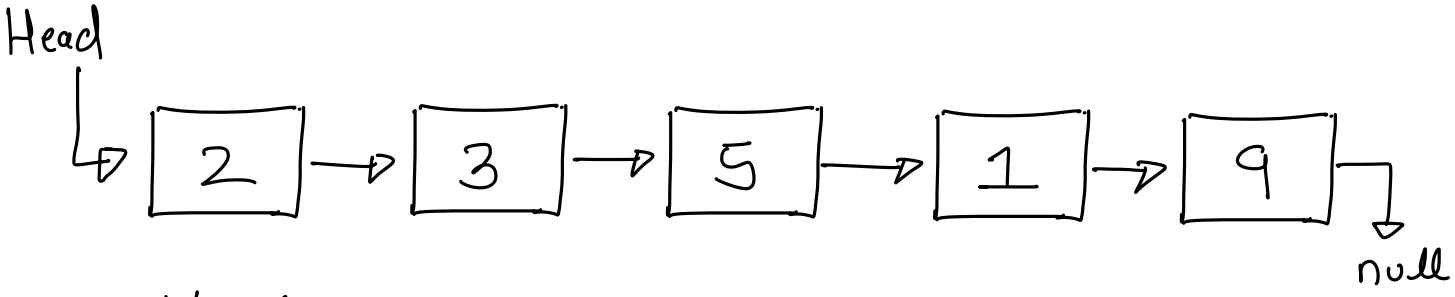
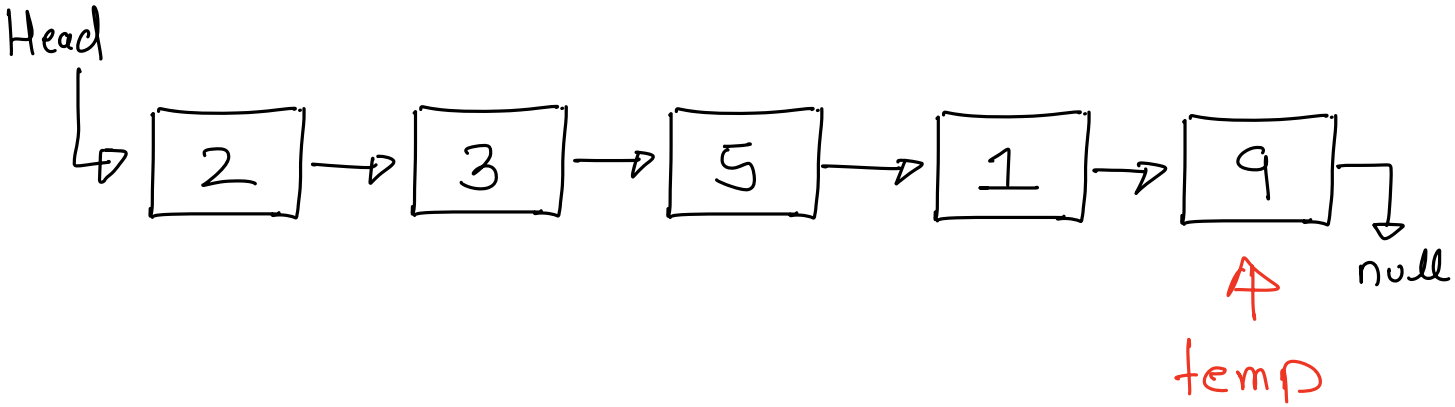
- Linked list questions are usually not tough algorithmically.
- Coding LL questions can be tricky
- Always think about Edge Cases
  1) When linked list is null
  2) When linked list has just one node
  3) Problem specific edge cases.

Q1 Search for an element K in given linked list.
Head node will be given. Distinct elements present.



K = 1

→ true



K = 9

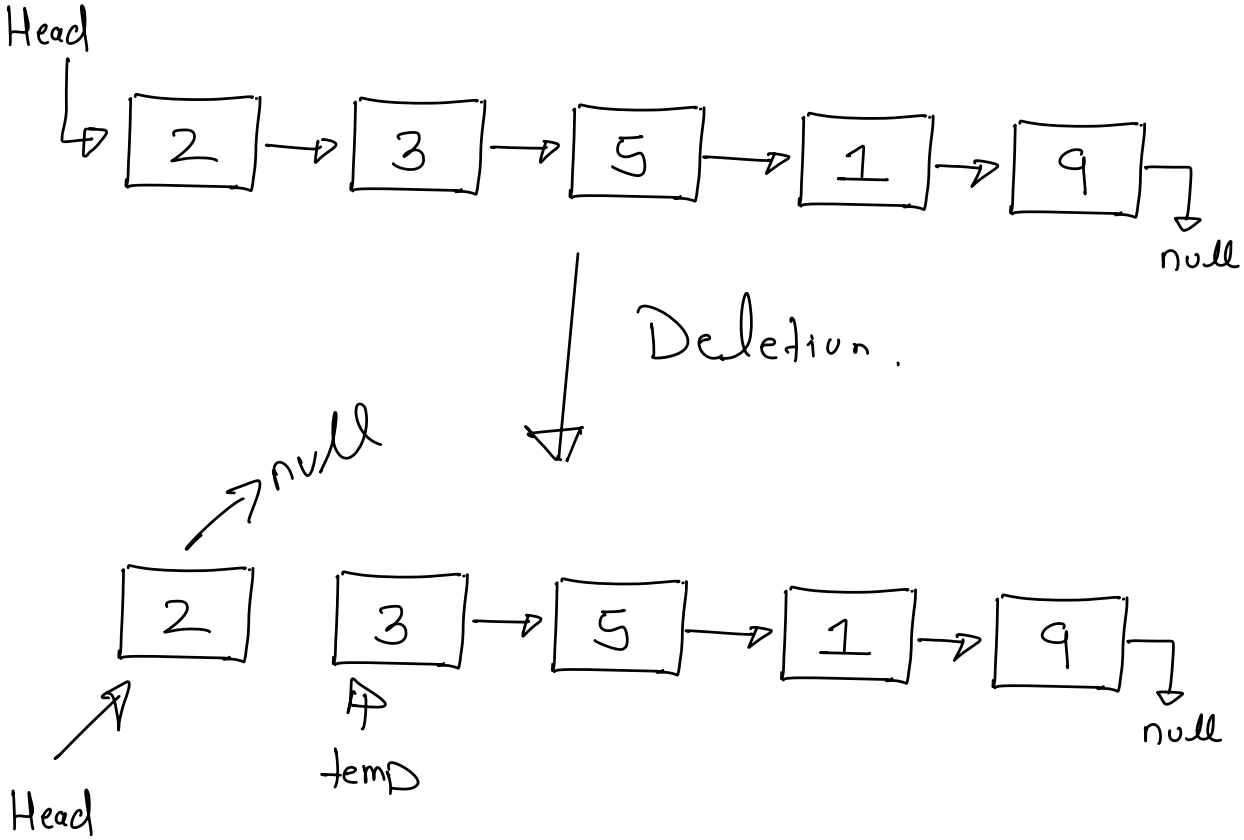
```
Node temp = head;
while (temp != null) {
    if (temp.val == k)
        return true;
    temp = temp.next;
}
return false;
```

