Linked List 3

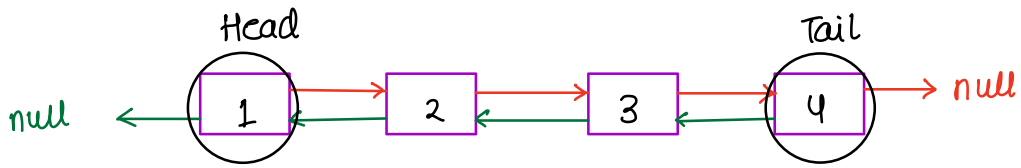
- Abhishek Sharma
- Akansh Nirmal
- amit khandelwal
- Bhaveshkumar
- Burhan
- Gagan Kumar S
- Gowtham
- Madhan Kumar M S
- Nikhil Pandey
- Pankaj
- Prajwal Khobragade
- PREETHAM
- Purusharth A
- Rajat Sharma
- Rajendra
- Sanket Agarwal
- Sanket Giri
- Saurabh Ruikar
- sharath r
- Shradha Srivastava
- Sneha L
- Subhashini
- Subhranil Kundu
- Sumit Adwani
- Sushant Patil
- Suyash Gupta
- Vasanth
- Venkata Sribhavana Nandiraju
- Vimal Kumar
- Vishal Mosa

- Doubly Linked List
- Insert a node
- Delete first occurrence
- LRU cache *
- Shallow Copy vs Deep Copy
- Copy List.

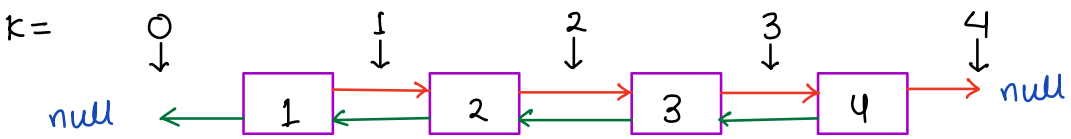
Current PSP

63% → 70%

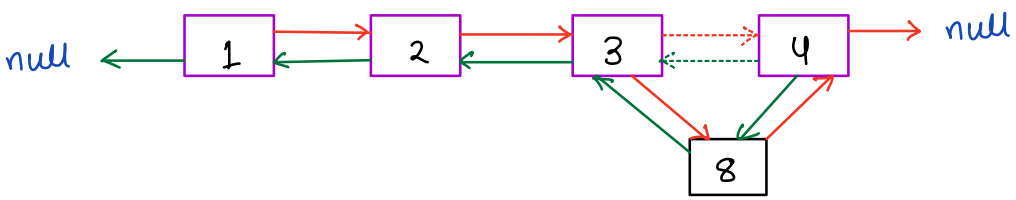
Doubly Linked List



Given a doubly LL. Insert a node with data X at a position K [0<= K <= N]



X = 8 , K = 3

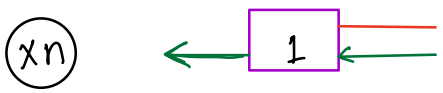


Corner Cases

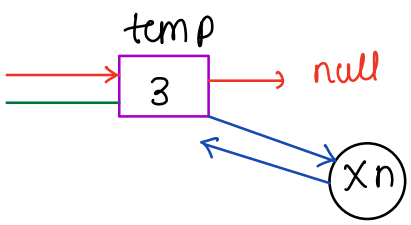
1. Head == null
2. K == 0
3. insert at tail

Node

```
insertNode ( head , k , x ) {
xn = new DLL(x)  ⟶ new node to be inserted

if ( head == null )  return xn
```

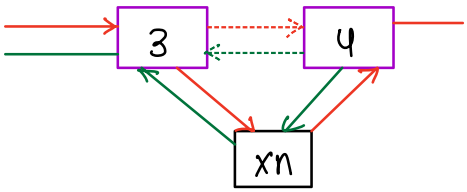



```
if( k==0 ) {
    xn.next = head
    head.prev = xn
    return xn
}

temp = head
for i ⟶ 1 to k-1 {
    temp = temp.next
}
```




```
if ( temp.next == null ) {  // insert at tail
    temp.next = xn
    xn.prev = temp.
    return head
}
```

temp. nxt




```
nxt = temp.next
temp.next = xn
xn.prev = temp
xn.next = nxt
nxt.prev = xn.

return head
```

