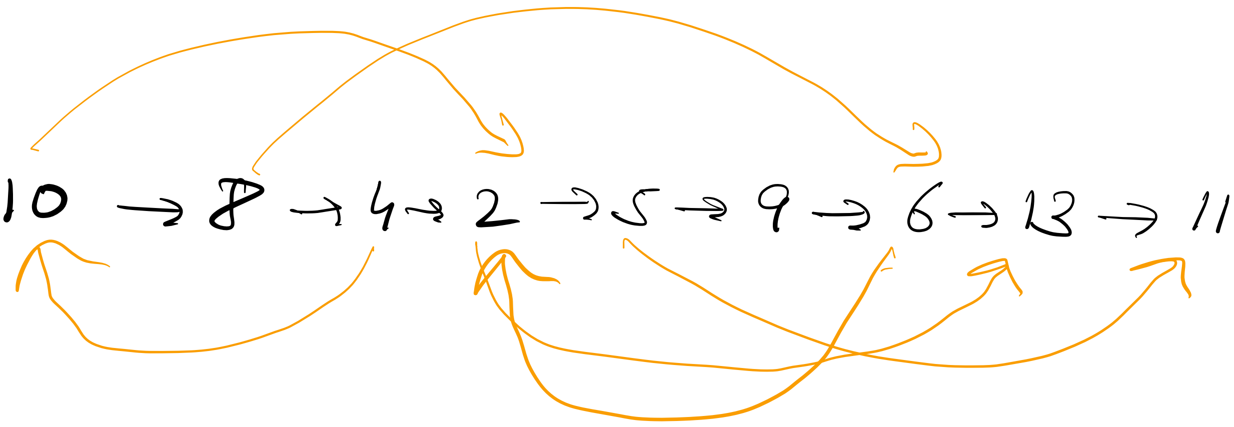
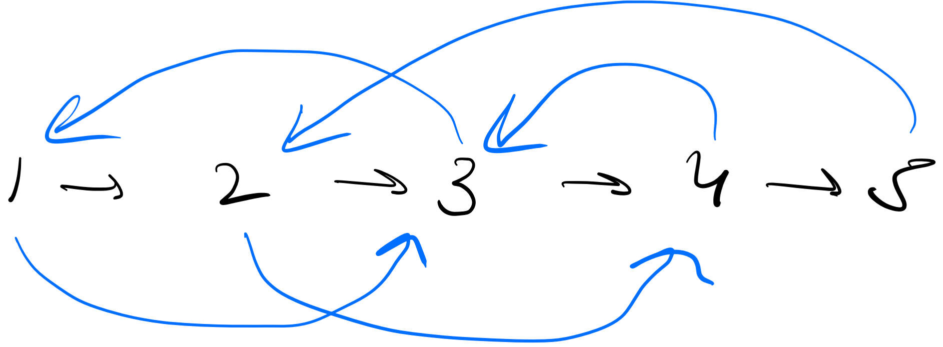
Q1 Clone Linked List

```
class Node {
  int data
  Node next
  Node random
}
```



Make a exact copy of this.

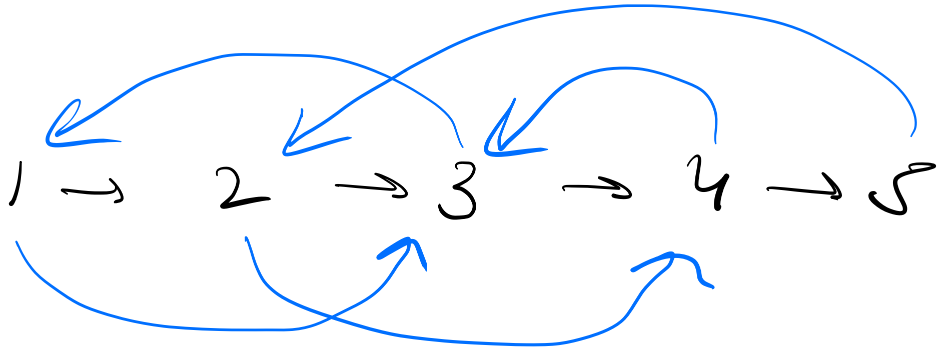
Brute : Create copies of each of the old nodes. And create a Hashmap of old-node , new_node

