1) Reversed linked list

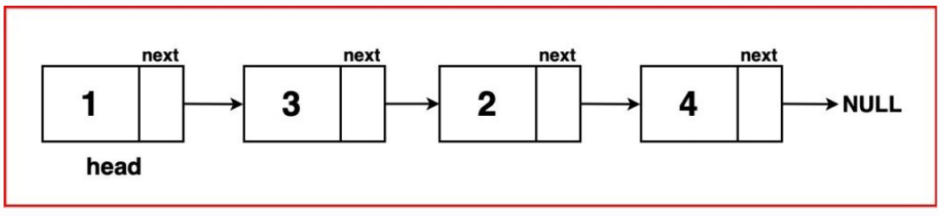
Problem Statement: Problem Statement: Given the **head** of a singly linked list, write a program to reverse the linked list, and return the **head** pointer to the **reversed list**.

Examples

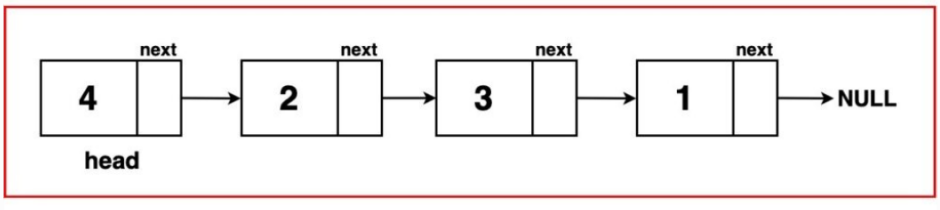
Example 1:

Input Format:

LL: 1 3 2 4



Output: 4 2 3 1

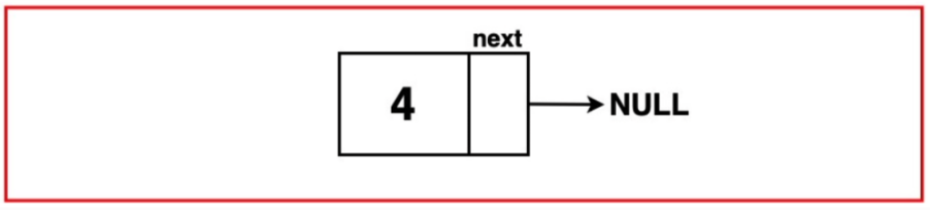


Explanation: After reversing the linked list, the new head will point to the tail of the old linked list.

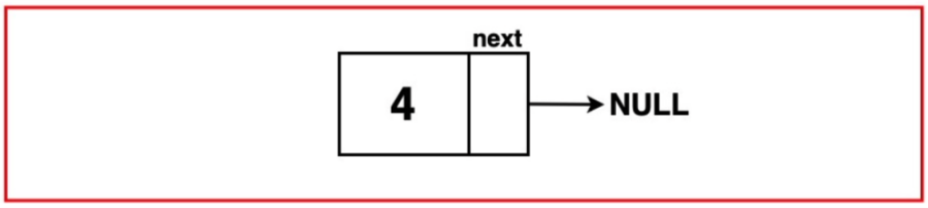
Example 2:

Input Format:

LL: 4



Output: 4



Explanation: In this example, the linked list contains only one node hence reversing this linked list will result in the same list as the original.

Approach :- * Take 2 nodes - prev as Null, curr as head.

* While Traversing Through curr, do the following steps.

i) declare curr.next as next_node

ii) declare prev as curr.next

iii) declare curr as prev

iv) declare next_node as curr

* return prev

Eg dry run

1 → 3 → 2 → 4 → Null

curr = 1, prev = Null

next_node = 3

curr.next = Null
prev = 1
curr = 3

1 → Null

next_node = 2
curr.next = 1
prev = 3
curr = 2

3 → 1 → Null

next_node = 4
curr.next = 3
prev = 2
curr = 4

2 → 3 → 1 → Null

next_node = Null
curr.next = 2
prev = 4
curr = Null

4 → 2 → 3 → 1 → Null

---

2) Merge 2 sorted linked list (use methods used

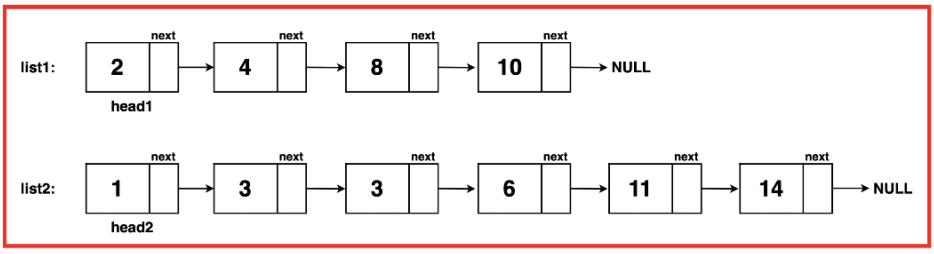
in mergesort)

