Lecture :- Linked List 3

Agenda

- Doubly linked list
- Insertion and deletion
- Maintain most recent integer
- Deep copy of DLL

Doubly linked list

SLL :- 1 → 2 → 3 → 4 → 5 → null.

DLL :- null ← 1 ⇄ 2 ⇄ 3 ⇄ 4 ⇄ 5 → null.

Structure of doubly linked list node

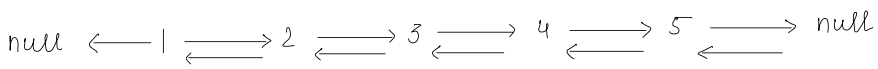
SLL.

```
class Node {
    int data;
    Node next;

    Node(int x) {
        data = x;
    }
}
```

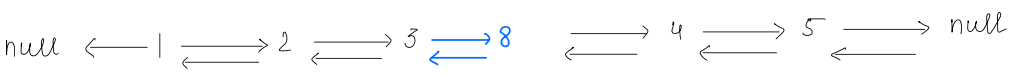
DLL

```
class Node {
    int data;
    Node next;
    Node prev;
    Node(int x) {
        data = x;
    }
}
```

Insert in a doubly linked list



val = 8
idx = 3.



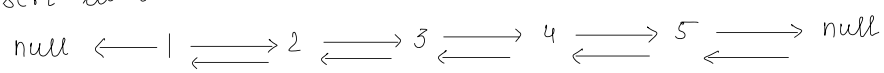
Approach

case1: when head == null.

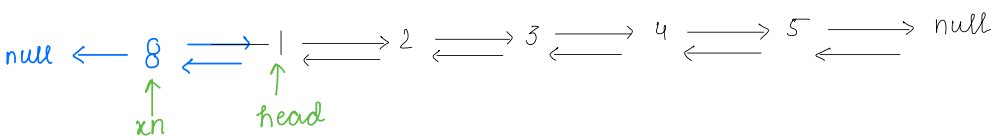
LL: null
val = 8 ⟶ head.
idx = 0

null ⟵ 8 ⟶ null.

case2: Insert at index = 0.

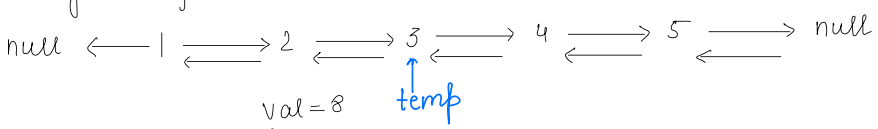


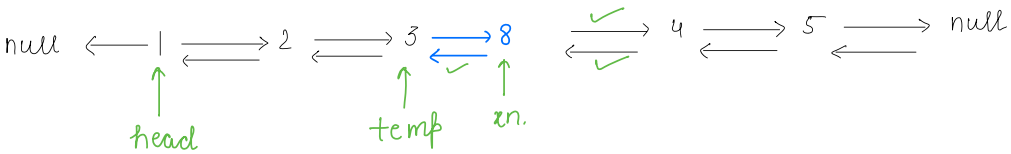
val = 8
idx = 0



xn.next = head; [8 ⟶ 1]
head.prev = xn [8 ⟵ 1]
head = xn;
return head;

case3 Anywhere from 1st idx to last idx.





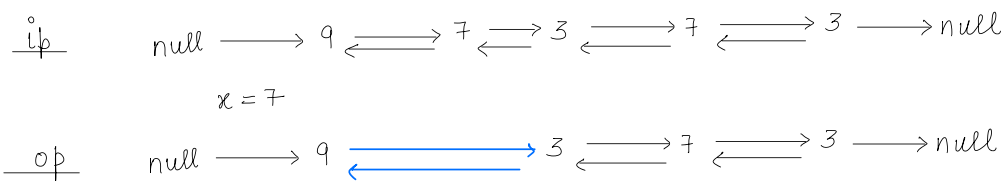

```
xn.next = temp.next   [ 8 ⟶ 4 ]
xn.prev = temp        [ 3 ⟵ 8 ]
if (temp.next != null) {
    temp.next.prev = xn;   [ 8 ⟵ 4 ]
}
temp.next = xn;   [ 3 ⟶ 8 ]
return head;
```

