Linked List: Basic Problems -2

1.Problem Statement

Given X and a linked list (head is given). Check if X is present in the linked list (LL) or not.

X =8 ans = true

X = 16 ans = false

Pseudocode

```
Boolean FindX(Nodehead, int X){
        node tmp = head
        while(tmp!= null){
                if(tmp.data == X) return true;
                tmp = tmp.next;
        }
        return false
}
```

2.Problem Statement - INSERT A NEW NODE IN LL

You are given **A** which is the head of a linked list. Also given is the **value B** and **position C**. Complete the function that should insert a new node with the said value at the given position.

Notes:

- In case the position is more than length of linked list, simply insert the new node at the tail only.
- In case the pos is 0, simply insert the new node at head only.
- Follow 0-based indexing for the node numbering.

Problem Constraints

$0 <= \text{size of linked list} <= 10^5$

$1 <= \text{value of nodes} <= 10^9$

$1 <= B <= 10^9$

$0 <= C <= 10^5$

Output Format: Return the head of the linked list

Input 1:

A = 1 -> 2

B = 3

C = 0

Output 1: 3 -> 1 -> 2

Example Explanation

The new node is add to the head of the linked list

10 → 20 → 6 → 4 → 15 → Null

X = 12

P = 3

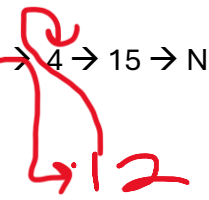
10 → 20 → 6 → 12 → 4 → 15 → Null

10 → 20 → 6 → 12 → 4 → 15 → Null

Pseudocode

```
Node tmp = head
 // when P==0
 if(P==0){
     Node NN = new node (k)
     NN.next = head;
      head = NN
      return
 }
 for(int i =1; i<=P-1; i++){          //my goal is to run a loop P-1 times & after this loop my tmp is
      if(tmp == null) return;         sitting at P-1 times // accessing .next → make sure it is not null
      tmp = tmp.next;
 }
 if(tmp == null) return;              //in LL we need to check null checker. whenever we are
 Node NN = new node (X);              //create a new node NN with value 12
 NN.next = tmp.next;
 tmp.next = NN;
```

10 → 20 → 6 → 4 → 15 → Null



TC = O(min(P, N) *//since we are running loop P-1 in 0^{th} index LL*

Actual Code -- value B and **position C**

```
public class Solution {
  public ListNode solve(ListNode A, int B, int C) {
    ListNode NN = new ListNode(B);
    if(A == null) return NN;
    if(C==0 && A != null){
```

```
        NN.next = A;
        A = NN;
        return NN;
      }
      ListNode tmp = A;
      for(int i =1; i<=C-1; i++){
        if(tmp != null){
          tmp = tmp.next;
        }
      }
      if(tmp != null){
        NN.next = tmp.next;
        tmp.next = NN;
      }else{
        ListNode tail = A; // Edge case: If position C is out of bounds, append at the end
        while(tail.next != null){
          tail = tail.next;
        }
        tail.next = NN;
      }
      return A;
    }
}
```

3.Problem Statement - Deletion in LL

You are given the head of a linked list **A** and an integer **B**. Delete the **B**-th node from the linked list. Note : Follow 0-based indexing for the node numbering.

Problem Constraints

$1 <= \text{size of linked list} <= 10^5$

$1 <= \text{value of nodes} <= 10^9$

$0 <= B < \text{size of linked list}$

Output Format: Return the head of the linked list after deletion

Example Input

A = 1 -> 2 -> 3

B = 1

Example Output: 1 -> 3

Example Explanation

The linked list after deletion is 1 -> 3.

head →1→ 8 → 12 → null

Pseudocode

```
if(head != null && head.data ==X){
      head = head.next
      return
}
Node tmp = head
while(tmp != null && tmp.next != null && tmp.next.data != X){
       tmp = tmp.next
}
if(tmp != null || tmp.next == null) return
tmp.next = tmp.next.next
return
```

Actual Code

```
public class Solution {
  public ListNode solve(ListNode A, int B) {
    ListNode tmp = A;
    if( B==0 && A != null){
      //A = tmp.next;
      return A.next;
    }
    for(int i =1; i <= B-1; i++){
      if(tmp == null || tmp.next ==null){
        return A;
      }
      tmp = tmp.next;
    }
    if(tmp.next != null){
      tmp.next = tmp.next.next;
    }
    return A;
  }
}
```

TC = O(N)
SC = O(1)

NOTE: In some languages unlike Java which have Garbage Collector to free up the space, we have to write Free(X') to free up the memory, where X = is any data.

4.Problem Statement – Remove all occurrences of X

In OnePlus firm, they have defective model. Remove all references of the defective model.

