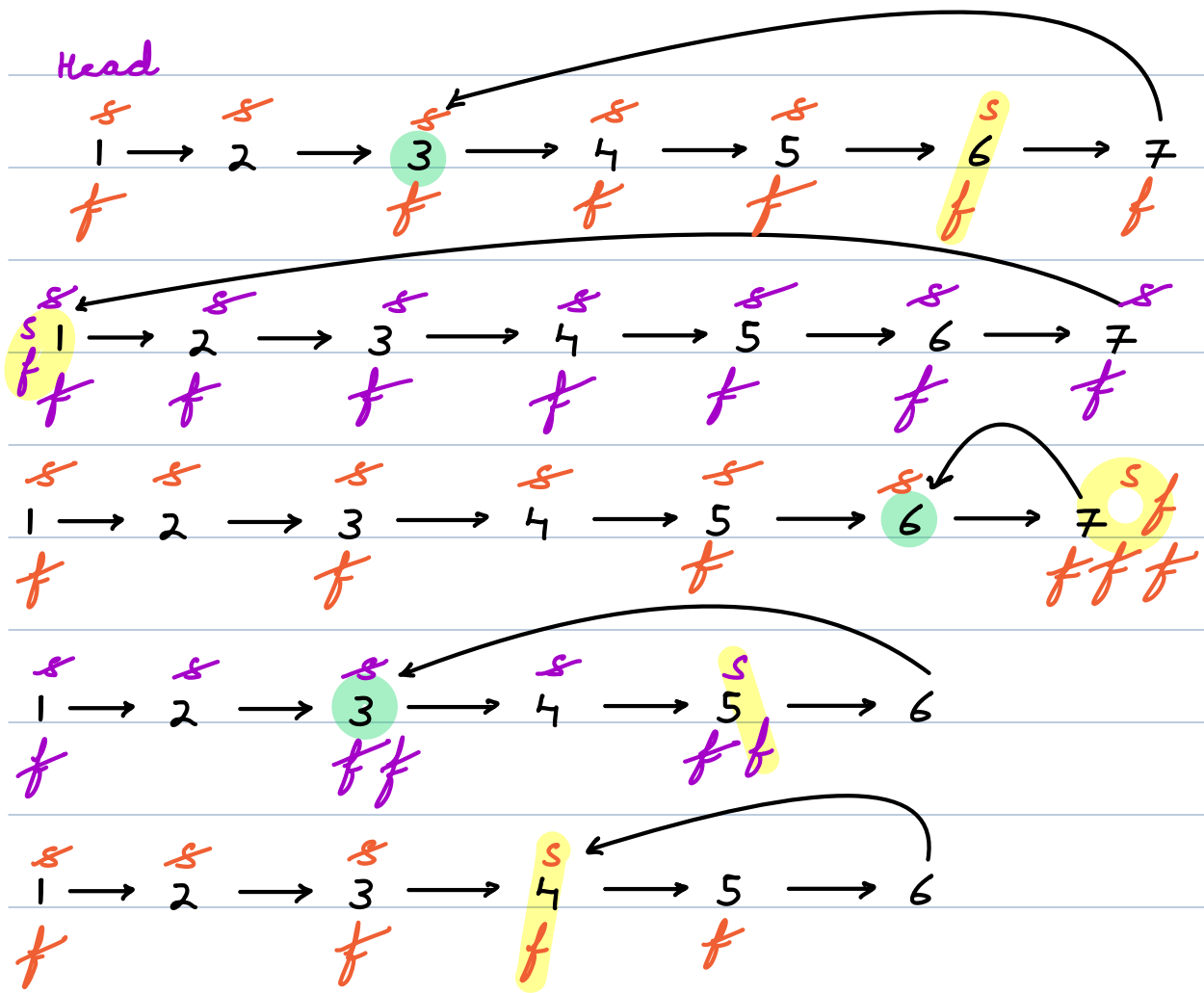
O(1)

If two people are running with different speeds on a circular track, they will 100% meet at some point.

