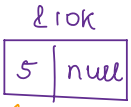
Lecture: LL-3

```
x = new Node(5);
```



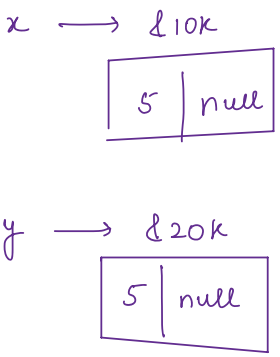

```
y = x
```

→ Shallow copy.

```
y.data = 10;
print(x.data)  // 10
```

```
x = new Node(5);
y = new Node(5);
y.data = 10;
print(x.data) // 5
```

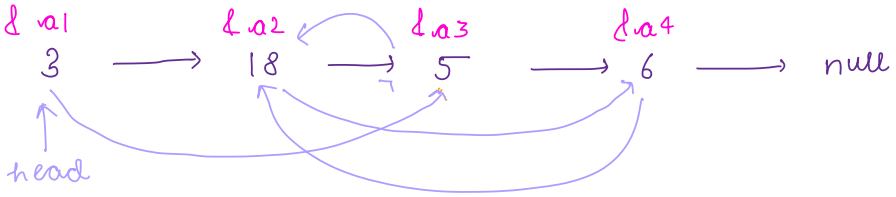
Deep copy



Qu. Given a LL, where every node has a random pointer along with next pointer. Create a deep copy of LL.

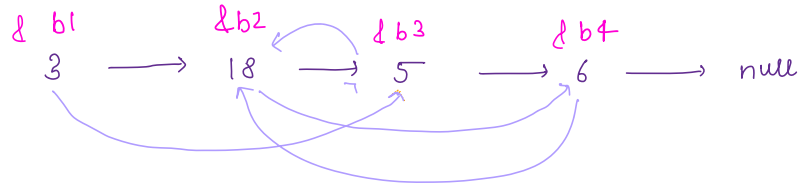
Ex:

ip:

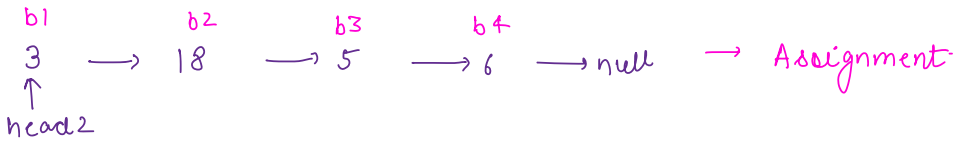



```
class Node {
    int data;
    Node next;
}
```

op:



Idea: 1. Iterate over original LL and create duplicate LL using deep copy. [medium]



2. Map :

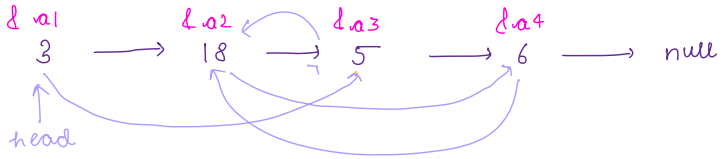
a1	:	b1
a2	:	b2
a3	:	b3
a4	:	b4

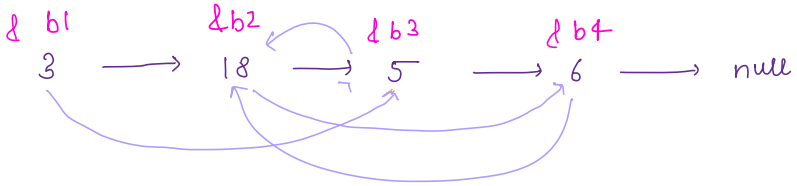
```
temp = head [a1]
h2 = head [b1]
rand-org = temp.random // 5 a3
h2.random = map.get(rand-org);
```

TC: O(n)
SC: O(n)