Pseudocode

```
while(tmp !=null){
    if(tmp.data == X){
```

```
            prev.next = tmp.next
            tmp = tmp.next
    }else{
            tmp = tmp.next
            prev = prev.next
    }
}
```

5.Problem Statement – Reverse LL

2 → 5→ 6 → 7 → null

↓ Reverse

2 ← 5 ← 6 ←7 ← null or 7 → 6 → 5 → 2 → null

Pseudocode

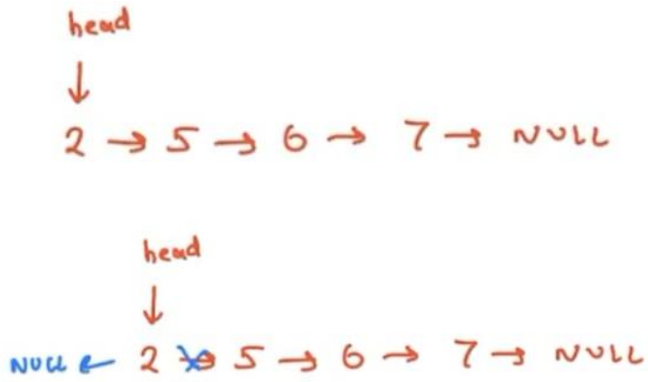
```
curr = head;
while(curr != null){
    tmp = curr.next;
    curr.next = prev;
    prev = curr;
    curr = tmp;
}
head = prev;
```

TC = O(N)

SC = O(1)

Dry Run

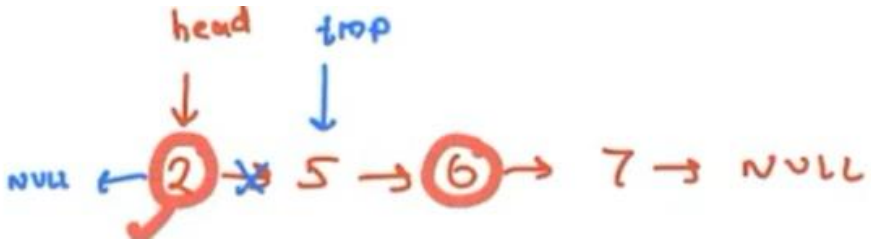
2→ 5 → 6 → 7 → Null



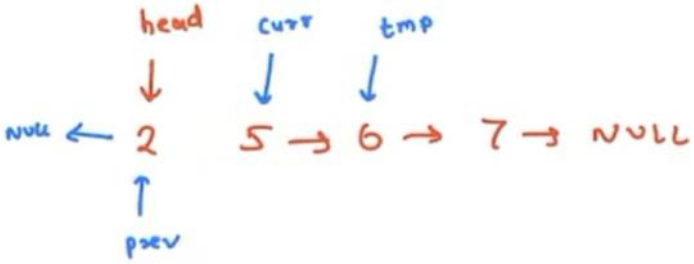
If I link 2 with null. I will lose access to the entire LL

So before referencing 2 to null. Have a reference to the next element tmp =5. So 2.next is sorted.

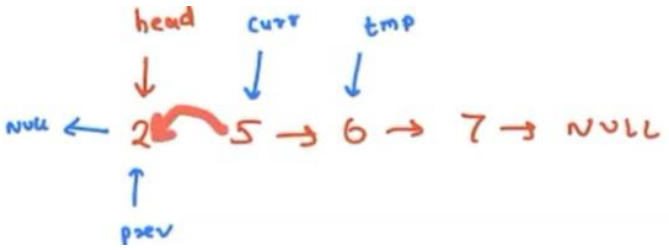
Now 5.next should refer to 2 but I do this, I will lose reference to entire LL.
tmp = curr.next



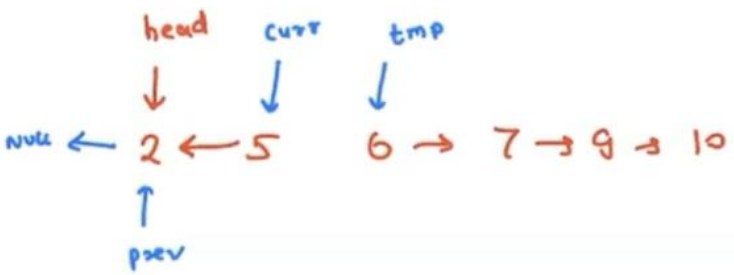
So I need a reference for 6 and 2 as well.



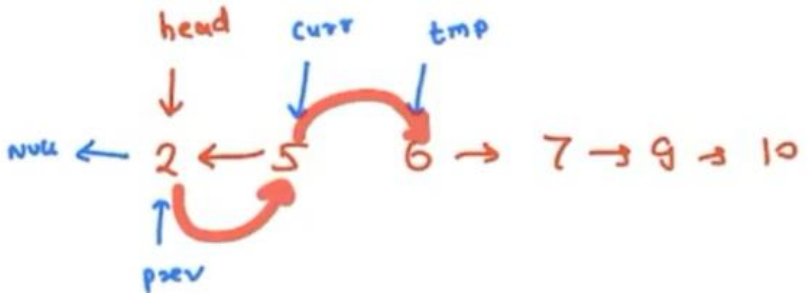
Now since I have stored the reference of the next element → 6, I can reference 5.next to prev.



