Linkedlist 3

"We don't just sit around and wait for other people. We just make, and we do."

Arlan Hamilton

Today's content

01. Doubly linkedlist

02. Insert a new node just before tail

03. Deletion in Doubly LL.

04. LRU Cache

05. Clone of a Linkedlist (If time permits)

Time complexity to merge two sorted LL

TC : O(n+m)

SC : O(1)

Starting point of cycle

→ Detect a meeting point if cycle is present

→ Move one of the pointer at head & then take one step with both slow & fast ptr.

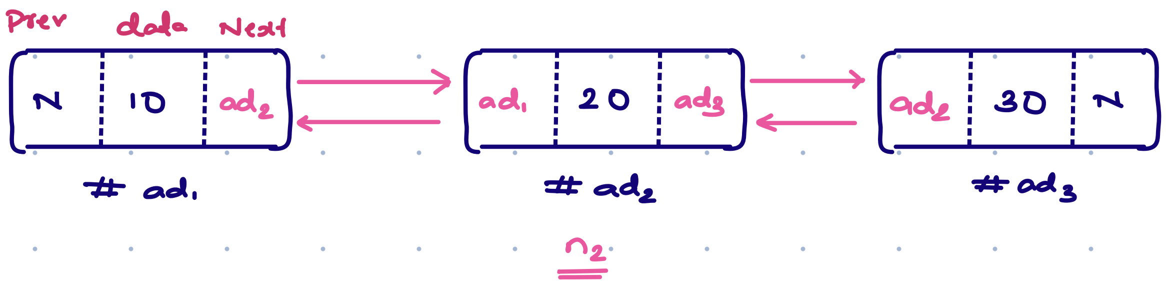
* Doubly Linkedlist

→ stores collection of element.

→ Class of DLL will contain one extra pointer which points towards the previous element.

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

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

```
Node  n1 = new  Node (10);      # head
Node  n2 = new  Node (20);
Node  n3 = new  Node (30);
```



$n_1.next = ad_2$

$n_2.prev = ad_1$

$n_2.next = ad_3$

$n_3.prev = ad_2$

* Spotify

Add new song → Add a new node & connect its previous ptr to me

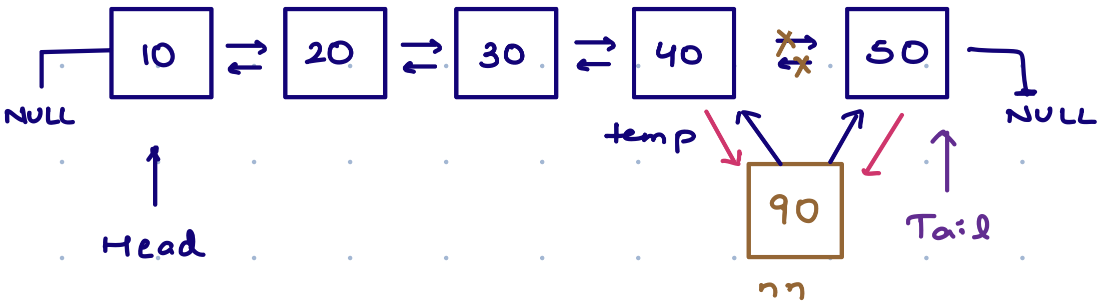
Play next song → node.next

play prev song → node.prev

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

Note :- Tail reference is given in input

Note :- No. of nodes ≥ 2



```
void insertback ( Node Tail , Node nn){

    Node temp = tail.prev

    nn.prev= temp;
    nn.next = tail;

    temp.next = nn;

    tail.prev= nn;

}
```

TC : O(1)