Problem Statement: Given two sorted linked lists, merge them to produce a combined sorted linked list. Return the head of the final linked list created.

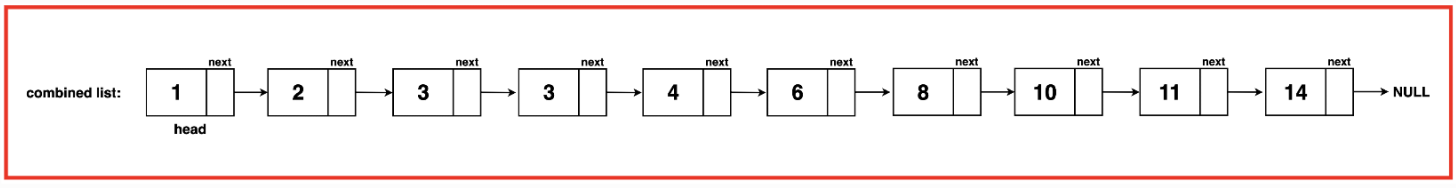
Examples

Example 1:

Input: List1: 2 4 8 10, List2: 1 3 3 6 11 14



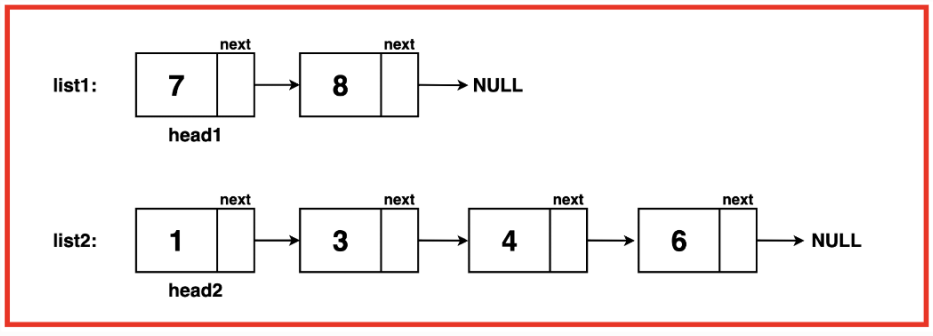
Output: Combined List: 1 2 3 3 6 8 10 11 14



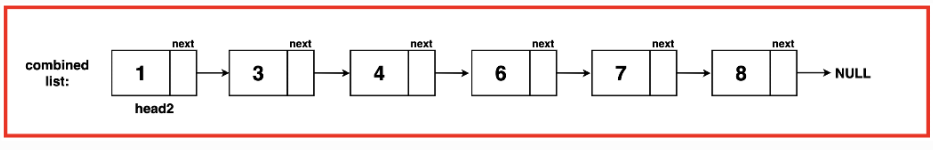
Explanation: The values in the first list are [2, 4, 8, 10] and in the second list are [1, 3, 3, 6, 11, 14], when combined and sorted give us [1, 2, 3, 3, 6, 8, 10, 11, 14].

Example 2:

Input:List1: 7 8, List2: 1 3 4 6



Output: 1 3 4 6 7 8



Explanation: The values in the first list are [7. 8] and in the second list are [1, 3, 4, 6], when combined and sorted give us [1, 3, 4, 6, 7, 8].

Approach :-

* Declare 2 linked lists — dummy & res.
  dummy = None
  res = dummy

  dummy is for returning the final array, whereas res is used for traversing.

* Later, use the same merge logic as used in mergesort to merge the 2 linked lists into the dummy/res list.

