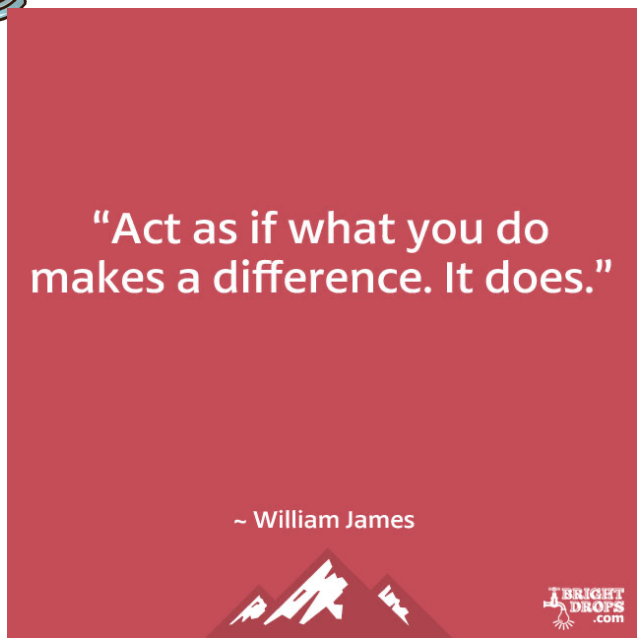
LINKEDLIST 3



Good Evening

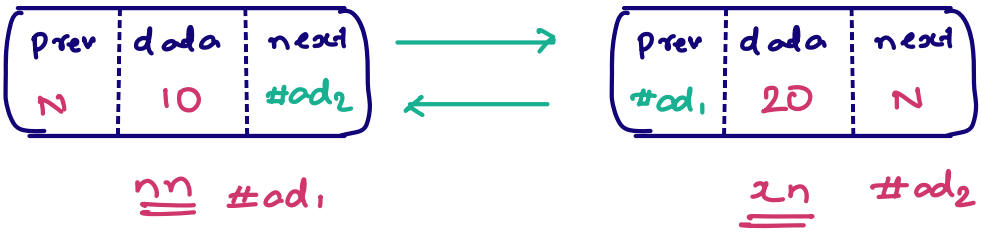
Today's content

01. Doubly linkedlist
02. Deletion in Doubly linkedlist → data, → obj reference
03. Insert a new node just before tail
04. LRU Cache
05. Clone a Linkedlist

Doubly Linked list

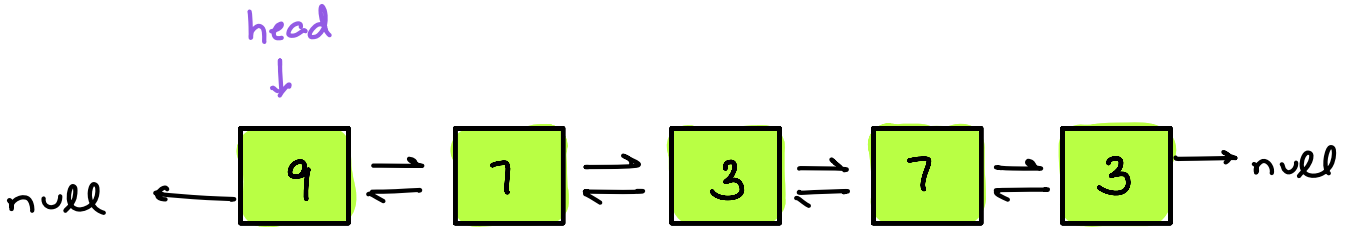

```
class Node {
    int data
    Node next;
    Node prev;
    Node ( x ) {
        data = 0;
        next = null;
        prev = null;
    }
}
```

```
Node nn = new Node (10);
Node xn = new Node (20);
```




```
nn.next = xn
xn.prev = nn;
```

Q Delete the first occurrence of data x in given DLL. (If not present → no update)

