Day 15

Segrate Even and ODD in linked List

To split the linked list, traverse the Original linked list and move all odd node to a seprate linked list of all odd nodes. At the end of loop, the original list will have all the even nodes and the odd node list. To keep the ordering of all nodes Same.

Programme:

```
head = None
Class Node:
    def __init__(self,data):
        self.data = data
        self.next = None
```

```
def segrateEvenOdd():
    global head
    evenstart = None
    evenEnd = None
    oddstart = None
    oddend = None
    currNode = head
    while(currNode != None):
        val = currNode.data
        if(val % 2 ==0)
                                    None
            if(evenstart = currNode):
                evenstart = currNode
                evenend = evenstart
            else:
                evenend.next = currNode
                evenend = evenend.Next
        else:
            if(oddstart ==None):
                oddstart = currNode
                oddEnd = oddstart
```

```
        else :
            oddEnd.next = currNode
            oddEnd = oddEnd.next
    currNode = currNode.next
If (oddstart == None (or) evenstart == None):
    return
evenend.next = oddstart
oddend.next = None
head = evenstart

def push(self, new_data):
    global head
    node = head
    while (node != None):
        print(node.data, end = " ")
        node = node.next
    print()
```

Input :

```
push(11)
push(10)
push(9)
push(6)
push(4)
push(1)
push(0)

print("original linked list")
printlist()
segrateEvenOdd()
print("modified linked list")
printlist()
```

Output :

Original linked list

0 1 4 6 9 10 11

modified linked list

0 4 6 10 1 9 11

Time Complexity :

O(n)

Function to get nth node in linked list

* initalize count = 0
* loop through linked list
  * if count is equal to the passed index then return current node
    * Increment Count
    * Change current to point to next to point.

```
class Node :
    def __init__(self,data):
        self.data = data
        self.next = None

class linked list:
    def __init__(self):
        self.head = None
```

```
def push(self, new_data):
    new_node = Node(new_data)
    new_node.next = self.head
    self.head = new_node

def getNth(self, index):
    current = self.head
    count = 0
    while (current):
        if (count == index):
            return current.data
        count += 1
        current = current.next

assert(false)
return 0

If __name__ == '__main__':
    L = linkedList()
    L.push(1)
    L.push(4)
    L.push(1)
    L.push(12)
    L.push(1)
    n = 3
    print(l.getNth(n))
```

Output:

4

time Complexity:

O(n)