Tc: O(n)

Sc: O(1)

U2 Delete Node

→ at Head



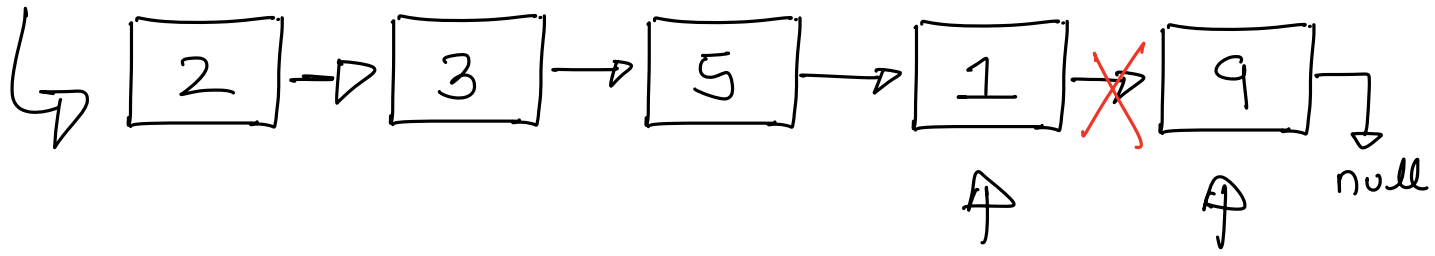

```
if (head.==null)
    return null;

Node temp = head
temp = temp.next
head.next = null
head = temp
return head;
```

TC : O(1)

SC : O(1)

at tail.

Head




```
if (head == null)
    return null;

if (head.next == null)
    return null;

Node temp = head
Node prev = null

while (temp.next != null) {
    prev = temp;
    temp = temp.next;
}

prev.next = null

del (temp) ——> optional. Depends on how memory
                management works in your language.
                Ask Chat gpt about it.

return head;
```

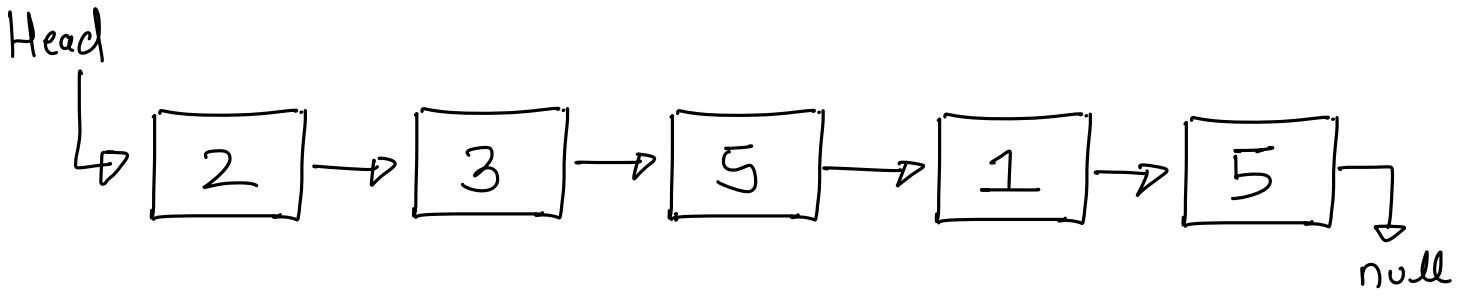
TC: O(n)

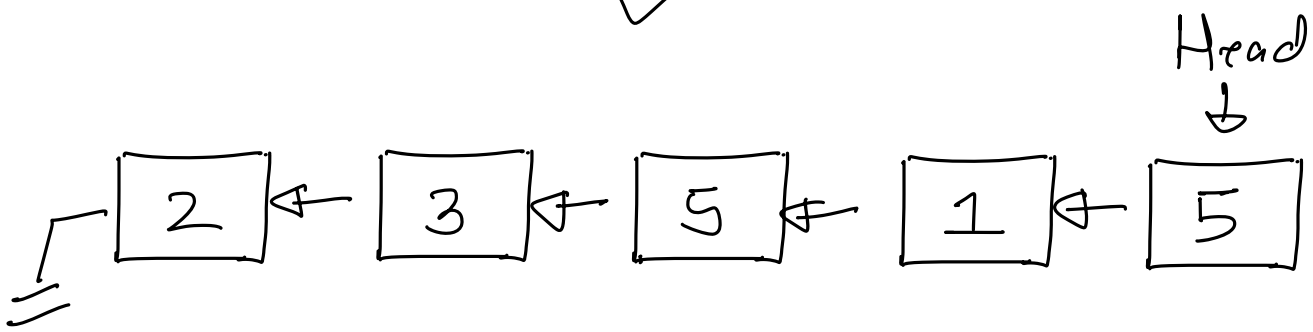
SC: O(1)