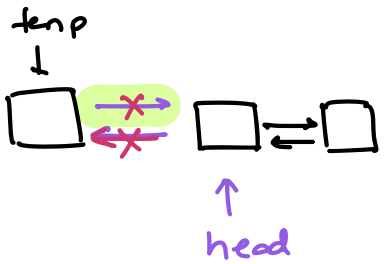


x = 7

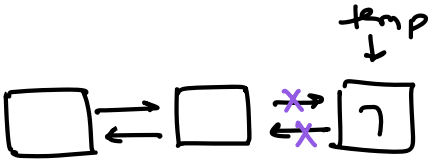
```
temp = head;
while (temp != null){
    if (temp.data == x) break;
    temp = temp.next;
}
```

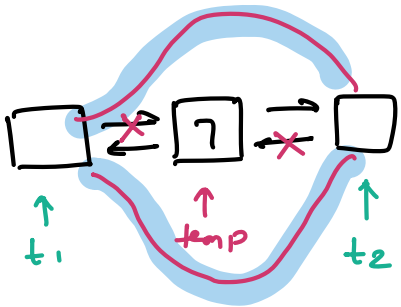
Search for x

```
if (temp == null) return head;
else if (temp.prev == null && temp.next == null)
    return null;
else if (temp.prev == null)
    head = temp.next
    temp.next = null;
    head.prev = null;
```



```
else if (temp.next ==null)
    temp.prev.next =null
    temp.prev =null;
}
else {
    Node t1 = temp.prev
    Node t2 = temp.next;
    t1.next = t2
    t2.prev= t1
    temp.prev= null:
    temp.next =null;
}
```





TC : O(n)

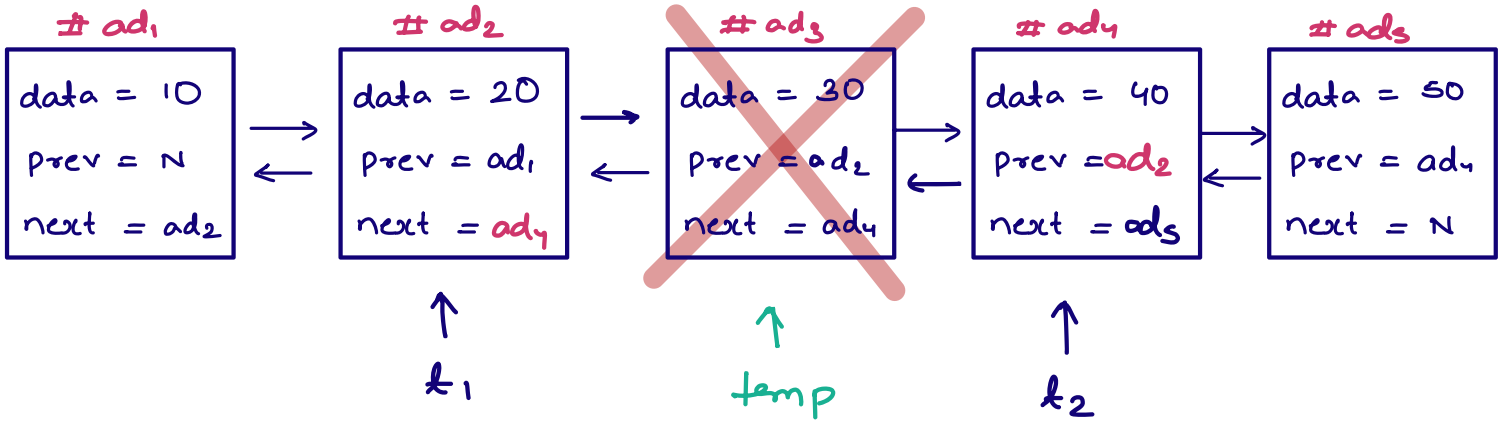
SC : O(1)