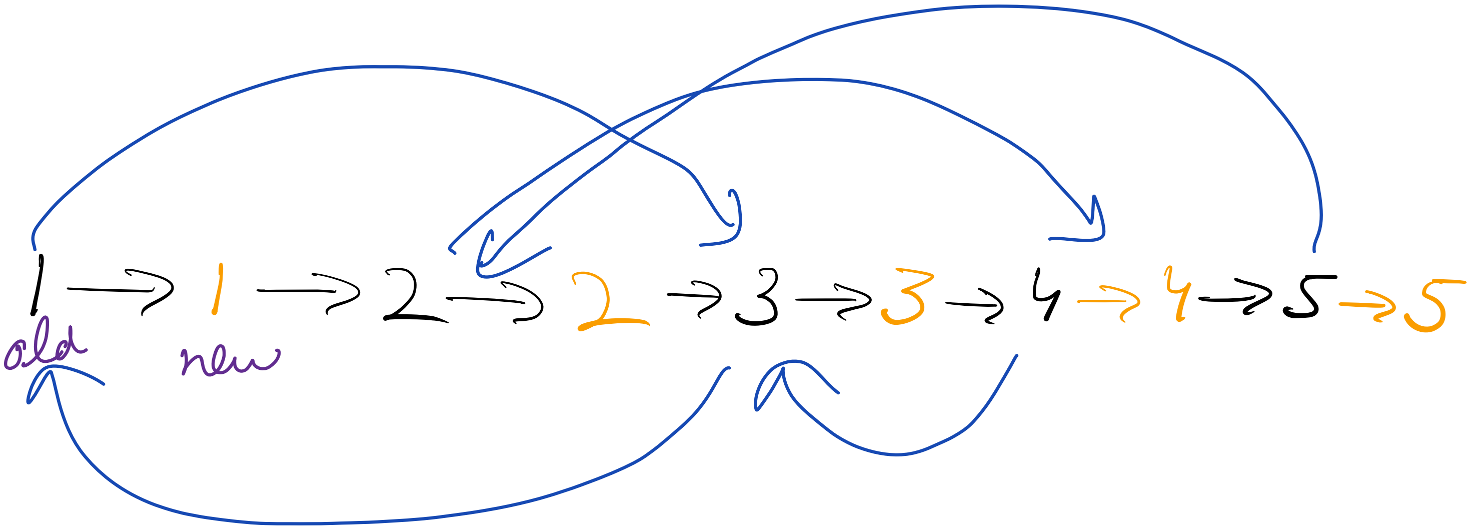


1 2 3 4 5

```
new_node .next = hm[old_node.next]
new_node .random= hm[old_node.random]
```

Challenge : Create the new list without the hm.





