Assignment 14 Solutions

Q1 Given a linked list of **N** nodes such that it may contain a loop.

A loop here means that the last node of the link list is connected to the node at position X(1-based index). If the link list does not have any loop, X=0.

Remove the loop from the linked list, if it is present, i.e. unlink the last node which is forming the loop.

Example 1:

Input:

N = 3

value[] = {1,3,4}

X = 2

Output:1

Explanation:The link list looks like

1 -> 3 -> 4 ^ | |____|

A loop is present. If you remove it

successfully, the answer will be 1.

Example 2:

Input:

N = 4

value[] = {1,8,3,4}

X = 0

Output:1

Explanation:The Linked list does not

contains any loop.

Example 3:

Input:

N = 4

value[] = {1,2,3,4}

X = 1

Output:1

Explanation:The link list looks like

1 -> 2 -> 3 -> 4 ^ | |_____________|

A loop is present.

If you remove it successfully,

the answer will be 1.

In [50]:
```python
# Example 1
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def removeLoop(head):
    if not head or not head.next:
```

```
        return head

    slowPtr = head
    fastPtr = head
    loopExists = False

    while fastPtr.next and fastPtr.next.next:
        slowPtr = slowPtr.next
        fastPtr = fastPtr.next.next
        if slowPtr == fastPtr:
            loopExists = True
            break

    if not loopExists:
        return head

    ptr1 = head
    while ptr1.next != slowPtr.next:
        ptr1 = ptr1.next
        slowPtr = slowPtr.next

    slowPtr.next = None

    return head
```

In [51]:
```
def createLinkedList(values):
    if not values:
        return None

    head = Node(values[0])
    curr = head
    loopNode = None

    for i in range(1, len(values)):
        curr.next = Node(values[i])
        curr = curr.next
        if i == X - 1:
            loopNode = curr

    curr.next = loopNode

    return head


N = 3
values = [1, 3, 4]
X = 2

head = createLinkedList(values)

head = removeLoop(head)

while head:
    print(head.data, end=" ")
    head = head.next
```

```
1 3 4
```

In [52]:
```
# Example 2
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def removeLoop(head):
    if not head or not head.next:
        return head

    slowPtr = head
    fastPtr = head
    loopExists = False

    while fastPtr.next and fastPtr.next.next:
        slowPtr = slowPtr.next
        fastPtr = fastPtr.next.next
        if slowPtr == fastPtr:
            loopExists = True
            break

    if not loopExists:
        return head

    ptr1 = head
    while ptr1.next != slowPtr.next:
```

```
        ptr1 = ptr1.next
        slowPtr = slowPtr.next

    slowPtr.next = None

    return head
```

In [53]:
```
def createLinkedList(values):
    if not values:
        return None

    head = Node(values[0])
    curr = head

    for i in range(1, len(values)):
        curr.next = Node(values[i])
        curr = curr.next

    return head


N = 4
values = [1, 8, 3, 4]
X = 0

head = createLinkedList(values)

head = removeLoop(head)

while head:
    print(head.data, end=" ")
    head = head.next
```

```
1 8 3 4
```

In [54]:
```
# Example 3
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def removeLoop(head):
    if not head or not head.next:
        return head

    slowPtr = head
    fastPtr = head
    loopExists = False

    while fastPtr.next and fastPtr.next.next:
        slowPtr = slowPtr.next
        fastPtr = fastPtr.next.next
        if slowPtr == fastPtr:
            loopExists = True
            break

    if not loopExists:
        return head

    ptr1 = head
    while ptr1.next != slowPtr.next:
        ptr1 = ptr1.next
        slowPtr = slowPtr.next

    slowPtr.next = None

    return head
```

In [55]:
```
def createLinkedList(values):
    if not values:
        return None

    head = Node(values[0])
    curr = head
    loopNode = None

    for i in range(1, len(values)):
        curr.next = Node(values[i])
        curr = curr.next
        if i == X - 1:
            loopNode = curr

    curr.next = loopNode
```

```
    return head


# Given inputs
N = 4
values = [1, 2, 3, 4]
X = 1

head = createLinkedList(values)

head = removeLoop(head)

while head:
    print(head.data, end=" ")
    head = head.next
```

```
1 2 3 4
```

Q2 A number **N** is represented in Linked List such that each digit corresponds to a node in linked list. You need to add 1 to it.