* Return dummy.next

---

3) Remove the N-th node from the back of Linked List

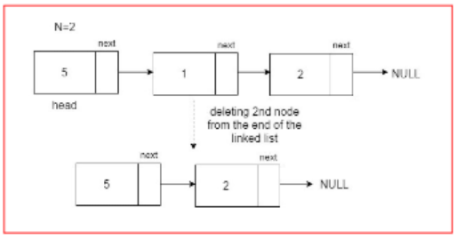
Problem Statement: Given a linked list and an integer N, the task is to **delete the Nth node from the end** of the linked list and print the updated linked list.

Examples

Example 1:

Input Format: 5->1->2, N=2

Result: 5->2



Explanation: The 2nd node from the end of the linked list is 1. Therefore, we get this result after removing 1 from the linked list.

Example 2:
Input Format: 1->2->3->4->5, N=3
Result: 1->2->4->5
Explanation: The 3rd node from the end is 3, therefore, we remove 3 from the linked list.

Approach :- * First count the nodes in the linked list.

* Then declare the position from the beginning of the array

* Go to that position and delete the node.

* Again, declare a linked list called dummy and after traversing, return dummy.next

---

4) Delete a given node when a node is given

(O(1) solution)

Problem Statement: Write a function to delete a node in a singly-

linked list. You will not be given access to the head of the list instead, you will be given access to the node to be deleted directly. It is guaranteed that the node to be deleted is not a tail node in the list.

Examples:

Example 1:
Input:
Linked list: [1,4,2,3]
Node = 2
Output:
Linked list: [1,4,3]
Explanation: Access is given to node 2. After deleting nodes, the linked list will be modified to [1,4,3].

Example 2:
Input:
Linked list: [1,2,3,4]
Node = 1
Output: Linked list: [2,3,4]
Explanation:
Access is given to node 1. After deleting nodes, the linked list will be modified to [2,3,4].

Approach:- If node is given,

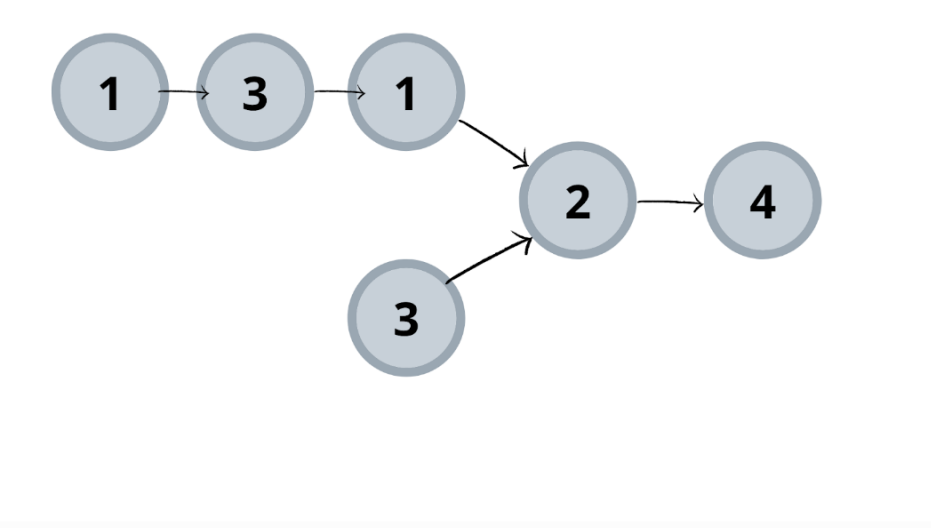
node.val = node.next.val
node.next = node.next.next

---

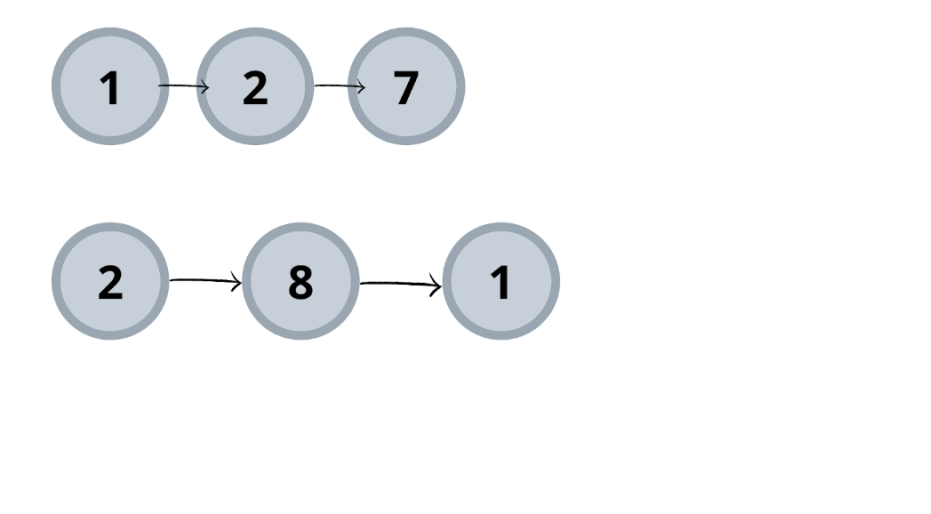
5) Find intersection point of Y Linked List