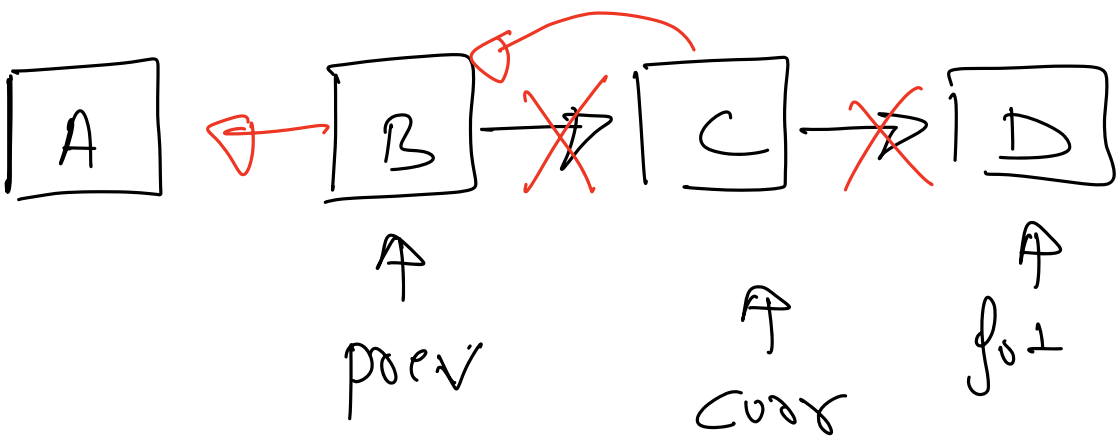
```
Node temp = head;

while (temp.next.next != null) {
    temp = temp.next;
}

temp.next = null
```

10:18pm

Q4 Reverse the linked list. SC: O(1)





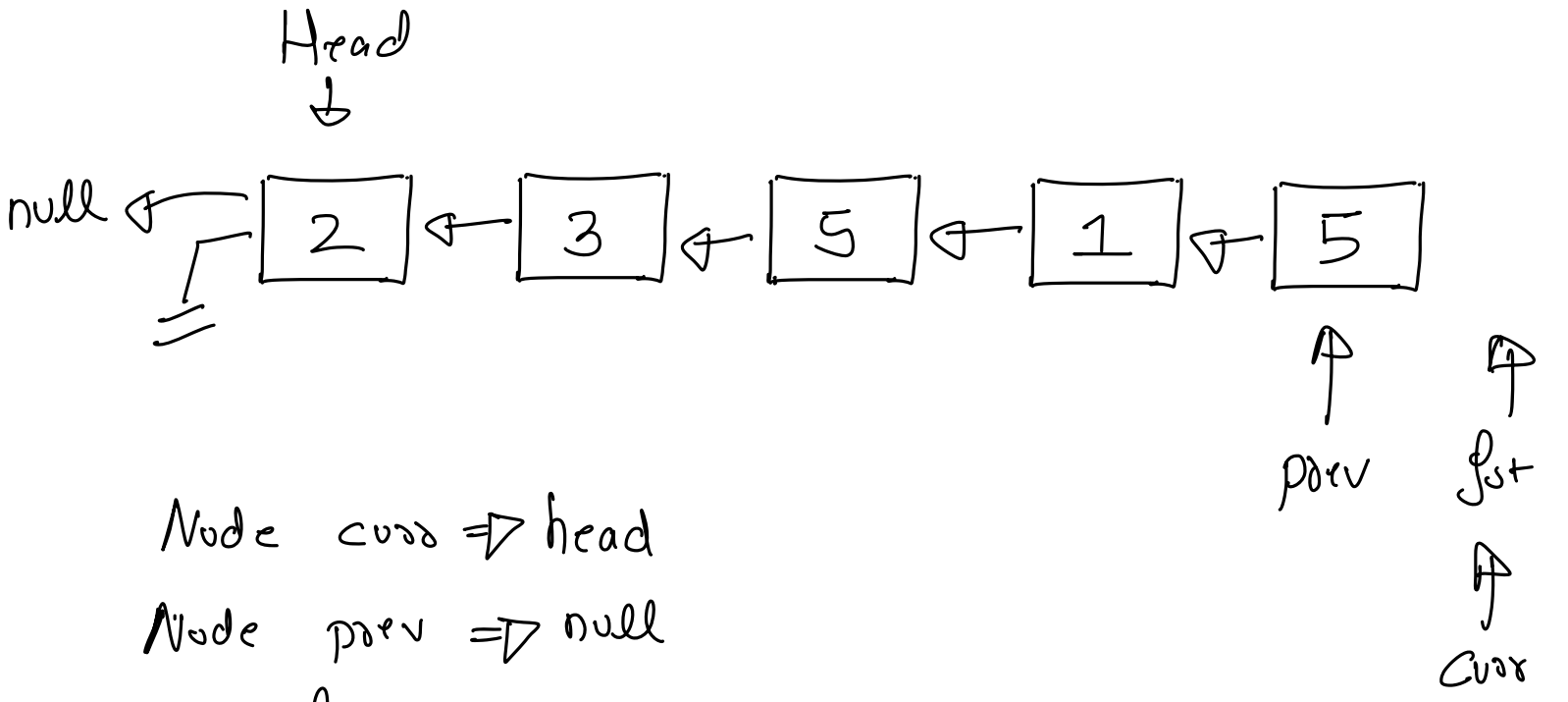


$fut = curr.next$

$curr.next = prev$

$prev = curr$

$curr = fut$



Node curr ⇒ head

Node prev ⇒ null

Node fut ⇒ null

```
while (curr != null)

    fut = curr.next
    curr.next = prev
    prev = curr
    curr = fut

}

return prev;
```

TC: O(n)