Example 1:

Input:

LinkedList: 4->5->6

Output:457

Example 2:

Input:

LinkedList: 1->2->3

Output:124

In [56]:
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def addOne(head):
    prev = None
    curr = head
    while curr:
        next_node = curr.next
        curr.next = prev
        prev = curr
        curr = next_node
    head = prev

    carry = 1
    curr = head
    while curr:
        sum = curr.data + carry
        curr.data = sum % 10
        carry = sum // 10
        if carry == 0:
            break
        curr = curr.next

    prev = None
    while head:
        next_node = head.next
        head.next = prev
        prev = head
        head = next_node
    head = prev

    return head
```

In [57]:
```
def printLinkedList(head):
    curr = head
    while curr:
        print(curr.data, end=" ")
        curr = curr.next
    print()
```

In [58]:
```
def createLinkedList(values):
```

```
    if not values:
        return None

    head = Node(values[0])
    curr = head

    for i in range(1, len(values)):
        curr.next = Node(values[i])
        curr = curr.next

    return head

# Example 1:
N1 = 3
values1 = [4, 5, 6]

head1 = createLinkedList(values1)

result1 = addOne(head1)

printLinkedList(result1)

# Example 2:
N2 = 3
values2 = [1, 2, 3]

head2 = createLinkedList(values2)

result2 = addOne(head2)

printLinkedList(result2)
```

```
4 5 7
1 2 4
```

Q3 Given a Linked List of size N, where every node represents a sub-linked-list and contains two pointers:(i) a **next** pointer to the next node,(ii) a **bottom** pointer to a linked list where this node is head.Each of the sub-linked-list is in sorted order.Flatten the Link List such that all the nodes appear in a single level while maintaining the sorted order. **Note:** The flattened list will be printed using the bottom pointer instead of next pointer.

Example 1:

Input:

5 -> 10 -> 19 -> 28 | | | | 7 20 22 35 | | | 8 50 40 | | 30 45

Output: 5-> 7-> 8- > 10 -> 19-> 20->

22-> 28-> 30-> 35-> 40-> 45-> 50.

Explanation:

The resultant linked lists has every

node in a single level.(Note:| represents the bottom pointer.)

Example 2:

Input:

5 -> 10 -> 19 -> 28 | | 7 22 | | 8 50 | 30

Output: 5->7->8->10->19->22->28->30->50

Explanation:

The resultant linked lists has every

node in a single level.

(Note:| represents the bottom pointer.)

In [59]:
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
```

```
        self.bottom = None

def mergeLists(list1, list2):
    if not list1:
        return list2
    if not list2:
        return list1

    merged = None

    if list1.data <= list2.data:
        merged = list1
        merged.bottom = mergeLists(list1.bottom, list2)
    else:
        merged = list2
        merged.bottom = mergeLists(list1, list2.bottom)

    return merged

def flattenLinkedList(head):
    if not head or not head.next:
        return head

    # Recursively flatten the rest of the linked list
    head.next = flattenLinkedList(head.next)

    # Merge the current list with the flattened list
    head = mergeLists(head, head.next)

    return head
```

In [60]:
```
def printLinkedList(head):
    curr = head
    while curr:
        print(curr.data, end="->")
        curr = curr.bottom
    print("NULL")
```