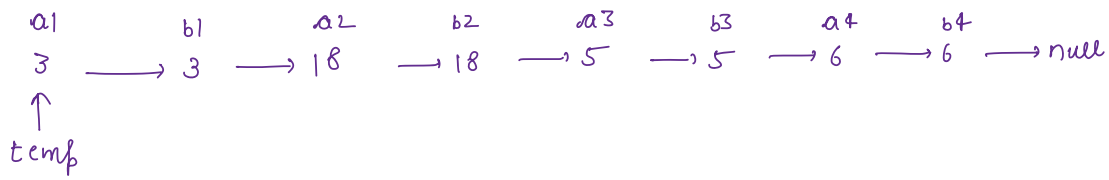
Approach 2 Do it in O(1) space. [Hara approach]

→





Step1: Attach original and dup. LL.



temp.next.random = temp.random.next

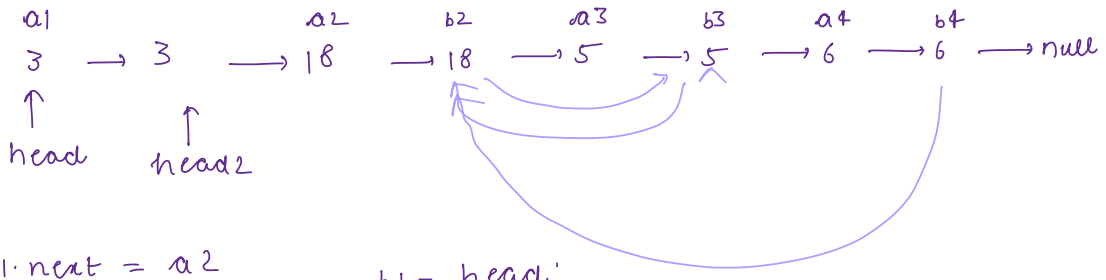
(temp.next = b1; temp.random = a3; a3.next = b3)

b1 random → b3

```
temp = head;
while( temp != null) {
    temp.next.random = temp.random.next
    temp = temp.next.next;
}
```

Edge case: 1. if (temp.random == null) {

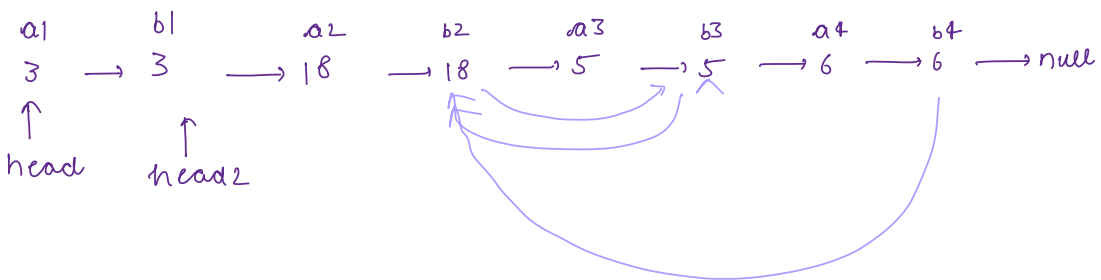
```
    temp.next.random = null;
}
```

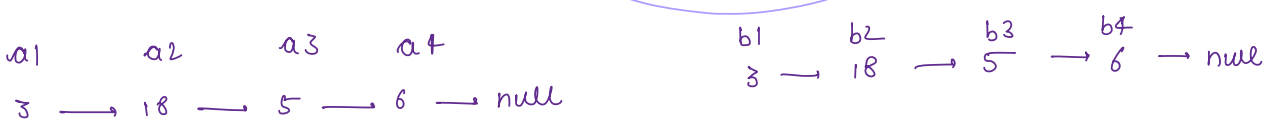


a1.next = a2

b1.next = b2

```
h1 = head;
h2 = head1[head.next]
while(h1 != null){
    h1.next = h1.next.next;
    h2.next = h2.next.next;
    h1 = h1.next;
    h2 = h2.next;
}
return head1;
}
```





