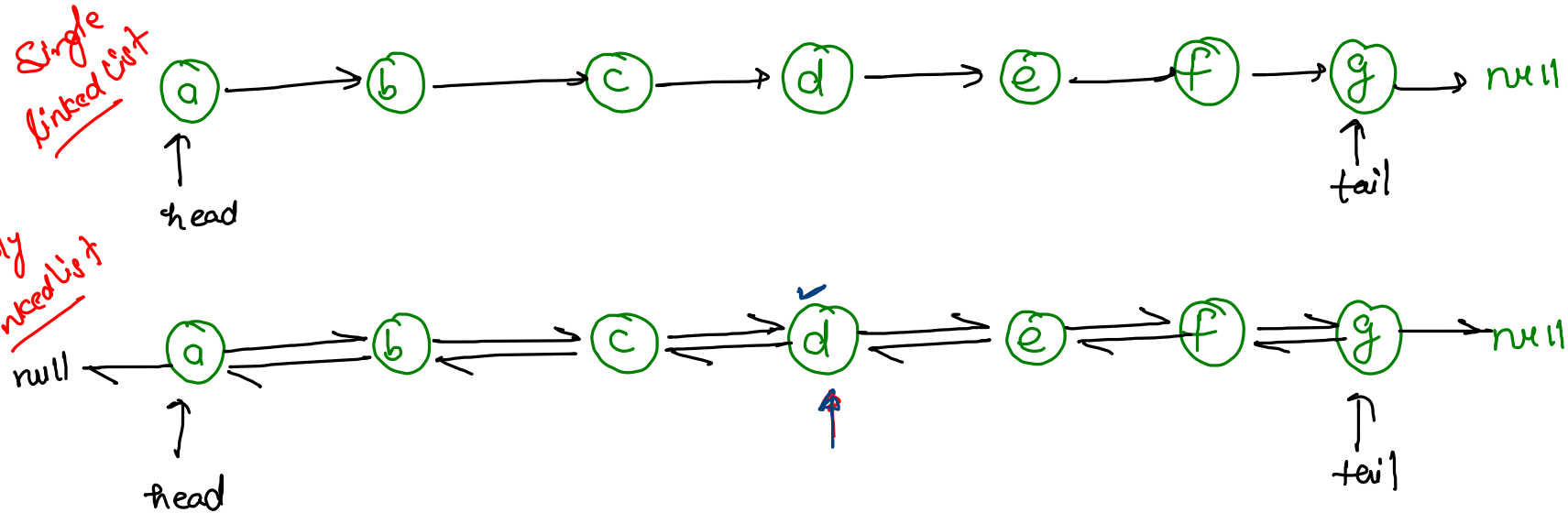
Doubly linked list:



Advantenge.

1. Remove last
2. Bi-direction traversal
   Forward + Backword
   ↑
   from any Node
3. If I have node them. after + Before
   - Remove
   - Add

   → I can Remove that node.

Disadvantage.

1. take one more pointer for every node. i.e → prev

Functions required in Doubly linked list :

1. Add First
2. Add Last
3. Add At
4. Add Before
5. Add After

→ Add, Node Give

6. Remove first
7. Remove Last
8. Remove At
9. Remove Before
10. Remove After

→ Remove, Node Give #

11. Get First
12. Get Last
13. GetAt

→ Get

14. Remove Node → Remove + Node

15. Forward Display
16. Backward Display.

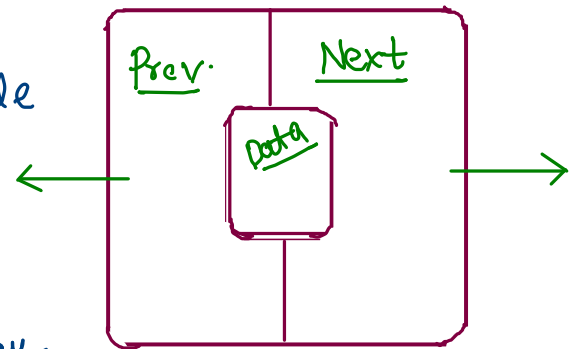
→ Display.