SC: O(1)

curr.next != null

fut != null

curr != null

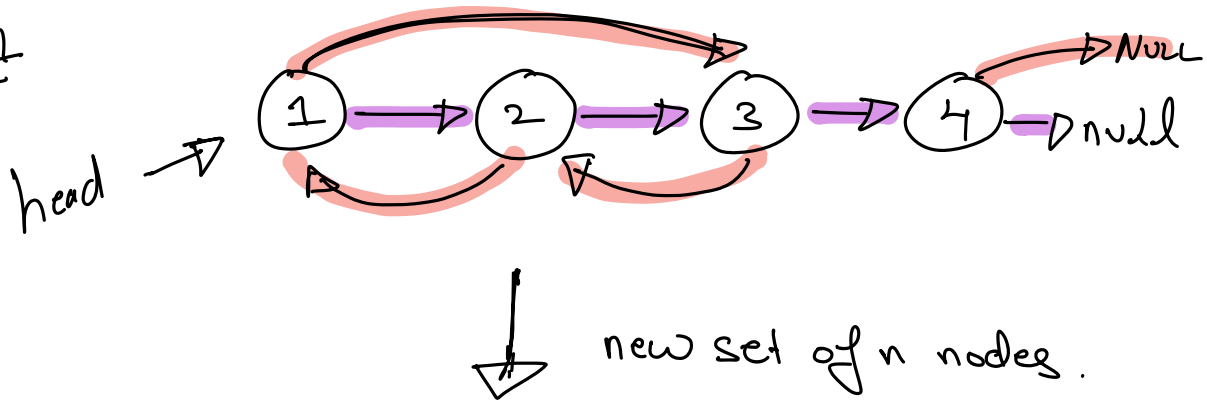
Q 5 Given a Linked list. Every node has 2 pointers.

1) next pointer : As usual.

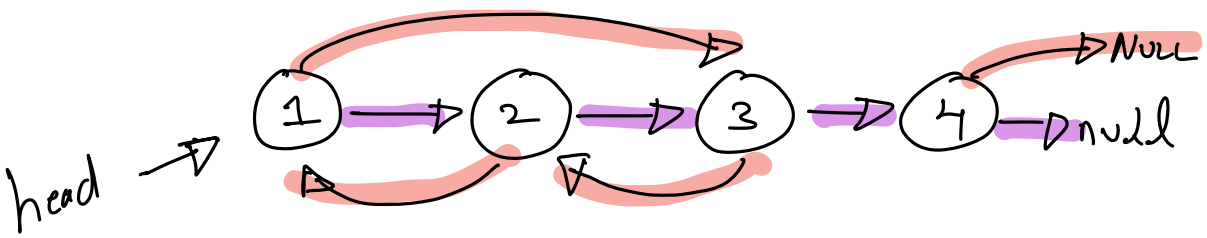
2) random pointer : Can point to any node or null

Make a clone of this. Brand new n nodes which the same structure.

Ex 1

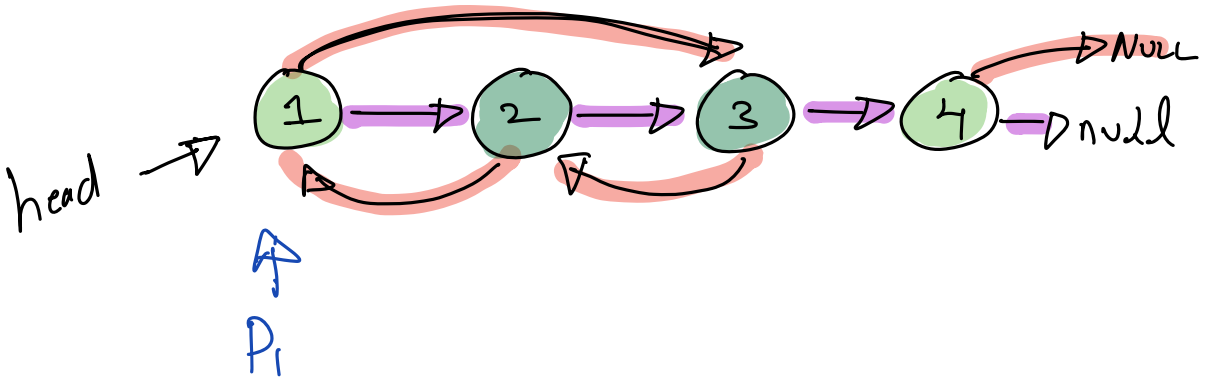


new set of n nodes.

