Todays Content

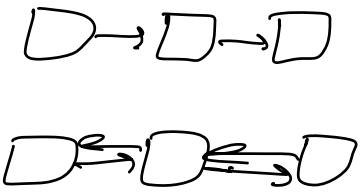
1) Doubly Linked List
2) LRU Cache
3) Clone Linked List

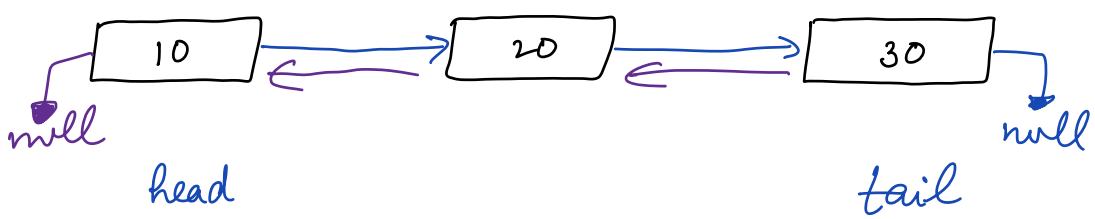
Frequently

Doubly linked List

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

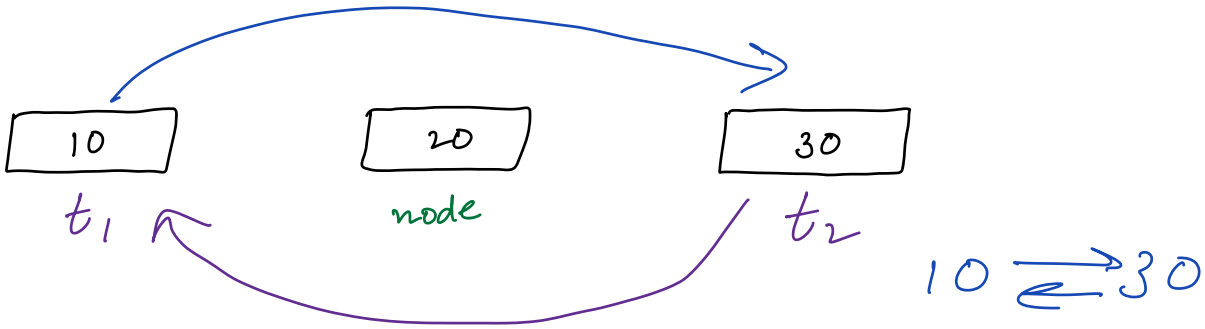
prev ← data → next



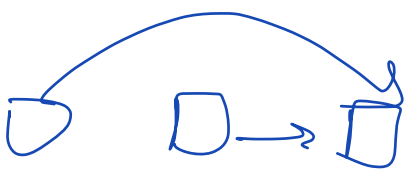


1) Delete a node from DLL

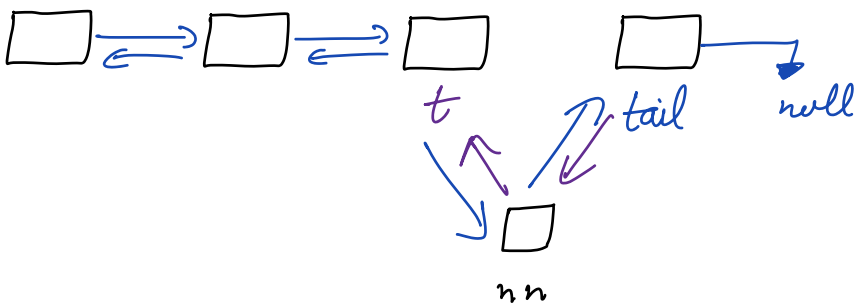
Note: Given node is not head / tail




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



Q Insert a new node just before tail node in a DLL



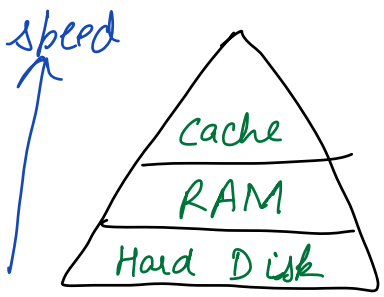

```
void insertBack (Node nn, Node tail) {
    Node t = tail.prev
    t.next = nn
    tail.prev = nn
    nn.prev = t
    nn.next = tail
}
```