}

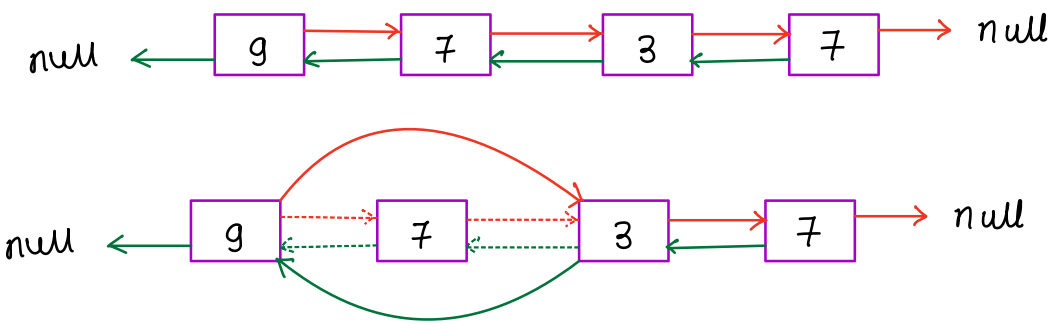
TC: O(k)

SC: O(1)

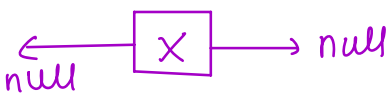
Q> Delete the first occurrence of data X in a given doubly linked list.

If not present then, no updates

X = 7

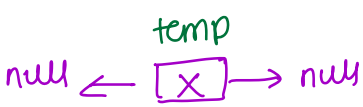


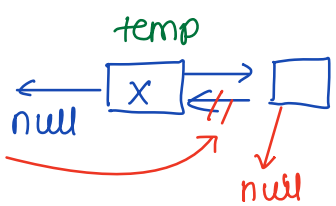
Corner Cases

1. X is not present
2. 

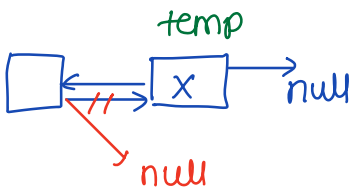
```
Node deleteNode (head, X) {
    temp = head

    while (temp != null) {
        if (temp.data == X) {
            break
        }
        temp = temp.next
    }

    // X not found
    if(temp == null) {
        return head
    }

    // Single node x
    if (temp.next == null &&
        temp.prev == null) {
        return null
    }

    // X is at head
    if (temp.prev == null) {
        temp.next.prev = null
        return temp.next
    }
```

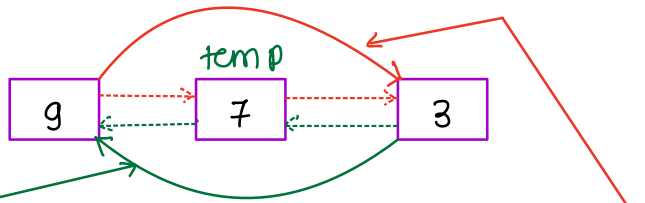
TC : O(N)





```
// X is at tail
if (temp.next == null) {
    temp.prev.next = null
    return head
}
```





temp.prev.next = temp.next

temp.next.prev = temp.prev .

return head

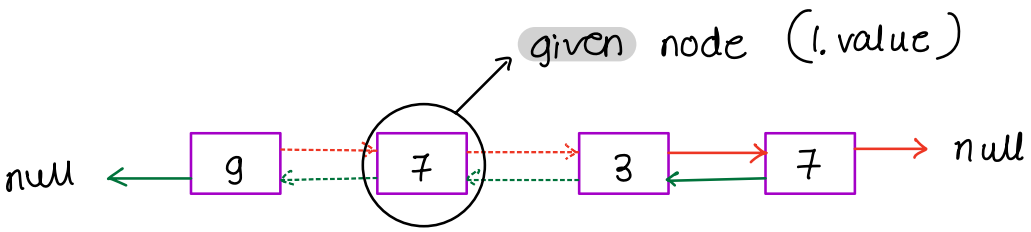
```
}
```

TC: O(N)

SC: O(1)

Given the node X to be deleted in a DLL

TC to delete node X ⟶ TC: O(1)