Pseudocode

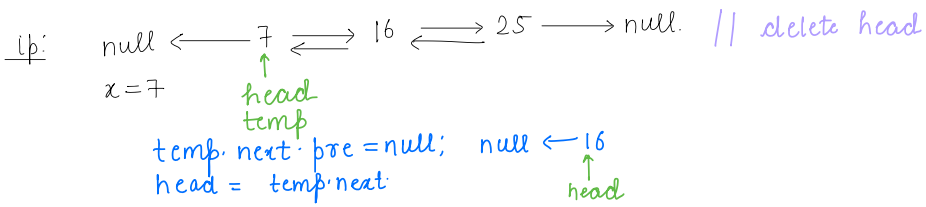
```
Node insert(Node head, int val, int x) {
    xn = new Node(val);
    if(head == null) {
        return xn;
    }
    if(x == 0) {
        xn.next = head;
        head.prev = xn
        head = xn;
        return head;
    }
    temp = head;
    for(i = 1; i <= x-1; i++) {
        temp = temp.next;
    }
    xn.next = temp.next
    xn.prev = temp
    if(temp.next != null) {
        temp.next.prev = xn;
    }
    temp.next = xn;   [3 → 8]
    return head;
}
```

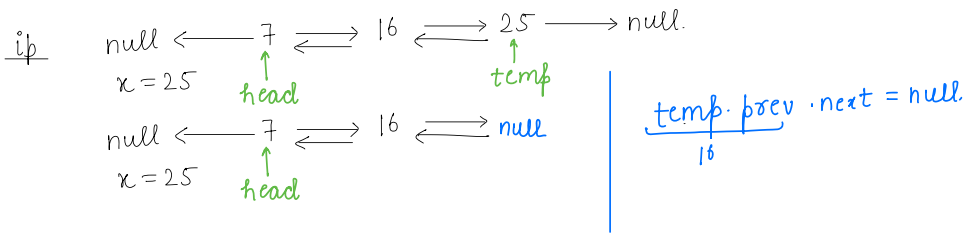
TC: O(n)
SC: O(1)

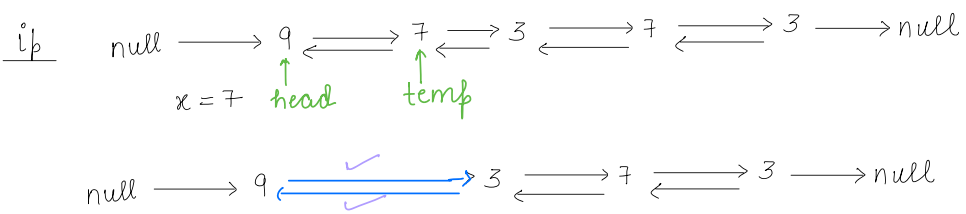
Qu Delete the first occurence from doubly linked list.



Approach ip : null ← 7 → null
val = 7
return null;







temp.prev.next = temp.next
9 3

temp.next.prev = temp.prev
3 9

Pseudocode

```
Node delete(Node head, int val) {
    temp = head;
    while(temp != null) {
        if(temp.data == val) {
            break;
        }
        temp = temp.next;
    }
    if(temp == null) {          → empty DLL && el not found.
        return head;
    }
    if(temp.prev == null && temp.next == null) {
        return null;
    }
    if(temp.prev == null) {
        temp.next.pre = null;
        head = temp.next;
    }
    else if(temp.next == null) {
        temp.prev.next = null;
    }
    else {
        temp.prev.next = temp.next;

        temp.next.prev = temp.prev;
    }
    return head;
}
```