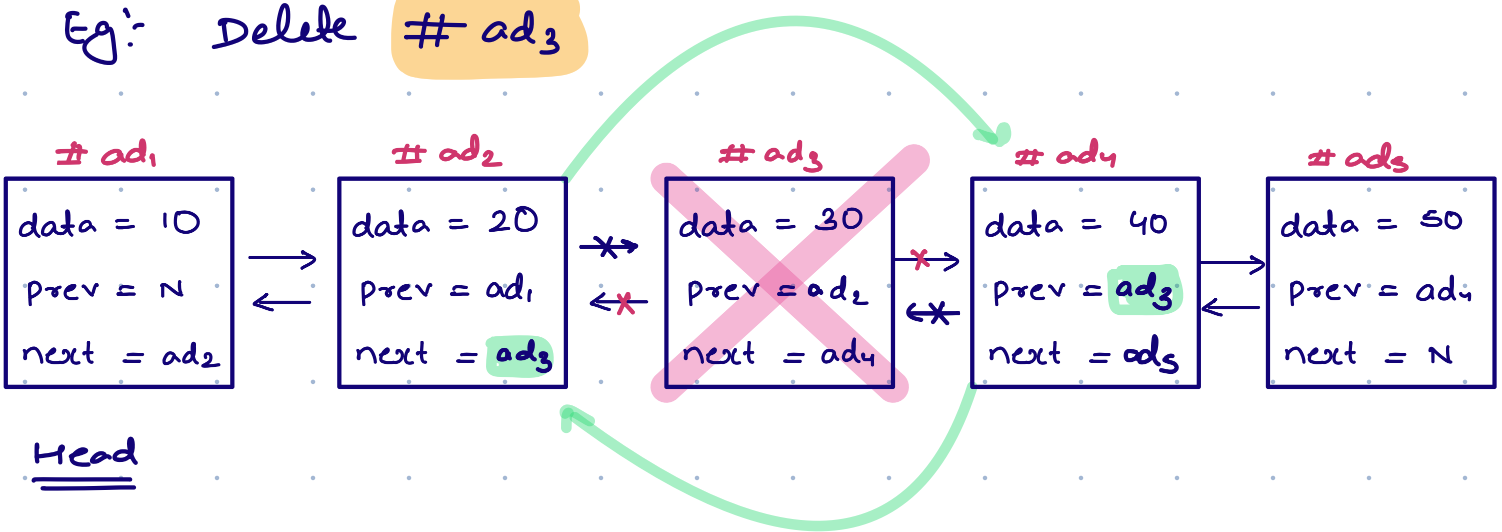
SC : O(1)