In [61]:
```
def createLinkedList(values):
    if not values:
        return None

    main_head = Node(values[0])
    main_curr = main_head

    for i in range(1, len(values)):
        sub_head = Node(values[i])
        main_curr.bottom = sub_head
        main_curr = main_curr.bottom

    return main_head

# Example 1:
N1 = 4
values1 = [5, 10, 19, 28]
sub_values1 = [[7, 20, 22, 35], [8, 50, 40], [30, 45]]

# Create the linked list with sub-linked lists
head1 = createLinkedList(values1)
curr1 = head1
for sublist in sub_values1:
    curr1.bottom = createLinkedList(sublist)
    curr1 = curr1.bottom

# Flatten the linked list
result1 = flattenLinkedList(head1)

# Print the flattened linked list
printLinkedList(result1)


# Example 2:
N2 = 3
values2 = [5, 10, 19]
sub_values2 = [[7, 22], [8, 50], [30]]

# Create the linked list with sub-linked lists
head2 = createLinkedList(values2)
curr2 = head2
for sublist in sub_values2:
    curr2.bottom = createLinkedList(sublist)
    curr2 = curr2.bottom
```

```
# Flatten the linked list
result2 = flattenLinkedList(head2)

# Print the flattened linked list
printLinkedList(result2)
```

```
5->7->8->30->45->NULL
5->7->8->30->NULL
```

Q4

You are given a special linked list with **N** nodes where each node has a next pointer pointing to its next node. You are also given **M** random pointers, where you will be given **M** number of pairs denoting two nodes **a** and **b** **i.e. a->arb = b** (arb is pointer to random node)**.**

Construct a copy of the given list. The copy should consist of exactly **N** new nodes, where each new node has its value set to the value of its corresponding original node. Both the next and random pointer of the new nodes should point to new nodes in the copied list such that the pointers in the original list and copied list represent the same list state. None of the pointers in the new list should point to nodes in the original list.

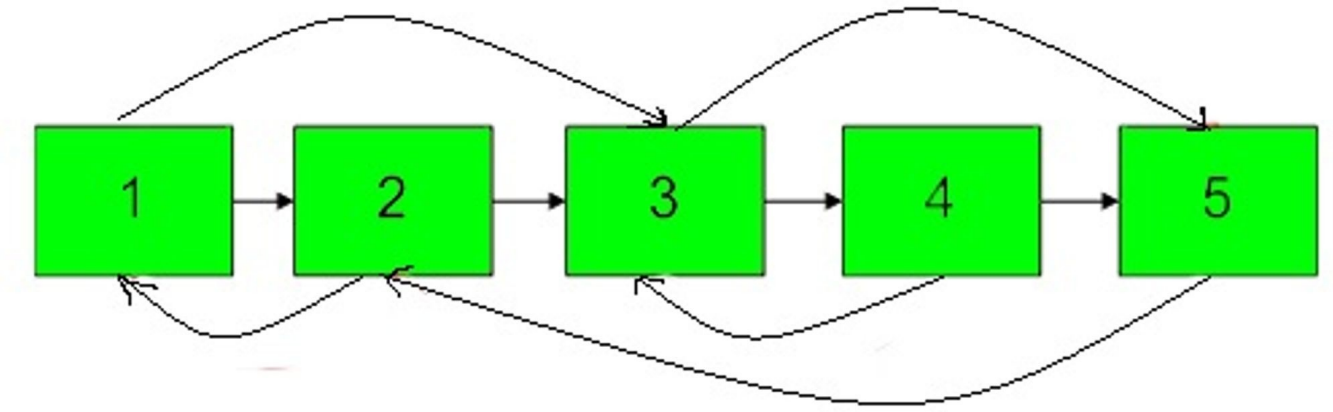
For example, if there are two nodes **X** and **Y** in the original list, where **X.arb --> Y**, then for the corresponding two nodes **x** and **y** in the copied list, **x.arb --> y.**

Return the head of the copied linked list.

Note :- The diagram isn't part of any example, it just depicts an example of how the linked list may look like.