17. Size.

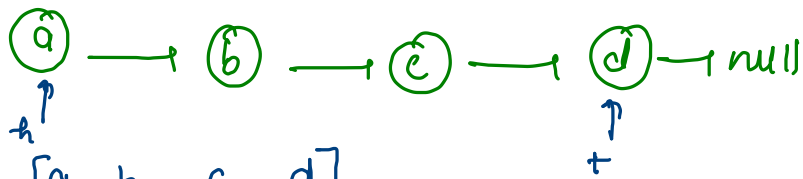
Data Member in DLL Class: →

1. Head, tail, size, Node class
2. Constructor.
3. 16 Member function [According to requirement]

Structure of Node in DLL →



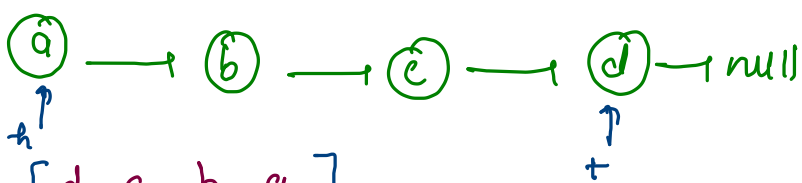
Forward display →



h = t = null

[ ]

Back ward display



h = t = null

[ ]

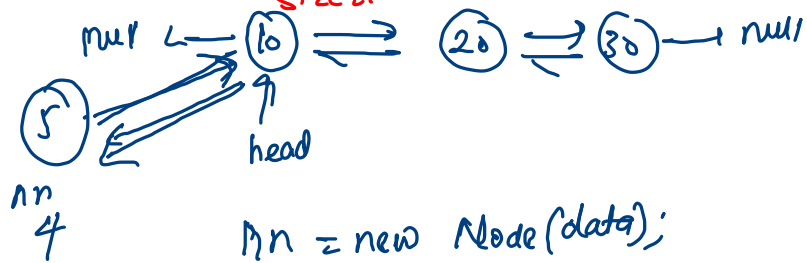
addFirst →

size = 0

nn = new Node(data)

head = tail = nn;

size = 1;



nn = new Node(data);

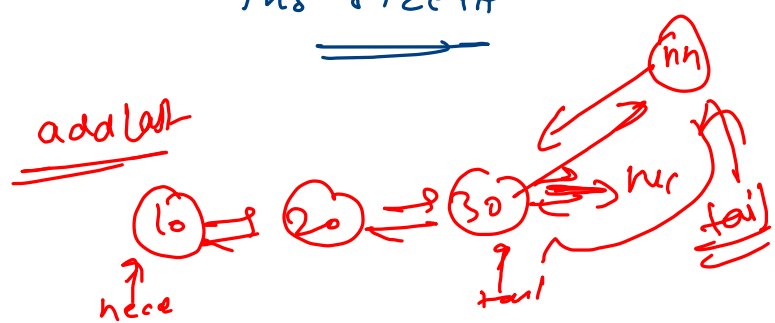
nn.next = head;

head.prev = nn

head = nn

Inc. size ++

addLast



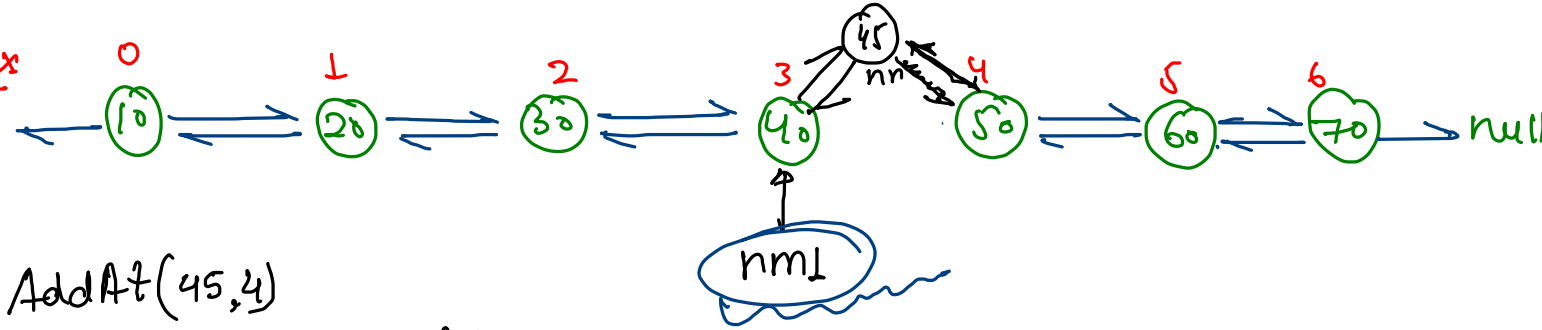
AddAt

pos<0 || size<pos
↳ Invalid pos.

pos = 0
↳ AddFirst

pos = size
↳ AddLast

otherwise.

