Day 13

Detect loop in linked list:

Traverse the list one by one and keep putting the node addresses in a Hash table. At any point, if Null is reached then return false and if next of current node points to any of the previously stored notes in hash then return True.

program:-

```
class Node:

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

```
class linkedlist:
    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 printlist(self):
        temp = self.head
        while(temp):
            print(temp.data, end = " ")
            temp = temp.next
    def detectloop(self):
        s = set()
        temp = self.head
        while(temp):
            if (temp in s):
                return true
            s.add(temp)
            temp = temp.next
        return False
```

Driver Program:

```
L = linkedlist()
L.push(20)
L.push(4)
L.push(15)
L.push(10)
L.head.next.next.next.next = L.head
If (L.detectloop()):
    print("loop found")
else:
    print("no loop")
```

Output:

loop found

Complexity Analysis:

O(n)

Auxilary space:

O(n)

Floyed cycle finding Algorithm:-

```
class Node:

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

class linked list:

    def __init__(self):
        self.head = ~~head~~ None

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

def detectloop(self):
    slow_p = self.head
    fast_p = self.head
    while(slow_p and fast_p and fast_p.next):
        slow_p = slowp.next
        fast_p = fast_p.next.next
        If slow_p == fast_p:
            return
```

```
def printlist(self):
    temp = self.head
    while (temp):
        print(temp.data)
        temp = temp.next
```

#driver code

```
L = linkedlist()
L.push(20)
L.push(4)
L.push(15)
L.push(10)
L.head.next.next.next.next = L.head
if (L.detectloop()):
    print("found loop")
else:
    print("NO loop")
```

Time Complexity : O(n)

Auxilary Complexity : O(1)

Find middle of linked list:

Count total number of nodes in the list.

Programme:-

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

class linkedlist:
    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 printmiddle(self):
    temp = self.head
    count = 0
    while (self.head):
        if (count &1):
            temp = temp.next
        self.head = self.head.next
        count += 1
    print (temp.data)

L = linkedlist()
L.push(1)
L.push(20)
L.push(100)
L.push(15)
L.push(35)
L.printmiddle()
```

Output:

100