In [62]:
```
from IPython import display
display.Image(r"C:\Users\hrush\OneDrive\Pictures\Saved Pictures\clone.jpg")
```

Out[62]:



Example 1:

Input:

N = 4, M = 2

value = {1,2,3,4}

pairs = {{1,2},{2,4}}

Output:1

Explanation:In this test case, there

are 4 nodes in linked list. Among these

4 nodes, 2 nodes have arbitrary pointer

set, rest two nodes have arbitrary pointer

as NULL. Second line tells us the value

of four nodes. The third line gives the

information about arbitrary pointers.

The first node arbitrary pointer is set to

node 2. The second node arbitrary pointer

is set to node 4.

Input:

N = 4, M = 2

value = {1,2,3,4}

pairs = {{1,2},{2,4}}

Output:1

Explanation:In this test case, there

are 4 nodes in linked list. Among these

4 nodes, 2 nodes have arbitrary pointer

set, rest two nodes have arbitrary pointer

as NULL. Second line tells us the value

of four nodes. The third line gives the

information about arbitrary pointers.

The first node arbitrary pointer is set to

node 2. The second node arbitrary pointer

is set to node 4.

Example 2:

Input:

N = 4, M = 2

value[] = {1,3,5,9}

pairs[] = {{1,1},{3,4}}

Output:1

Explanation:In the given testcase ,

applying the method as stated in the

above example, the output will be 1.

In [63]:
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.random = None

def copyRandomList(head):
    if not head:
        return None

    mapping = {}

    curr = head
    while curr:
        mapping[curr] = Node(curr.data)
        curr = curr.next

    curr = head
    while curr:
        new_node = mapping[curr]
        new_node.next = mapping.get(curr.next)
        new_node.random = mapping.get(curr.random)
        curr = curr.next

    return mapping[head]

def printLinkedList(head):
    curr = head
```

```
    while curr:
        print("Value:", curr.data, end=" ")
        if curr.random:
            print("Random:", curr.random.data, end=" ")
        else:
            print("Random: None", end=" ")
        print()
        curr = curr.next

def createLinkedList(values, pairs):
    if not values:
        return None

    nodes = {}
    head = Node(values[0])
    curr = head
    nodes[values[0]] = curr

    for i in range(1, len(values)):
        curr.next = Node(values[i])
        curr = curr.next
        nodes[values[i]] = curr

    for pair in pairs:
        if pair[0] in nodes and pair[1] in nodes:
            nodes[pair[0]].random = nodes[pair[1]]

    return head

# Example 1:
N1 = 4
M1 = 2
values1 = [1, 2, 3, 4]
pairs1 = [[1, 2], [2, 4]]

head1 = createLinkedList(values1, pairs1)

result1 = copyRandomList(head1)

printLinkedList(result1)


# Example 2:
N2 = 4
M2 = 2
values2 = [1, 3, 5, 9]
pairs2 = [[1, 1], [3, 4]]

head2 = createLinkedList(values2, pairs2)

result2 = copyRandomList(head2)