null ← 7 → null
val = 7
↑ temp

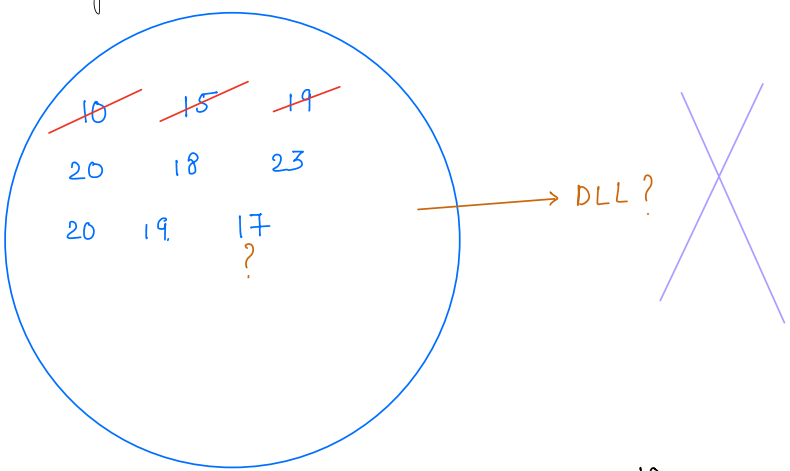
TC: O(n)
SC: O(1)

Maintain the most recent integer { hard }

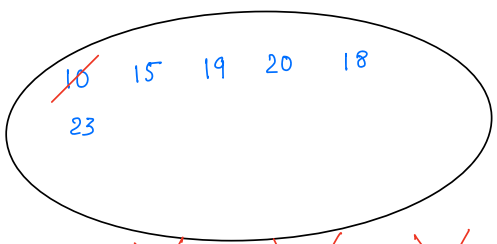
We have been given a running stream of integers and fixed memory size of m. We have to maintain most recent m elements. In case the memory is full, we have to delete the least recent element and insert the current data into the memory (as the most recent item).

Eg: 10 15 19 20 18 23 20 19 17

memory size = 5

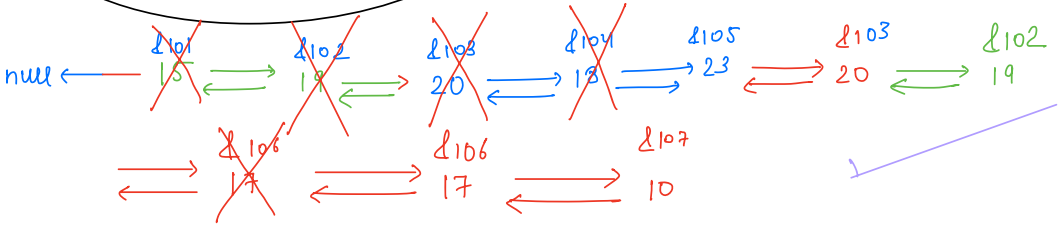


10 15 19 20 18 23 20 19 17 17 10



Map

~~10 — &100~~
~~15 — &101~~
19 — &102
20 — &103
~~18 — &104~~
23 — 105
17 — &106
10 — &107