01. Delete a given node from DLL

Note:- Node reference / address is given

Note:- Given node is not head / tail node

Eg:- Delete #ad3




```
void deleteNode (Node temp)
{
    Node t1 = temp.prev;
    Node t2 = temp.next;

    t1.next = t2
    t2.prev = t1

    temp.prev = null ;
    temp.next = null ;
}
```

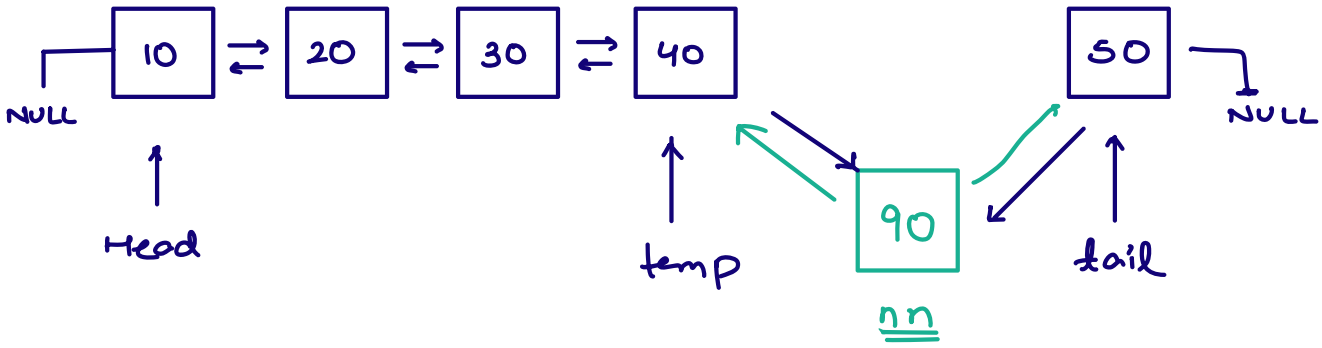
TC: O(1)

SC: O(1)

02. Insert a new node just before tail of doubly linked list

Note :- Tail reference is given in input

Note :- No. of nodes ≥ 2



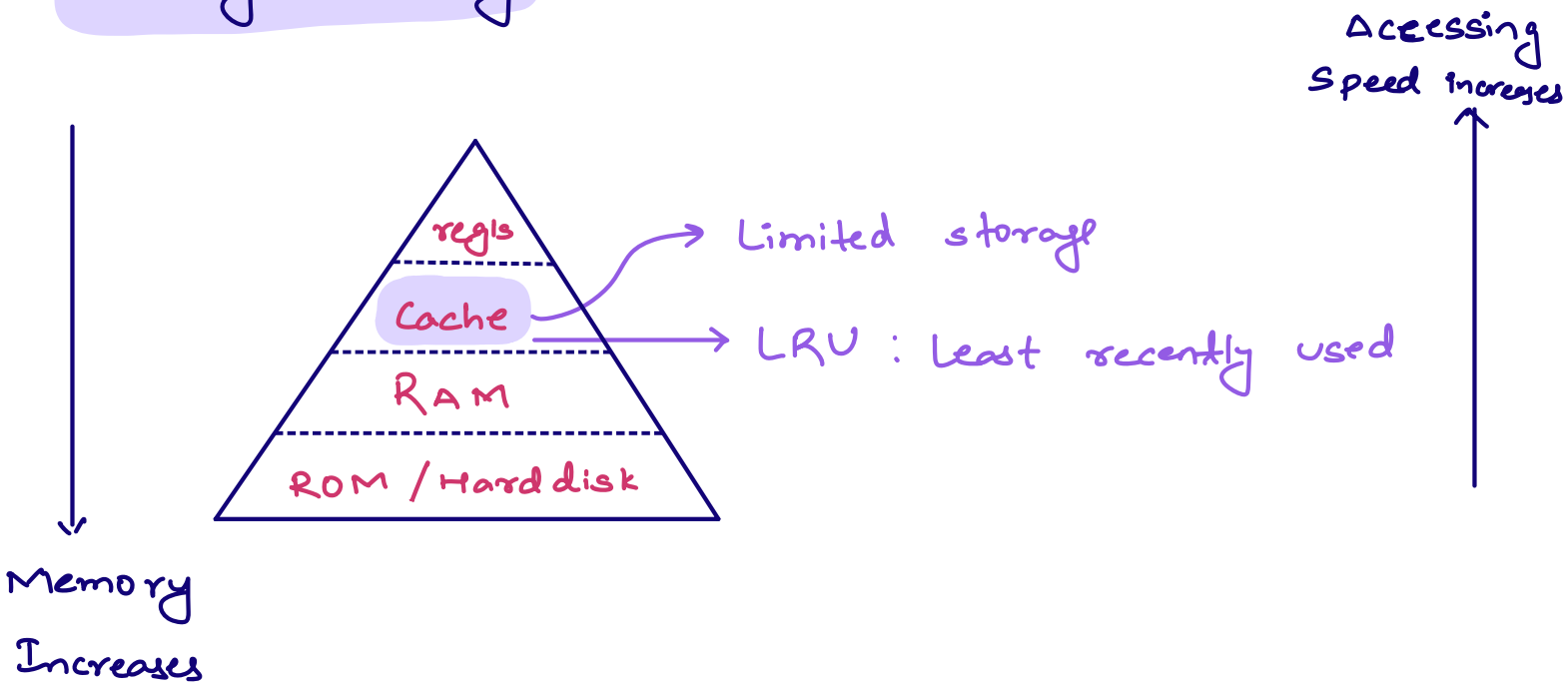
Insert 90

```
void insertback (Node nn, Node tail)
{
    Node temp = tail.prev

    nn.next = tail

    nn.prev = temp

    temp.next = nn;

    tail.prev = nn;
}
```