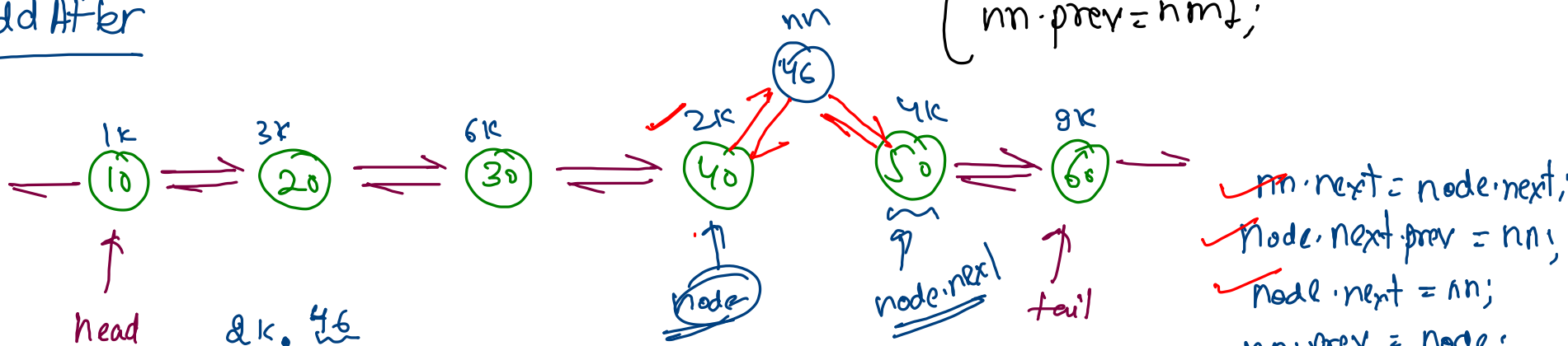


AddAt(45,4)

GetNodeAt → (3) ← Par-1

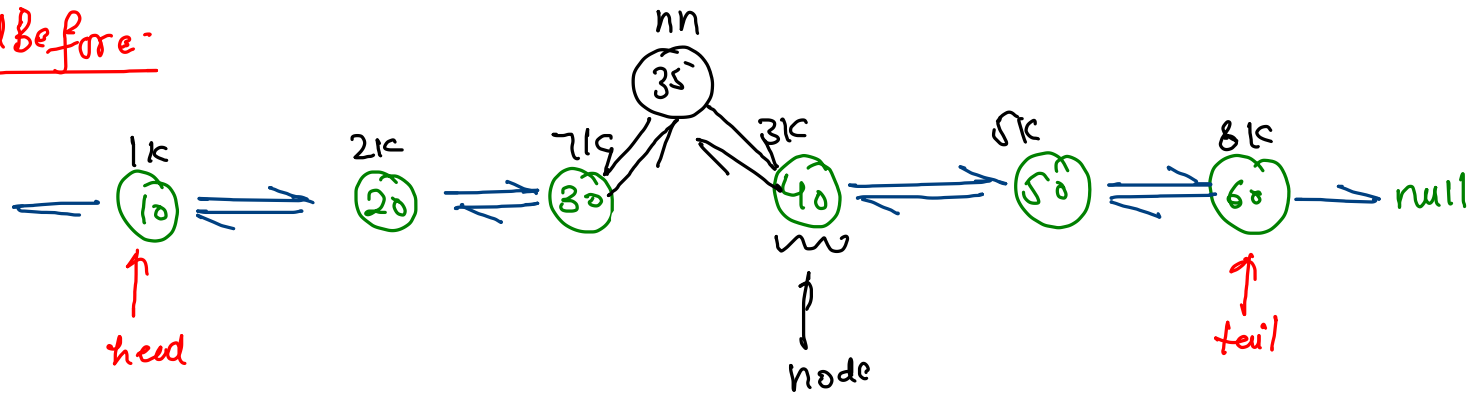
```
nn.next = nm1.next
nm1.next.prev = nn

nm1.next = nn;
nn.prev = nm1;
```

AddAfter




```
nn.next = node.next;
node.next.prev = nn;
node.next = nn;
nn.prev = node;
```

AddBefore:



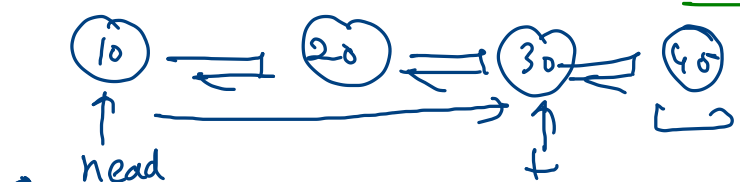
Node = 3k

data = 35

Edge case → Node == head

↓

addFirst

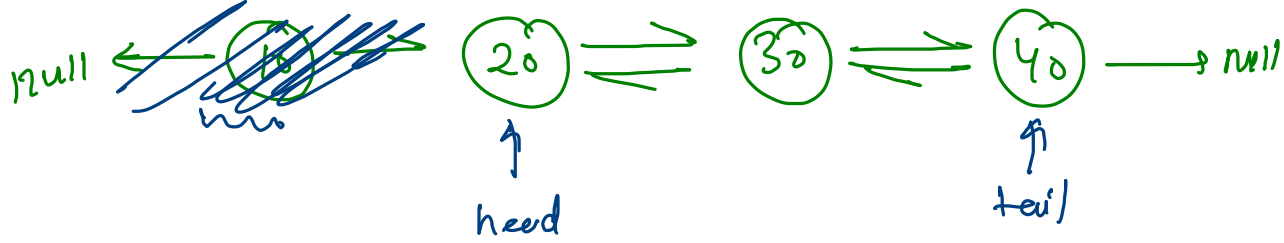


[val,

10, 20, 30, 40]

```
Node nn = new Node(data);
nn.prev = node.prev
node.prev.next = nn;
nn.next = node;
node.prev = nn;
```

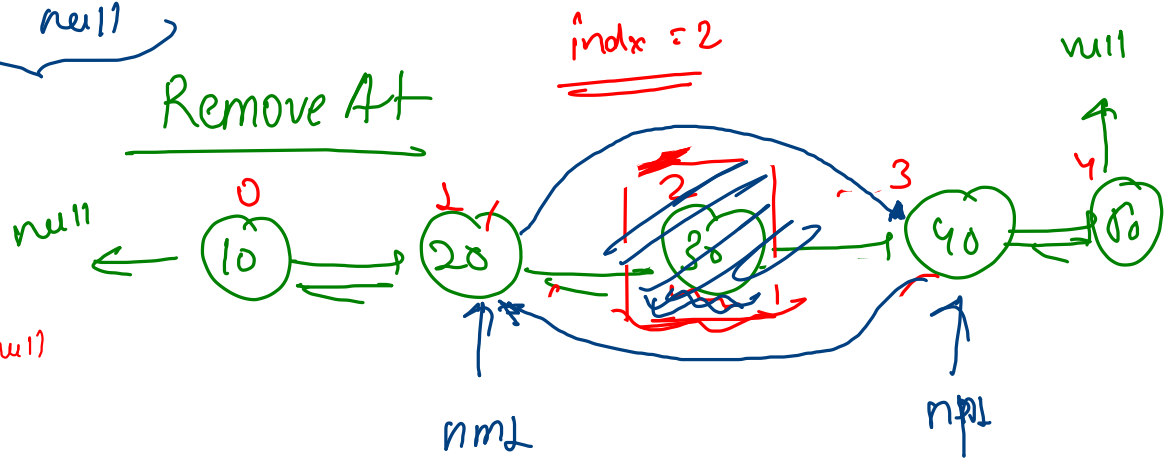
Remove first




```
val = this.head.data;
this.head = this.head.next;
this.head.prev = null
```

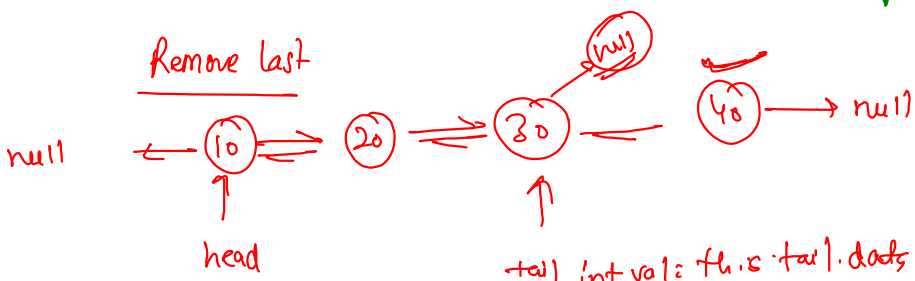
val = 10

Remove At




```
nm1.next = np1;
np2.prev = nm1;
```

Remove last




```
tail int val = this.tail.data
tail = tail.pre
tail.next = null
```