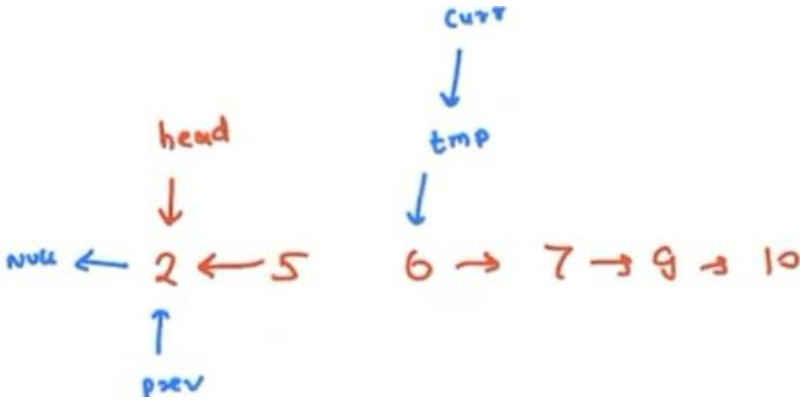
So curr.next = prev



Before referencing 6.next to 5, Now move curr and prev



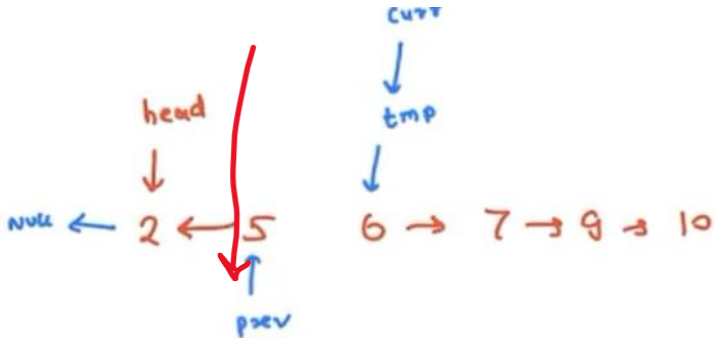
curr.next = prev



1st move prev then curr else I will lose reference to the LL

prev = curr

curr = tmp

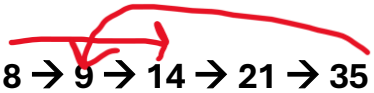


Actual Code

```
public class Solution {
  public ListNode reverseList(ListNode A) {
    ListNode curr = A;
    ListNode prev = null;
    while(curr != null){
      ListNode tmp = curr.next;
      curr.next = prev;
      prev = curr;
      curr = tmp;
    }
    A = prev;
    return A;
```

```
    }
}
```

6.Problem Statement – Deep copy of LL with random nodes



8 → 9 → 14 → 21 → 35 → null

- You are given a linked list A
- Each node in the linked list contains two pointers: a next pointer and a random pointer
- The next pointer points to the next node in the list
- The random pointer can point to any node in the list, or it can be NULL
- Your task is to create a deep copy of the linked list A
- The copied list should be a completely separate linked list from the original list, but with the same node values and random pointer connections as the original list
- You should create a new linked list B, where each node in B has the same value as the corresponding node in A
- The next and random pointers of each node in B should point to the corresponding nodes in B (*rather than A*)

Problem Constraints

$0 <= |A| <= 10^6$

Output Format: Return a pointer to the head of the required linked list.

Example Input

Given list

1 -> 2 -> 3 with random pointers going from

1 -> 3

2 -> 1

3 -> 1

Example Output

1 -> 2 -> 3

with random pointers going from

1 -> 3

2 -> 1

3 -> 1

Example Explanation

You should return a deep copy of the list. The returned answer should not contain the same node as the original list, but a copy of them. The pointers in the returned list should not link to any node in the original input list.

Pseudocode for Cloning LL

```
cur = head;
Node X = new Node (curr.data);
head' = X;
prev = head';
curr = curr.next;
while(curr != null){
        Node X = new node (curr.data);
        prev.next = X;
        curr = curr.next;
        prev = prev.next;
}
```

curr

head → 10 → 15 → 16 → 12 → null

h' → 10 → 15 → 16 → 12 →

prev

Now for attaching random references ➔ we can use a hashmap. But hashmap will take O(N) space.

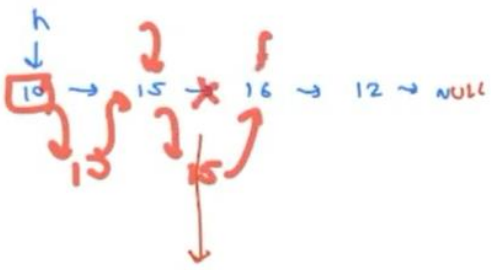
10 old ➔ 10 new
15 old ➔ 15 new
16 old ➔ 16 new
12 old ➔ 12 new

Another method is a 3 step solution. In this no extra space is required other than the cloned LL

STEP1: Create a LL like this

10 → 10 → 15 → 15 → 16 → 16 → 12 → 12 → null

old new



10→10→15→15→16→16→12→12→NULL

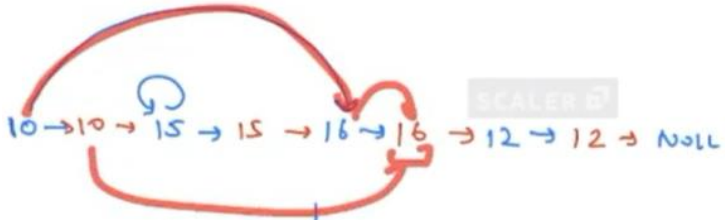
When you are at 10, add a new 10, when you are at 15, add a new 15.

