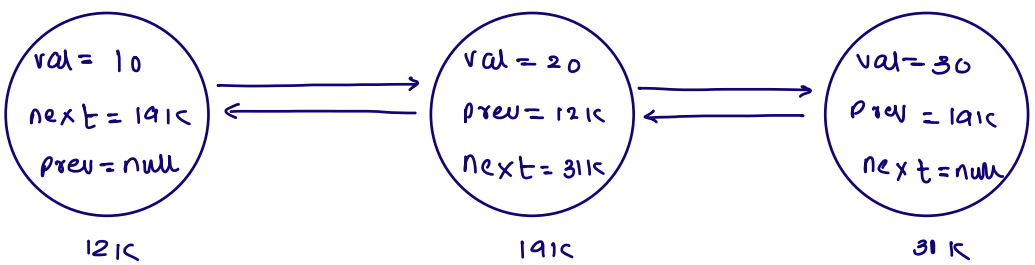
Agenda

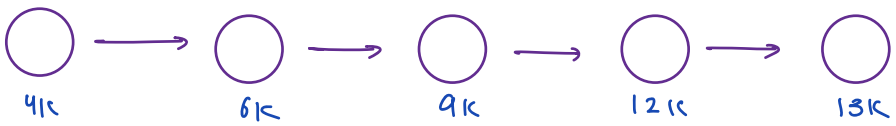
1) Basics of DLL (Doubly Linked List)
   i) removeNode
   ii) addBeforeTail
2) Implement LRU cache (LRU: Least Recently used)
3) Copy LinkedList with Random pointers

DLL Basics

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

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

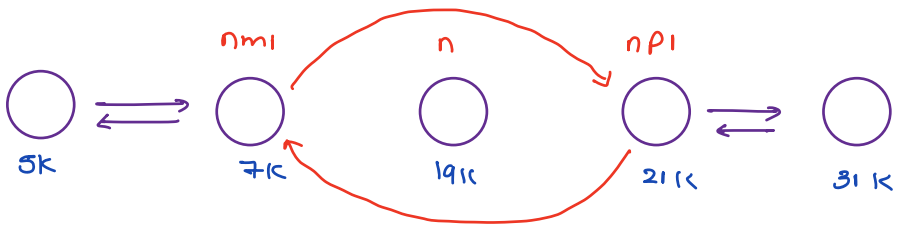


remove given node



n = 9K

TC: O(n) search for node whose next is equals n.



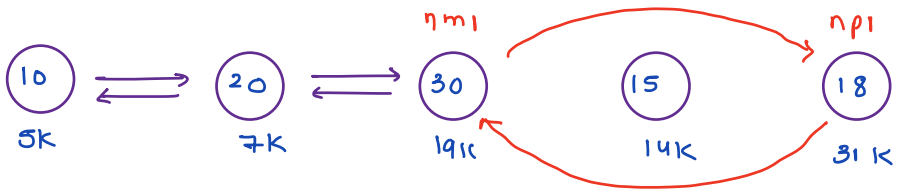
n = 19K

TC: O(1)

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

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

The node given is not equals to the 1st and last node of DLL.

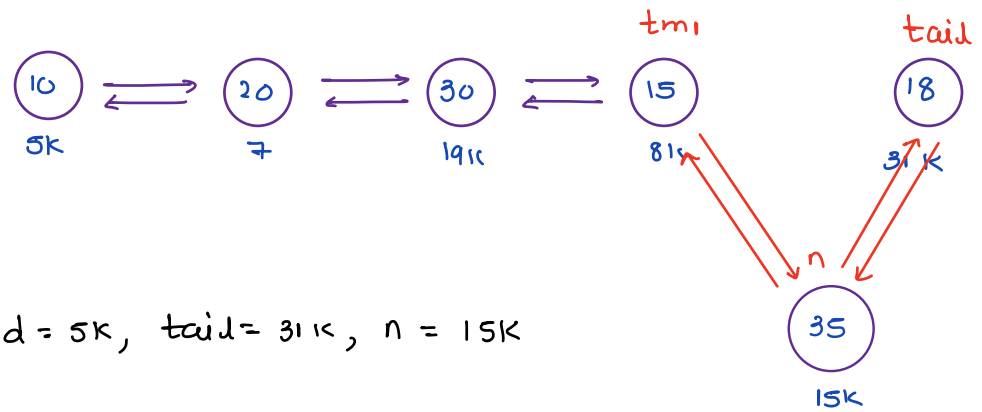


head = 5K , n = 14K

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

    Node nm1 = n.prev;