Problem Statement: Given the heads of two singly linked-lists headA and headB, return the node at which the two lists intersect. If the two linked lists have no intersection at all, return null.

Examples:
Example 1:
Input:
List 1 = [1,3,1,2,4], List 2 = [3,2,4]
Output:
2
Explanation: Here, both lists intersecting nodes start from node 2.



Example 2: Input: List1 = [1,2,7], List 2 = [2,8,1] **Output:** Null
Explanation: Here, both lists do not intersect and thus no intersection node is present.



Approach :- * Count the number of nodes in both the lists, and find the difference between the 2 counts.

* If head A has more no. of elements, then traverse through head A list till diff & no. of elements are traversed.

* Same applies to B.

* If both lists have the same no. of nodes, then no need to do anything.

* Later, taking the new lists, traverse through both the lists. If the values are equal, return that node.

---

6) Reverse a linked list in a group of size k

Problem Statement: Given the head of a singly linked list of `n` nodes and an integer `k`, where k is less than or equal to `n`. Your task is to reverse the order of each group of `k` consecutive nodes, if `n` is not divisible by `k`, then the last group of remaining nodes should remain unchanged.

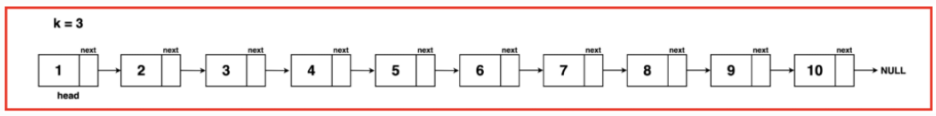
Examples

Example 1:

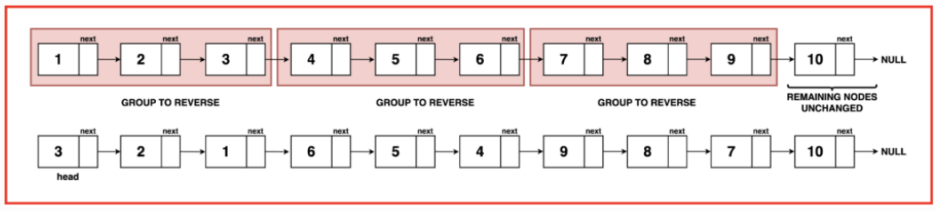
Input Format:

LL: 1 2 3 4 5 6 7 8 9 10

K = 3



Output: 3 2 1 6 5 4 9 8 7 10



Explanation:

Group 1: Reverse nodes 1 -> 2 -> 3 to become 3 -> 2 -> 1.

Group 2: Reverse nodes 4 -> 5 -> 6 to become 6 -> 5 -> 4.

Group 3: Reverse nodes 7 -> 8 -> 9 to become 9 -> 8 -> 7.

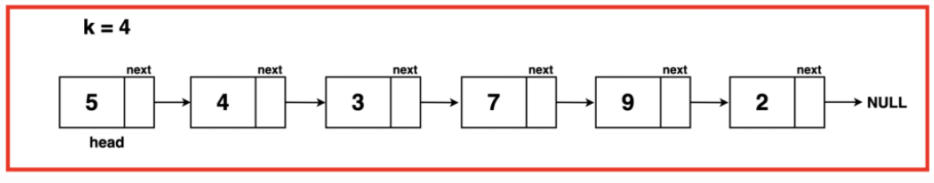
Node 10 remains as is since there are no more groups of size K remaining.

