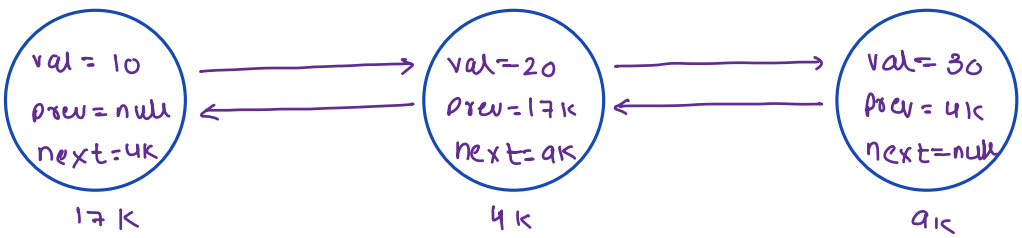
Agenda

1) Basics of DLL (Doubly Linked List)
   i) removeNode
   ii) addBeforeTail
2) Implement LRU cache (LRU: Least Recently used)
3) Copy LinkedList with Random pointers

DLL Basics

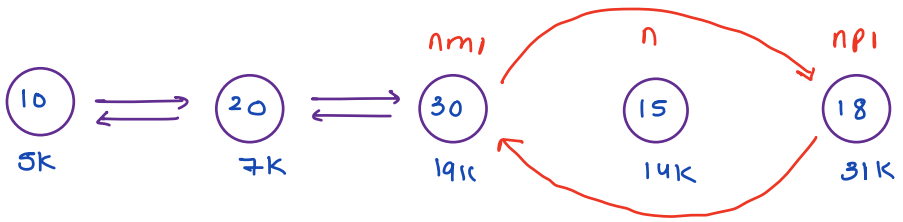
```
class Node{
    int val;
    Node next;
    Node prev;

    Node (int val) {
        this.val= val;
    }
}
```



Q- Given head of a DLL and reference of a Node, remove this node from DLL.

i) node with given reference, is present in DLL.

ii) the given node to be removed can't be 1st & last node of DLL.

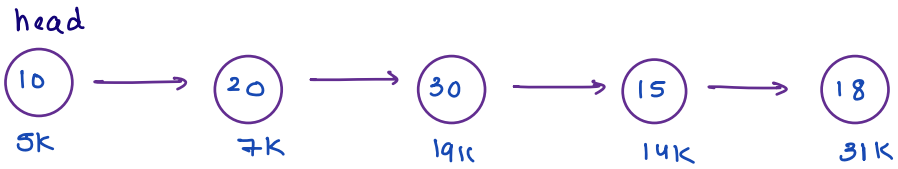


n = 14K

```
void removeNode ( Node head, Node n ) {

    Node nm1 = n.prev;
    Node np1 = n.next;

    nm1.next = np1;
    np1.prev = nm1;

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

TC: O(1)

Same question in singly LL.



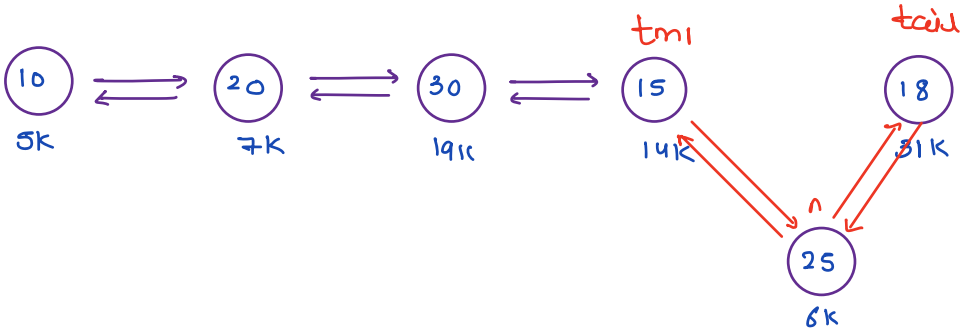
n = 14K

we need to go on node whose next is equals to n, and for that we need to travel.

TC : O(n)

Q. Given head & tail of a DLL and reference of a Node. Add this node just before tail of DLL.

i) The node whose ref. is given is not already present in DLL.

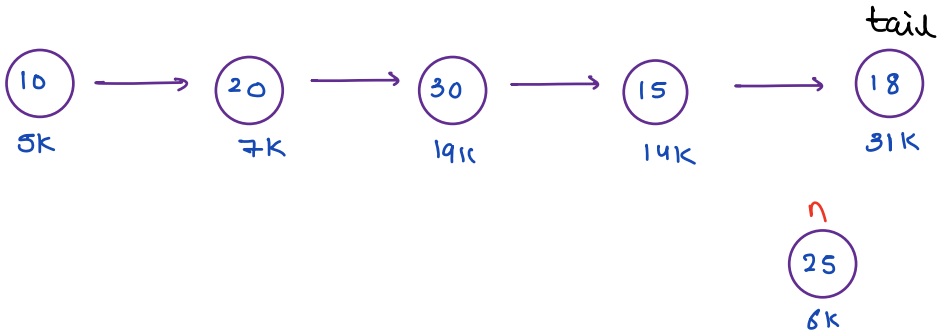