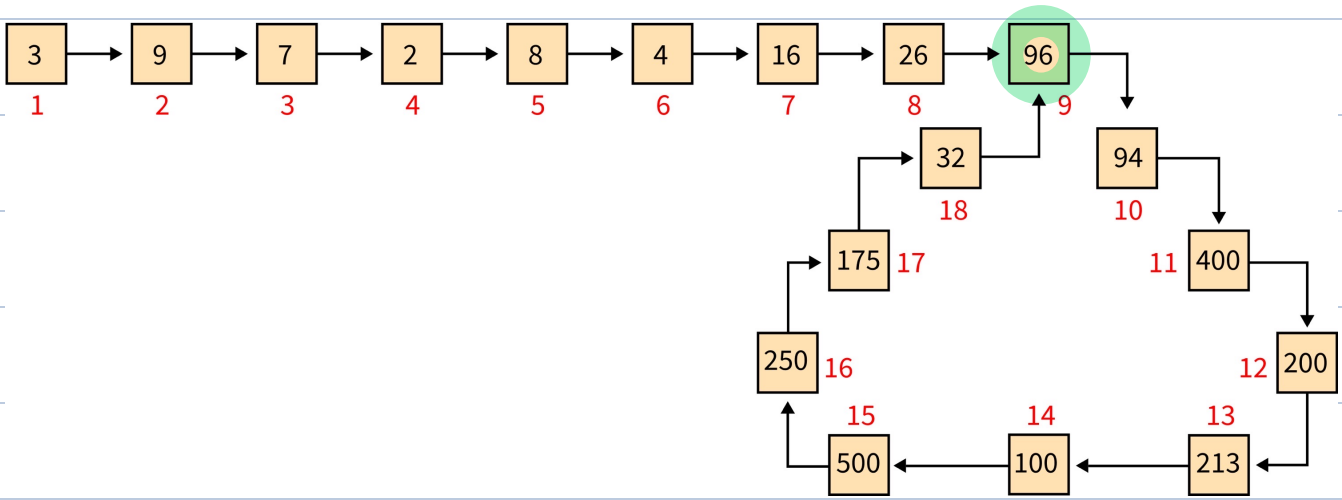


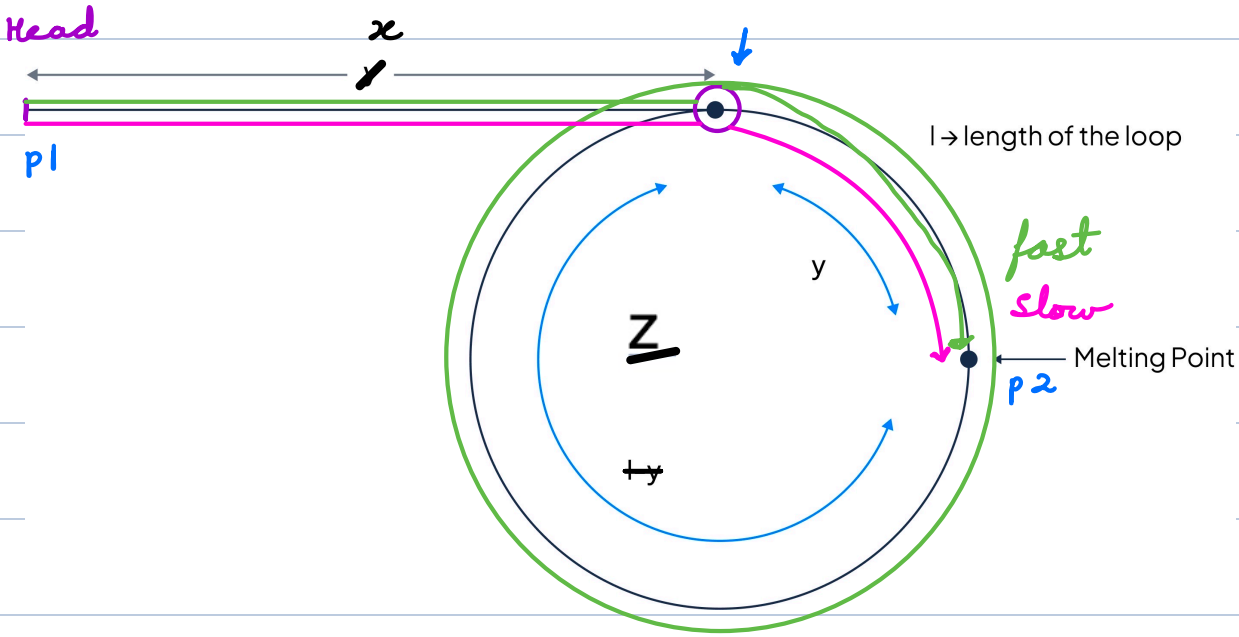
```
if (Head == null)   return false
s = f = Head
while ( f != null && f.next != null) {
    s = s.next
    f = f.next.next
    if (s == f)   return true
}
return false
```