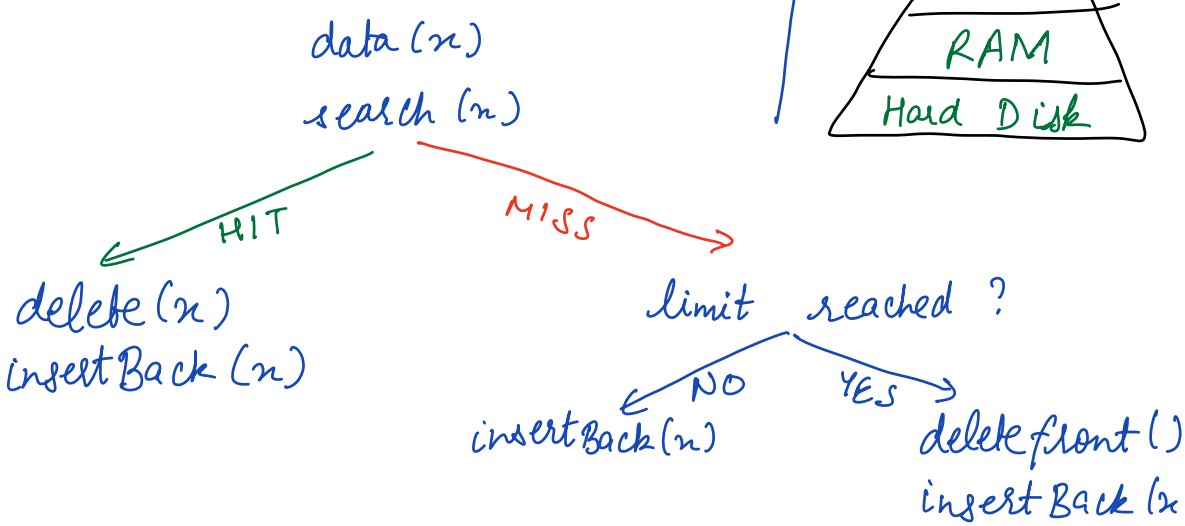
Cache: Fast access memory (Limited capacity)

LRU: Least Recently used (सबसे पुराना)

7 3 9 2 6 10 14 2 10 15 8 14 (HIT)

Cache (5) | 2 | 10 | 15 | 8 | 14 |

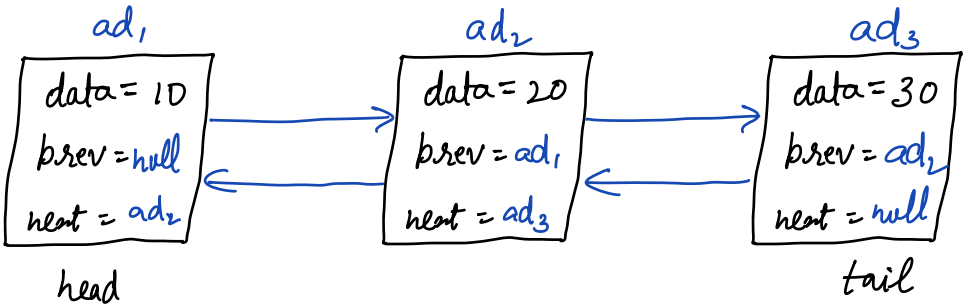




Note: Duplicates not allowed

GS LRU

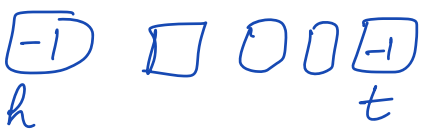
Operation	Arraylist	DLL + Hashmap
search (x)	O(n)	O(1)
remove (x)	O(n)	O(1)
insertBack (x)	O(1)	O(1)
delete front ( )		O(1)



