Assignment-14

22 June 2023 22:09

💡 **Question 1**

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.

```
class Solution:
    #Function to remove a loop in the linked list.
    def removeLoop(self, head):
        # code here
        # remove the loop without losing any nodes
        slow=fast=head
        while(fast!=None and fast.next!=None):
            slow=slow.next
            fast=fast.next.next
            if(slow==fast):
                slow=head
                if slow==fast:
                    while(fast.next!=slow):
                        fast=fast.next
                else:
                    while(slow.next!=fast.next):
                        slow=slow.next
                        fast=fast.next
                fast.next=None
```

💡 **Question 2**

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

```
class Solution:
    def addOne(self,head):
        #Returns new head of linked List.
        temp=head
        res=""
        while temp:
          res=res+str(temp.data)
          temp=temp.next
        res=int(res)+1
        res=str(res)
        dummy=Node(0)
        cur=dummy
        i=0
        while i<len(res):
          cur.next=Node(res[i])
          cur=cur.next
          i+=1
        return dummy.next
```

💡 **Question 3**

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.)

```
def flatten(root):
    #Your code here
    a = []
    l = Node(-1)
    while root:
        a.append(root.data)
        cur = root.bottom
        while cur :
          a.append(cur.data)
          cur= cur.bottom


        root = root.next
    a.sort()
    c = l
    for i in a:
        c.bottom = Node(i)
        c = c.bottom
    return l.bottom
```

💡 **Question 4**

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.

Example1:

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.

```
class Solution:
    #Function to clone a linked list with next and random pointer.
    def copyList(self, head):
        h=Node(-1)
        h2=h
        h1=head
        while h1:
            temp=Node(h1.data)
            h2.next = temp
            h2=temp
            h1=h1.next

        original = head
        clone = h.next
        m = {None:None}
        while original and clone:
            m[original]=clone
            original = original.next
            clone = clone.next

        original = head
        clone = h.next

        while original and clone:
            clone.arb=m[original.arb]
            clone=clone.next
            original = original.next

        return h.next
```

💡 **Question 5**

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:

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

```
class Solution:
    def oddEvenList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if head == None or head.next == None or head.next.next == None:
            return head

        odd,even = head, head.next
        pointer1,pointer2 = odd,even
        prev = None
        while(pointer1 != None and pointer2 != None):
            pointer1.next = pointer2.next
            prev = pointer1
            pointer1 = pointer1.next
            if pointer1 == None:
                pointer2.next = None
            else:
                pointer2.next = pointer1.next
            pointer2 = pointer2.next
        if pointer1 == None:
            prev.next = even
        else:
            pointer1.next = even
        return odd
```

💡 **Question 6**

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

```
class Solution:
    # Function to rotate a linked list.
    def rotate(self, head, k):
        # Helper function to calculate the length of the linked list.
        def get_length(root):
            length = 0
            while root:
                length += 1
                root = root.next
            return length

        # Get the length of the linked list.
        length = get_length(head)

        # Check if rotation is not needed.
        if k == 0  or not head or head.next is None:
            return head
        else:
            k = k % length
            node = head

            # Find the node at the new head position after rotation.
            for _ in range(k):
                node = node.next

            cur = head

            # Traverse to the last node of the original list.
            while cur.next:
                cur = cur.next

            # Connect the last node to the original head to form a circular list.
            cur.next = head

            # Update the new head position.
            head = node
```

```
# Find the node at the new tail position after rotation.
for _ in range(length-1):
    head = head.next

# Set the next pointer of the new tail to None to break the circular list.
head.next = None
return node
```

💡 **Question 7**

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:

Input: head = [2,1,5]

Output: [5,5,0]

```
def nextLargerNodes(self, head: Optional[ListNode]) -> List[int]:
ans = []
stack = []
i = 0
curr = head
while(curr):
# just for the length of the linked list.
ans.append(0)
curr = curr.next
while(head):
while(stack and stack[-1][1] < head.val):
index, _ = stack.pop()
ans[index] = head.val
stack.append([i, head.val])
i += 1
head = head.next
return ans
```

💡 **Question 8**

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.

```
class Solution:
def removeZeroSumSublists(self, head: Optional[ListNode]) -> Optional[ListNode]:

dummy = ListNode(0,head)
pre = 0
dic = {0: dummy}

while head:
pre+=head.val
dic[pre] = head
head = head.next

head = dummy
pre = 0
while head:
pre+=head.val
head.next = dic[pre].next
head = head.next
```

```
        return dummy.next
```