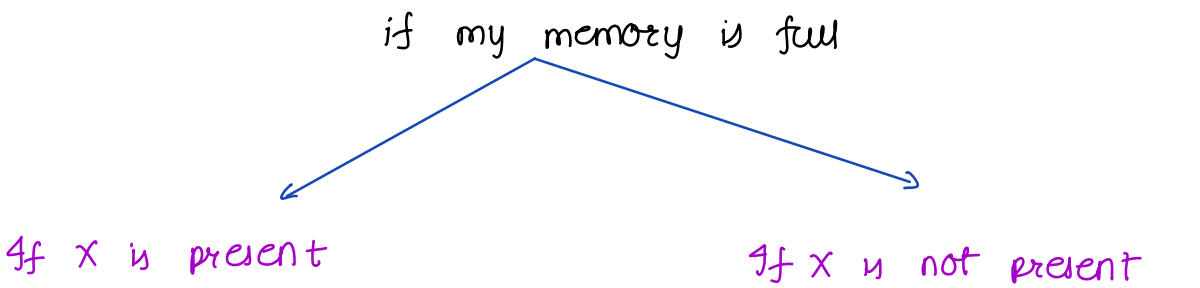
{least Recently used} LRU cache*****

Q> Given a running stream of integer and a fixed memory of size M. we have to maintain the most recent M elements.

In case the memory is full, we have to delete least recent element and insert current data into the memory {as most recent}

M = 5

~~10~~ ~~15~~ ~~19~~ ~~20~~ ~~18~~ 23 20 19 ~~17~~ 17 10



Break 22:20

What is the best data structure to do the checking part ie if X is present or not ?

~~HashSet~~ OR HashMap

To maintain order ⟶ ~~Array~~, ~~ArrayList~~, Linked List

SLL, DLL

TC to delete: O(N) | O(N) | O(N)

If node to be deleted is known

TC: O(1)

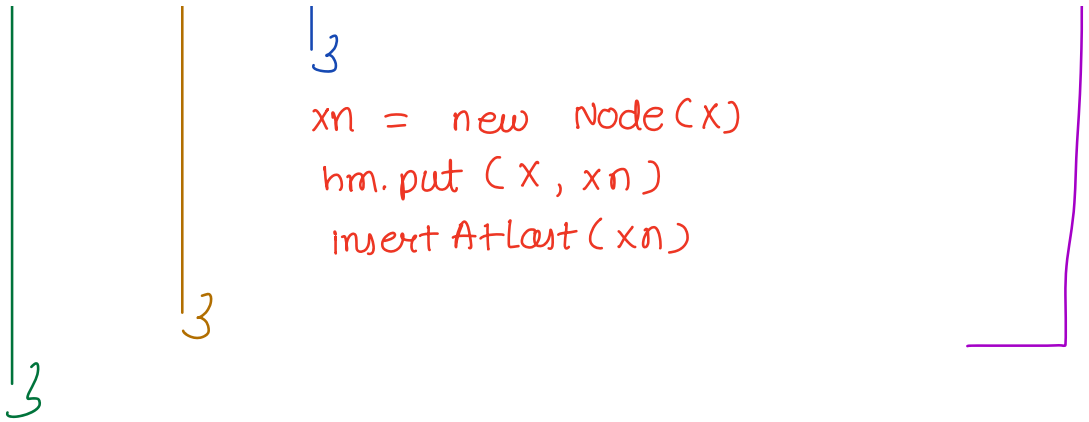
which is DLL

HashMap <Integer, Node>

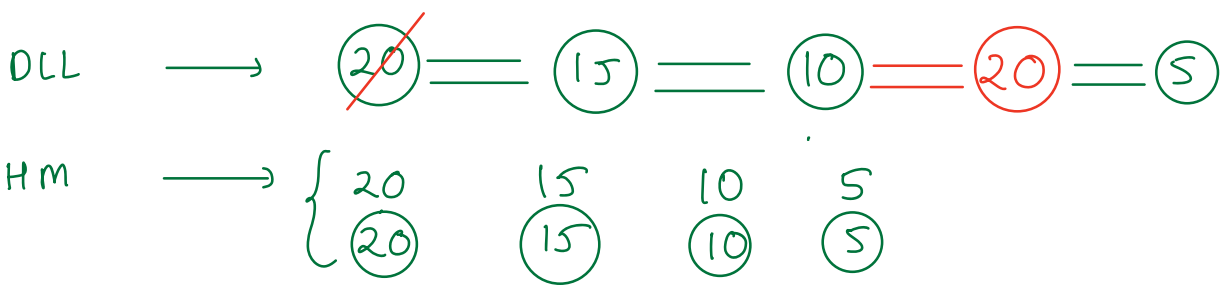
head = null
tail = null

→ global

```
void addX(X) {
    if (hm.containsKey(X)) {
        node = hm.get(X) // return Node
        deleteNode(node)
        insertAtLast(node)
    }
    else {
        if (hm.size() == M) {
            hm.remove(head.data)
            deleteNode(head)
```



M = 5



TC per addition of X : O(1)