Storage in Hashmap

Hashmap <int, address of Node>

<10, ad1>
<20, ad2>
<30, ad3>



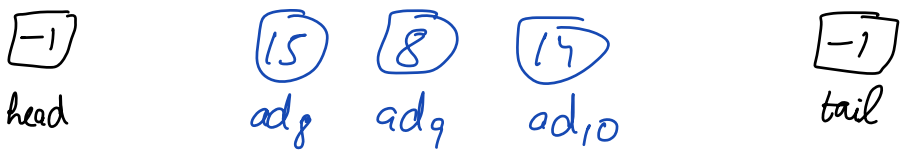
<node data, node address>

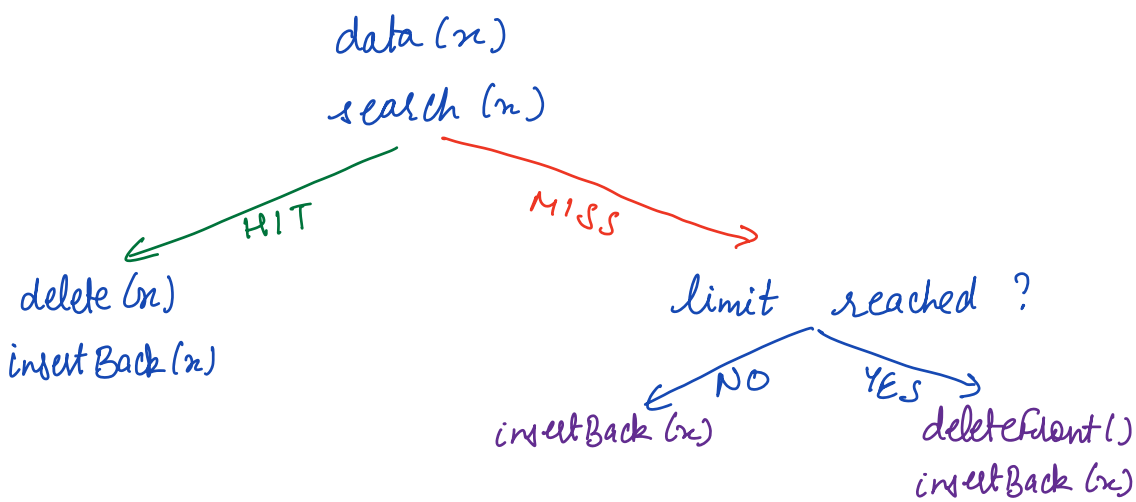
LRU cache using DLL + Hashmap

7 3 9 10 14 9 10 15 8 14

Cache (3)

Hashmap ⇒ { {14, ad_{10}} , {15, ad_8} , {8, ad_9} }





Code

```
Node head = new Node(-1)
Node tail = new Node(-1)
head.next = tail
tail.prev = head
HashMap<int, Node> hm
```

-1 ⟷ -1

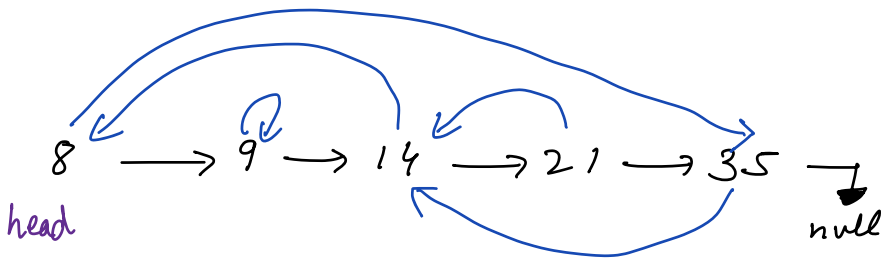
head ⟵ tail

```
void LRU(int x, int limit) {   //insert
    if(hm.containsKey(x)) {          HIT
        Node t = hm.get(x)
        DeleteNode(t)
        Node nn = new Node(x)
        insertBack(nn, tail)
        hm.put(x, nn)
    }
    else {          // MISS
        if(hm.size() == limit) {
            Node t = head.next
            int value = t.value
            DeleteNode(t)
            hm.remove(value)
        }
        Node nn = new Node(x)
        insertBack(nn, tail)
        hm.put(x, nn)
    }
}
```