Doubly linked list

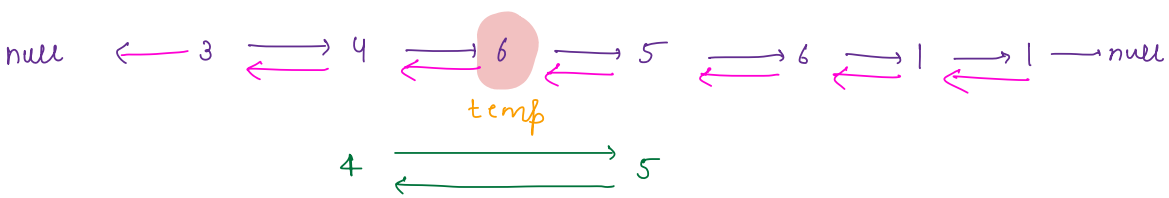
Singly LL: 1 → 3 → 7 → 9 → null

Doubly LL: null ← 1 ⇄ 3 ⇄ 7 ⇄ 9 → null

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

Qu. Delete the first occ of given el in DLL.

Sol:



temp.prev (4) . next = temp.next (5)

temp.next (5) . prev = temp.prev (4)

```
Node deleteFirstOccurrence(Node head, int x) {
    temp = head;
    while(temp != null) {                    // find el.
        if(temp.data == x) {
            break;
        }
        temp = temp.next;
    }
    if(temp == null) {                       // x is not in DLL
        return head;
    }
```

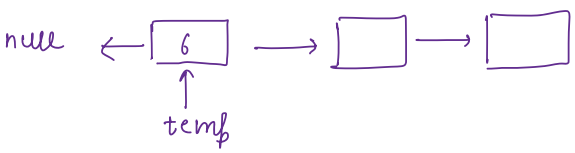
case1:

temp.prev (4) . next = temp.next (5)

temp.next (5) . prev = temp.prev (4)

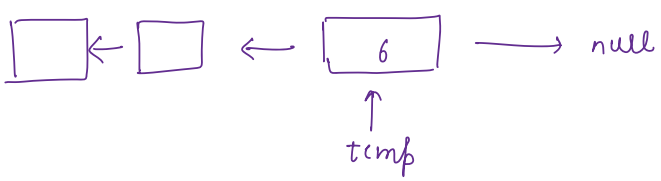
```
if ( temp.prev == null && temp.next == null ) {
    return null;
}
```

case2:




```
if ( temp.prev == null ) {
    temp.next.prev = null;
    head = head.next;
    return head;
}
```

case3:




```
if ( temp.next == null ) {
    temp.prev.next = null;
    return head;
}
```

case4:

```
else {
    temp.prev (4) . next = temp.next (5)
    temp.next (5) . prev = temp.prev (4)
}
return head;
```

LRU Cache

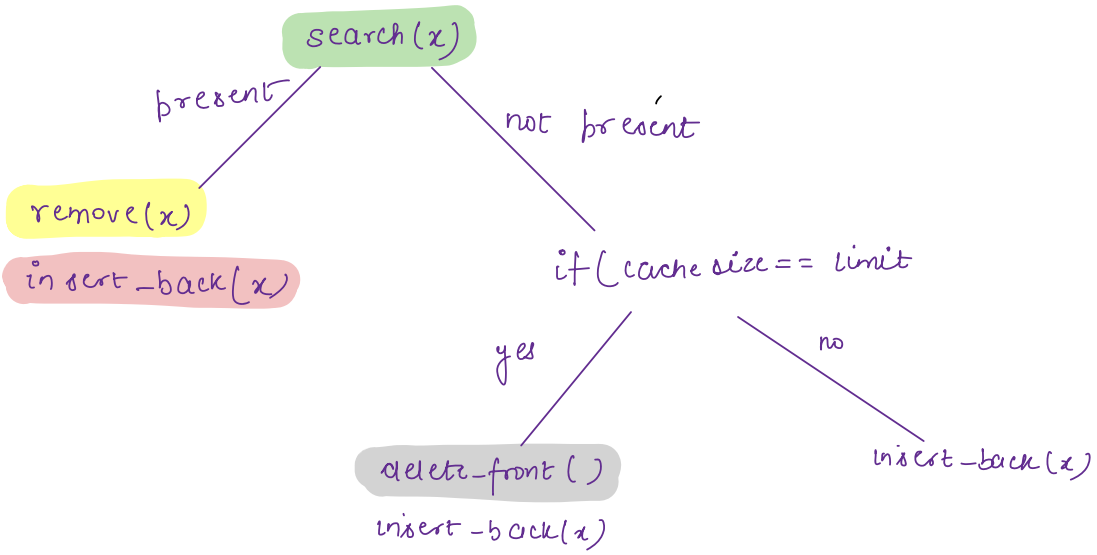
Least recently used cache.