STEP2: Point random links

old 15.random.next → new node 15

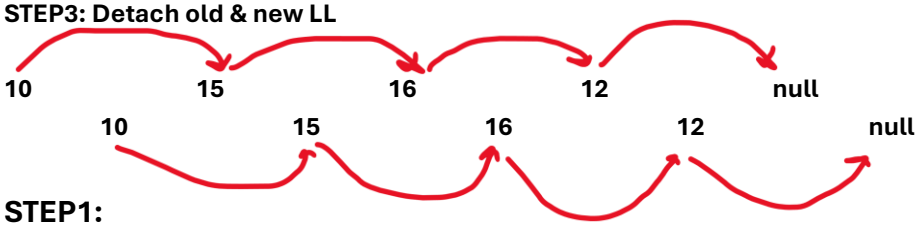
wherever is your old random → go .next → it will have new node

10 → 10 → 15 → 15 → 16 → 16 → 12 → 12 → null

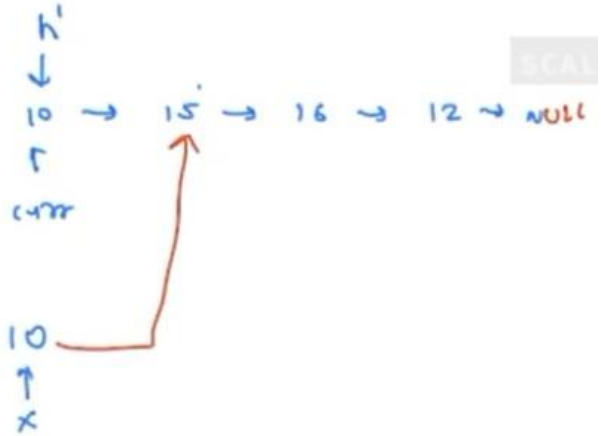


Coz of the nature of the current LL, new 16 will be always at the next of old 16. old.random.next will give you the new random of this new 10.

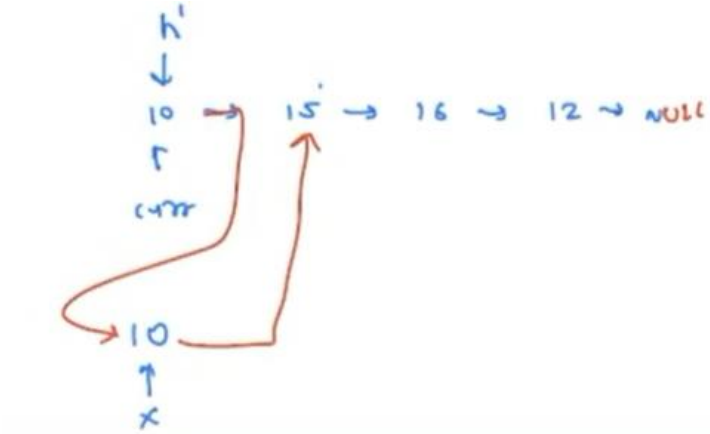
STEP3: Detach old & new LL



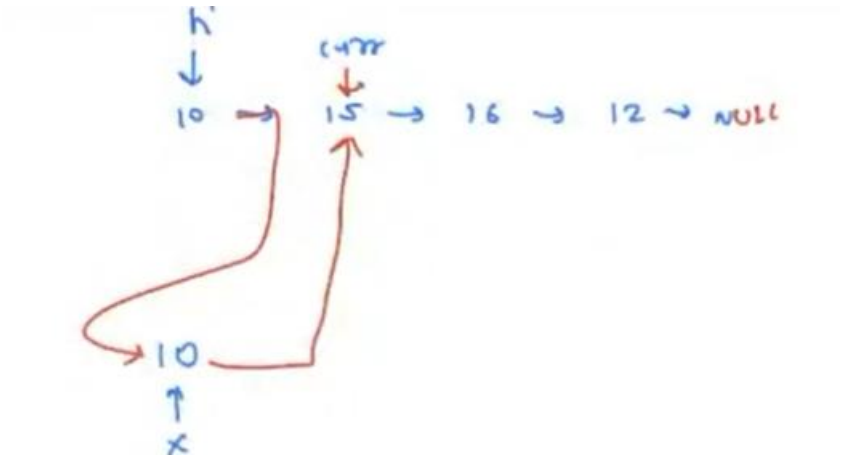
STEP1:



X is new node 10. First we will connect X.next to curr.next so that we don't lose reference of the LL(If we would have connected 10 old.next to 10.new



Now curr.next = X.data



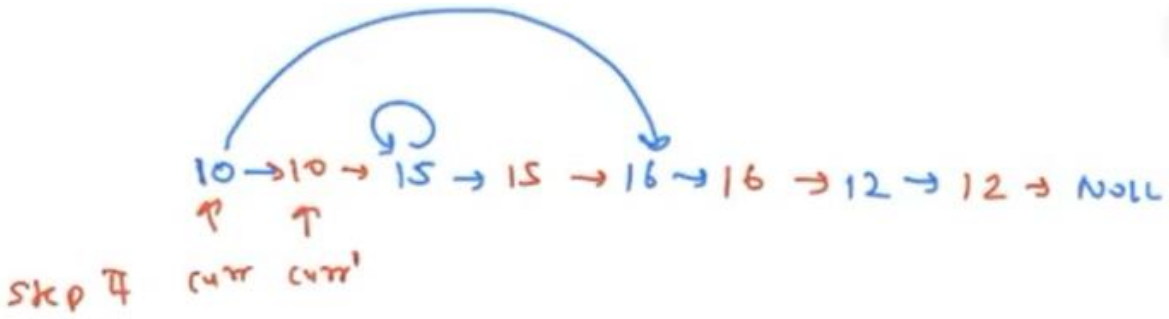
Now moving the curr pointer to curr = X.next. Now keep repeating the process

Pseudocode

```
curr = head
while (curr != null){
    Node X = new node(curr.data);
    X.next = curr.next;
    curr.next = X;
    curr = X.next;
}
```