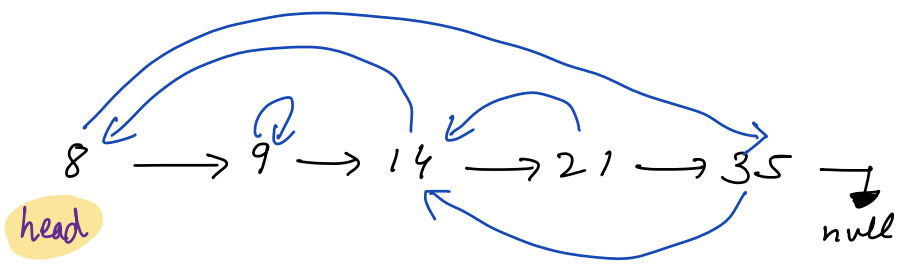
make a copy

Q. Clone Linked List

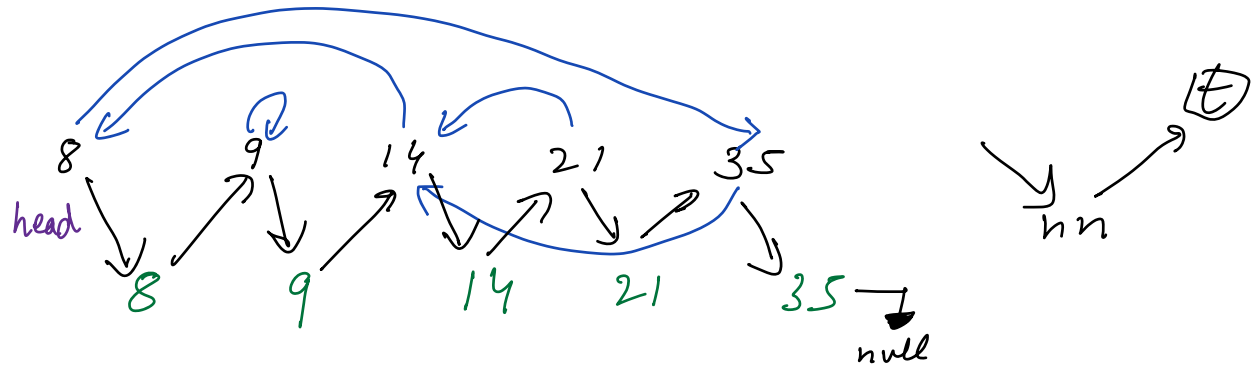
```
class Node {
    int data
    Node next
    Node rand      // pointing to any random node in LL
}
```