TC: O(1)

SC: O(1)

Memory hierarchy



Data : 7 3 9 2 6 10 14 2 10 15

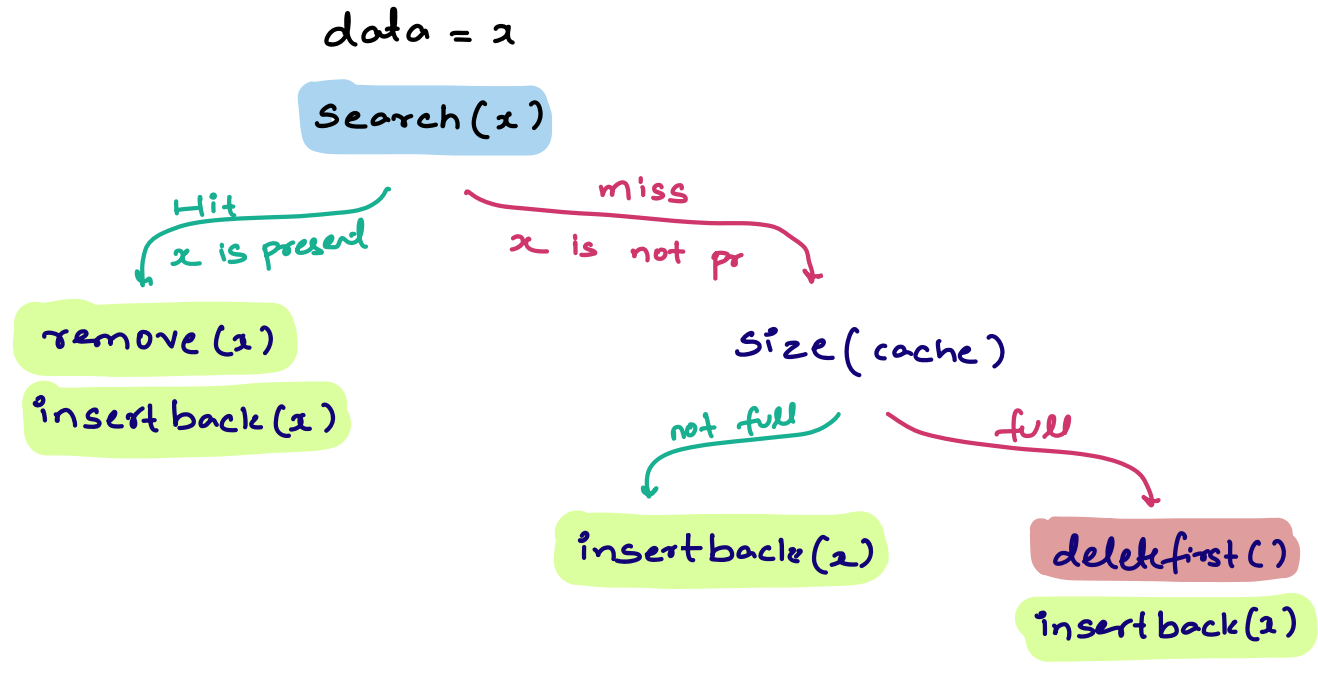
del 7 del3

ins 10 ins 14

Cache : 5

~~7~~ ~~3~~ ~~9~~ ~~2~~ 6 ~~10~~ 14 2 10 15

Flowchart



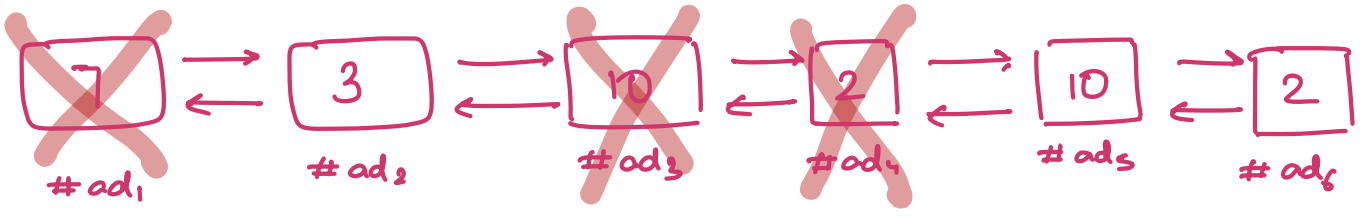
Operation	AL	SLL	SLL + Hashset
search(x)	O(n)	O(n)	O(1)
remove(x)	O(n)	O(n)	O(n) // iterate from start
insertback(x)	O(1)	O(1) → tail is given	O(1)
deletefirst(x)	O(n)	O(1)	O(1)

Operation	DLL + HashMap<int, Node> (data, address)
search(x)	O(1)
remove(x)	O(1)
insertback(x)	O(1)
deletefirst(x)	O(1)

