Assignment-12

14 June 2023 14:52

Question 1

Given a singly linked list, delete **middle** of the linked list. For example, if given linked list is 1->2->**3**->4->5 then linked list should be modified to 1->2->4->5.If there are **even** nodes, then there would be **two middle** nodes, we need to delete the second middle element. For example, if given linked list is 1->2->3->4->5->6 then it should be modified to 1->2->3->5->6.If the input linked list is NULL or has 1 node, then it should return NULL

Example 1:

Input:

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

Output:1 2 4 5

Example 2:

Input:

LinkedList: 2->4->6->7->5->1

Output:2 4 6 5 1

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

        slow=head
        fast=head.next.next

        while fast and fast.next:
            slow=slow.next
            fast=fast.next.next

        slow.next=slow.next.next
        return head
```

Question 2

Given a linked list of **N** nodes. The task is to check if the linked list has a loop. Linked list can contain self loop.

Example 1:

```
Input:
N = 3
value[] = {1,3,4}
x(position at which tail is connected) = 2
Output:True
Explanation:In above test case N = 3.
The linked list with nodes N = 3 is
given. Then value of x=2 is given which
means last node is connected with xth
node of linked list. Therefore, there
exists a loop.
```

Example 2:

Input: N = 4 value[] = {1,8,3,4} x = 0 Output:False Explanation:For N = 4 ,x = 0 means then lastNode->next = NULL, then the Linked list does not contains any loop.

```
class Solution:
    def detectLoop(self, head):
        if head==None: return False
        map={}
        while(head):
            if head not in map:
                map[head]=1
                head=head.next
            else:
                return True
        return False
```

Question 3

Given a linked list consisting of **L** nodes and given a number **N**. The task is to find the **N**th node from the end of the linked list.

Example 1:

```
Input:
N = 2
LinkedList: 1->2->3->4->5->6->7->8->9
Output:8
Explanation:In the first example, there
are 9 nodes in linked list and we need
to find 2nd node from end. 2nd node
from end is 8.
```

Example 2:

Input:

N = 5

LinkedList: 10->5->100->5

Output:-1

Explanation:In the second example, there

are 4 nodes in the linked list and we

need to find 5th from the end. Since 'n'

is more than the number of nodes in the

linked list, the output is -1.

```
def getNthFromLast(head,n):
    first=head
    second=head

    for i in range(1,n):
        second=second.next
        if second==None: return -1

    while second.next:
        first=first.next
        second=second.next
    return first.data
```

💡 **Question 4**

Given a singly linked list of characters, write a function that returns true if the given list is a palindrome, else false.

Examples:

Input: R->A->D->A->R->NULL

Output: Yes

Input: C->O->D->E->NULL

Output: No

```
class Solution:
    def isPalindrome(self, head):
        if head==None or head.next==None:
            return True
        def reverse(head):
            newHead=None
            while head:
                next=head.next
                head.next=newHead
                newHead=head
                head=next
            return newHead

        slow=head
        fast=head
        while fast.next and fast.next.next:
            slow=slow.next
            fast=fast.next.next
        slow.next=reverse(slow.next)

        slow=slow.next

        while slow:
            if head.data!=slow.data:
```

```
            return False
        head=head.next
        slow=slow.next
    return True
```

💡 **Question 5**

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.

```
class Solution:
    #Function to remove a loop in the linked list.
    def removeLoop(self, head):
        if(head is None and head.next is None):
            return False



        slow = head
        fast = head
        interSectionNode = None

        #Step1: Fine inter Section Point
        while(fast is not None):

            slow = slow.next
            fast = fast.next
            if(fast is not None):
                fast = fast.next

            if(slow == fast):
                interSectionNode = slow
                break
        if(interSectionNode is not None):
            #Step2: Fine Loop Starting Point
            slow = head
            while(slow != interSectionNode):
                slow = slow.next
                interSectionNode = interSectionNode.next

            #Step3: Fine last node of loop
            while(slow.next != interSectionNode):
                slow = slow.next

            #Step4: Mark last node's next to None to remove the loop
            slow.next = None

            return head

        return False
```

💡 **Question 6**

Given a linked list and two integers M and N. Traverse the linked list such that you retain M nodes then delete next N nodes, continue the same till end of the linked list.

Difficulty Level: Rookie

Examples:

Input:

M = 2, N = 2

Linked List: 1->2->3->4->5->6->7->8

Output:

Linked List: 1->2->5->6

Input:

M = 3, N = 2

Linked List: 1->2->3->4->5->6->7->8->9->10

Output:

Linked List: 1->2->3->6->7->8

Input:

M = 1, N = 1

Linked List: 1->2->3->4->5->6->7->8->9->10

Output:

Linked List: 1->3->5->7->9

```
class Solution:
    def skipMdeleteN(self, head, M, N):
        # Code here
        curr = head
        while(curr):
            for count in range(1, M):
                if curr is None:
                    return
                curr = curr.next
            if curr is None :
                return
            t = curr.next
            for count in range(1, N+1):
                if t is None:
                    break
                t = t.next
            curr.next = t
            curr = t
```

💡 **Question 7**

Given two linked lists, insert nodes of second list into first list at alternate positions of first list. For example, if first list is 5->7->17->13->11 and second is 12->10->2->4->6, the first list should become 5->12->7->10->17->2->13->4->11->6 and second list should become empty. The nodes of second list should only be inserted when there are positions available. For example, if the first list is 1->2->3 and second list is 4->5->6->7->8, then first list should become 1->4->2->5->3->6 and second list to 7->8.

Use of extra space is not allowed (Not allowed to create additional nodes), i.e., insertion must be done in-place. Expected time complexity is O(n) where n is number of nodes in first list.

```
def mergeList(head1, head2):
    # Code here
    p_curr = head1
    q_curr = head2
    while p_curr != None and q_curr != None:
        p_next = p_curr.next
        q_next = q_curr.next
        q_curr.next = p_next  # change next pointer of q_curr
        p_curr.next = q_curr  # change next pointer of p_curr
        # update current pointers for next iteration
        p_curr = p_next
        q_curr = q_next
        head2 = q_curr
    return [head1, head2]
```

💡 **Question 8**

Given a singly linked list, find if the linked list is circular or not.

A linked list is called circular if it is not NULL-terminated and all nodes are connected in the form of a cycle. Below is an example of a circular linked list.

```
def isCircular(head):
    # Code here
    slow = head
    fast = head

    if not head:
        return 1

    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            return 1
    return 0
```