Q2. Delete a given node from DLL

Note :- Node reference / address is given

Note :- Given node is not head / tail node

Eg :- Delete # ad₃



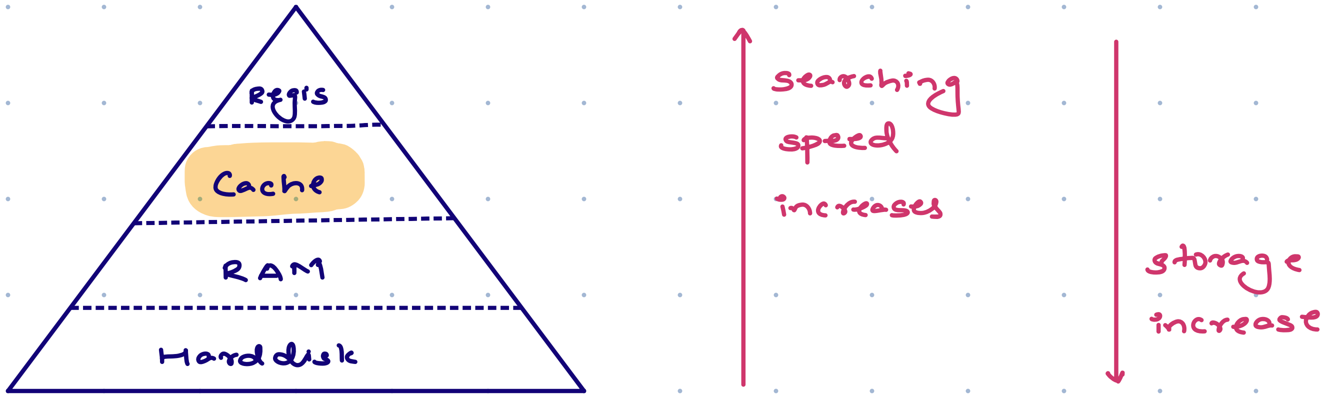
Node to be deleted

```
void deleteNode ( Node head, Node temp){

        Node t₁ = temp.prev;
        Node t₂ = temp.next;

          t₁.next = t₂
          t₂.prev = t₁

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

Tc : O(1)

Sc : O(1)

* Memory Hierarchy



Cache → Limited storage

Maintain most recently used data, & remove least recently used items.

LRU Cache Memory

Q Given a running stream of Integer & fix memory of size M, we have to maintain the most recent M elements.

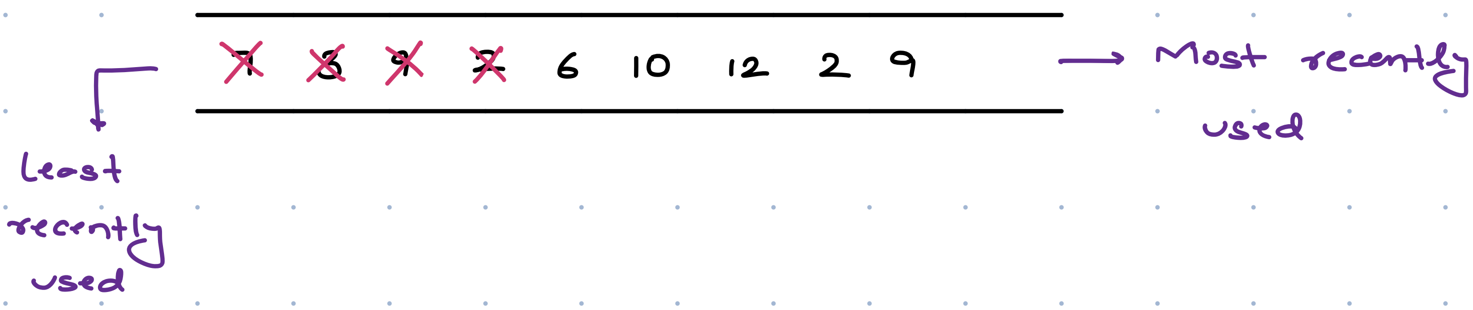
Note → If memory is full, delete the least recently used item from the memory.

Data : 7 3 9 2 6 10 12 2 9