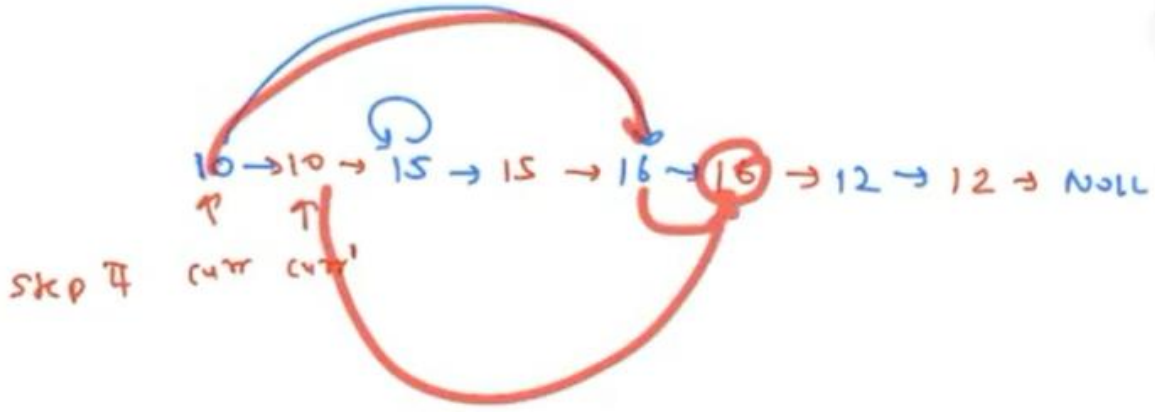
Step 1

STEP2:

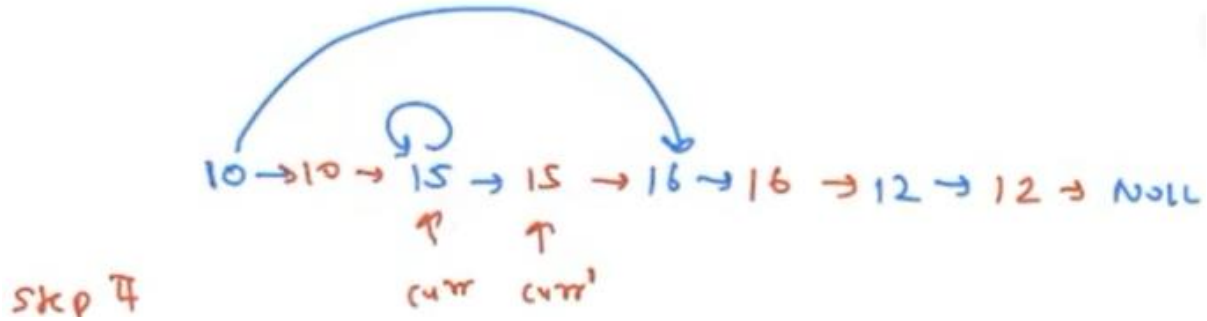


curr = h

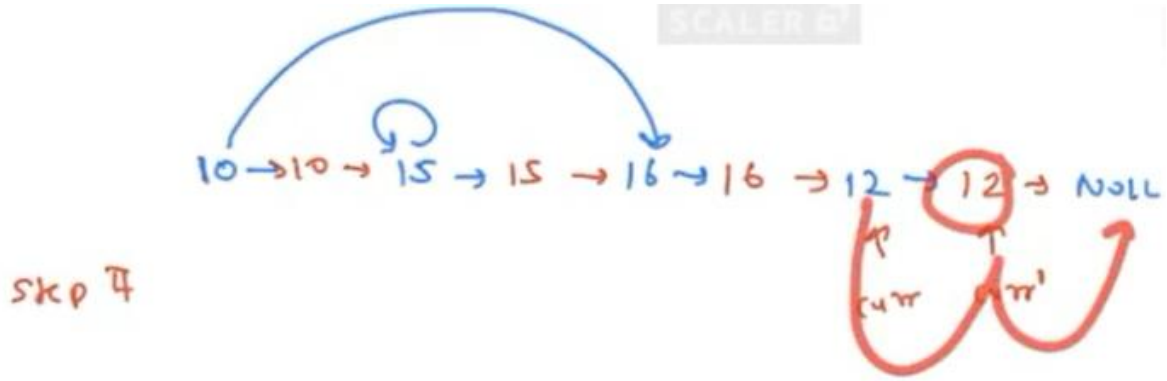
curr' = h.next



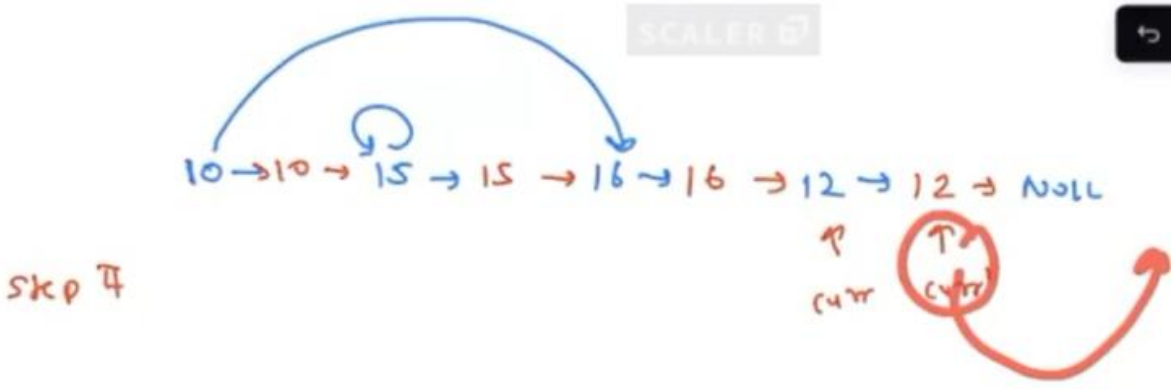
Wherever curr.random.next is that should be curr'.random
curr'.random = curr.random.next