SC : O(M)

Deep Copy & Shallow Copy

```
x = new Node(5)
y = x
y.val = 10
print(x.val) ⟶ 10
```

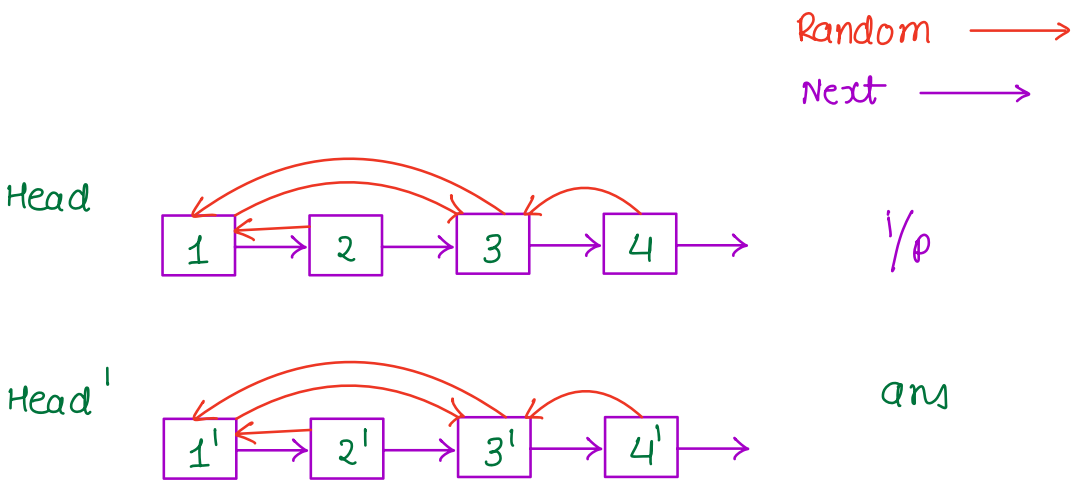
shallow

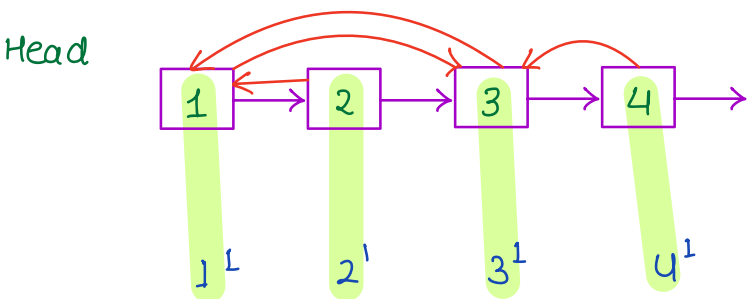
```
x = new Node(5)
y = new Node(x.val)
y.val = 10
print(x.val) ⟶ 5
```

Deep Copy

```
class Node {
    data
    next
    random
}
```

Q) Create a deep copy of given LL with random pointers





HM { Node : Node }

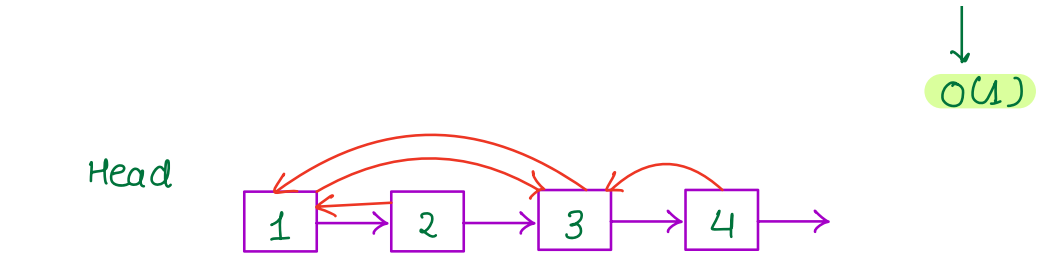
↓ original ↓ copy

Pseudocode

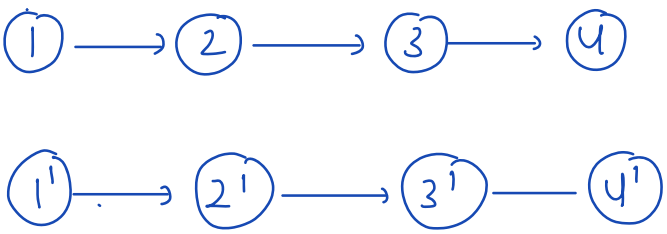
```
Node deepCopy (Node head) {

    HashMap<Node, Node> hm = new HashMap<>()
        hm.put (null, null)
        HM { Node : Node }
                    ↓       ↓
                 original  copy

    temp = head
    while (temp != null) {
        hm.put (temp, new Node(temp.data))
        temp = temp.next
    }

    temp = head

    while (temp != null) {
        // update next pointer of copy
        hm.get(temp).next = hm.get(temp.next)
        // update next pointer of copy
        hm.get(temp).random= hm.get(temp.random)
        temp = temp.next
    }

    return hm.get(head)
}
```