Data = 7, 3, 10, 2, 10, 2 Cache = 3

HashMap { <7, #ad1> , <3, #ad2> , <10, #ad3> <2, #ad4> }

DLL =



* Create a dummy head & dummy tail to save yourself from the edge cases

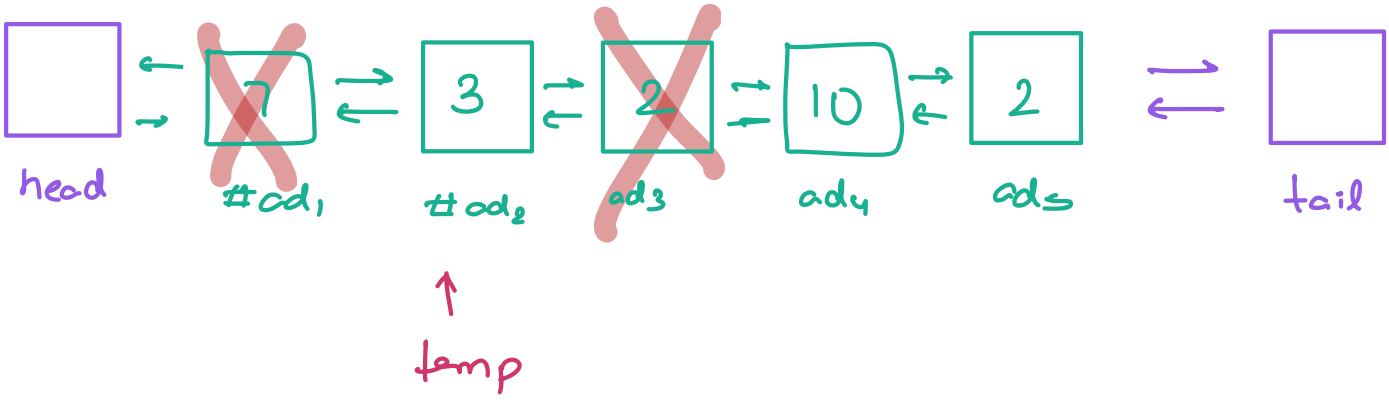
```
Node head = new Node(-1)
Node tail = new Node(-1)
head.next = tail;
tail.prev = head;
```

```
..... LRU (int x, int limit)
    if (hm.containskey(x) == true)
        Node tmp = hm.get(x)  // address of x
        deleteNode(temp);
        Node nn = new Node(x);
        insertback(nn, tail);
        hm.put(x, nn);
    }
    else {
        if (hm.size() == limit)
            Node temp = head.next;
            deleteNode(temp);
            hm.remove(temp.data);
        }
        Node nn = newNode(x)
        insertback(nn, tail);
        hm.put(x, nn);
    }
```