printLinkedList(result2)
```

```
Value: 1 Random: 2
Value: 2 Random: 4
Value: 3 Random: None
Value: 4 Random: None
Value: 1 Random: 1
Value: 3 Random: None
Value: 5 Random: None
Value: 9 Random: None
```

Q5

Given the `head` of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*.

The **first** node is considered **odd**, and the **second** node is **even**, and so on.

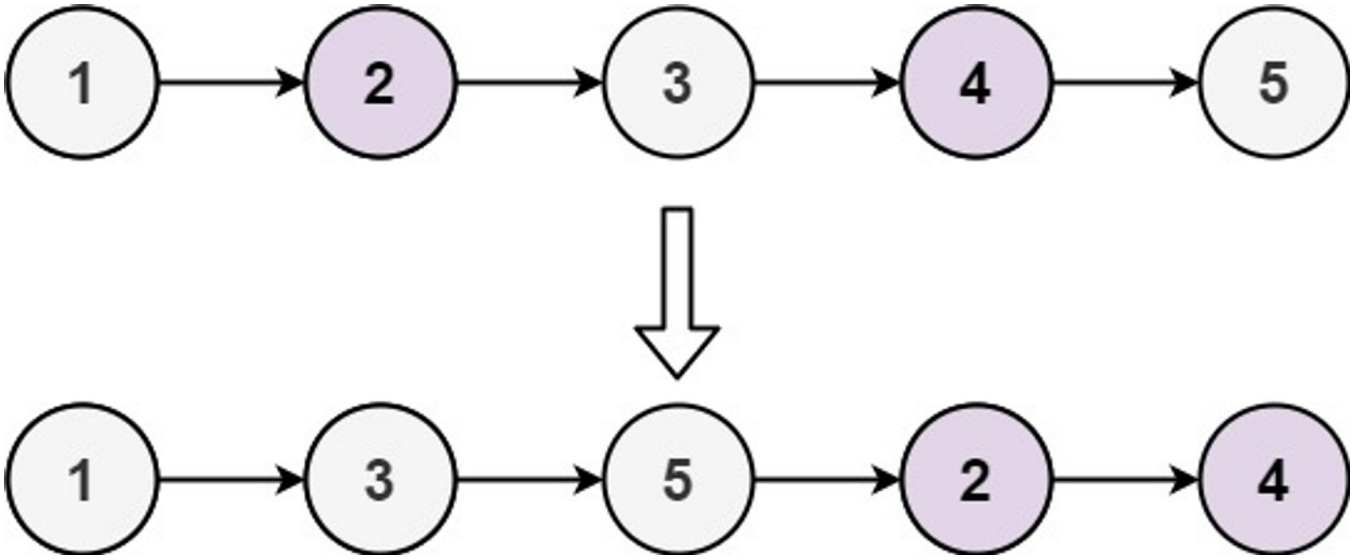
Note that the relative order inside both the even and odd groups should remain as it was in the input.

You must solve the problem in `O(1)` extra space complexity and `O(n)` time complexity.

Example 1:

In [64]:
```
from IPython import display
display.Image(r"C:\Users\hrush\OneDrive\Pictures\Saved Pictures\oddeven-linked-list.jpg")
```

Out[64]:



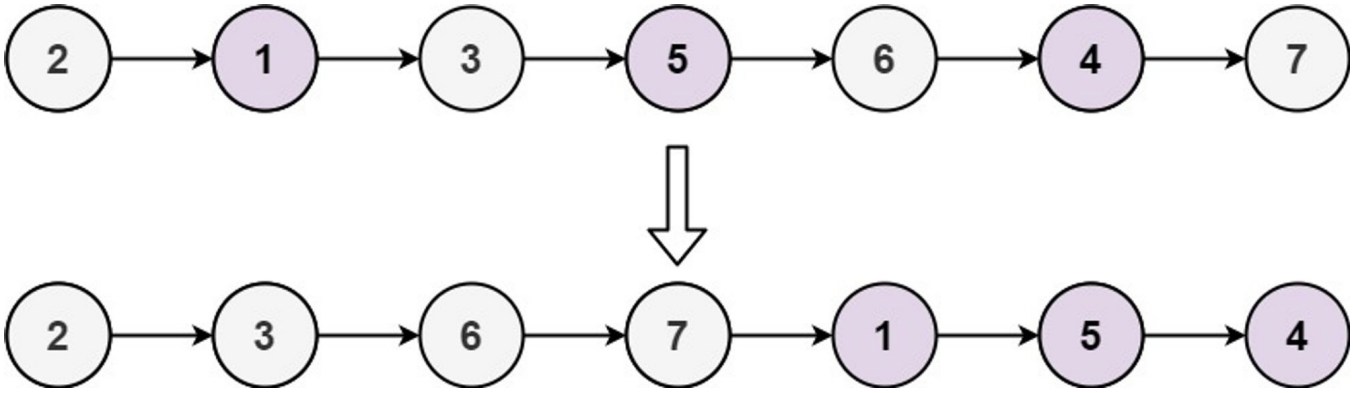
Input: head = [1,2,3,4,5]

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

Example 2:

In [65]:
```
from IPython import display
display.Image(r"C:\Users\hrush\OneDrive\Pictures\Saved Pictures\oddeven2-linked-list 1.jpg")
```

Out[65]:



Input: head = [2,1,3,5,6,4,7]

Output: [2,3,6,7,1,5,4]

In [66]:
```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def oddEvenList(head):
    if not head or not head.next:
        return head