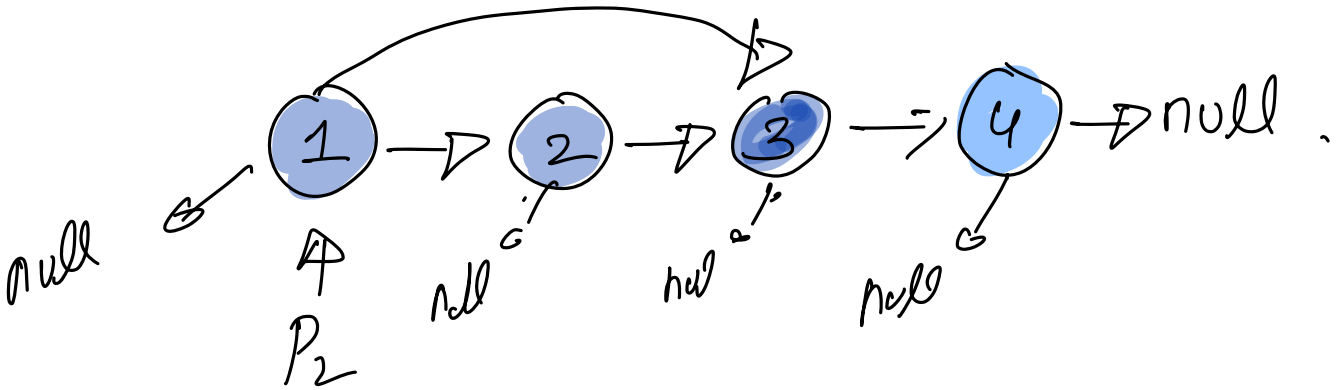


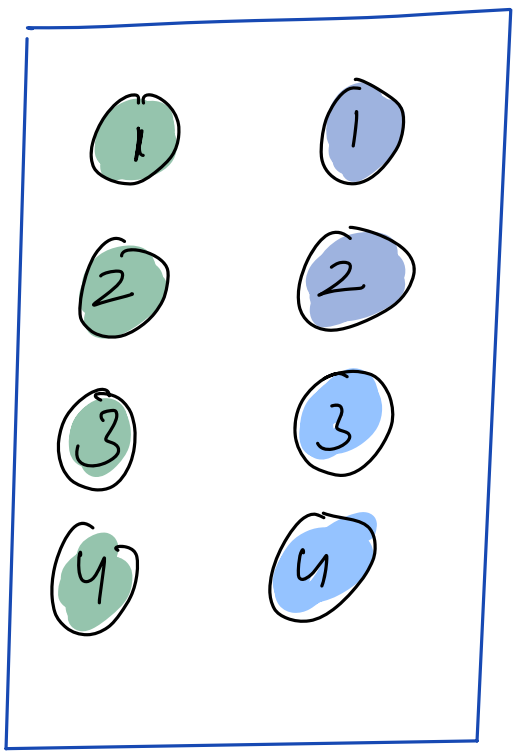
```
class Node {
    int val;
    Node next;
    Node random;

    Node(int val) {
        this.val = val;
        this.next = null;
        this.random = null;
    }
}
```

Approach 1:



P_1





p2.random ⇒

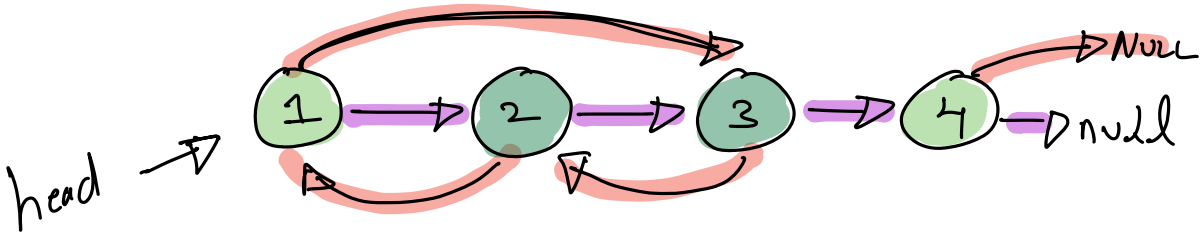
⇒ mp.get(p1.random)!

TC : O(n)

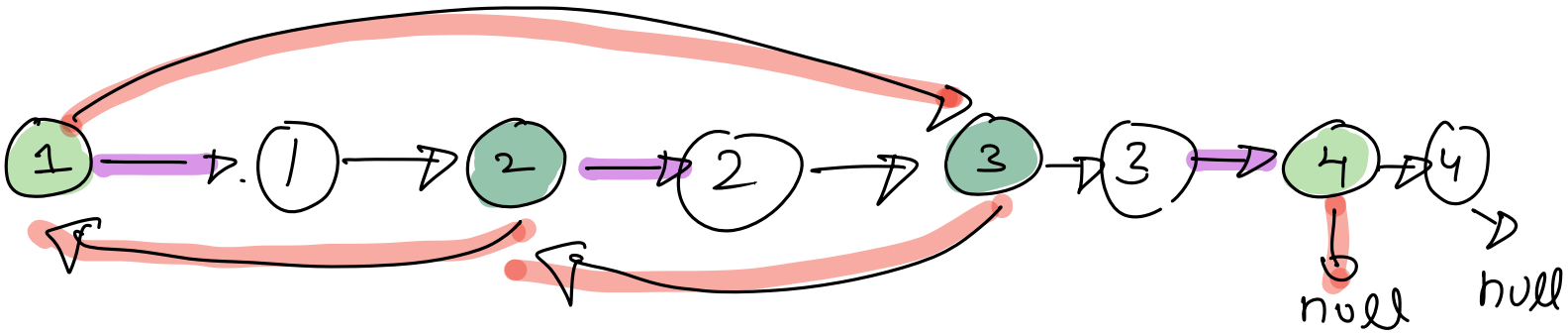
SC : O(n)

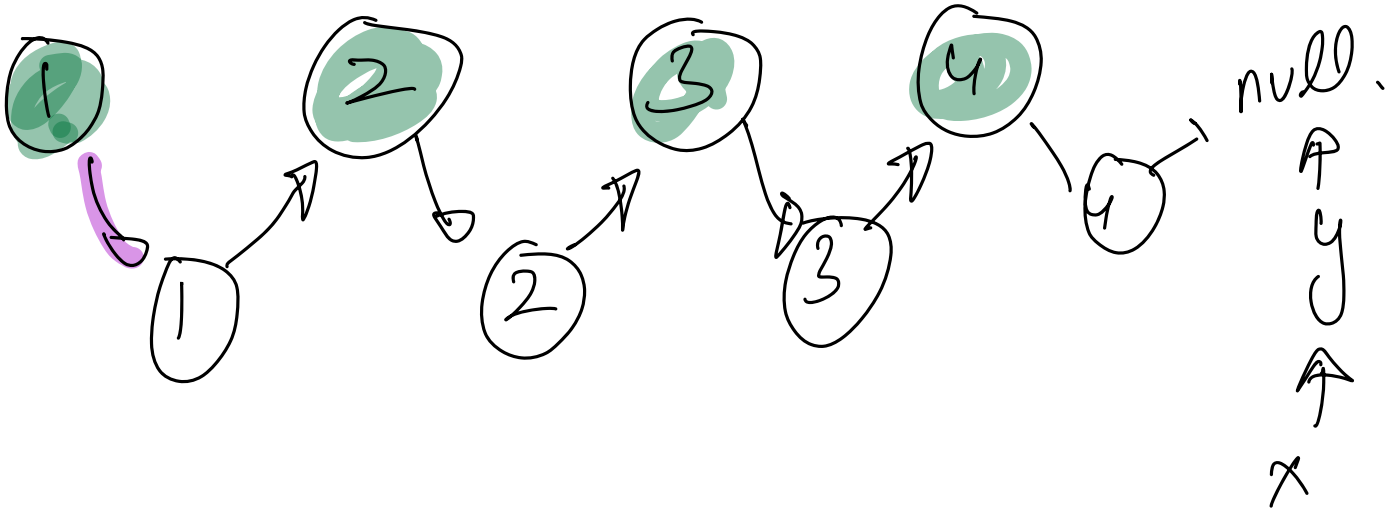
Map<Node, Node> mp

Approach 2 [Reduce Space].