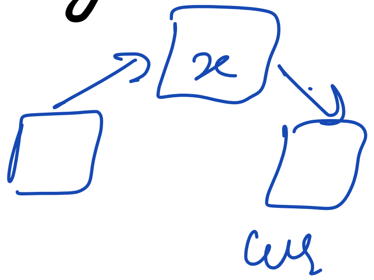
new.next =

new.random = old.random.next

How is new & old related

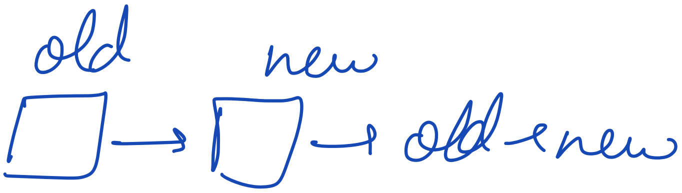
new = old.next

Step-1 Create new nodes & arrange them



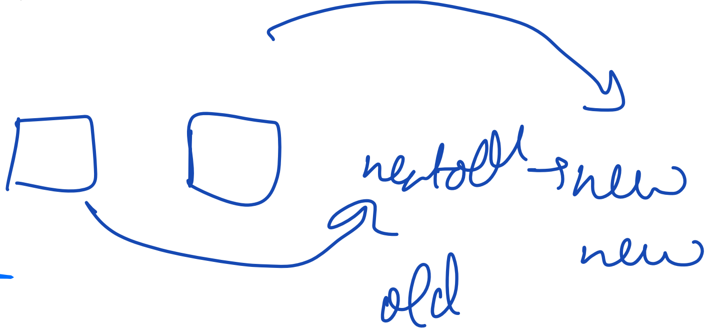
```
cur = head
while (curr != null) {
    x = new node (curr.data)
    x.next = cur.next
    curr.next = x
    curr = x.next
}
```

Step-2 Update random pointers



```
old = head
new = old.next
while ( old != null) {
    new.random = old.random.next
    old = new.next
    if ( new.next != null)
        new = new.next.next
}
```

Step-3 Get the new list seperated from old list



```
old = head
Node ans = old.next
new = curr.next
while (old != null) {
    old.next = new.next
    old = old.next
    if (new.next != null) {
        new.next = new.next.next
        new = new.next
    }
}
return ans
```

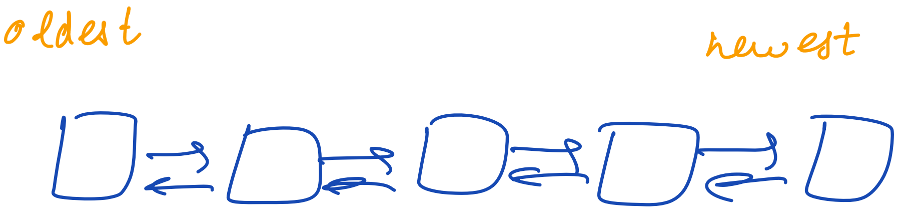
{done}

Q2 LRU Cache → high speed storage } used for temp storage

Least Recently used

Capacity = 5

7 3 9 2 6 10 14 2 10 15 8 14



HIT → present

MISS → not present

↓ put(x)

search(x)

HIT

remove

insert

MISS

size == capacity_full

yes

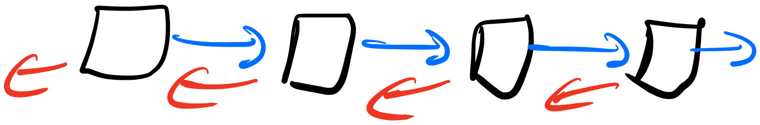
remove head → insert at tail

no

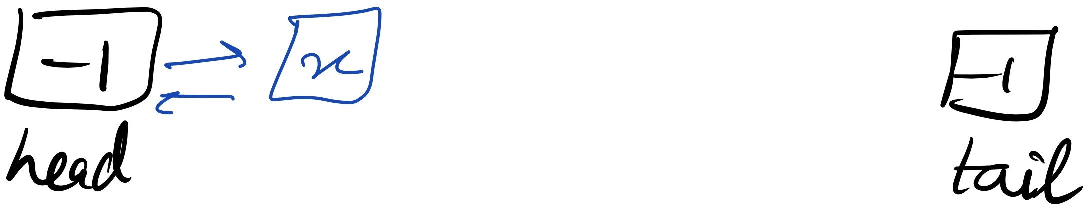
insert at tail

Doubly Linked List

```
class Node {
    int data
    Node next
    Node prev
}
```