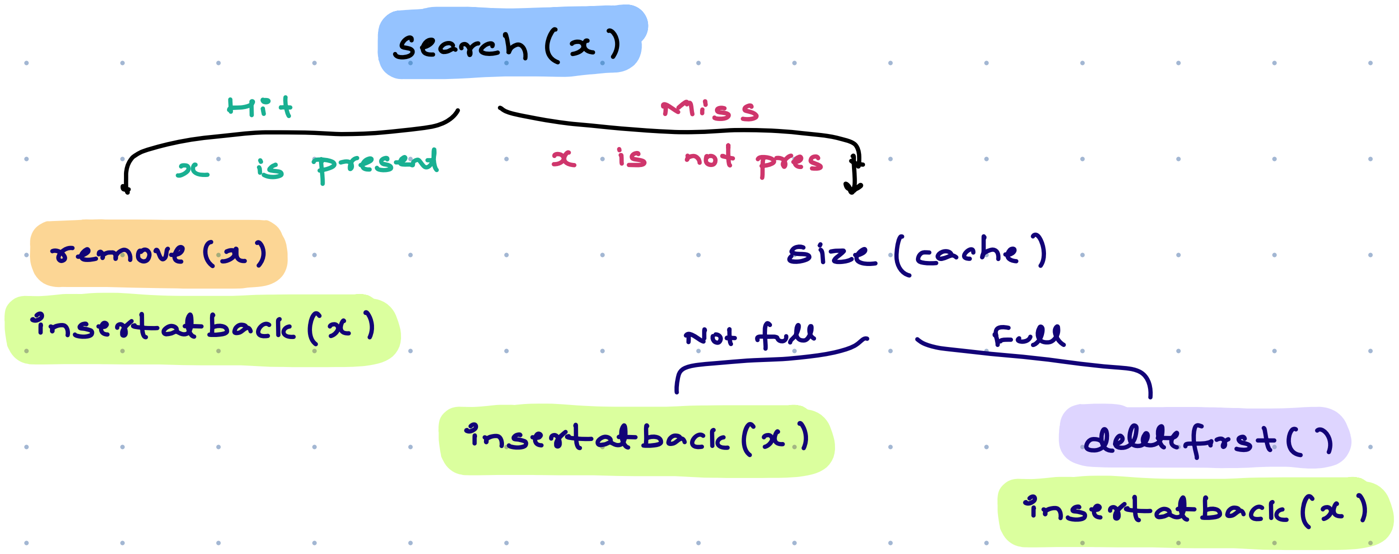
Cache : 5

del 7 del 3 del 2

add 10 add 12 add 2



data = x



Operations	ArrayList	Singly LL	LL + Hashset
search (x)	O(n)	O(n)	O(1)
remove (x)	O(n)	O(n)	O(n) // iterate on LL
insertatback(x)	O(1)	O(1) tail is given	O(1)
deletefirst( )	O(n)	O(1)	O(1)

Operations

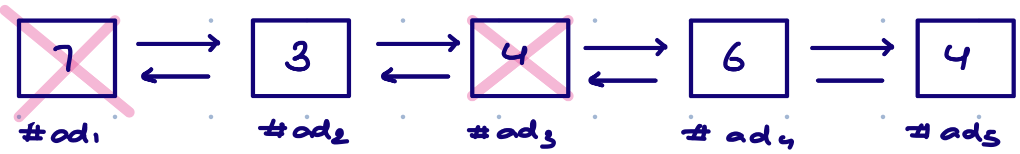
Doubly Linkedlist + HashMap $<\overset{\text{data}}{\text{Integer}}, \overset{\text{address}}{\text{Node}}>$

Search(x)	O(1)
remove (x)	O(1)
insertatback(x)	O(1)
deletefirst( )	O(1)

* Data : 7 3 4 6 4 Cache = 3

HashMap = { <7, #ad$_1$>, <6, ad$_4$> <3, #ad$_2$> <4, ad$_5$> }

DLL =

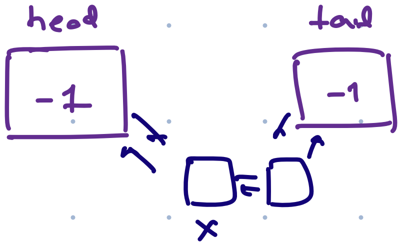


* Create two dummy nodes, as dummy head & dummy tail

save ourself from

- → Addition for first time
- → Deletion from front
- → Deletion from tail
- → Addition at tail

