```
class LRUCache {
    int capacity;
    Map<Integer, Node> mp;
    Node head = null;


    public LRUCache(int capacity) {
        this.capacity = capacity;
        this.mp = new HashMap<>();
    }


    public void remove(Node node) {
        if (node.next == node) {
            head = null;
        } else {


            if (node == head) {
                head = head.next;
            }
        node.prev.next = node.next;
        node.next.prev = node.prev;
    }
}


public void insertAtFront(Node node) {
    if (head == null) {
        node.next = node;
        node.prev = node;
        head = node;
    } else {
        Node tail = head.prev;
        tail.next = node;
        node.prev = tail;
        head.prev = node;
        node.next = head;
        head = node;
    }
}


public int get(int key) {
    if (!mp.containsKey(key))
        return -1;
        Node nn = mp.get(key);
        remove(nn);
        insertAtFront(nn);
        return nn.value;
    }


    public void put(int key, int value) {
            if(mp.containsKey(key)){
                remove(mp.get(key));
                mp.remove(key);
            }
            if(mp.size() == capacity){
                Node tail = head.prev;
                remove(tail);
                mp.remove(tail.key);
            }
            Node nn = new Node(key , value);
            insertAtFront(nn);
            mp.put(key,nn);
    }
}
class Node{
    int key , value;
    Node next , prev;
    Node(int key , int value){
        this.key = key;
        this.value = value;
    }
}
```