Stream of data: 7, 3, 9, 2, 6, 10, 14, 2, 10, 15, 13, 14

LRU cache: [5]

[~~7~~, ~~3~~, ~~9~~, ~~2~~, ~~6~~, ~~10~~, ~~14~~, 2, 10, 15, 13, 14]

flow chart



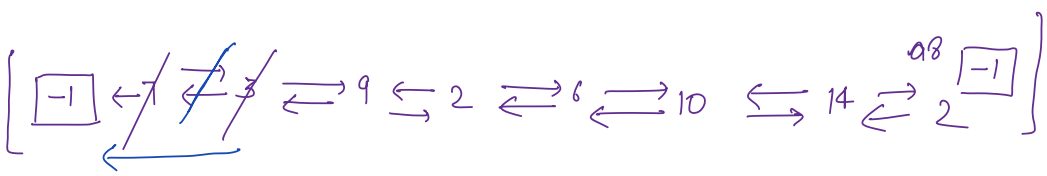
Operations	Dynamic Array	Singly LL	Map<Integer, Address> + DLL
search(x)	O(n)	O(n)	O(1)
remove(x)	O(n)	O(n)	~~O(n)~~ O(1)
insert_back(x)	O(1)	O(n) → head O(1) → tail	O(1)
delete_front()	O(n)	O(1)	O(1)

Dry run:

Data: 7 3 9 2 6 10 14 2 10 15 8 14

cache size = 5.

cache: DLL:



Map:

~~⟨7, a1⟩~~
~~⟨3, a2⟩~~
⟨9, a3⟩
⟨2, ~~a4~~ a8⟩
⟨6, a5⟩
⟨14, a6⟩
⟨10, a7⟩
. . . . . .

Initialisation

```
Node h = new Node(-1);
Node t = new Node(-1);
h.next = t
t.prev = h.
Map<Integer, Node> map;

LRU(x, limit) {
    if(map.containskey(x)) {
        Node add = map.get(x);
        deleteNode(add);
        map.remove(x);
        Node xn = new Node(x);
        insert_back(xn.tail);
        map.put(x, xn);
    } else {
        if(map.size() == limit) {
            Node temp = h.next;
            map.remove(temp.data);
            deletenode(temp, );
        }
        Node xn = new Node(x);
        insert_back(xn, t );
        map.insert(x, xn);
    }
}
```

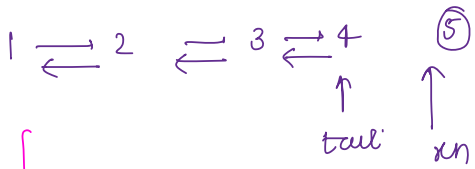
void deleteNode(Node temp) {

temp.prev (4) . next = temp.next (5)

temp.next (5) . prev = temp.prev (4)

}

void insert_back(Node xn, Node t) {



Look out yourself

tail.next = xn [4 ⟷ 5]

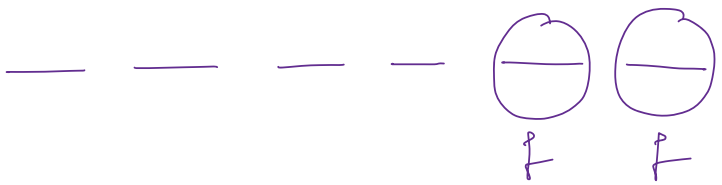
xn.prev = tail

tail = xn

}

Thankyou :)

Doubt



```
if (f.next ==null)    // odd
        return s;
} else{                    //even.
    return s.next;
}
```