Data = 7, 3, 2, 10, 2

limit = 3

HashMap = { <7, ad1> <3, ad2> <2, ad3> <10, ad4> }



temp = ad3

10:31 pm → 10:41 pm

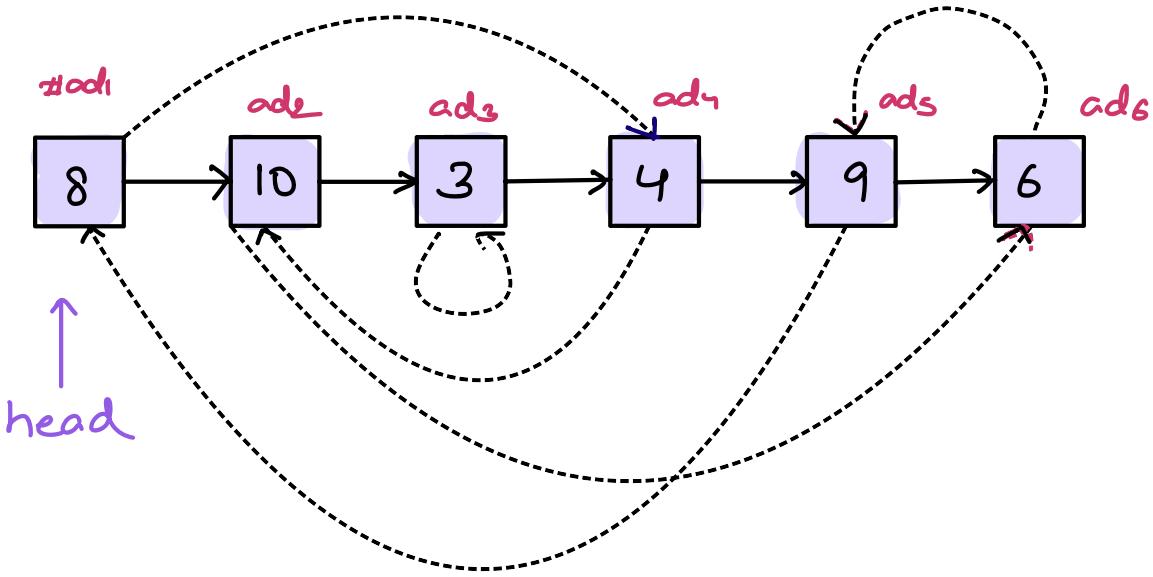
clone LinkedList

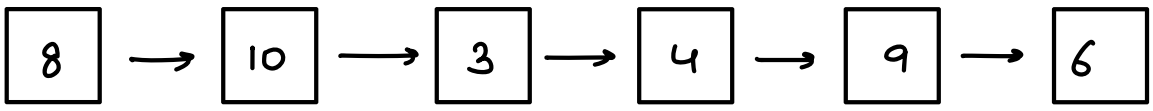
```
Class Node {
    int data;
    Node next; // pointing to next node
    Node rand; // pointing to any one node in LL.
}
```

Note :- Rand is not null