Once this is fixed, now move curr = curr.next.next
curr' = curr.next.next



No when you are at the last node, curr will jump twice to null



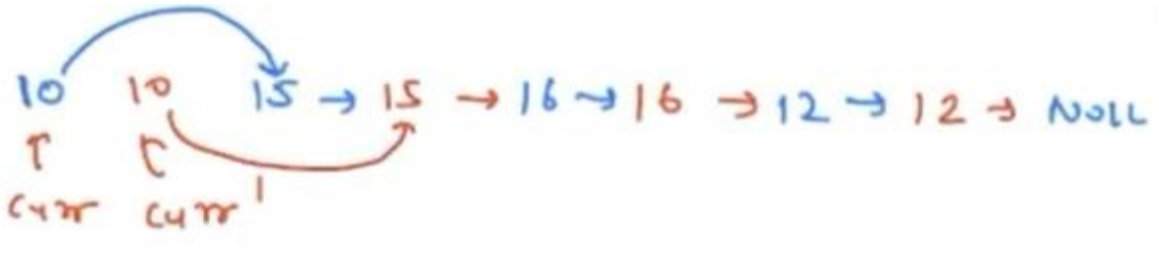
But curr' cannot jump twice. So last jump should be done carefully.

```
curr = head;
curr' = h.next;
while(curr != null){
    curr'.random = curr.random.next
    curr = curr.next.next;
    if(curr !=null){
        curr' = curr'.next.next
    }
}
```

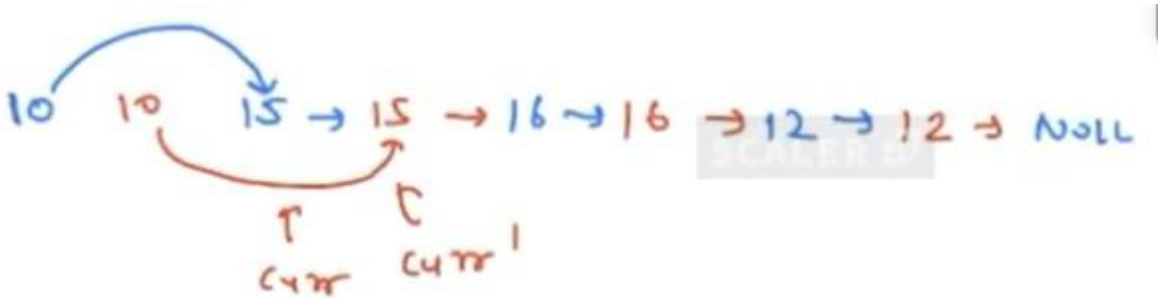
Step2

STEP3:

Connecting curr 10 to node 15



Now curr'.next = curr'.next.next



Now moving pointers curr and curr'
curr = curr.next
curr' = curr'.next

```
curr = h;
curr' = h.next;
while(curr != null){
    curr.next = curr'next;
    if(curr.next != null){
        curr'.next = curr'.next.next
    }
    curr = curr.next
    curr' = curr'.next
}
```

Step 3

1. Problem Statement - Reverse Linked List

Problem Description

You are given a singly linked list having head node **A**. You have to reverse the linked list and return the head node of that reversed list.

NOTE: You have to do it **in-place** and in **one-pass**.

Problem Constraints

1 <= Length of linked list <= 10^5

Value of each node is within the range of a 32-bit integer.

Input Format

First and only argument is a linked-list node **A**.

Output Format

Return a linked-list node denoting the head of the reversed linked list.

Example Input

Input 1:

A = 1 -> 2 -> 3 -> 4 -> 5 -> NULL

Input 2:

A = 3 -> NULL

Example Output

Output 1:

5 -> 4 -> 3 -> 2 -> 1 -> NULL

Output 2:

3 -> NULL

Example Explanation

Explanation 1:

The linked list has 5 nodes. After reversing them, the list becomes : 5 -> 4 -> 3 -> 2 -> 1 -> NULL

Explanation 2:

The linked list consists of only a single node. After reversing it, the list becomes : 3 -> NULL

Actual Code

```
public class Solution {
  public ListNode reverseList(ListNode A) {
    ListNode curr = A;
    ListNode prev = null;
    while(curr != null){
      ListNode tmp = curr.next;
      curr.next = prev;
      prev = curr;
      curr = tmp;
```

```
    }
    A = prev;
    return A;
  }
}
```

2. Problem Statement - Insert in Linked List

Problem Description

You are given **A** which is the head of a linked list. Also given is the **value B** and **position C**. Complete the function that should insert a new node with the said value at the given position.

Notes:

- In case the position is more than length of linked list, simply insert the new node at the tail only.
- In case the pos is 0, simply insert the new node at head only.
- Follow 0-based indexing for the node numbering.

Problem Constraints

$0 <= \text{size of linked list} <= 10^5$

$1 <= \text{value of nodes} <= 10^9$

$1 <= B <= 10^9$

$0 <= C <= 10^5$

Input Format

The first argument **A** is the **head** of a linked list.

