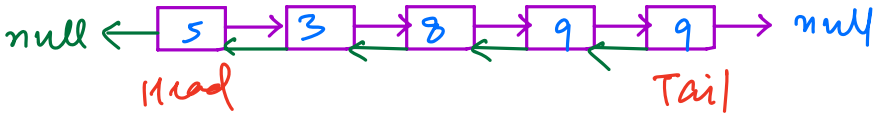
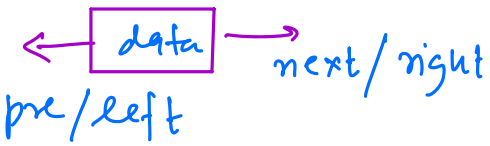
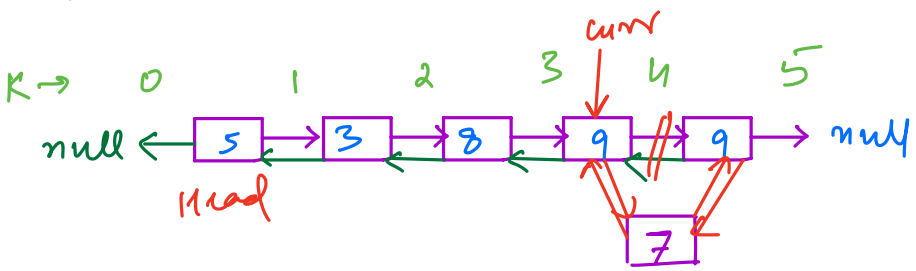
Linked List 3 : Problems & Doubly Linked List






```
class Node {
    int data;
    Node next, pre;
    Node (x) {
        data = x
        next = pre = null
    }
}
```

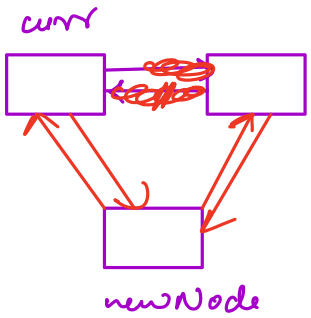
Ques → Given a doubly LL. A node is to be inserted with data X at index K where $0 <= K <= N$



eg K=4, X=7

```
newNode = new Node(x)
if (Head == null) { return newNode }
if (K == 0) {
    newNode.next = Head
    Head.pre = newNode
    return newNode
```

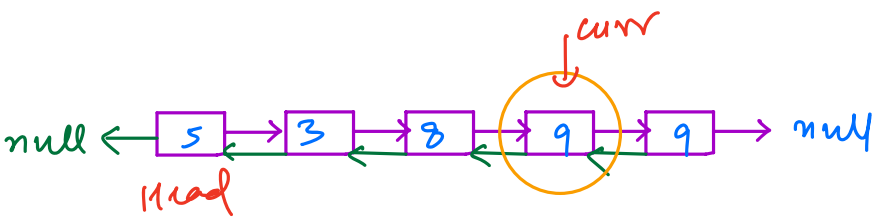
```
}
curr = Head
for (i=1 to k-1) { curr = curr.next }
newNode.next = curr.next
newNode.pre = curr
if(curr.next != null) {
    curr.next.pre = newNode
}
curr.next = newNode
return Head
```



TC = O(K)

SC = O(1)

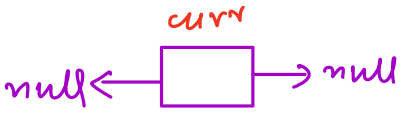
Ques → Given a doubly LL, delete the first occurance of X. If not present → no change.

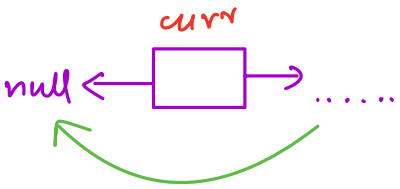


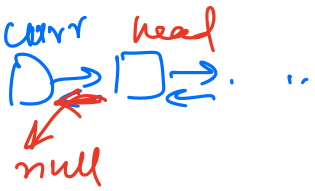
X = 9

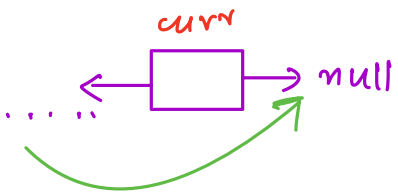
```
// Search
curr = Head
while(curr != null) {
    if (curr.data == X)  break
    curr = curr.next
}
```