    oddHead = head
    evenHead = head.next
    oddTail = oddHead
    evenTail = evenHead

    curr = evenHead.next
    isOdd = True

    while curr:
        if isOdd:
            oddTail.next = curr
            oddTail = oddTail.next
        else:
            evenTail.next = curr
            evenTail = evenTail.next

        curr = curr.next
        isOdd = not isOdd

    evenTail.next = None

    oddTail.next = evenHead
```

```
    return oddHead
```

In [67]:
```
def createLinkedList(values):
    if not values:
        return None

    head = ListNode(values[0])
    curr = head

    for val in values[1:]:
        curr.next = ListNode(val)
        curr = curr.next

    return head
```

In [68]:
```
def linkedListToList(head):
    result = []
    curr = head

    while curr:
        result.append(curr.val)
        curr = curr.next

    return result

# Example 1
head1 = createLinkedList([1, 2, 3, 4, 5])
reordered1 = oddEvenList(head1)
print(linkedListToList(reordered1))

# Example 2
head2 = createLinkedList([2, 1, 3, 5, 6, 4, 7])
reordered2 = oddEvenList(head2)
print(linkedListToList(reordered2))
```

```
[1, 3, 5, 2, 4]
[2, 3, 6, 7, 1, 5, 4]
```

Q6

Given a singly linked list of size **N**. The task is to **left-shift** the linked list by **k** nodes, where **k** is a given positive integer smaller than or equal to length of the linked list.

Example 1:

Input:

N = 5

value[] = {2, 4, 7, 8, 9}

k = 3

Output:8 9 2 4 7

Explanation:Rotate 1:4 -> 7 -> 8 -> 9 -> 2

Rotate 2: 7 -> 8 -> 9 -> 2 -> 4

Rotate 3: 8 -> 9 -> 2 -> 4 -> 7

Example 2:

Input:

N = 8

value[] = {1, 2, 3, 4, 5, 6, 7, 8}

k = 4

Output:5 6 7 8 1 2 3 4

In [69]:
```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def leftShift(head, k):
```

```
    if not head or not head.next or k == 0:
        return head

    length = 0
    curr = head

    while curr:
        length += 1
        curr = curr.next

    actualShift = k % length

    if actualShift == 0:
        return head

    curr = head
    for _ in range(actualShift):
        curr = curr.next

    newHead = curr
    prev = None

    while curr.next:
        curr = curr.next

    curr.next = head

    for _ in range(length - actualShift):
        prev = head
        head = head.next

    prev.next = None

    return newHead
```

In [70]:
```
def createLinkedList(values):
    if not values:
        return None

    head = ListNode(values[0])
    curr = head

    for val in values[1:]:
        curr.next = ListNode(val)
        curr = curr.next

    return head
```

In [71]:
```
def linkedListToList(head):
    result = []
    curr = head

    while curr:
        result.append(curr.val)
        curr = curr.next

    return result

# Example 1
head1 = createLinkedList([2, 4, 7, 8, 9])
shifted1 = leftShift(head1, 3)
print(linkedListToList(shifted1))
# Example 2
head2 = createLinkedList([1, 2, 3, 4, 5, 6, 7, 8])
shifted2 = leftShift(head2, 4)
print(linkedListToList(shifted2))
```