The second argument **B** is an integer which denotes the **value of the new node**

The third argument **C** is an integer which denotes the **position of the new node**

Output Format

Return the head of the linked list

Example Input

Input 1:

A = 1 -> 2

B = 3

C = 0

Input 2:

A = 1 -> 2

B = 3

C = 1

Example Output

Output 1:

3 -> 1 -> 2

Output 2:

1 -> 3 -> 2

Example Explanation

For Input 1:

The new node is add to the head of the linked list

For Input 2:

The new node is added after the first node of the linked list

Actual Code

```
public class Solution {
  public ListNode solve(ListNode A, int B, int C) {
    ListNode NN = new ListNode(B);
    if(A == null) return NN;
    if(C==0 && A != null){
      NN.next = A;
      A = NN;
      return NN;
    }
    ListNode tmp = A;
    for(int i =1; i<=C-1; i++){
      if(tmp != null){
        tmp = tmp.next;
      }
    }
    if(tmp != null){
      NN.next = tmp.next;
      tmp.next = NN;
    }else{
      ListNode tail = A; // Edge case: If position C is out of bounds, append at the end
      while(tail.next != null){
        tail = tail.next;
      }
      tail.next = NN;
    }
    return A;
  }
}
```

3. Problem Statement - Delete in Linked List

Problem Description

You are given the head of a linked list **A** and an integer **B**. **Delete the B-th node from the linked list.**

Note : Follow 0-based indexing for the node numbering.

Problem Constraints

1 <= size of linked list <= 10^5

1 <= value of nodes <= 10^9

0 <= B < size of linked list

Input Format

The first argument A is the head of a linked list.

The second arguement B is an integer.

Output Format

Return the head of the linked list after deletion

Example Input

Input 1:

A = 1 -> 2 -> 3

B = 1

Input 2:

A = 4 -> 3 -> 2 -> 1

B = 0

Example Output

Output 1:

1 -> 3

Output 2:

3 -> 2 -> 1

Example Explanation

For Input 1:

The linked list after deletion is 1 -> 3.

For Input 2:

The linked list after deletion is 3 -> 2 -> 1.

Actual Code

```
public class Solution {
  public ListNode solve(ListNode A, int B) {
    ListNode tmp = A;
    if( B==0 && A != null){
      return A.next;
    }
    for(int i =1; i <= B-1; i++){
```

```
        if(tmp == null || tmp.next ==null){
           return A;
        }
        tmp = tmp.next;
      }
      if(tmp.next != null){
        tmp.next = tmp.next.next;
      }
      return A;

   }
}
```

4.Problem Statement – Copy List

Problem Description

- You are given a **linked list A**
- Each node in the linked list contains two pointers: a **next** pointer and a **random** pointer
- The **next** pointer points to the **next node** in the **list**
- The **random** pointer can point to **any node** in the **list**, or it can be **NULL**
- Your task is to create a **deep copy** of the **linked list A**
- The copied list should be a completely separate linked list from the original list, but with the same **node values** and **random** pointer connections as the original list
- You should create a **new linked list B**, where each **node** in **B** has the same value as the corresponding **node** in **A**
- The **next** and **random** pointers of each **node** in **B** should point to the corresponding **nodes** in **B** (***rather than A***)

Problem Constraints

$0 <= |A| <= 10^6$

Input Format

The first argument of input contains a pointer to the head of linked list **A**.

Output Format

Return a pointer to the head of the required linked list.

Example Input

Given list

1 -> 2 -> 3

with random pointers going from

1 -> 3

2 -> 1

3 -> 1

Example Output

1 -> 2 -> 3

with random pointers going from

1 -> 3

2 -> 1

3 -> 1

Actual Code

```
public class Solution {
   public RandomListNode copyRandomList(RandomListNode head) {
      if(head ==null) return null;
      RandomListNode curr = head;
      while(curr != null){
         RandomListNode X = new RandomListNode(curr.label);
         X.next = curr.next;          // First we will connect X.next to curr.next so that we don't lose
reference of the LL
         curr.next = X;
         curr = X.next;
      }
      curr = head;
      while(curr != null){
         if(curr.random != null){
            curr.next.random = curr.random.next;
         }
         curr = curr.next.next;

      }
      curr = head;
      RandomListNode curr1head = head.next;
      RandomListNode curr1 = curr1head;
      while(curr != null){
         curr.next = curr.next.next;          // when you are at the last node of curr, curr will jump twice and
points to null
         if(curr1.next != null){
            curr1.next = curr1.next.next;    // curr' cannot jump twice. So last jump should be done
carefully.
         }
         curr = curr.next;
         curr1 = curr1.next;
      }
      return curr1head;
   }
```

}

5.Problem Statement – Remove Duplicates from Sorted List

Problem Description

Given a **sorted** linked list, delete all duplicates such that each element appears only once.

Problem Constraints

0 <= length of linked list <= 10^6

Input Format

First argument is the head pointer of the linked list.

Output Format

Return the head pointer of the linked list after removing all duplicates.

Example Input

Input 1:

1->1->2

Input 2:

1->1->2->3->3

Example Output

Output 1:

1->2

Output 2:

1->2->3

Example Explanation

Explanation 1:

Each element appear only once in 1->2.

Actual Code

```
public class Solution {
  public ListNode deleteDuplicates(ListNode A) {
    ListNode tmp = A;
    while(tmp != null && tmp.next !=null){
      if(tmp.val == tmp.next.val){
        tmp.next = tmp.next.next;
      }else{
        tmp = tmp.next;
      }
    }
    return A;
  }
}
```

6. Problem Statement - Remove Nth Node from List End

Problem Description

Given a linked list A, remove the B-th node from the end of the list and return its head.