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



Find the start point of the loop





Distance

slow = x + y

fast = x + y + z + y

Fast is all double speed ⇒

x + y + z + y = 2 * (x + y)

x + y + z + y = x + y + x + y

⇒ z = x

x

```
if (Head == null)   return false
s = f = Head
while ( f != null && f.next != null) {
    s = s.next
    f = f.next.next
    if (s == f)   break
}

p1 = Head
p2 = s
while (p1 != p2) {
    p1 = p1.next
    p2 = p2.next
}

return p1
```

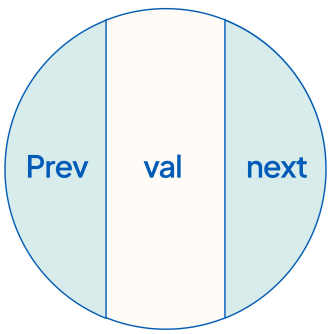
TC = O(N) SC = O(1)

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

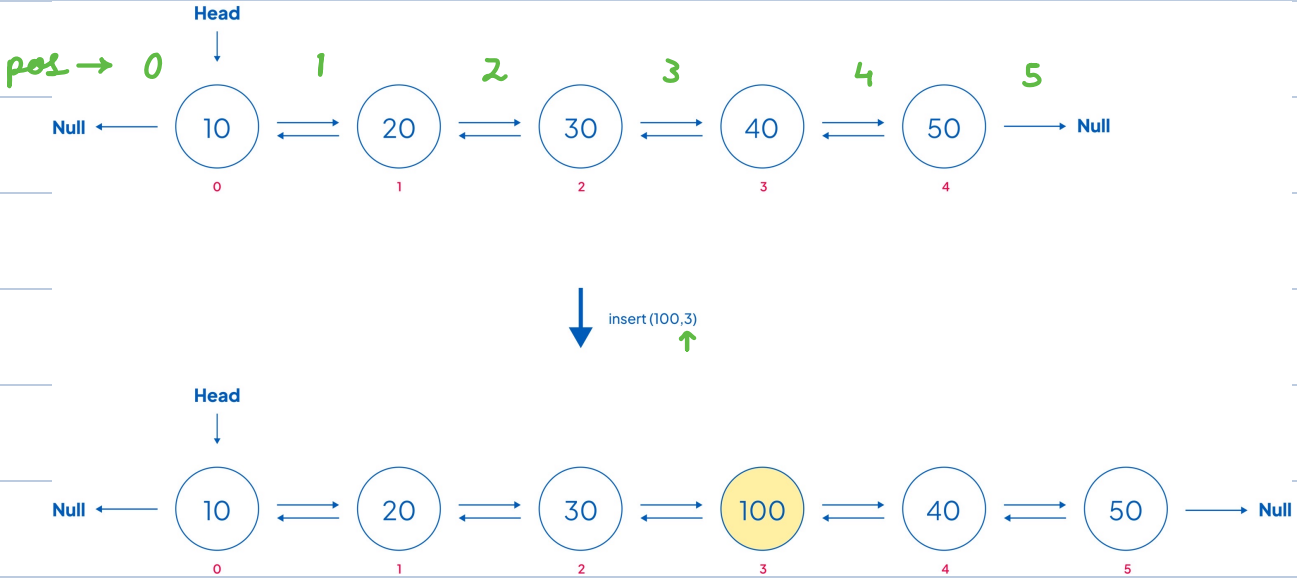
Spotify wants to enhance its user experience by allowing users to navigate through their music playlist seamlessly using "next" and "previous" song functionalities.

