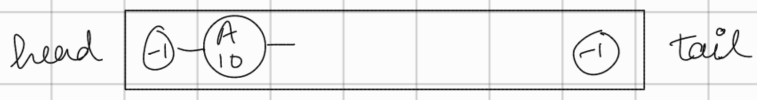
- Design LRU cache 146
- Design LFU cache 460
- Copy Linked List with random pointer 138
  - ↳ with extra space
  - ↳ without extra space

① LRU cache {Least Recently used}

Eg: whatsapp chat ranking → latest chat first.

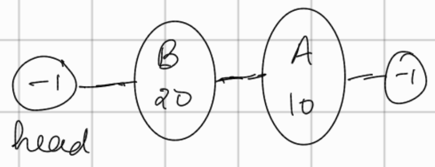
Removal: Can be from any point → ~~Array~~ O(n) → LL ✓

Insertion: Only from head → LL → O(1)



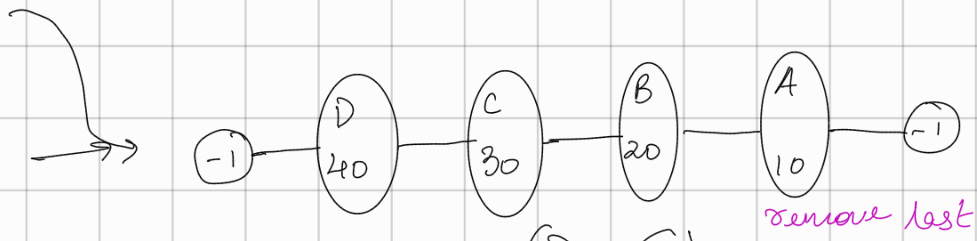
① put (A, 10)

② put (B, 20) →

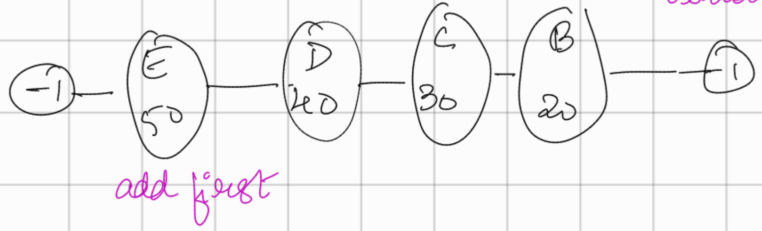


③ put (C, 30)

④ put (D, 40) →



⑤ put (E, 50) →



(put = addFirst)

⑥ get(A) => -1 (Not in cache) but will LL, gives O(n)

So to improve complexity, use HashMap.

→ O(1)

⑦ get (C) : Process: Check in Hash Map & if present return value.

If we use HashSet, this operation will take O(capacity) because we need to traverse list which is not acceptable for current problem. So use HashMap.

Once node is found, remove Node & add First the remove Node. → this will also take O(n) in SLL

So use DLL (Doubly Linked List)

⑧ put (D, 50) → check if D is present in HashMap O(1) &

if present, remove from it's current position and add Front (D, 50). → $\underline{\underline{O(1)}}$

⑨ put (E, 60) → check if E is present in Hash Map O(1).

if present, remove from it's current position and add front (E, 60) . → $\underline{\underline{O(1)}}$

⑩ put (F, 60) → check if F is present in Hash Map O(1)

if not present, then removeLast () → O(1)

and add First (F, 60). → O(1)

To remove last node, No need to update tail node,

just do curr. prev. next = curr. next
curr. next. prev = curr. prev

| | | | |
|---|---|---|---|
| Hashmap → | Insert | → avg case → | O(1) |
| | | → worst case → | O(n) |

UNORDERED MAP

Page / Frame Not found in cache :→ Page Fault or Cache miss

" FOUND " :→ Cache hit

☆ DLL Operations

Remove First

```
curr = head.next
curr.prev.next = curr.next
curr.next.prev = curr.prev
```

Remove Last

```
curr = tail.prev
curr.prev.next = curr.next
curr.next.prev = curr.prev
```

Remove At

```
curr.prev.next = curr.next
curr.next.prev = curr.prev
```

Add First

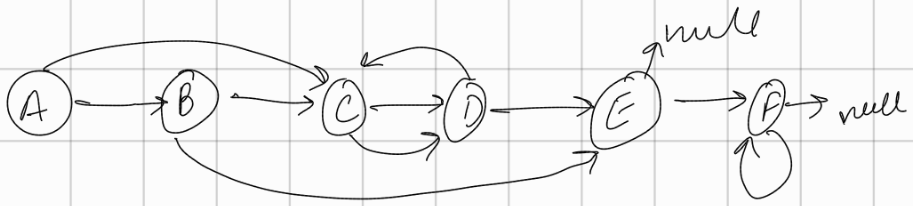
```
Node curr = new Node(val);
curr.prev = head;
Curr.next = head.next;
head.next = curr ;
Curr.next.prev = curr;
```

Add Last

```
Node curr = newNode(val)
curr.next = tail
curr.prev = tail.prev
curr.next.prev = curr
curr.prev.next = curr
```

② Copy Linked List with random pointer   Deep Copy

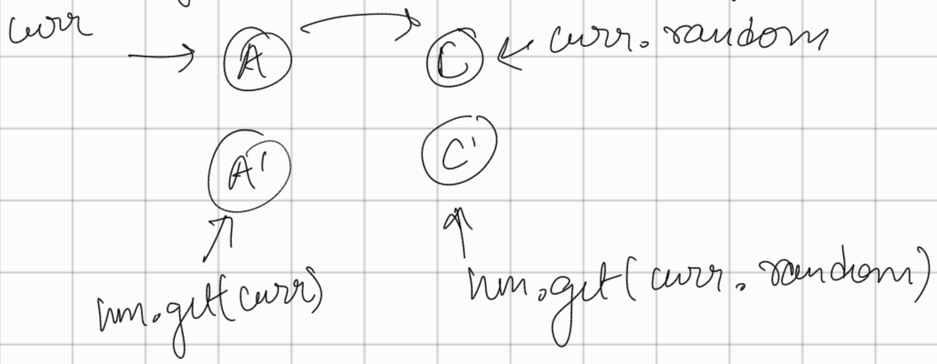
If no random pointer: Simple traversal & addLast → O(N)



$$\begin{tikzcd} A \arrow[r] & B \arrow[r] & C \arrow[r] & D \arrow[r] & E \arrow[r] & F \arrow[r] & null \end{tikzcd}$$

Use hashmap <Node , Node>
old node ↗   ↑ cloned node

for each node after cloning without random pointers
hm.get(curr). random = hm. get(curr.random);



Constant Extra Space:
Insert duplicate node just after it's original one