TC: O(N)

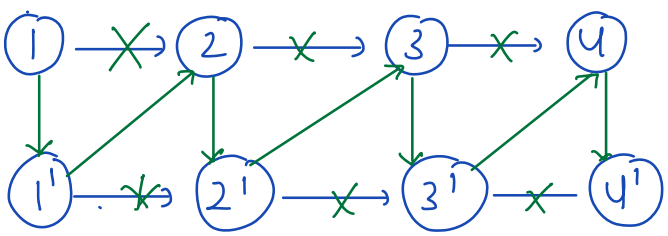
SC: O(N)

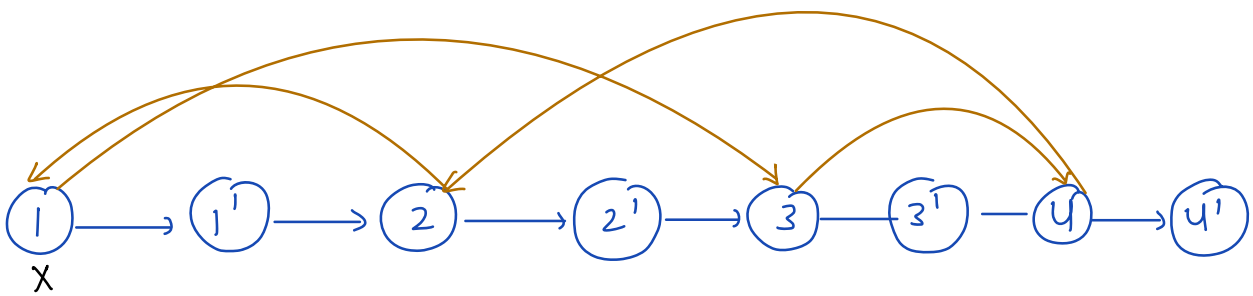


Step 1



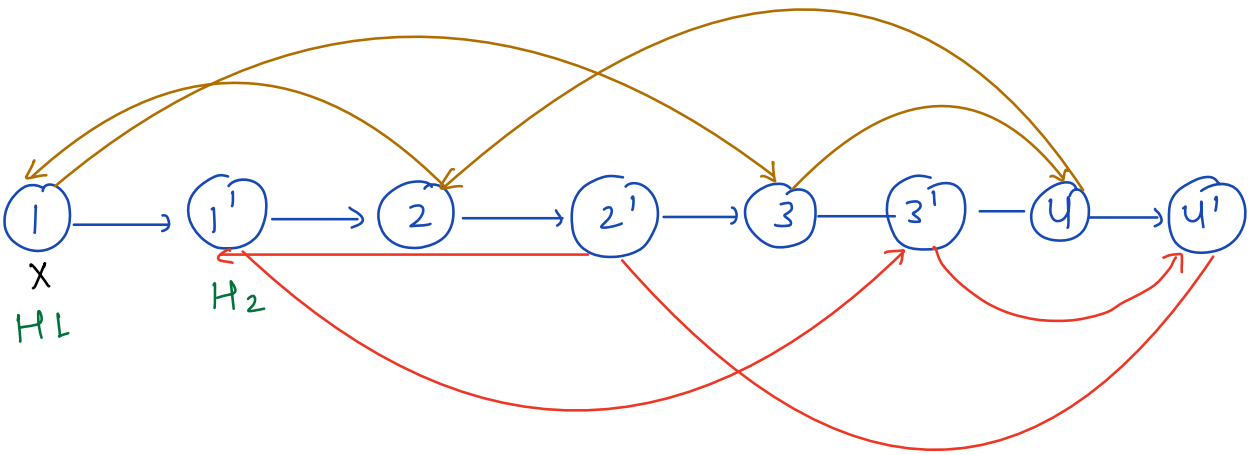
Step 2





How to connect random pointers

```
while (x != null)
    y = x.next
    y.random = x.random.next
    x = x.next.next
}
```


1
1'
2
2'
3
3'
4
4'
X
HL
H2