DAY-36

① Design a data structure for "LRU Cache"

[Least Recently Used (LRU) cache organized item in order of use, allowing you to quickly identify which item hasn't been used for the longest amount of time.]

it should (in this part) support the following operation :-

1. get(key) → Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.
2. set(key, value) → Set or insert the value if the key is not already present.

→ When the cache reached its capacity, it should invalidate the least recently used item before inserting them.

Approach :-

→ The key to solve this problem is using a double linked list which enables us to quickly move nodes.

→ The LRU cache is a hash with keys and double linked list nodes.

REDMI NOTE 9
AI QUAD CAMERA

⇒ The hash map makes the time of get() to be O(1). The list of double linked list nodes make the nodes adding / removal operations O(1).

Implementation:-

```
class DLLNode:
    def __init__(self, key, val):
        self.val = val
        self.key = key
        self.prev = None
        self.next = None
class LRUCache:
    def __init(self, capacity):
        self.capacity = capacity
        self.map = {}
        self.head = DLLNode(0,0)
        self.tail = DLLNode(0,0)
        self.head.next = self.tail
        self.tail.prev = self.head
        self.count = 0

    def deleteNode(self, node):
        node.prev.next = node.next
        node.next.prev = node.prev
    def addToHead(self, node):
        node.next = self.head.next
        node.next.prev = node
        node.prev = self.head
        self.head.next = node
```

```
def get(self, key):
    if key in self.map:
        node = self.map[key]
        result = node.val
        self.deleteNode(node)
        self.addToHead(node)
        print("{} result key found")
        return result
    print('{}', format(key))
    return -1
def set(self, key, value):
    if key in self.map:
        node = self.map[key]
        node.value = value
        self.deleteNode(node)
        self.addToHead(node)
    else:
        node = DLLNode(key, value)
        self.map[key] = node
        if self.count < self.capacity:
            self.count += 1
            self.addToHead(node)
        else:
            del self.map[self.tail.prev.key]
            self.deleteNode(self.tail.prev)
            self.addToHead(node)
```

REDMI NOTE 9
AI QUAD CAMERA