    Node npl = n.next;

    nm1.next = npl;

    npl.prev = nm1;

    n.next = n.prev = null;

}
```

TC: O(1)

Q. Given head & tail of DLL as well as reference of a node (not present in DLL). Add this node before tail.



head = 5K, tail = 31K, n = 15K

```
void addBeforeTail(Node head, Node tail, Node n) {
    Node tmi = tail.prev;
    tmi.next = n;
    n.prev = tmi;
    n.next = tail;
    tail.prev = n;
}
```

TC: O(1)

{because tail was given}

Implement LRU cache

Least recently used (LRU)

capacity = 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
```

Cap = 2

get (key) → O(1)

set (key, val) → O(1)

DLL

HashMap

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

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

DLL

HM <Integer, Node> map

cap = 3

set (1,1) ✓

set (2,2) ✓

set (1,5) ✓

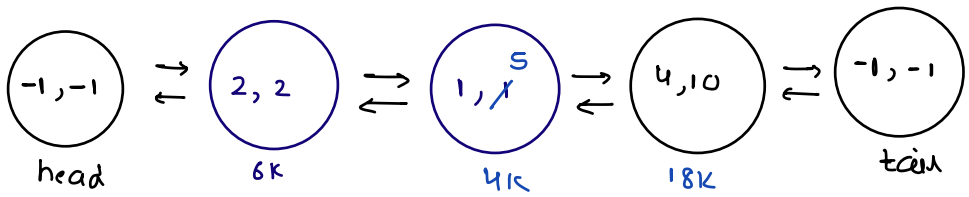
set (3,3) ✓

get (2) ✓ → 2

get (1) ✓ → 5

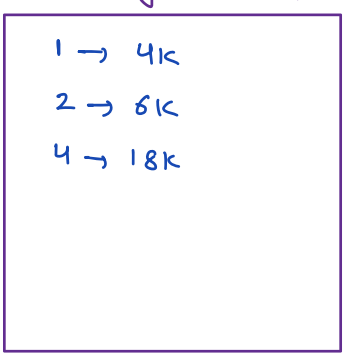
set (4,10) ✓

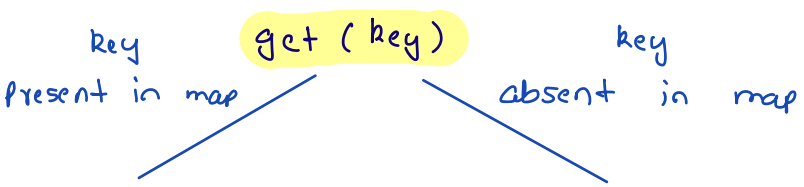
get (3) → -1



↱ (key)

Integer vs Node





make the node infront of key as most recent.

```
Node n = map.get(key);
removeNode(n);
addBeforeTail(n);
```

set (key, val)

key present in map

```
Node n = map.get(key);
n.val = val;
removeNode(n);
addBeforeTail(n);
```

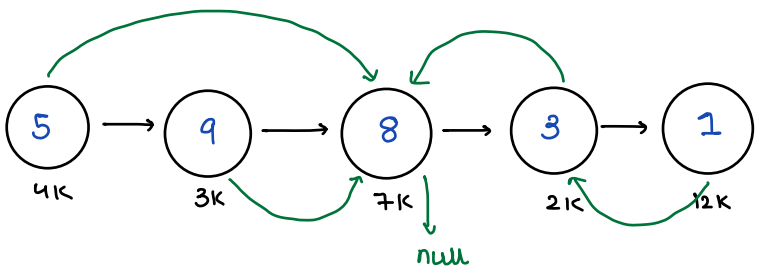
making it most recent

key absent in map

```
Node nn = new Node(key, val);
addBeforeTail(nn);
map.put(key, nn);

if(map.size() > cap) {
  // remove least recently used
  Node n = head.next;
  removeNode(n);
  map.remove(n.key);
}
```