Given a LinkedList, create & return clone of it.

Expected SC: O(1)





Brute force

TC: $O(n^2)$

SC: O(1)

01. Create a clone ll just by using next ptr

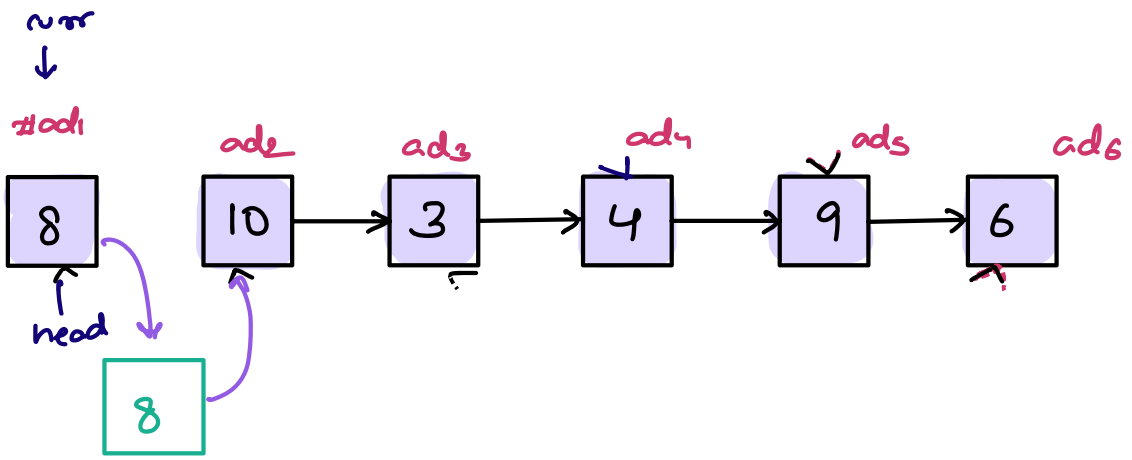
02. For every node in a given LL, we need to get the random ptr

03. Iterate on cloned LL & search for the data

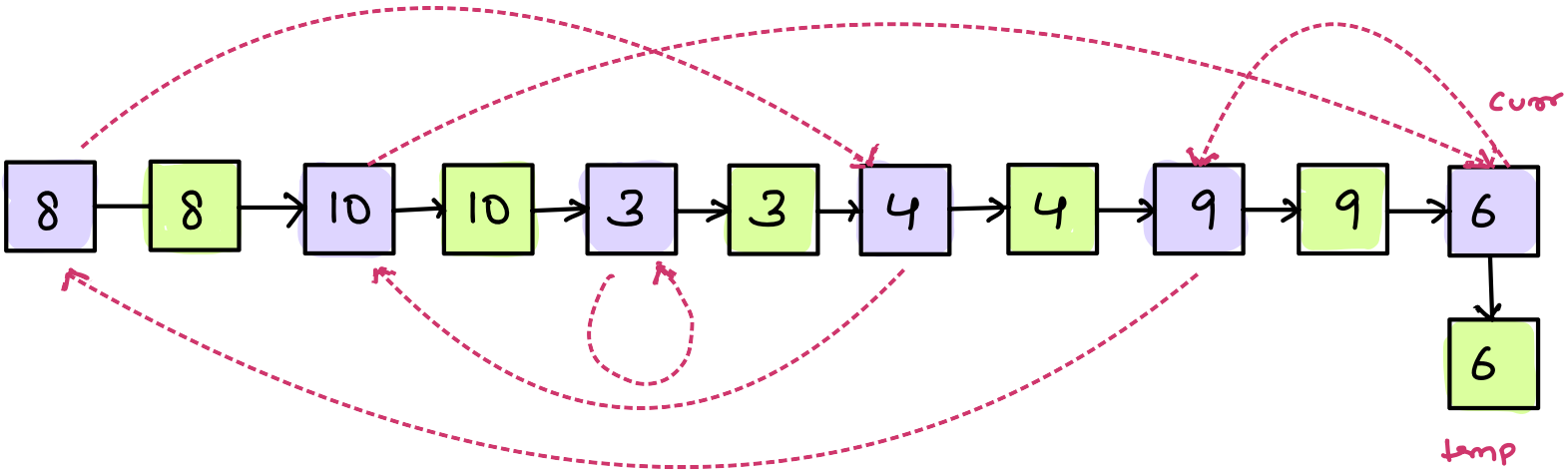
04. Create the random link with that node.