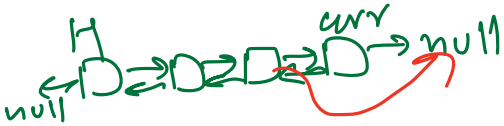
```
// not present
if(curr == null)   return Head

if( curr.next == null && curr.pre == null )
      return null

if( curr.pre == null ) { // delete head
      curr.next.pre = null
      return curr.next // new head node
}

if( curr.next == null ) {
      curr.pre.next = null
      return Head
}
else {
      curr.next.pre = curr.pre
      curr.pre.next = curr.next
}
return Head
```

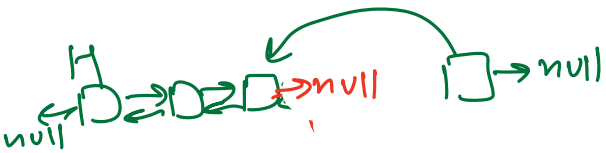


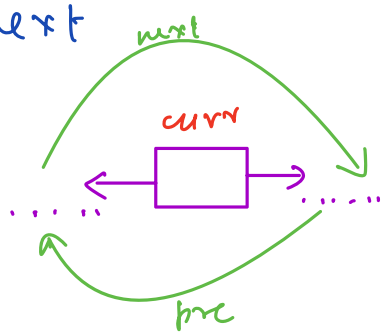












TC = O(N)

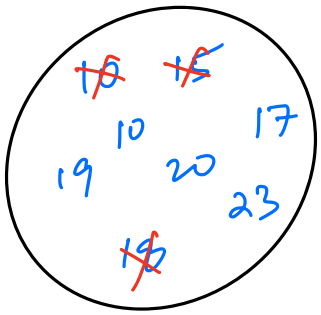
SC = O(1)

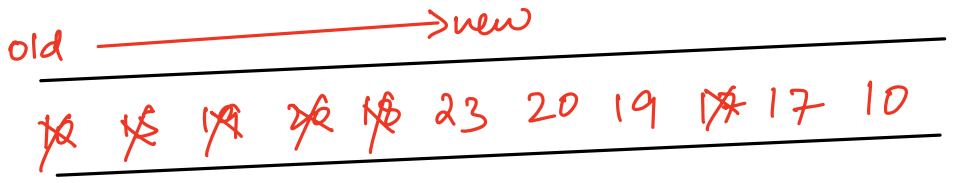
Ques → Given a running stream of integers & a fixed memory of size M (SC = O(M))

Maintain the latest M elements in memory. In case the memory is full, delete the least recent element.

eg → 10 15 19 20 18 23 20 19 17 17 10

M = 5





Once memory is full

if intake X

if X is not present

1. Delete least recent item.
2. Insert X as most recent item.

X is present

1. Delete X from its position
2. Insert X as most recent item.

Requirements :

<data, location of data>

1. Search of intake X → ~~Hashset~~ / Hashmap

2. Maintain order of recency → ~~Array~~, ~~Stack~~, ~~Queue~~, Linked list

delete curr node ⇒ doubly LL

in doubly LL case :

<data, node of data>

```
Hashmap<Integer, Node> hm = new Hashmap<>();
Head = Tail = null
for(# input : X ){
    if ( hm.containsKey(X)) {
        temp = hm.get(X) // node
        Head = deleteNode(Head, temp)
        insertLastNode(Tail, temp)
    }
    else {
        if( hm.size() == M) {
            hm.remove(Head.data)
```