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:

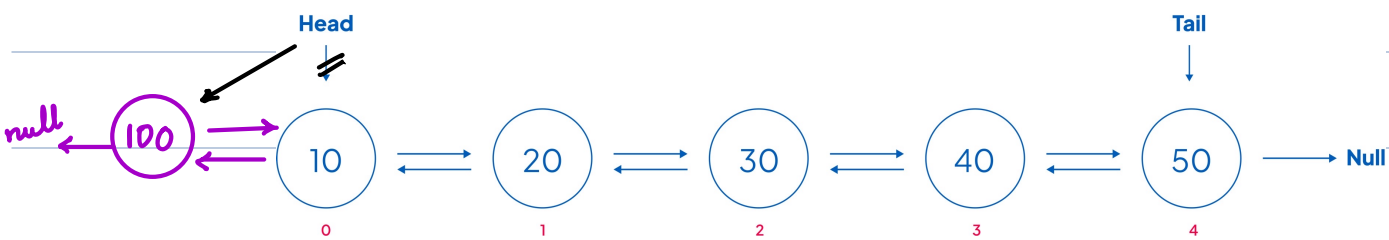
- **Add Song :** Insert a new song into the playlist. If the playlist is currently empty, this song becomes the "Current song".
- **Play Next Song :** Move to the next song in the playlist and display its details.
- **Play Previous Song :** Move to the previous song in the playlist and display its details.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

Insertion in Doubly LinkedList

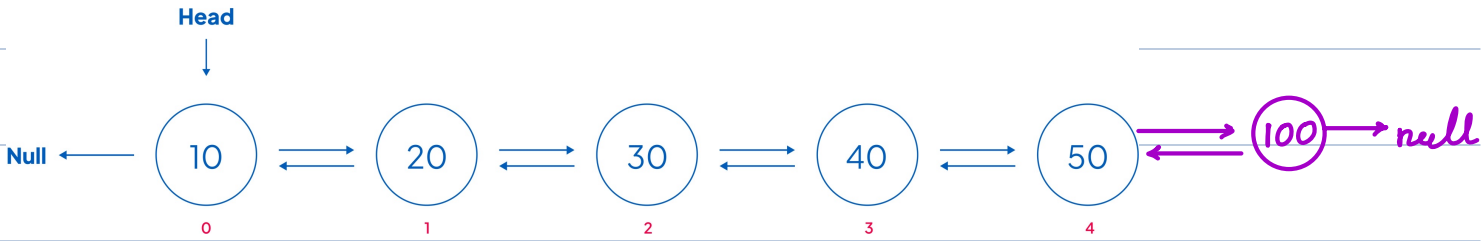


Edge - Case

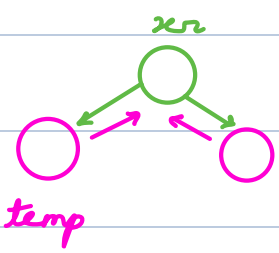
- **At index 0**



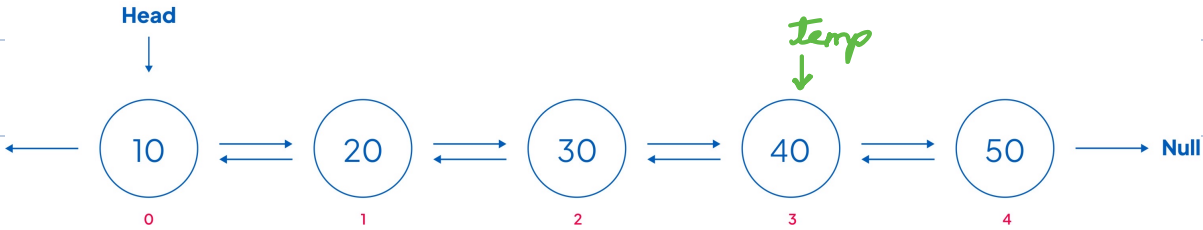
- **At index 5**




```
// 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)
```