```
class   LRU {

    HashMap<I, Node> hm = new HashMap<>();

     Node  head = new  Node (-1);
     Node  tail  = new  Node (-1);

          head.next = tail;
          tail. prev =  head

   public  add (int x, int limit)

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



10:29 → 10:35 pm

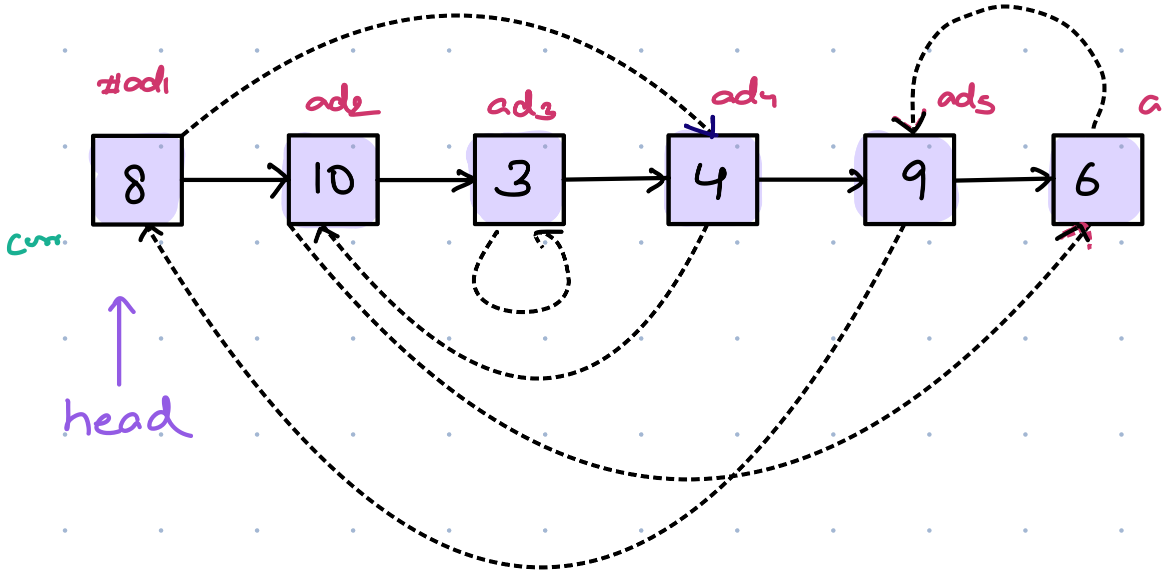
clone Linkedlist

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

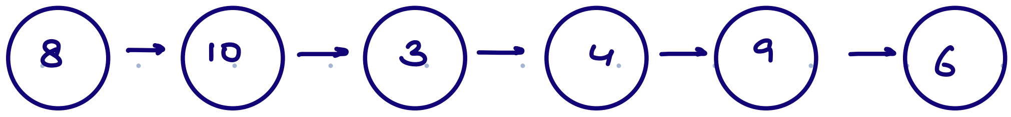
Note :- Rand is not null.

Given a LinkedList, create & return clone of it.

Expected SC:O(1)



Brute force



01. Create a clone of LL just by using next pointer.

02 For every node in given LL, we need to go & get

random ptr.

03. Iterate on cloned LL & get that random node.

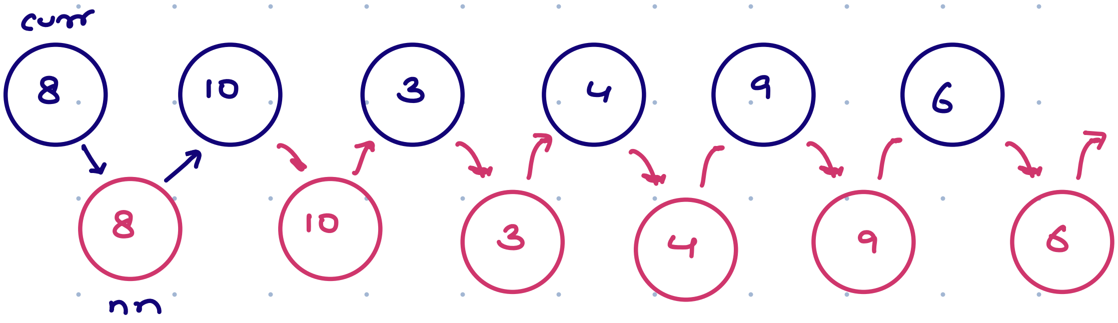
04. Create a random pointer in cloned LL.