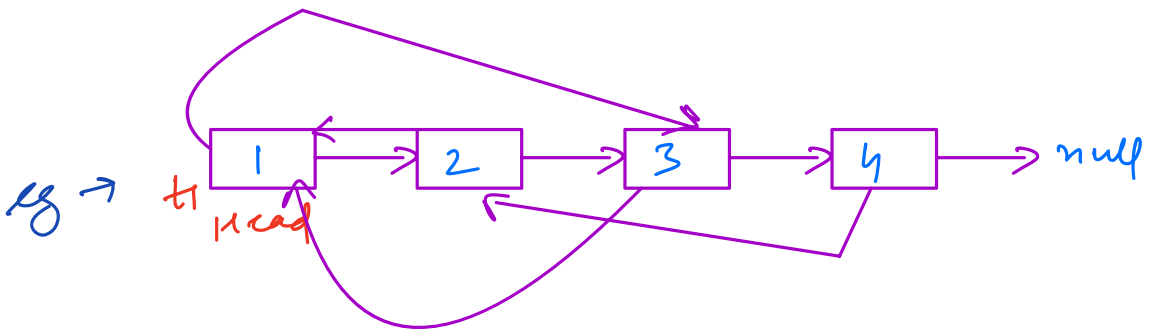
```
        Head = deleteHead(Head)
    }
    newNode = new Node(X)
    hm.put(X, newNode)
    insertLastNode(Tail, newNode)
}
}
```

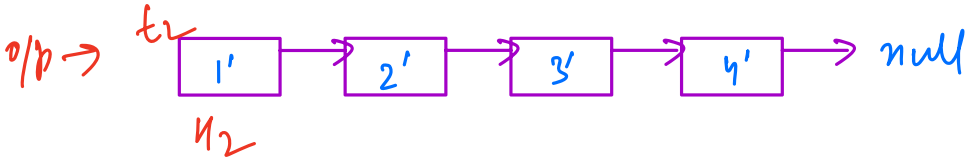
TC = O(1) per input

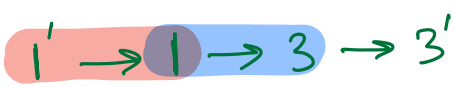
SC = O(M)

Qus → Given a LL with next & random pointer.

Create a deep copy of this LL.



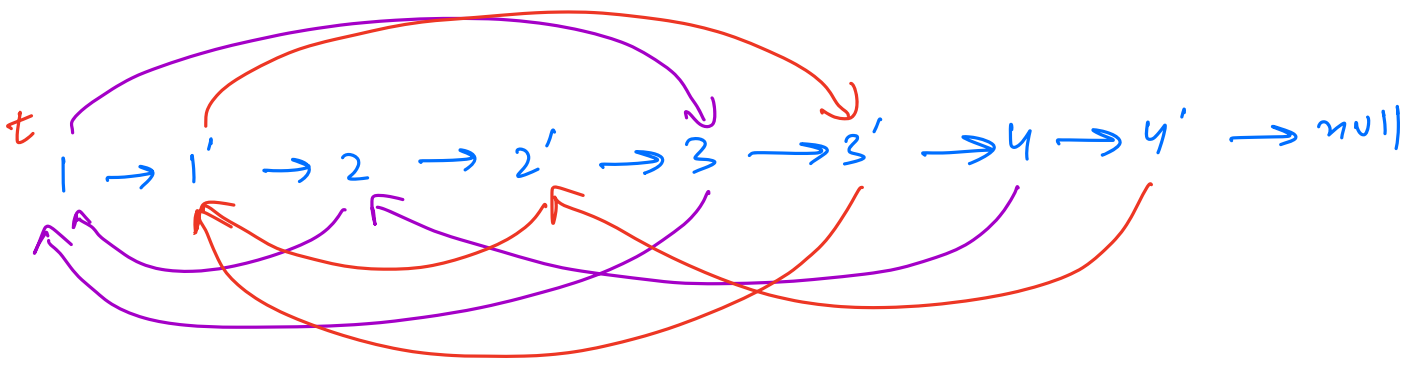




Soln → use HashMap <node, copy of node>

```
h2 = new Node(head.data)
t1 = head, t2 = h2
hm.insert(t1,t2)
while (t1.next != null) {
    t2.next = new Node(t1.next.data)
    hm.insert(t1.next, t2.next)
    t1 = t1.next
    t2 = t2.next
}
t2 = h2 , t1 = head
while (t2 != null) {
    t2.random = hm.get(t1.random)
    t1 = t1.next
    t2 = t2.next
}
```

TC = O(N) SC = O(N) ↓ O(1) ?



t.next.random = t.random.next

copy node copy node

t = t.next.next

Detach logic

```
H2 = Head.next
t1 = Head,   t2 = H2
while (t1 != null) {
    t1.next = t1.next.next
    if (t2.next != null) {
        t2.next = t2.next.next
    }
    t1 = t1.next
    t2 = t2.next
}
```

TC = O(N)

SC = O(1)