10 15 19 20 15 18 23 20



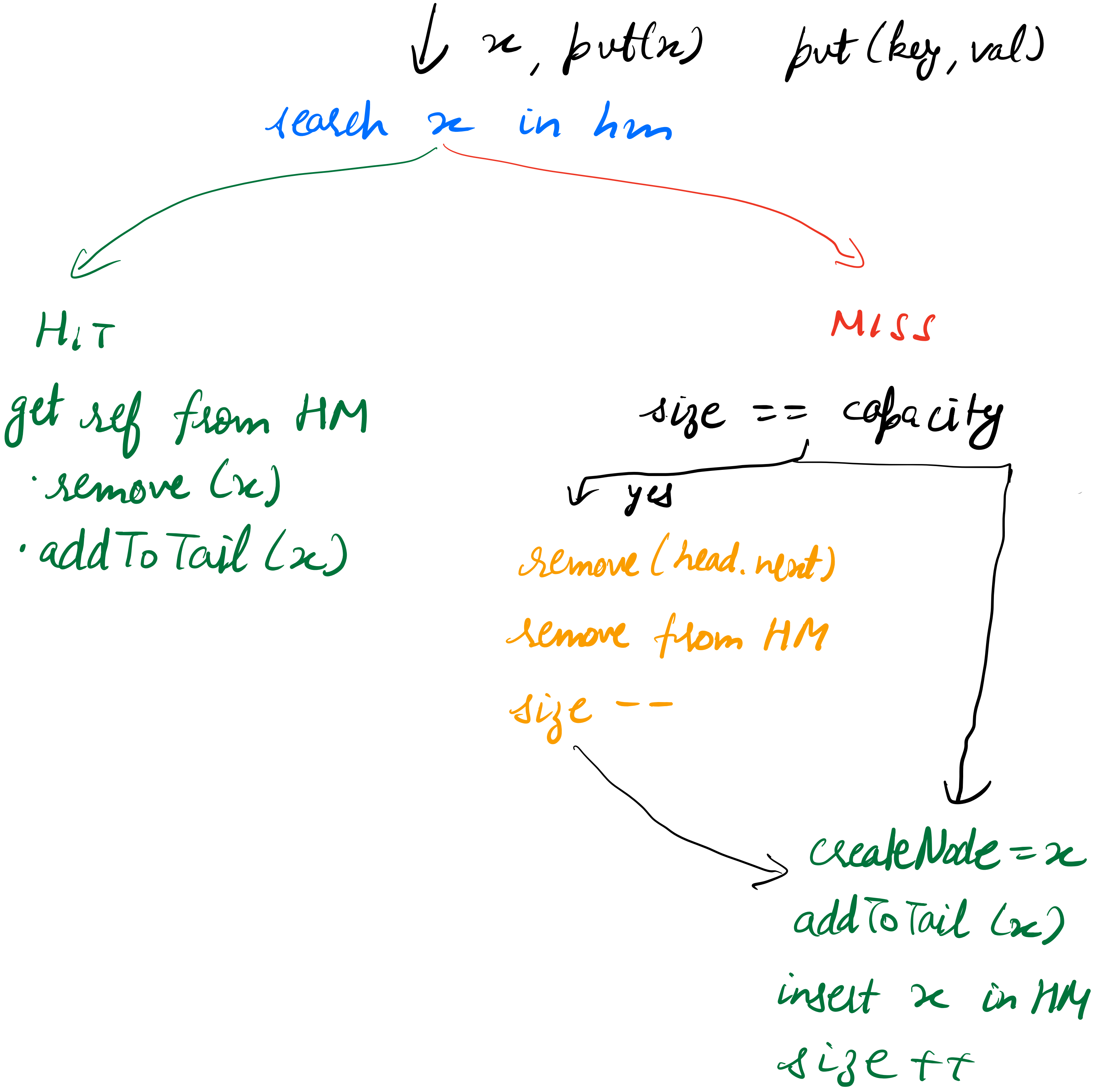
2 dummy nodes

```
addToTail(Node x){
    prev_node = tail.prev
    x.next = tail
    x.prev = prev_node
    tail.prev = x
    prev_node.next = x
}
```

HM

<10, add>
<15, add>
<19, add>
<20, add>

```
remove head ( Node head ) {
    x = head. next
    remove (x)
}
```