Idea 2

01. Interleaved all the new nodes in between the old linkedlist




```
Node curr = head :

while (curr != null)
|
|     Node nn = new Node (curr.data)
|
|       nn.next = curr.next ;
|
|       curr.next = nn ;
|
|       curr = curr.next.next ;
|
3
```



02 Populate the random ptr for green LL

temp.random = curr.random.next

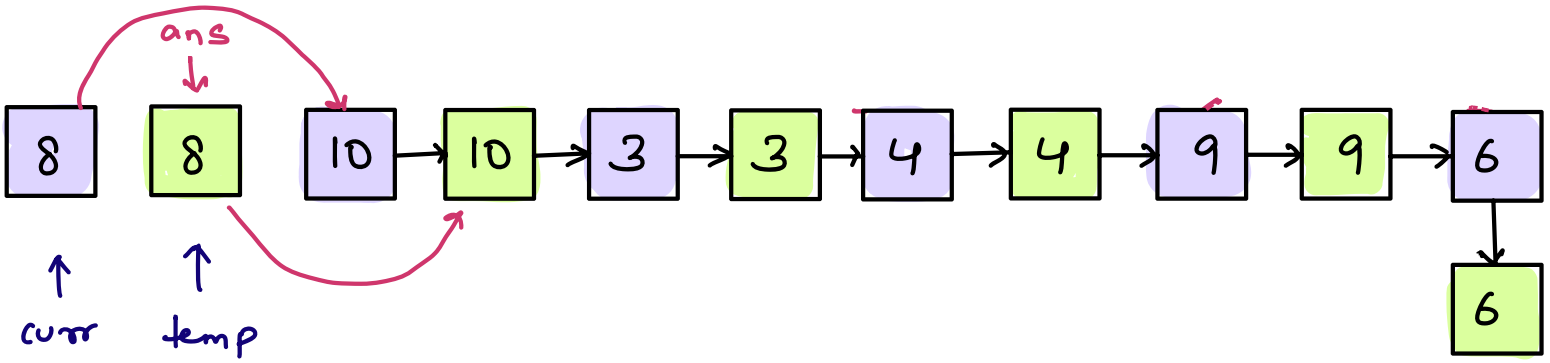
4

4

```
Node curr = head;
Node temp = head.next;

while (curr != null) {
    temp.rand = curr.random.next

    curr = curr.next.next;
    if (temp.next != null) {
        temp = temp.next.next;
    }
}
```

// handle edge case if any

03. Detatch both the LL



curr = head

temp = head.next

ans = temp

while (curr != null)

curr.next = curr.next.next

if (temp.next != null) {

temp.next = temp.next.next; // handle the edge case

}

curr = curr.next;

temp = temp.next;

}

return ans;

TC: O(n)

SC: O(1)