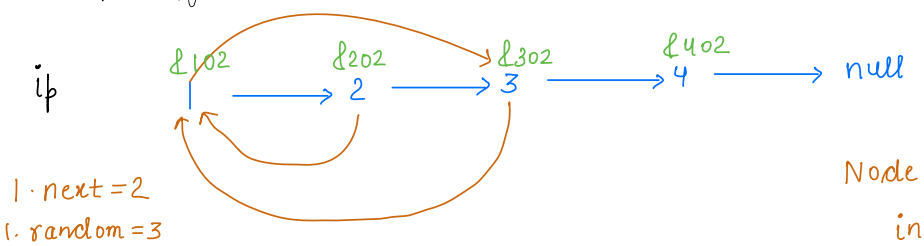


Pseudocode

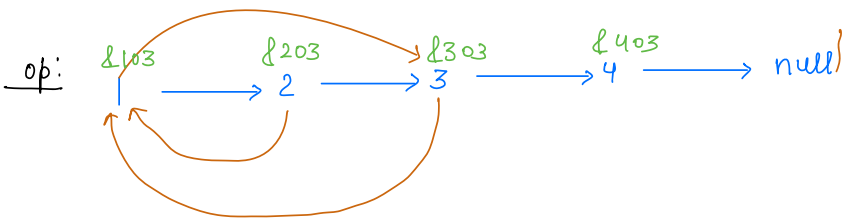
```
Map<Integer, Node> map;
if (map.contains(x)) {
    // Remove old x from memory.
    xn = map.get(x);
    head = delete(head, xn);
    // insert it in LL
    tail.next = xn;
    xn.prev = tail;
    tail = xn
} else {
    // full memory
    if (map.size == m) {
        hm.remove(head.data);
        delete(head);
    }
    xn = new Node(x);
    map.put(x, xn);

    tail.next = xn;
    xn.prev = tail;
    tail = xn;
}
```

Break: 8:51–9:00

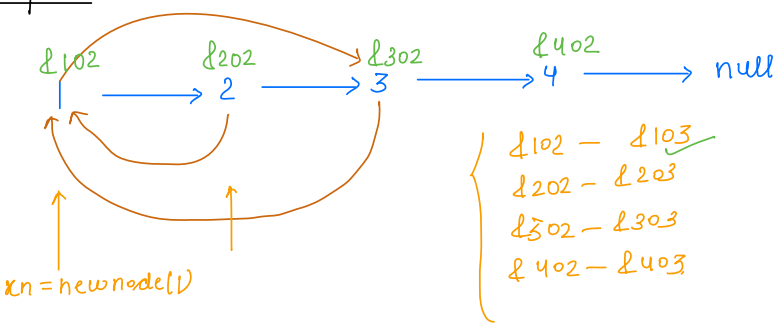
Qu Deep copy of a linked list with random pointers { medium-hard }

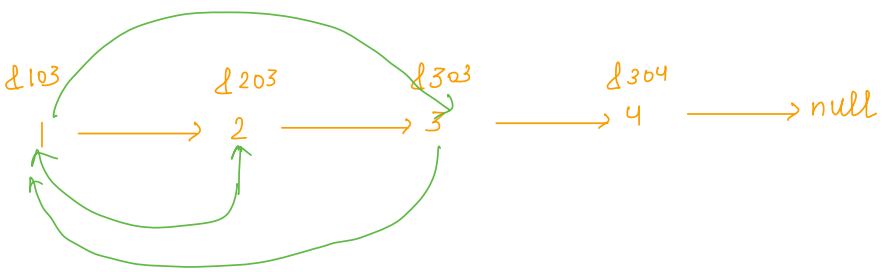


Node {
int data;
Node next;
Node random;
}



Brute force:

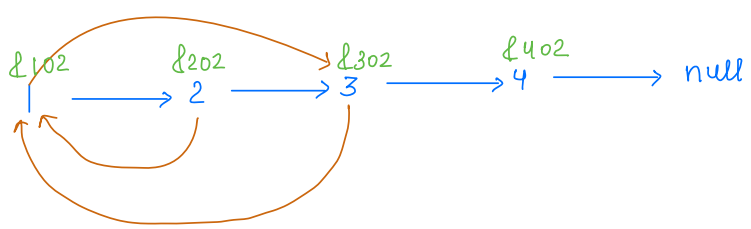




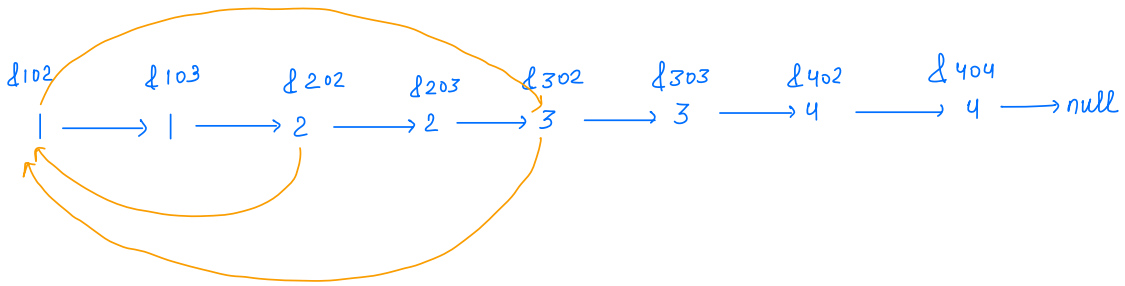
TC: O(n)
SC: O(n) — map

Expectation

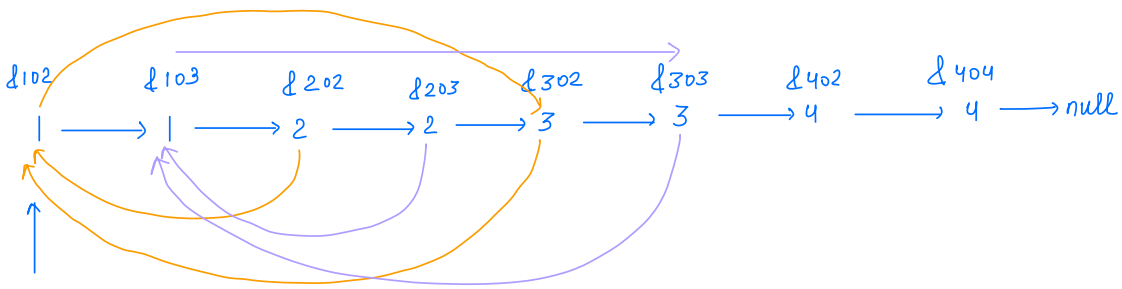
TC: O(n)
SC: O(1)



step1:



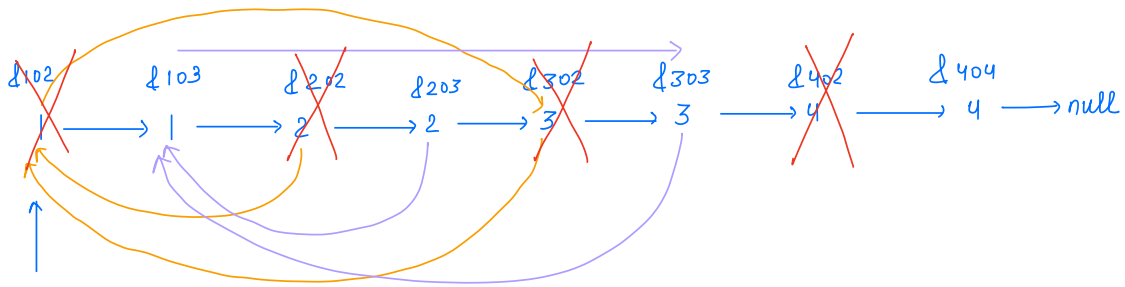
step2:



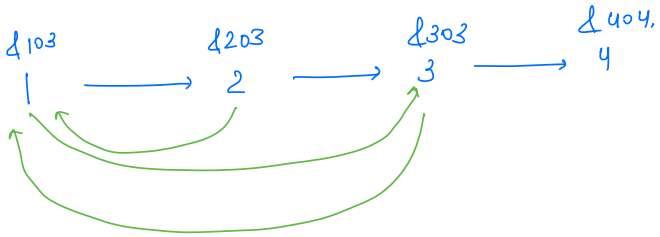
temp.random = &302.
&102

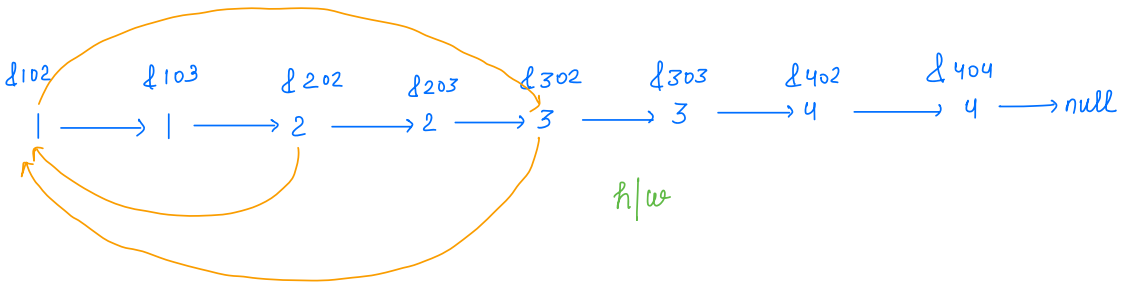
&103.random = &303
temp.next temp.random.next

Step3

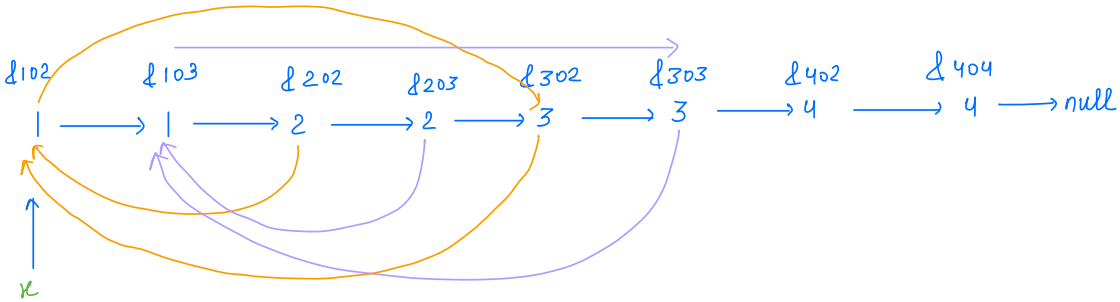

l102
l103
l202
l203
l302
l303
l402
l404
1
1
2
2
3
3
4
4
null

Step4:


l103
l203
l303
l404,
1
2
3
4

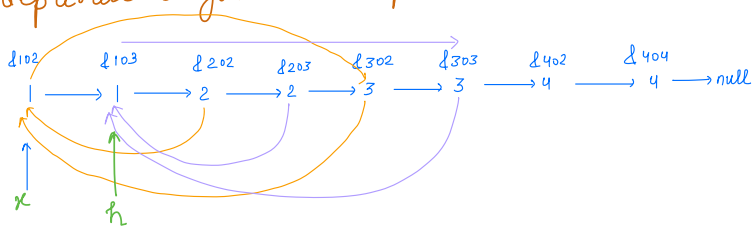


h/w

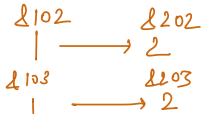



```
x = head;
while( x!=null) {
    y = x.next;
    y.random = x.random.next;
    x = x.next.next;
}
```

// Separate original & output LL

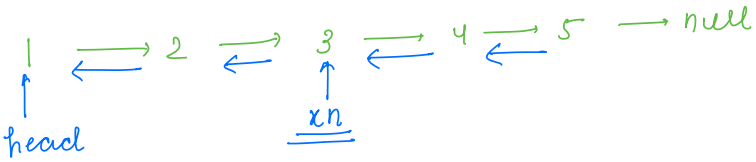



```
h = head.next;  // &103  (1)
x = head;   // &102 (1)
while(x!=null) {
    y = x.next   // &103
    x.next = x.next.next
    &102        &202.
    y.next = y.next.next
    &103       &203
    x = x.next;
```



Thankyou :)

Doubts



$$\frac{xn.prev}{2}.next = \frac{xn.next}{4}$$

$$\frac{xn}{4}.next.prev = \frac{xn.prev}{2}$$