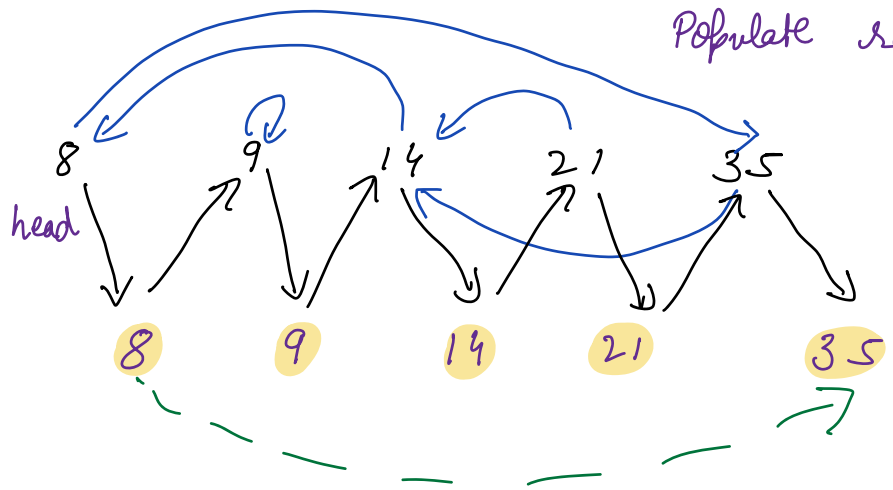
Make a copy of this



Idea



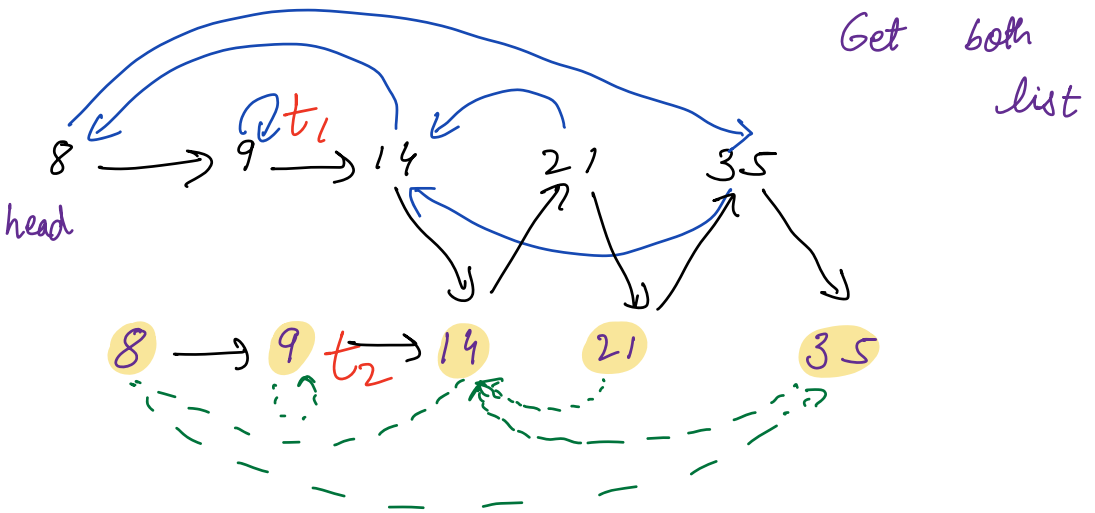

```
Node temp = head
while (temp != null) {
    Node nn = new Node(temp.value)
    nn.next = temp.next
    temp.next = nn
    temp = nn.next
}
```



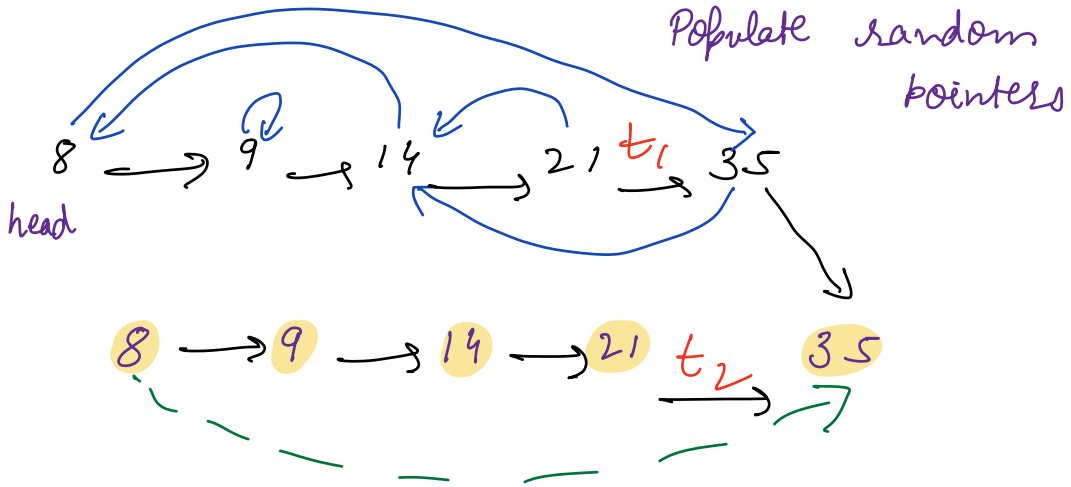

```
old.next.random = old.random.next

Node old = head
while (old != null) {
|
```

```
    old.next.random = old.random.next
    old = old.next.next
}
```




```
dh = h.next                    // duplicate head
t1 = h
t2 = h.next
while (t1 != null) {
    t1.next = t2.next
    t1 = t1.next
    t2.next = t1.next
    t2 = t2.next
}
return dh
```



$t_1.next = t_2.next$

$t_1 = t_1.next$

$t_2.next = t_1.next$

$t_2 = t_2.next$

done

$t_1.next = t_1$

$t_1 = t_1.next$