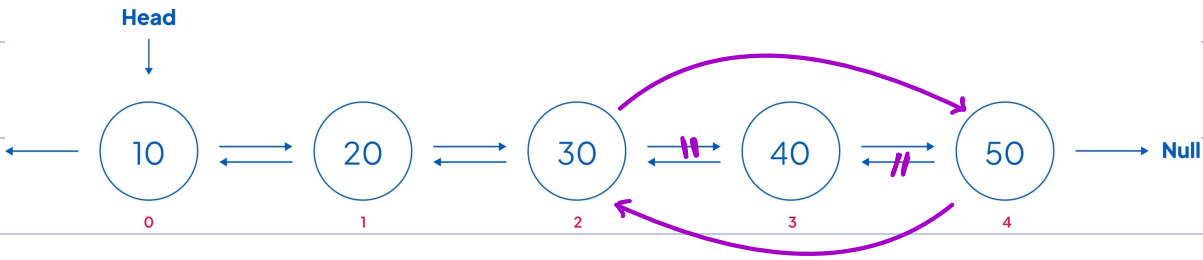


Delete a node from D.L.L



Val = 40

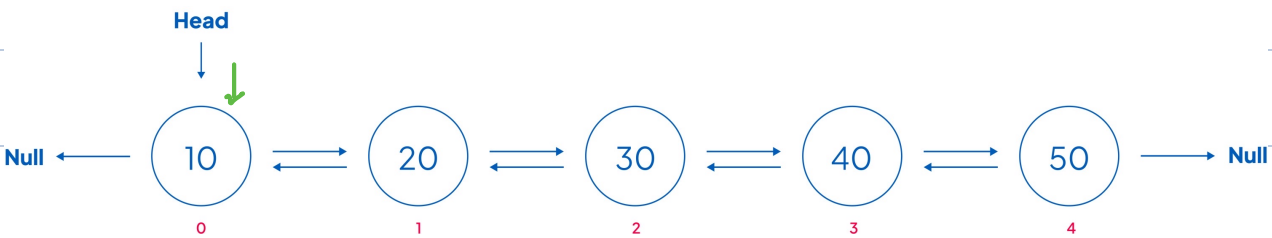
Delete the node with val = 40 →

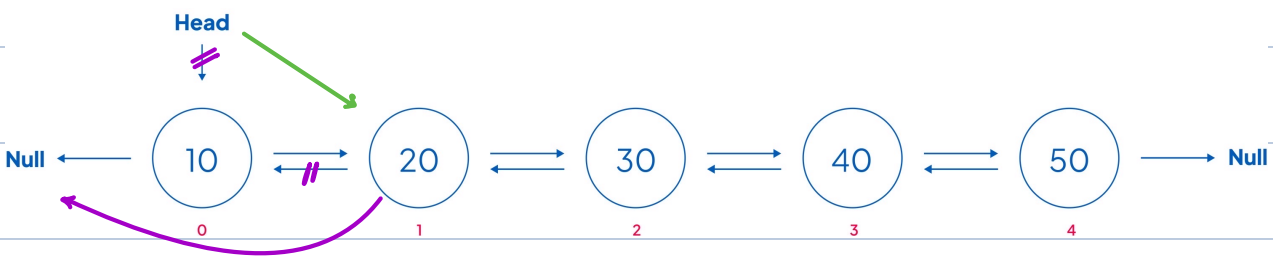


Edge - Case

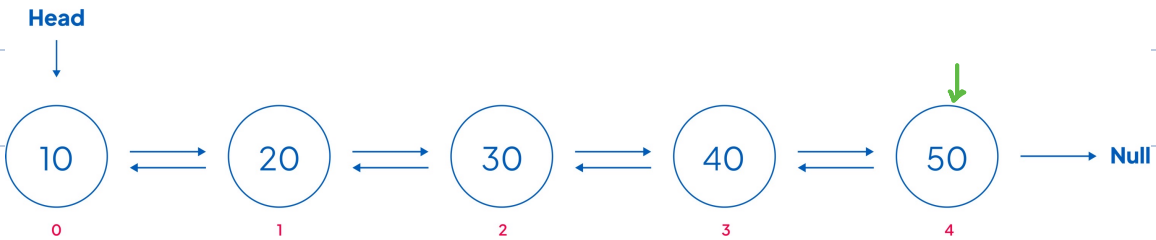
- **idx → 0**

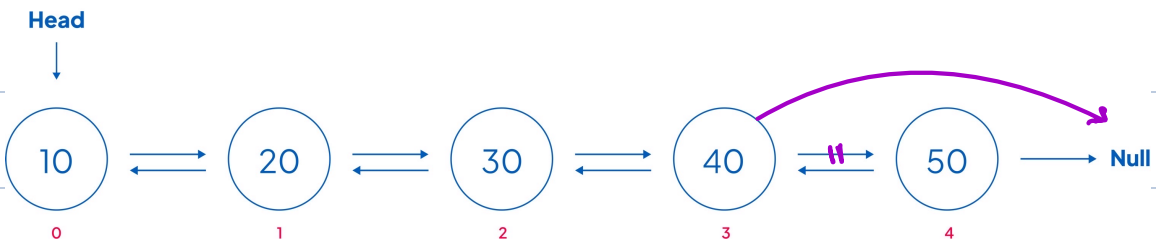
val → 10



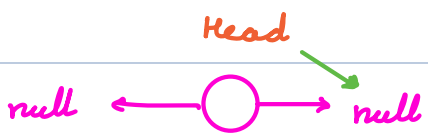


- **last - node** **val → 50**