```
[8, 9, 2, 4]
[5, 6, 7, 8, 1, 2, 3, 4]
```

Q7

You are given the `head` of a linked list with `n` nodes.

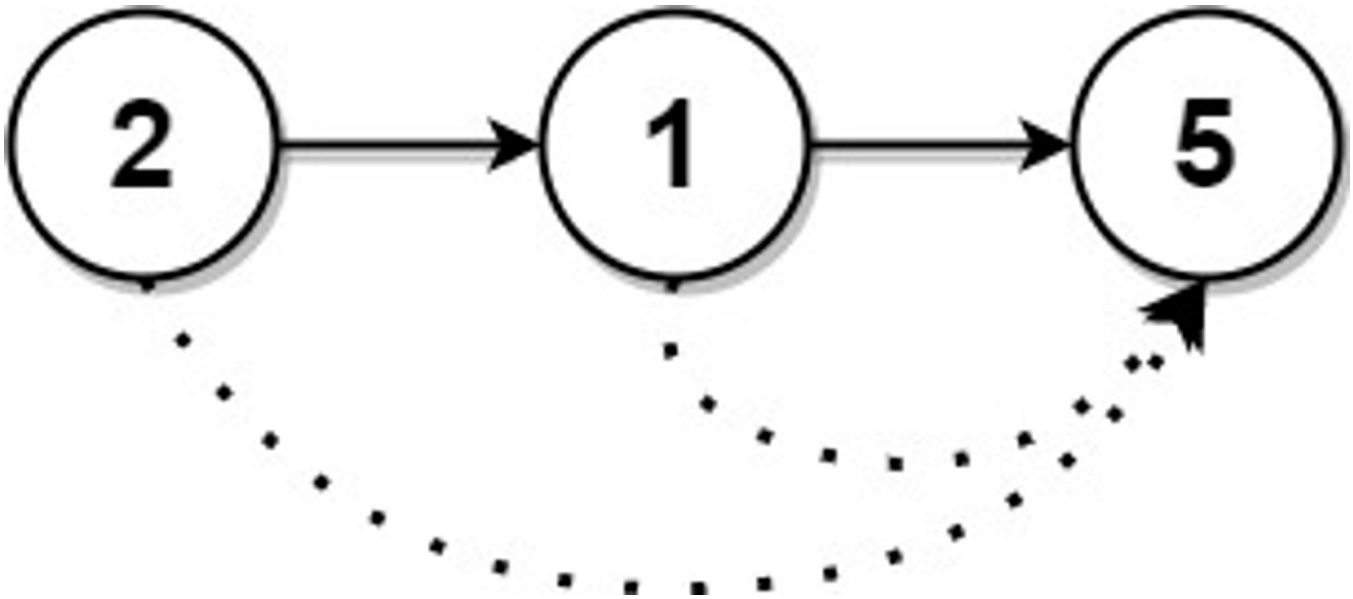
For each node in the list, find the value of the **next greater node**. That is, for each node, find the value of the first node that is next to it and has a **strictly larger** value than it.

Return an integer array `answer` where `answer[i]` is the value of the next greater node of the `ith` node (**1-indexed**). If the `ith` node does not have a next greater node, set `answer[i] = 0`.

Example 1:

In [72]:
```
from IPython import display
display.Image(r"C:\Users\hrush\OneDrive\Pictures\Saved Pictures\linkedlistnext1.jpg")
```

Out[72]:



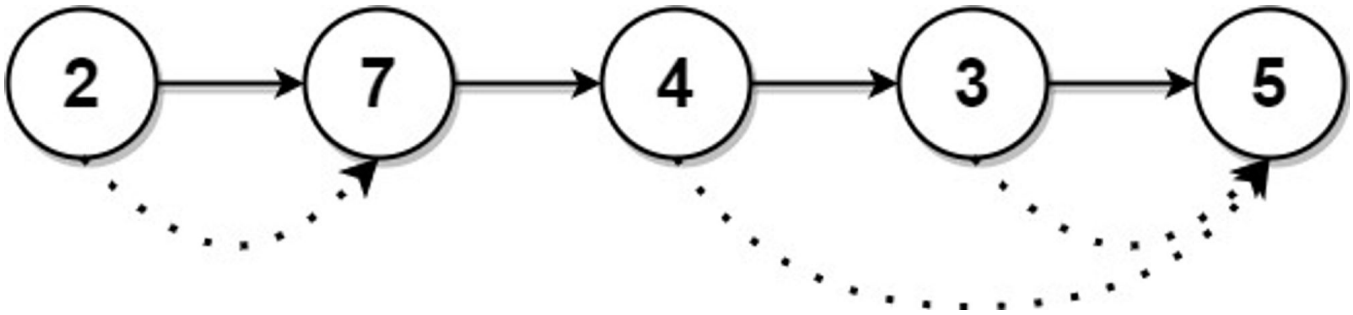
Input: head = [2,1,5]

Output: [5,5,0]

Example 2:

In [73]:
```
from IPython import display
display.Image(r"C:\Users\hrush\OneDrive\Pictures\Saved Pictures\linkedlistnext2.jpg")
```

Out[73]:



Input: head = [2,7,4,3,5]

Output: [7,0,5,5,0]

In [74]:
```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def nextLargerNodes(head):