↓ x, put(x)

put (key, val)

search x in hm

HIT

get ref from HM

- remove (x)
- addToTail (x)

MISS

size == capacity

yes

remove (head. next)

remove from HM

size --

createNode = x

addToTail (x)

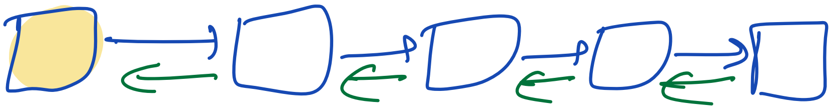
insert x in HM

size ++

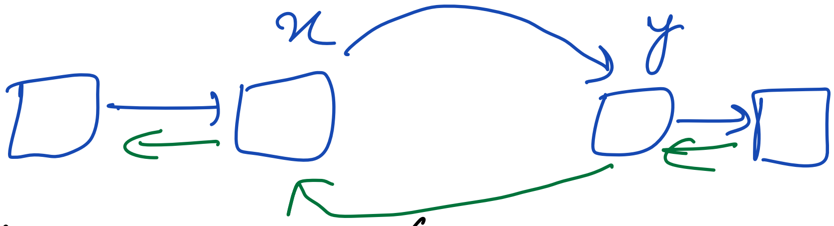
```
head= new Node(-1)        tail = new Node(-1)
head.next = tail          tail.prev = head
hashmap<int,Node> hm
num-of-nodes = 0
for(i=0;i<n;i++){
    if(hm.contains(arr[i])==true){  //hit
        cur_node = hm.get(arr[i])
        remove(cur_node)
        new_node = new Node(arr[i])
        addToTail(new_node)
        hm.put(arr[i], new_node)
    }
    else{        // miss
        if(num_of_nodes < capacity){
            new_node = new Node(arr[i])
            addToTail(new_node)
            hm.put(arr[i], new_node)

            num-of-nodes++
        }
    else {
        removehead ( head)
        new_node = new Node (arr[i])
        addToTail (new_node )
        hm.but( arr [i], new_node)
    }
}
```

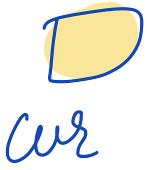
Doubly linked list

```
Node {
  int data
  Node next
  Node prev
}
```



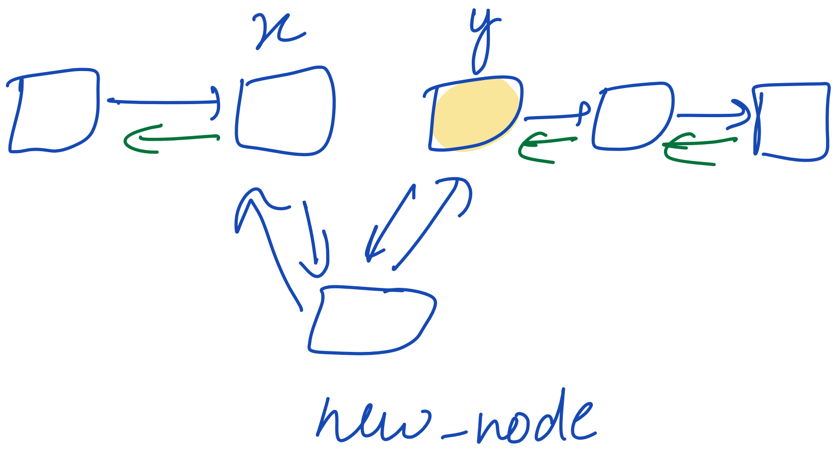
a) Remove a node





```
void remove (Node cur) {
    x = cur. prev
    y = cur. next
    x. next = y          (if x not NULL)
    y. prev = x          (if y not NULL)
    cur. next = cur. prev = null
}
```

b) Add given node before a certain node



new_node.next = y

new_node.prev = x

x.next = new_node

y.prev = new_node