TC : O(n^2)

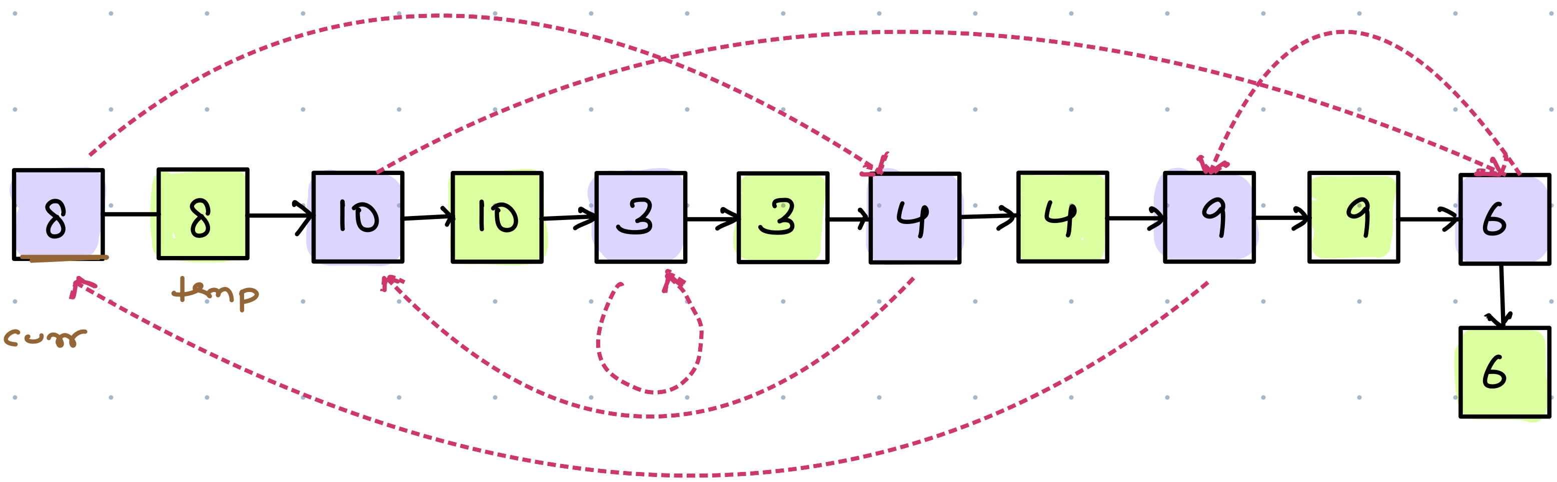
SC : O(1)

* Idea 2 Optimal Approach

01. Interleave all the new node in between old nodes



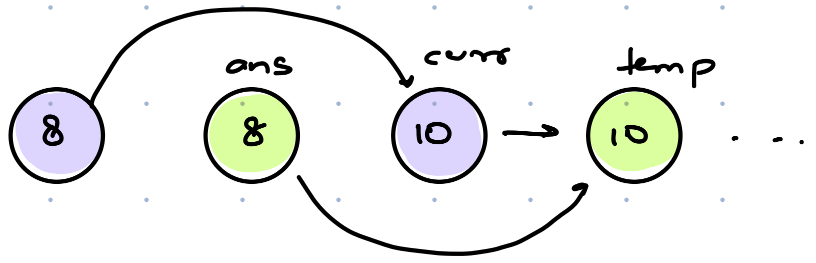
```
Node curr= head

while (curr != null) {
    Node nn= new Node (curr.val);
      nn.next = curr.next ;
      curr. next = nn ;
     curr= nn.next;
}
```



* populate the random ptr for given Linkedlist

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

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

    curr= curr.next.next;

    if (temp.next != null) temp= temp.next.next;

}
```

* Detatch both old & new Linkedlist



TC: O(n)

SC: O(1)

```
Node curr= head

Node temp= head.next

Node ans = temp


while (curr != null){

    curr.next = curr.next.next;

    if (temp.next != null) temp.next = temp.next.next;

    curr= curr.next ;

    temp = temp.next ;

}
return ans;
```

Tries → DSA 4.2