class LRUCache {

  private LinkedHashMap<Integer,Integer>map;

     private final int CAPACITY;

public LRUCache(int capacity) {

CAPACITY=capacity;

map=new LinkedHashMap(capacity,0.75f,true){

protected boolean removeEldestEntry(Map.Entry eldest){

     return size()>CAPACITY;

}

};

}

public int get(int key) {

  return map.getOrDefault(key,-1);

}

public void put(int key, int value) {

     map.put(key,value);

}

}

class Node{

Node prev,next;

int key,val;

Node(int key, int val){

     this.key=key;

     this.val=val;

}

}

class LRUCache {

Node head=new Node(0,0);

Node tail=new Node(0,0);

Map<Integer,Node> map=new HashMap();

int capacity;

public LRUCache(int capacity) {

  this.capacity =capacity;

  head.next=tail;

  tail.prev=head;

}

public int get(int key) {

     if(map.containsKey(key)){

         Node node=map.get(key);

         remove(node);

         insert(node);

         return node.val;

     } else{

         return -1;

     }

}

void remove(Node node){

map.remove(node.key);

node.prev.next=node.next;

node.next.prev=node.prev;

}

void insert(Node node){

map.put(node.key, node);

Node headNext=head.next;

head.next=node;

node.prev=head;

headNext.prev=node;

node.next=headNext;

}

public void put(int key, int value) {

     if(map.containsKey(key)){

         remove(map.get(key));

     }

     if(map.size()==capacity){

         remove(tail.prev);

     } insert(new Node(key,value));

}

}