```
void addBeforeTail(Node head, Node tail, Node n) {
    Node tmp = tail.prev;
    //connect tmp & n
    tmp.next = n;
    n.prev = tmp;
    //connect n & tail
    n.next = tail;
    tail.prev = n;
}
```

TC: O(1)

{ TC is O(1) because tail is given }

Same question is single LL

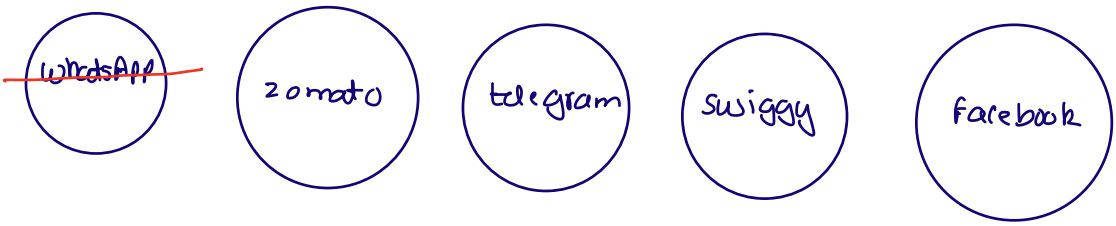


Even if tail is given, we need to travel from start to find tm1 and that's why TC: O(n)

Implement LRU cache

LRU : Least recenty used

Cap = 4



LRU Cache

Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: get and set.

- `get(key)` - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return `-1`.
- `set(key, value)` - Set or insert the value if the key is not already present. When the cache reaches its capacity, it should invalidate the least recently used item before inserting the new item.

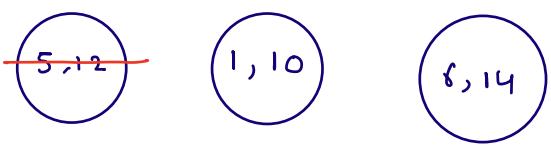
The LRUCache will be initialized with an integer corresponding to its capacity. Capacity indicates the maximum number of unique keys it can hold at a time.

Definition of "least recently used" : An access to an item is defined as a get or a set operation of the item. "Least recently used" item is the one with the oldest access time.

> ***NOTE:*** *If you are using any global variables, make sure to clear them in the constructor.*

Example :

```
Input :
        capacity = 2
        set(1, 10)
        set(5, 12)
        get(5)      returns 12
        get(1)      returns 10
        get(10)     returns -1
        set(6, 14)  this pushes out key = 5 as LRU is full.
        get(5)      returns -1
```



get → O(1)

Set → O(1)

} single time TC for get, set.

cap = 3

Set (1,1)

Set (2,2)

Set (1,5)

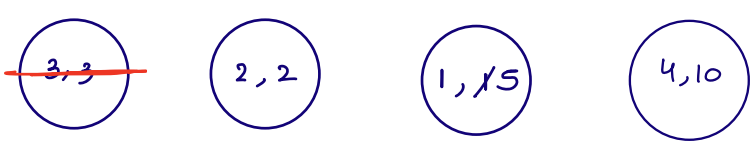
Set (3,3)

get (2) → 2

get (1) → 5

Set (4,10)

get (3) → -1



DLL

HashMap <Integer, Node>