Q. Copy LinkedList with random pointers.

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

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



— : next

— : random

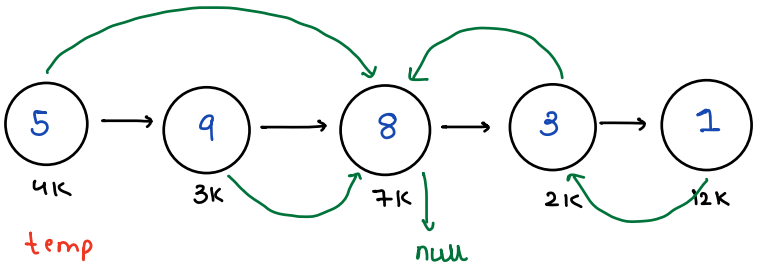
n = 3K

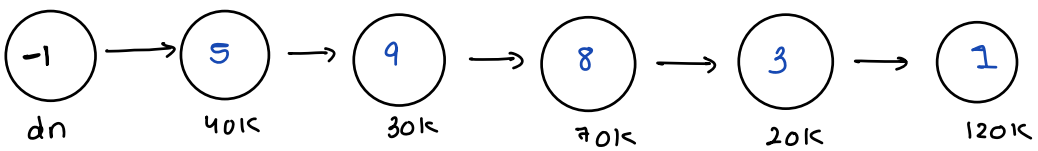
n.random.val = 8

n = 2K

n.next.random.val = 3

Create a deep copy (LL with new nodes) of existing linked List and return head of that deep copy.





copying val and next is easy.

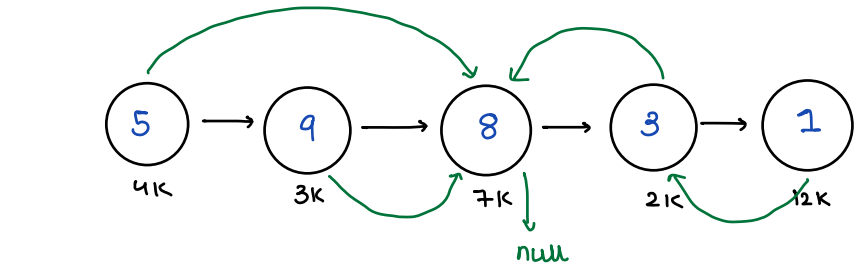
[TC: O(n)]

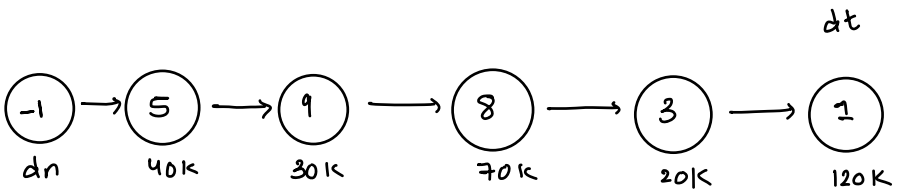
HashMap < Node, Node > map;

to copy random pointers

old_node vs new_node

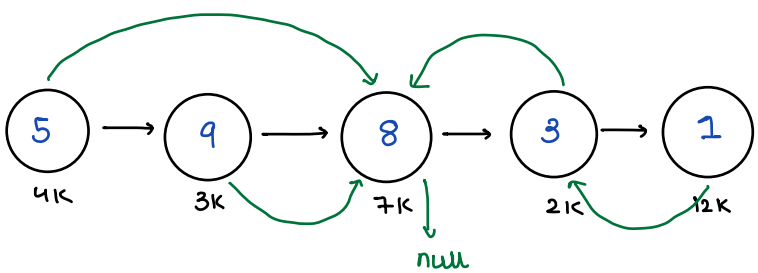
step1 : copy val and next , during this process create map

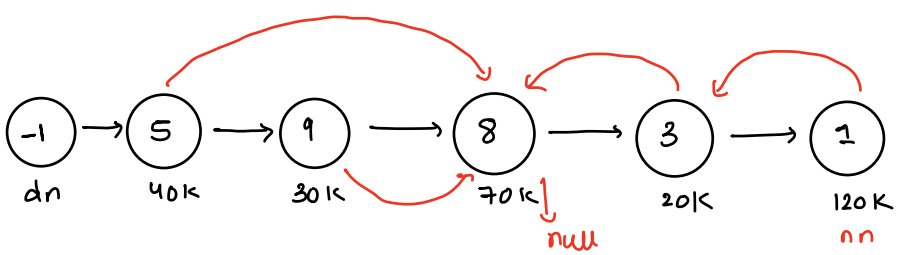




old node	new node
4K	40K
3K	30K
7K	70K
2K	20K
12K	120K

step2 : copy the random pointers





old node	new node
4K	40K
3K	30K
7K	70K
2K	20K
12K	120K

```
Node old = temp;
Node nn = map.get(old);
nn.random = map.get(old.random);
```

TC: O(n)

SC: O(n)

syllabus for contest : 5 → two pointers, Strings, LinkedList