Doubly Linked list

* Deleting a node at a given pos^n

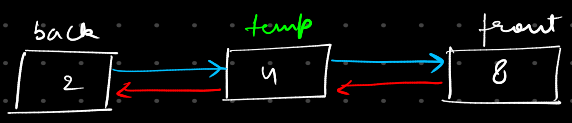
the pos^n must be from 1

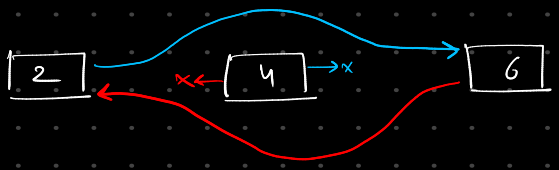
↳ In deletion of a node, what we basically do is moving of pointers, (next and previous).

↳ If head element is deleted, then previous pointer for head.next is set to null & next pointer of head is nullified.

↳ similarly on deleting the tail, the next pointer of tail.previous is nullified & previous pointer of tail is set to null.

↳ & for deleting element in between,





back = temp.previous, front = temp.next;

back.next = front;

front.previous = back;

back = null;

front = null;

But how to get to kth element?

→ using counters.

```
if(head == null) return null;
Node temp = head;
int count = 0;
while(temp != null){
    count++;
    if(count == k) break;
    temp = temp.next;
}
if(temp == null) return head;
Node front = temp.next, back = temp.previous;
if(front == null && previous == null){
    return null;
}
else if(front == null){
    back.next = null;
    back = null;
}
else if(back == null){
    front.previous = null;
    head = front;
    front = null;
}
else{
    back.next = front;
    front.previous = back;
    back = null;
    front = null;
}
return head;
```

this is how we get to k^{th} element.

⇒ if k^{th} el. is not present, then temp becomes null. → for handling tail →

(nothing is deleted)

single element case →

case of deleting tail →

Case of deleting head →

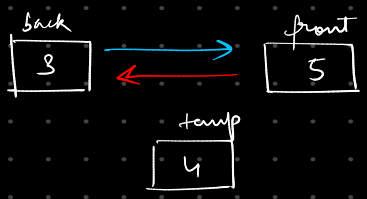
* Inserting a node at a given pos^n:

↳ similar to deleting a node.

↳ for inserting at head:
create a temp node & assign it's next to head & head's previous to temp node.
→ then move head to temp.

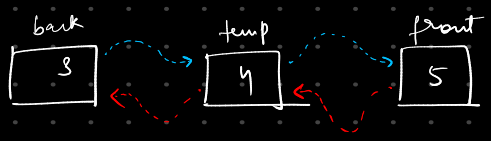
↳ for inserting at tail:
create a temp node & assign it's previous to tail & tail's next to temp &
→ then move tail to temp.

↳ for inserting in b/w two nodes: (before front)



back = front.prev.

→



→

Done ←

```
back.next = temp;
temp.next = front;
temp.previous = back;
back = temp;
```

↳ How to get to front?

↳ using counter.

→ follow similar steps to Deletion.