For example, given linked list: 1->2->3->4->5, and B = 2.

After removing the second node from the end, the linked list becomes 1->2->3->5.

NOTE: If B is greater than the size of the list, remove the first node of the list.

Try doing it using constant additional space.

Problem Constraints

$1 <= |A| <= 10^6$

Input Format

The first argument of input contains a pointer to the head of the linked list. The second argument of input contains the integer B.

Output Format

Return the head of the linked list after **deleting the B-th element from the end.**

Example Input

Input 1:

A = 1->2->3->4->5

B = 2

Input 2:

A = 1

B = 1

Example Output

Output 1: 1->2->3->5

Output 2:

Example Explanation

Explanation 1:

In the first example, 4 is the second last element.

Explanation 2:

In the second example, 1 is the first and the last element.

Actual Code

```
public class Solution {
  public ListNode removeNthFromEnd(ListNode A, int B) {
    if (A == null) return null;

    ListNode curr = A;
    int size =0;
    while(curr != null){
      size++;
      curr = curr.next;
    }
    int cnt =0;
    ListNode tmp = A;
    if(size <= B){        // If B is greater than the size of the list, remove the first node of the list
```

```
        return A.next;
      }
      curr = A;
      ListNode prev = curr;
      while (curr != null){
        cnt++;
        if (cnt == (size - B+1)){
          prev.next = curr.next;
          return A;
        }
        prev = curr;
        curr = curr.next;
      }
      return A;
    }
}
```

7. Problem Statement – Reverse Link List II

Problem Description

Reverse a linked list **A** from position **B** to **C**.

NOTE: Do it in-place and in one-pass.

Problem Constraints

$1 <= |A| <= 10^6$

$1 <= B <= C <= |A|$

Input Format

The first argument contains a pointer to the head of the given linked list, A.

The second arugment contains an integer, B.

The third argument contains an integer C.

Output Format

Return a pointer to the head of the modified linked list.

Example Input

Input 1:

A = 1 -> 2 -> 3 -> 4 -> 5

B = 2

C = 4

Input 2:

A = 1 -> 2 -> 3 -> 4 -> 5

B = 1

C = 5

Example Output

Output 1: 1 -> 4 -> 3 -> 2 -> 5

Output 2: 5 -> 4 -> 3 -> 2 -> 1

Example Explanation

Explanation 1:

In the first example, we want to reverse the highlighted part of the given linked list : 1 -> **2 -> 3 -> 4** -> 5

Thus, the output is 1 -> 4 -> 3 -> 2 -> 5

Explanation 2:

In the second example, we want to reverse the highlighted part of the given linked list : **1 -> 4 -> 3 -> 2 -> 5**

Thus, the output is 5 -> 4 -> 3 -> 2 -> 1

Actual Code

```
public class Solution {
  public ListNode reverseBetween(ListNode A, int B, int C) {
    ListNode curr = A, from = null, to = null, first = null, last = null;
    int cnt =0;
    while(curr != null){
      cnt +=1;
      if(cnt < B){
        first = curr;
      }
      if(cnt ==B){
        from = curr;
      }
      if(cnt ==C){
        to = curr;
        last = to.next;
        break;
      }
      curr = curr.next;
    }
    to.next = null;    //Detach the Segment to Reverse
    reverse(from);
    if (first != null){
      first.next = to;
    }else{
      A = to;
    }
    from.next = last;
    return A;
  }
```

```
  private void reverse(ListNode from){
    ListNode curr = from.next;
    ListNode prev = from;
    while (curr !=null){
      ListNode nxt = curr.next;
      curr.next = prev;
      prev = curr;
      curr = nxt;
    }

  }
}
```

8.Problem Statement - K reverse linked list - Unsolved

Problem Description

Given a singly linked list **A** and an integer **B**, reverse the nodes of the list **B** at a time and return the modified linked list.

Problem Constraints

$1 <= |A| <= 10^3$

B always divides A

Input Format

The first argument of input contains a pointer to the head of the linked list.

The second arugment of input contains the integer, B.

Output Format

Return a pointer to the head of the modified linked list.

Example Input

Input 1:

A = [1, 2, 3, 4, 5, 6]

B = 2

Input 2:

A = [1, 2, 3, 4, 5, 6]

B = 3

Example Output

Output 1: [2, 1, 4, 3, 6, 5]

Output 2: [3, 2, 1, 6, 5, 4]

Example Explanation

Explanation 1:

For the first example, the list can be reversed in groups of 2.

[[1, 2], [3, 4], [5, 6]]

After reversing the K-linked list

[[2, 1], [4, 3], [6, 5]]

Explanation 2:

For the second example, the list can be reversed in groups of 3.

[[1, 2, 3], [4, 5, 6]]

After reversing the K-linked list

[[3, 2, 1], [6, 5, 4]]

Actual Code

```
public class Solution {
  public ListNode reverseList(ListNode A, int B) {
    ListNode curr = A;
    ListNode prev = null;
    for(int i =1; i <=B-1; i++){
      prev=curr;
      curr= curr.next;
      reverse(prev, curr);
    }
    return A.next;
  }
  private void reverse(ListNode prev, ListNode curr){
    while(curr !=null){

      ListNode tmp = curr.next;
      curr.next = prev;
      prev = curr;
      curr = tmp;
    }
  }
}
```