```
class Node{
    int key;
    int val;
    Node next;
    Node prev;

    Node ( int key, int val) {
        this.key = key;
        this.val = val;
    }
}
```

DLL

HashMap <Integer, Node> map

cap = 3

Set (1,1) ✓

Set (2,2) ✓

Set (1,5) ✓

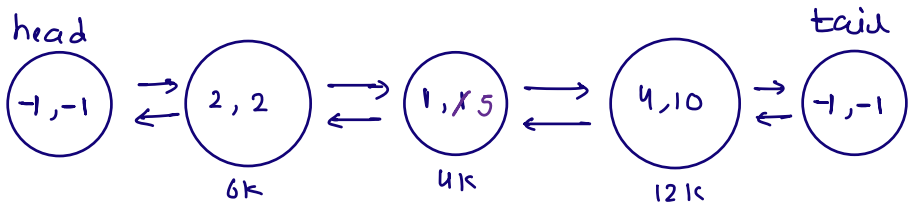
Set (3,3) ✓

get (2) ✓ → 2

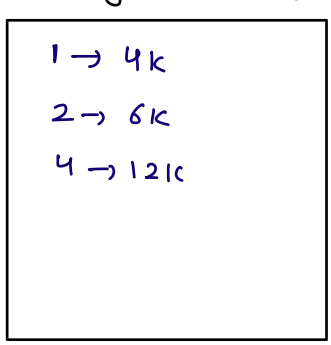
get (1) ✓ → 5

Set (4,10) ✓

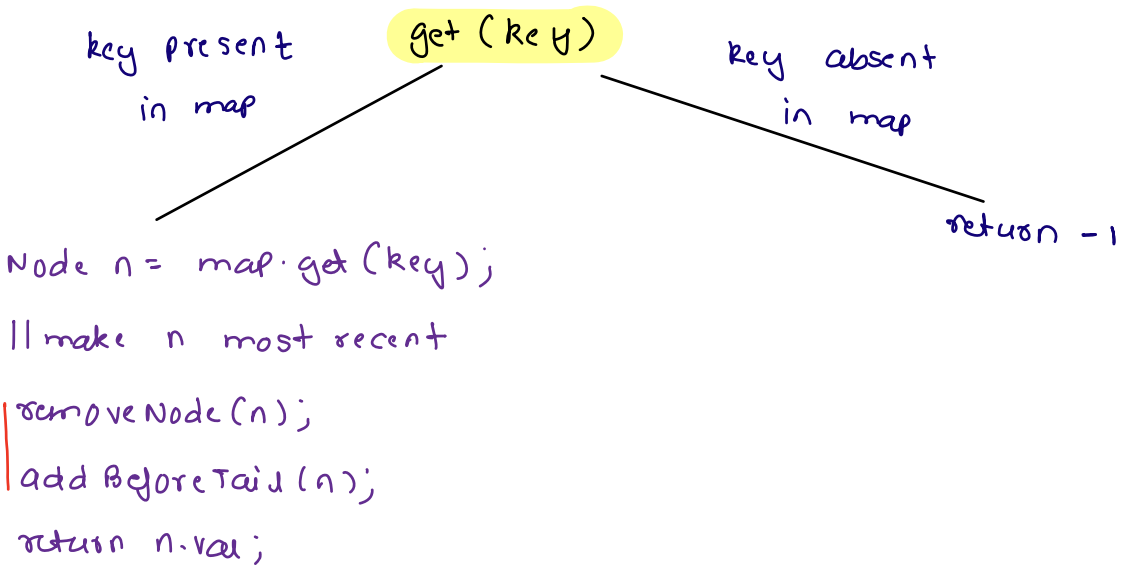
get (3) ✓ → -1

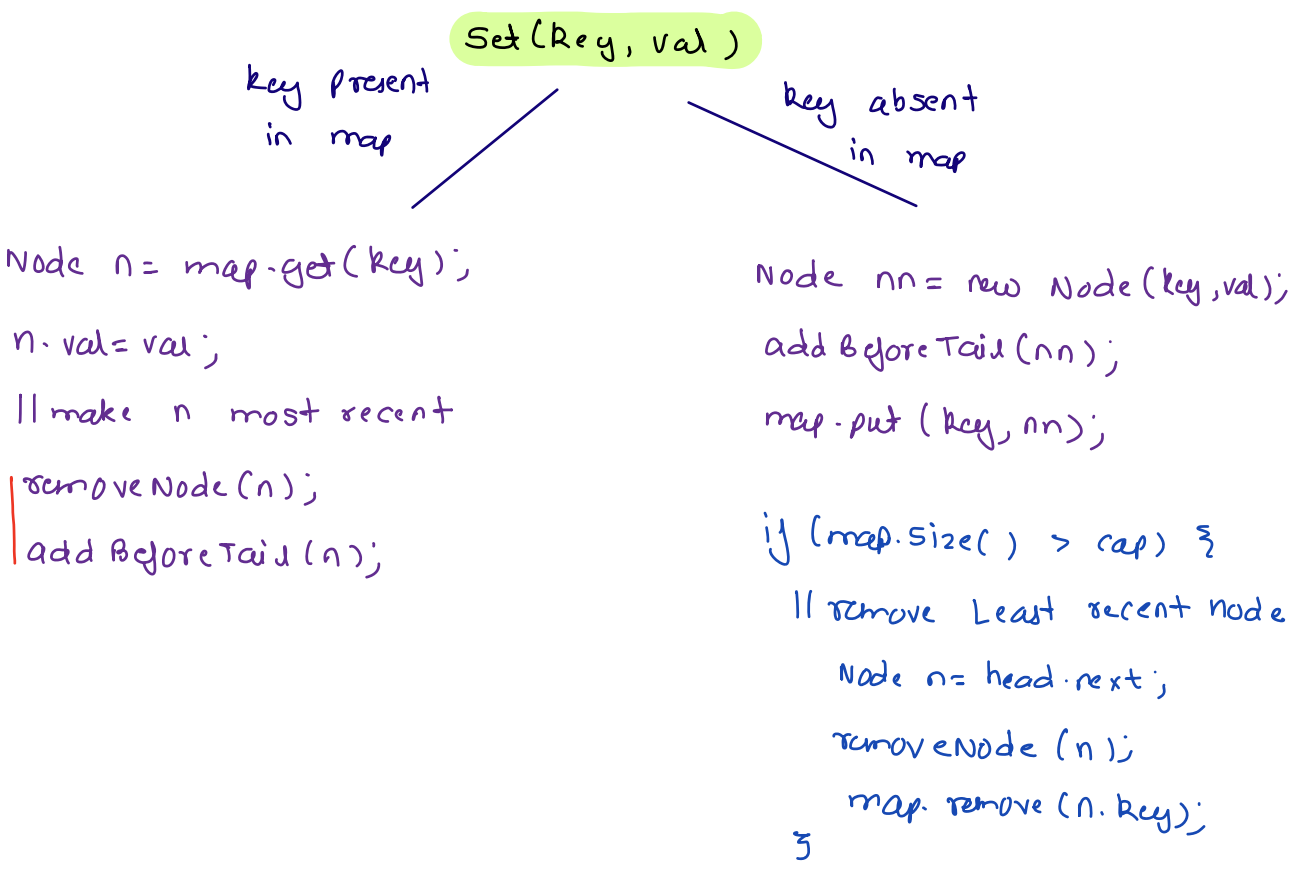


Integer vs Node



map


get (key)
key present in map
key absent in map
return -1
Node n = map.get(key);
// make n most recent
removeNode(n);
addBeforeTail(n);
return n.val;


Set(key, val)
key present in map
key absent in map
Node n = map.get(key);
n.val = val;
// make n most recent
removeNode(n);
addBeforeTail(n);
Node nn = new Node(key, val);
addBeforeTail(nn);
map.put(key, nn);
if (map.size() > cap) {
// remove Least recent node
Node n = head.next;
removeNode(n);
map.remove(n.key);
}

Syllabus: Strings and LinkedList

contest time: 9pm to 10:30pm

discussion: 10:30 pm onwards

Q. Copy LinkedList with random pointers.

```
class Node{
    int val;
    Node next;
    Node random;

    Node (int val) {
        this.val= val;
    }
}
```