```
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) ✓

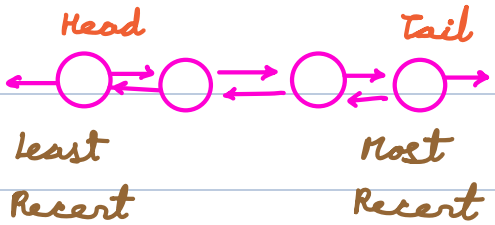
deleteNode(xn)

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)

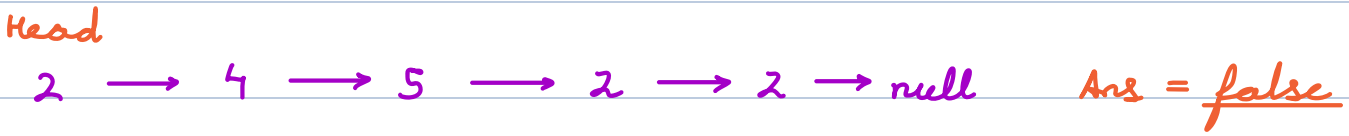
TC = O(1) SC = O(M)

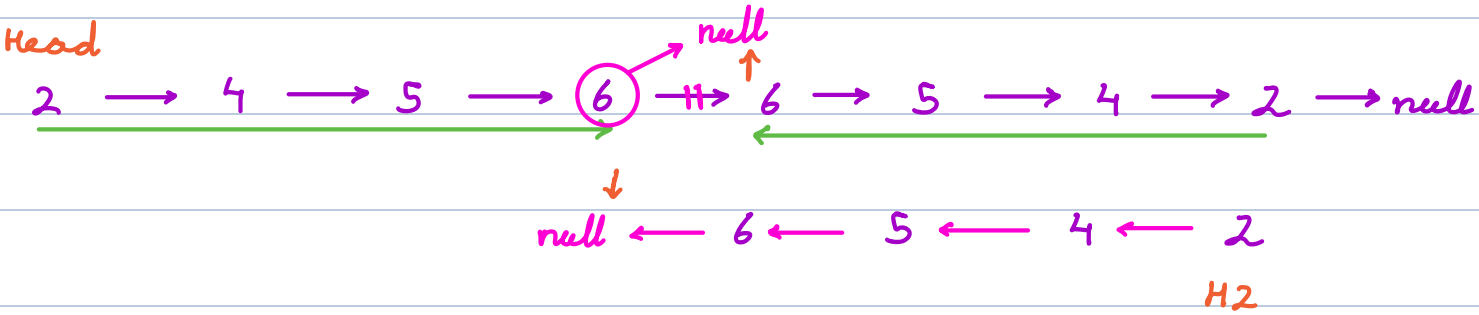
Q → check if the given linked list is palindrome.

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)