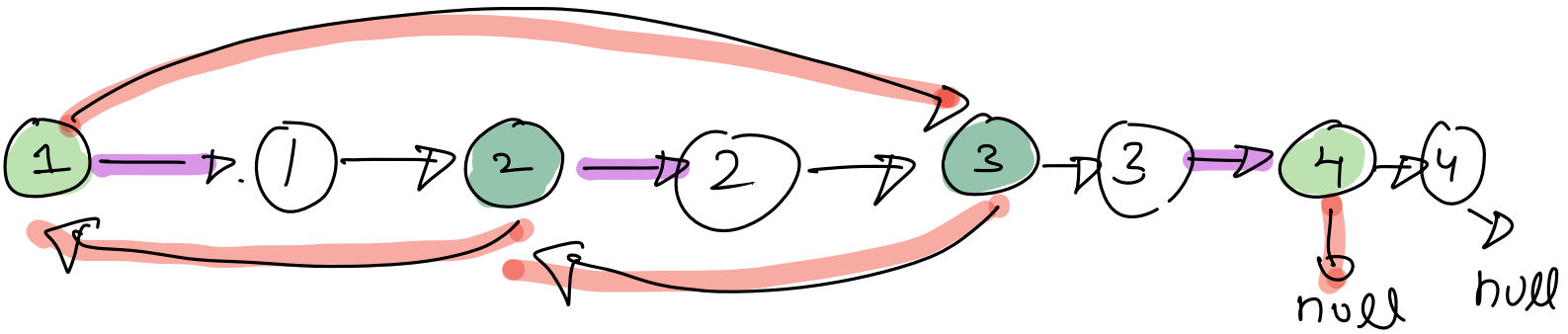


Step 1 : Add new nodes after old nodes in the same linked list

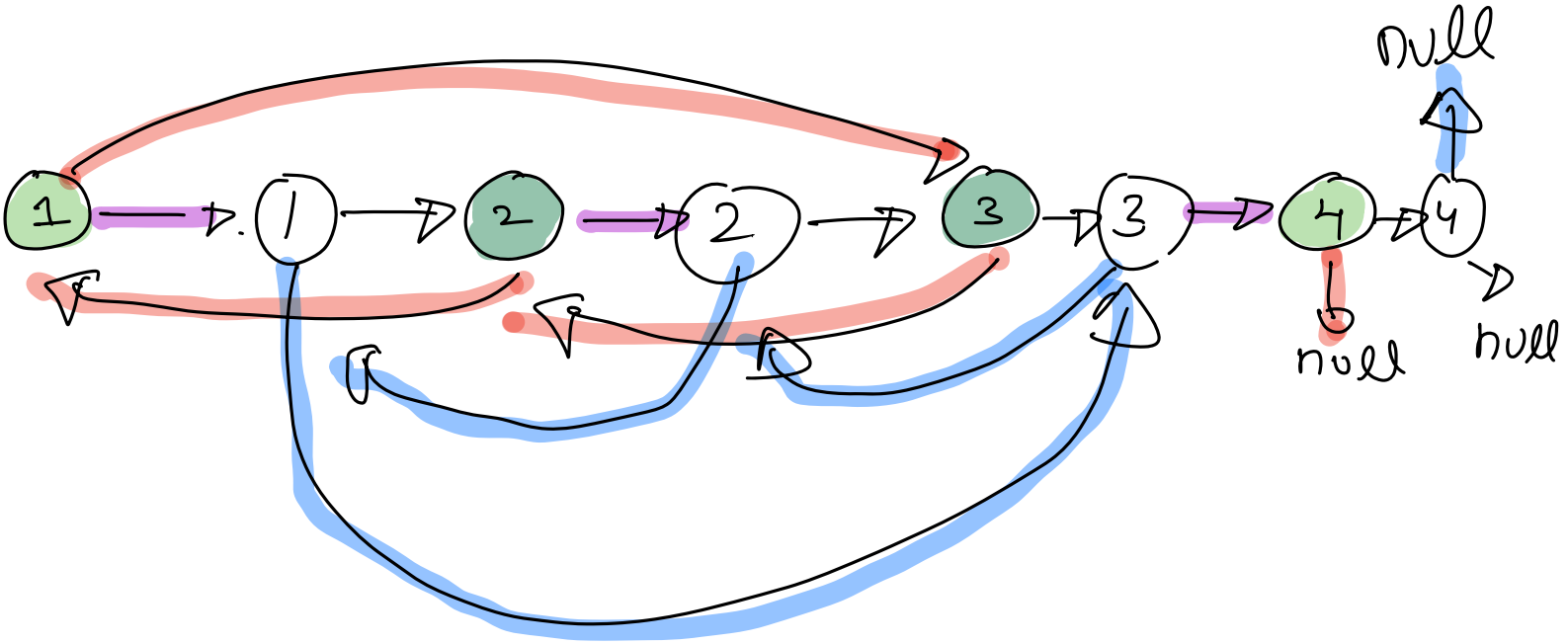


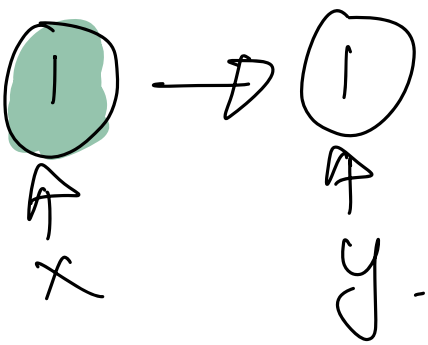


```
x => head;
y => head.next.

while (x != null) {
    x.next = new Node(x.val)
    x.next.next = y.
    x = y.
    if(y != null) {
        y = y.next.
    }
}
```



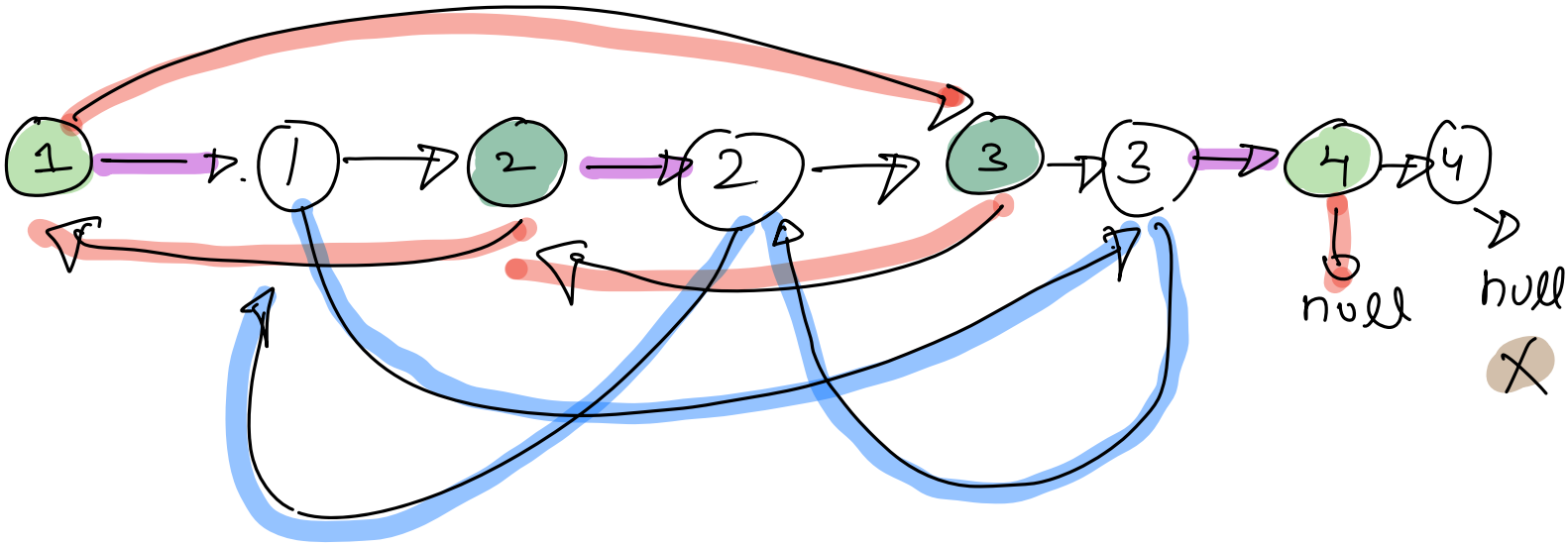
Step 2 : Point random pointer of new nodes correctly.



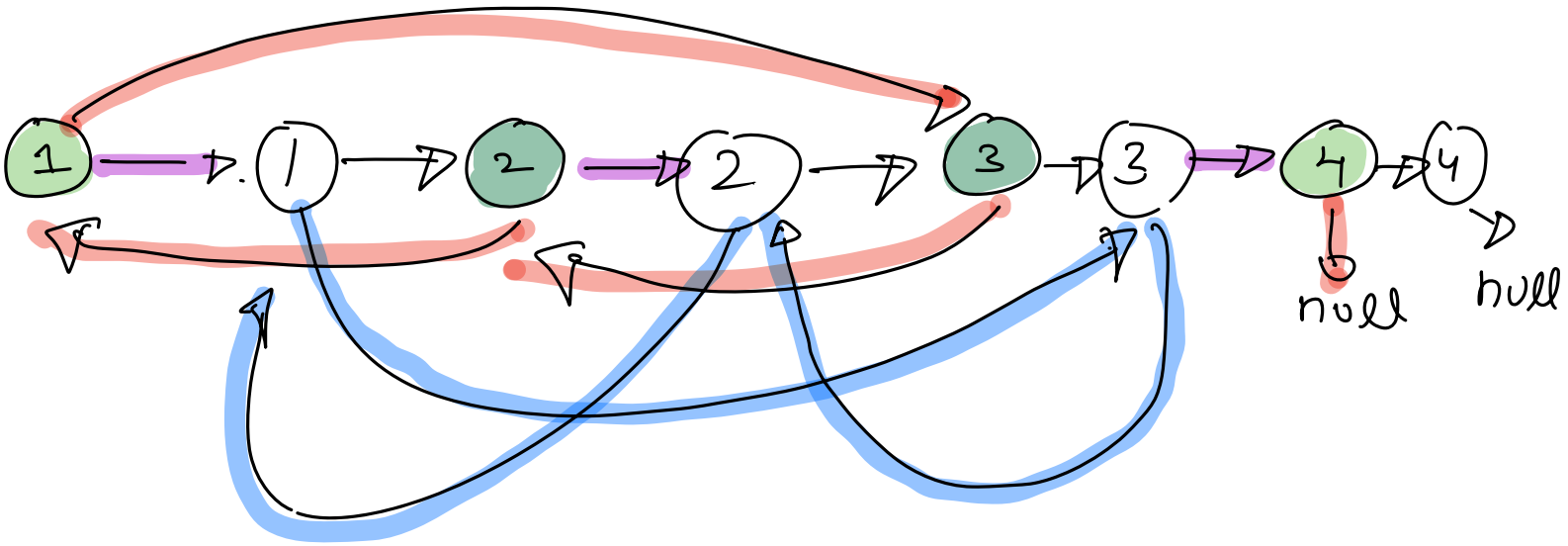


y.rand ⇒ x.rand.next

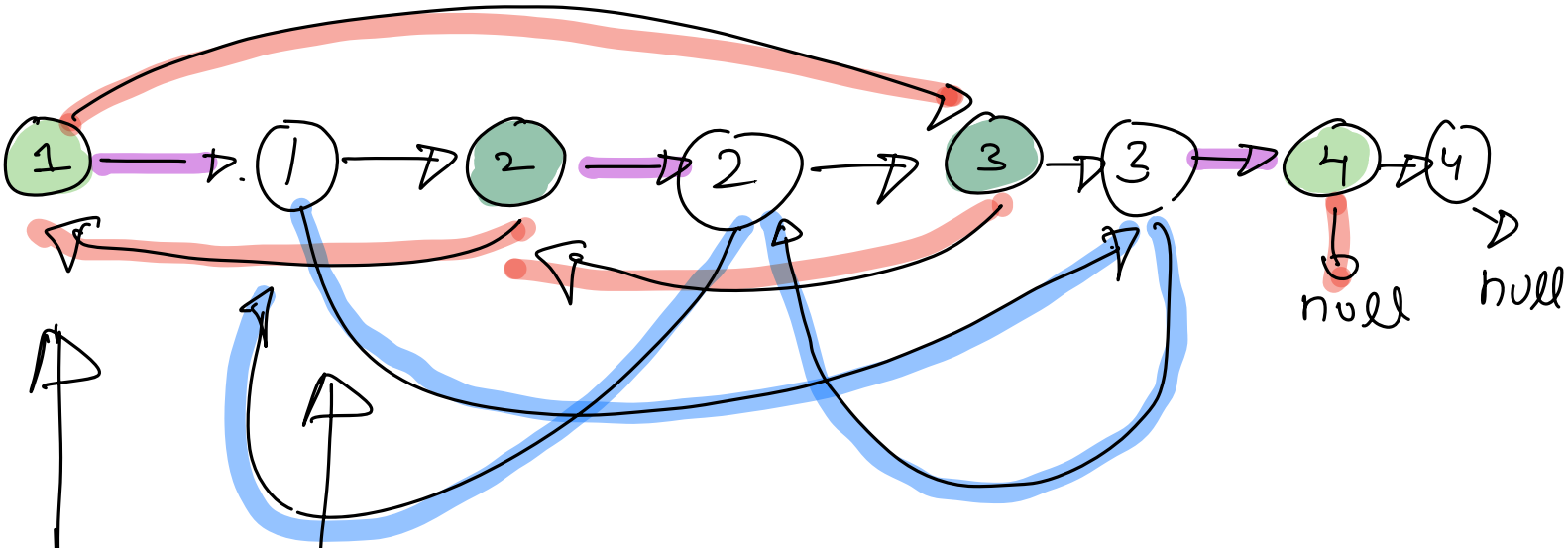
x.next.rand = x.rand.next




```
Node x = head;
while ( x != null ) {
    if (x.rand != null) {
        x.next.rand = x.rand.next
    }
    x = x.next.next;
}
```



Step 3 : Decouple the linked list



x

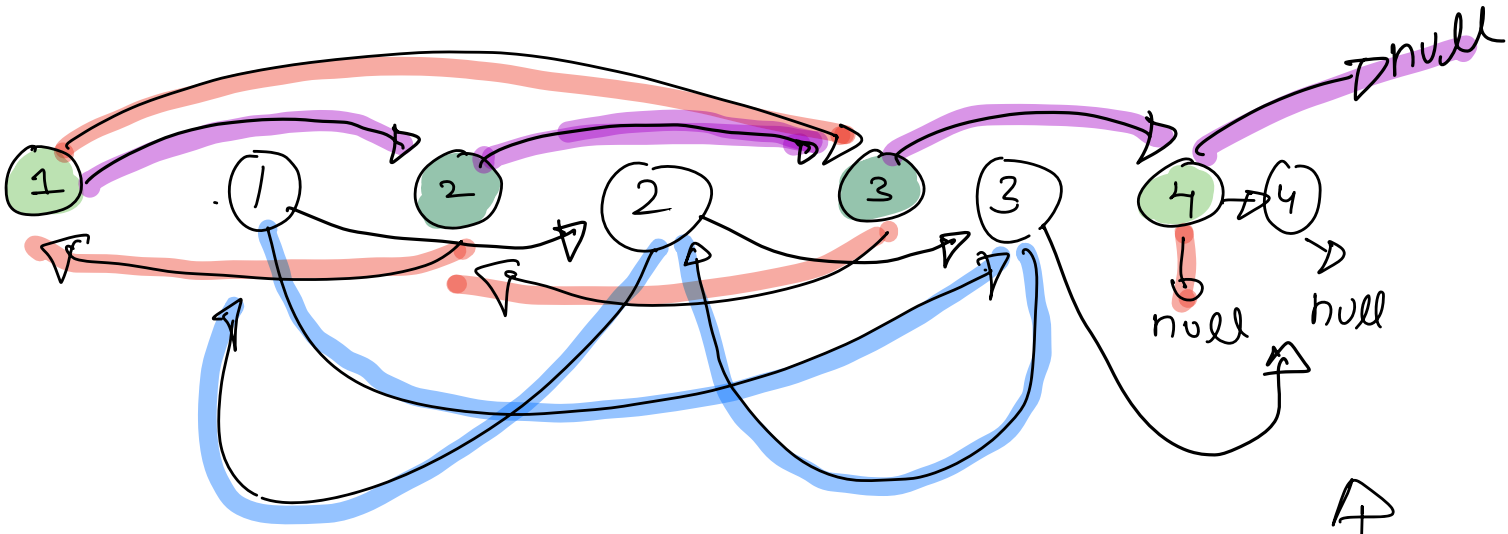
y

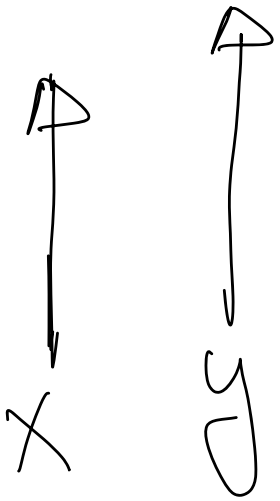
x.next = y.next

y.next = y.next.next

x = x.next

y = y.next






```
x = head;
y = head.next
while (x!=null ) {
    x.next = y.next
    if(y.next!=null) {
    y.next = y.next.next
    x = x.next
    y = y.next
}
```