Example 2:

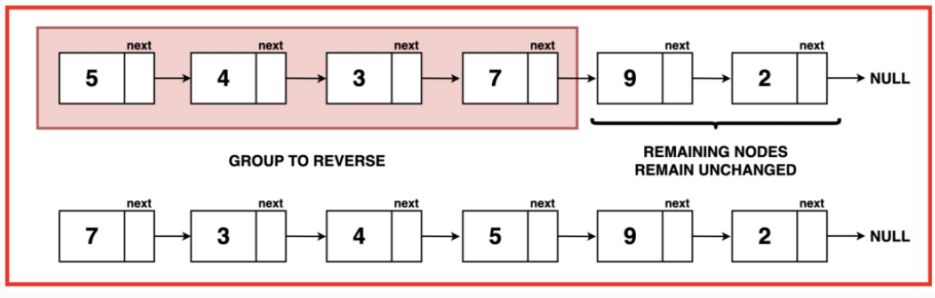
Input Format:

LL: 5 4 3 7 9 2

K = 4



Output: 7 3 4 5 9 2



Explanation:

Group 1: Reversed nodes 5 -> 4 -> 3 -> 7 to become 7 -> 3 -> 4 -> 5.

Group 2: Nodes 9 -> 2 remain unchanged as they are not a complete group of size K.

Approach:- * This approach involves the use of stack.

* While traversing through the list, push the node into the stack.

* If the length of the stack is k, then pop the element and form a new list

* After the entire list is traversed, then add new nodes to the stack from index 0 to n, and not by popping the stack.

---

7) Check if a linked list is a palindrome or not

Example 1:
Input Format:
LL: 1 2 3 2 1
Output: True
Explanation: A linked list with values "1 2 3 2 1" is a palindrome because its elements read the same from left to right and from right to left, making it symmetrical and mirroring itself.

Example 2:
Input Format:
LL: 1 2 3 3 2 1
Output: True
Explanation: A linked list with values "1 2 3 3 2 1" is a palindrome because it reads the same forwards and backwards.

Example 3:
Input Format:
LL: 1 2 3 2 3

Output: False
Explanation: The linked list "1 2 3 2 3" is not a palindrome because it reads differently in reverse order, where "3 2 3 2 1" is not the same as the original sequence "1 2 3 2 3."

Approach: * Add all the node values to a list l. Reverse the list l.

* Now, compare the reverse list values and the linked list values. If they matches, the linked list is a palindrome.

---

8) Rotate a linked list

Problem Statement: Given the head of a linked list, rotate the list to the right by k places.

Examples:

Example 1:
Input:
head = [1,2,3,4,5]
k = 2
Output:
head = [4,5,1,2,3]
Explanation:
We have to rotate the list to the right twice.

Example 2:
Input:
head = [1,2,3]
k = 4

Output:
head = [3,1,2]
Explanation:

Approach:
* If the list is empty or has only one element or k = 0, then return the original linked list

* Count the number of nodes in the list. If k is the multiple of the length, then return the original list. Also, make the list circular by giving the link of last element to the first.
* Else, declare actual as $l - (k \% l)$.

* Traverse the list actual - 1 number of times. Declare t.next as the new_head and Null as t.next. Return new_head

---

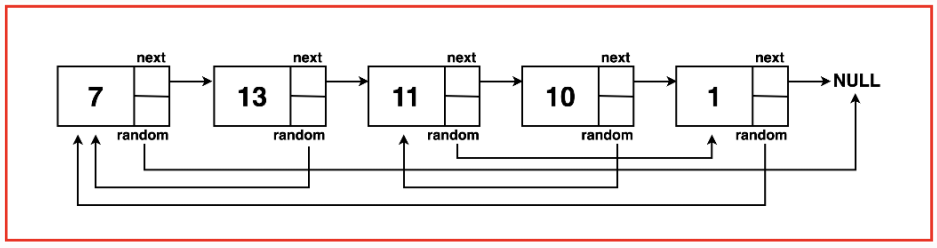
9) Clone a linked list with random and next pointer

Problem Statement: Given a linked list where every node in the linked list contains two pointers:

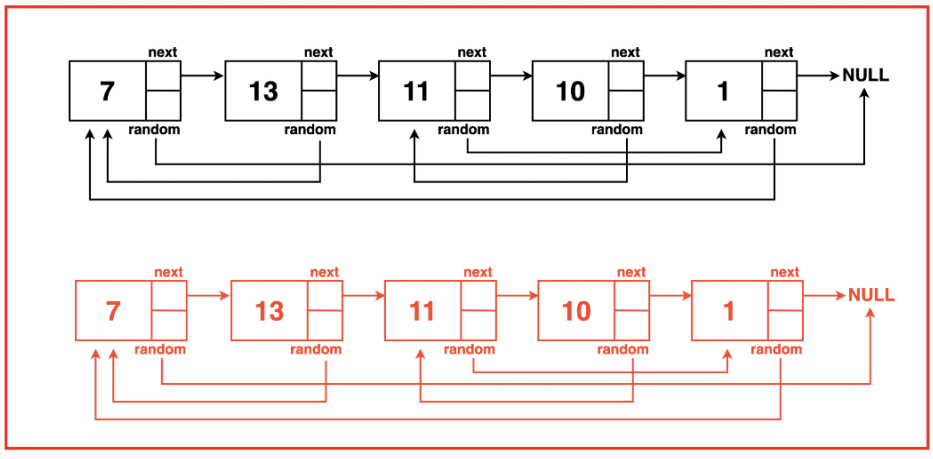
- 'next' which points to the next node in the list.
- 'random' which points to a random node in the list or 'null'.

Create a 'deep copy' of the given linked list and return it.

Examples
Example 1:
Input:



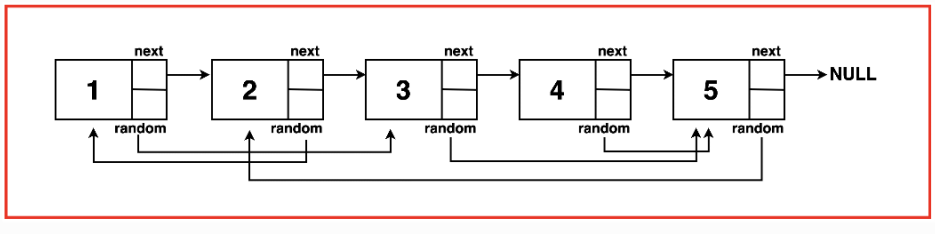
Output:



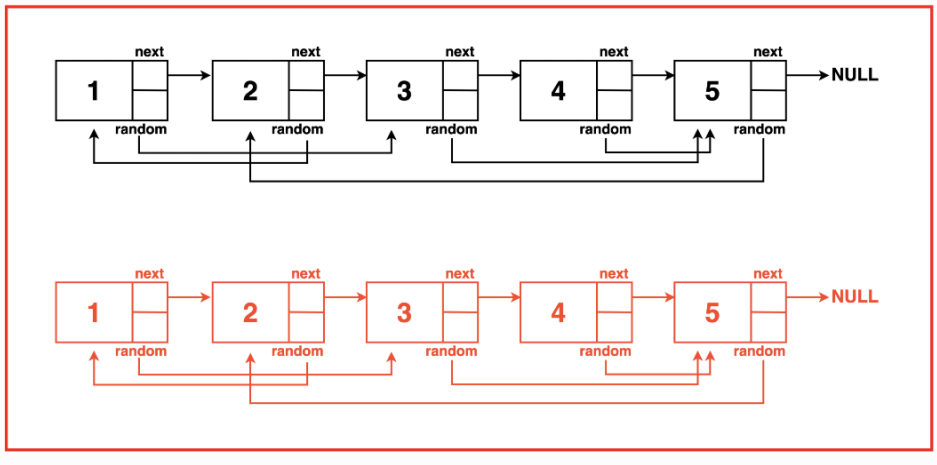
Explanation: A deep copy of the linked list has to be created while maintaining all ‘next’ and ‘random’ pointers to the appropriate new nodes. Additional memory allocation is done while creating a duplicate set of nodes and managing their pointer relationships.

Example 2:

Input:



Output:



Explanation: A deep copy of the linked list has to be created while maintaining all 'next' and 'random' pointers to the appropriate new nodes. Additional memory allocation is done while creating a duplicate set of nodes and managing their pointer relationships.

Approach: * Interleave copied nodes after original nodes.

* Later, change the random pointers of the copied nodes.

```
t = head
while t:
    if t.random:
        t.next.random = t.random.next
    t = t.next.next
```

* Next, Seperate the original linked list from the copied one.

```
t = head
res = head.next
while t:
    copy = t.next
    t.next = copy.next
    if copy.next:
        copy.next = copy.next.next
    t = t.next
return res.
```