    arr = []
    curr = head
    while curr:
        arr.append(curr.val)
        curr = curr.next

    n = len(arr)
    result = [0] * n
    stack = []

    for i in range(n - 1, -1, -1):
        while stack and stack[-1] <= arr[i]:
            stack.pop()

        if stack:
            result[i] = stack[-1]

        stack.append(arr[i])

    return result
```

In [75]:
```
def createLinkedList(values):
    if not values:
```

```
        return None

    head = ListNode(values[0])
    curr = head

    for val in values[1:]:
        curr.next = ListNode(val)
        curr = curr.next

    return head
```

In [76]:
```
# Example 1
head1 = createLinkedList([2, 1, 5])
print(nextLargerNodes(head1))

# Example 2
head2 = createLinkedList([2, 7, 4, 3, 5])
print(nextLargerNodes(head2))
```

```
[5, 5, 0]
[7, 0, 5, 5, 0]
```

Q8

Given the `head` of a linked list, we repeatedly delete consecutive sequences of nodes that sum to `0` until there are no such sequences.

After doing so, return the head of the final linked list. You may return any such answer.

(Note that in the examples below, all sequences are serializations of `ListNode` objects.)

Example 1:

Input: head = [1,2,-3,3,1]

Output: [3,1]

Note: The answer [1,2,1] would also be accepted.

Example 2:

Input: head = [1,2,3,-3,4]

Output: [1,2,4]

Example 3:

Input: head = [1,2,3,-3,-2]

Output: [1]

In [77]:
```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def removeZeroSumSublists(head):
    dummy = ListNode(0)
    dummy.next = head
    prefix_sum = 0
    prefix_sum_map = {0: dummy}

    while head:
        prefix_sum += head.val

        if prefix_sum in prefix_sum_map:
            prev = prefix_sum_map[prefix_sum]
            start = prev.next
            curr_sum = prefix_sum

            while start != head:
                curr_sum += start.val
                prefix_sum_map.pop(curr_sum)
                start = start.next

            prev.next = head.next
        else:
            prefix_sum_map[prefix_sum] = head

        head = head.next
```

```
    return dummy.next
```

In [78]:
```
def createLinkedList(values):
    if not values:
        return None

    head = ListNode(values[0])
    curr = head

    for val in values[1:]:
        curr.next = ListNode(val)
        curr = curr.next

    return head
```

In [79]:
```
def linkedListToList(head):
    result = []
    curr = head

    while curr:
        result.append(curr.val)
        curr = curr.next

    return result

# Example 1
head1 = createLinkedList([1, 2, -3, 3, 1])
result1 = removeZeroSumSublists(head1)
print(linkedListToList(result1))

# Example 2
head2 = createLinkedList([1, 2, 3, -3, 4])
result2 = removeZeroSumSublists(head2)
print(linkedListToList(result2))

# Example 3
head3 = createLinkedList([1, 2, 3, -3, -2])
result3 = removeZeroSumSublists(head3)
print(linkedListToList(result3))
```

```
[3, 1]
[1, 2, 4]
[1]
```

In [ ]: